A Methodology for Exposing Risk in Achieving Emergent System Properties

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

Determining whether systems achieve desired emergent properties, such as safety or reliability, requires an analysis of the system as a whole, often in later development stages when changes are difficult and costly to implement. In this article we propose the Process Risk Indicator (PRI) methodology for analyzing and evaluating emergent properties early in the development cycle. A fundamental assumption of system engineering is that risk mitigation processes reduce system risks, yet these processes may also be a source of risk: (1) processes may not be appropriate for achieving the desired emergent property; or (2) processes may not be followed appropriately. PRI analyzes development process artifacts (e.g., designs pertaining to reliability or safety analysis reports) to quantify process risks that may lead to higher system risk. We applied PRI to the hazard analysis processes of a network-centric, Department of Defense system-of-systems and two NASA spaceflight projects to assess the risk of not achieving one such emergent property, software safety, during the early stages of the development lifecycle. The PRI methodology was used to create measurement baselines for process indicators of software safety risk, to identify risks in the hazard analysis process, and to provide feedback to 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:
Lucas Layman, Victor R. Basili, and Marvin V. Zelkowitz. 2014. A methodology for exposing risk in achieving emergent system properties. ACM Trans. Softw. Eng. Methodol. 23, 3, Article 22 (May 2014), 28 pages.
DOI: http://dx.doi.org/10.1145/2560048

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. A major cost driver for these systems is their challenging nonfunctional requirements, such as safety, reliability, security, and performance. These *emergent properties* evolve during development and can only be fully tested when the system is complete. Corrective actions based on final tests or operations are difficult and costly to implement, nonetheless, failure to achieve these properties is costly and may be life-threatening at worst.

This research was supported by NASA OSMA SARP grant NNX08AZ60G to the Fraunhofer CESE.
Authors' addresses: L. Layman (corresponding author), V. R. Basili, M. V. Zelkowitz, Fraunhofer Center for Experimental Software Engineering, 2825 University Research Court Suite 1300, College Park, MD 20740-3823; email: llayman@fc-md.umd.edu.

DOI: http://dx.doi.org/10.1145/2560048

Emergent system properties are achieved through applying certain techniques, such as threat modeling for security [Torr 2005] or Failure Modes and Effects Analysis (FMEA) for reliability [Maier 1995; Lutz and Shaw 1999]. These techniques are wrapped in processes that guide their application to a project or within an organization. These processes must meet three 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.

In this article we define a risk measurement methodology for emergent system properties that goes beyond process conformance. The methodology leverages the assumption that processes meant to control risk, when inadequately or incorrectly applied (process risk), may lead to residual system risk (a product risk). The methodology is novel in that it focuses on the syntactic and semantic content of process artifacts to evaluate the preceding assumptions. Our contribution is also novel in that it can be applied *early* in the lifecycle because it is based on process, thus enabling systems engineers to measure, track, and respond to risks that an emergent property (e.g., safety, reliability, performance) will not achieve fruition during early phases of system development when the cost of change is manageable. The concept of risk used throughout this article describes process implementation failures that impact the ability of engineers to adequately perform traditional risk analysis and mitigation activities, thus increasing the probability that a failure will escape in the system.

We present our Process Risk Indicator (PRI) methodology with specific examples measuring *software safety* process risk on three case studies: the NASA Constellation project, a large U.S. Department of Defense network-centric system-of-systems, and a NASA Earth-observation satellite. We synthesize and expand on our previous publications[1] with the following new content: (1) the model behind our process risk measurement approach; (2) explicit guidance for each step of the methodology; (3) results from the NASA satellite study; (4) a side-by-side comparison of the application of the methodology to the three case studies.

The remainder of this work is organized as follows: Section 2 presents background and related work in risk management and process risk; Section 3 introduces the model of process risk measurement that underpins the PRI; Section 4 describes the steps of the PRI with explicit examples from our case studies; Section 5 presents common process risk themes we uncovered in our case studies and lessons learned for implementing software safety risk analysis methodologies on future programs; and we conclude in Section 6.

2. BACKGROUND AND RELATED WORK

To frame our discussion, we define two types of risk with respect to emergent properties: product risk and process risk. A *product 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 that causes a rocket to prematurely activate its launch booster is a product risk that is also a safety risk. A *process risk* is the risk created by the correct or incorrect application of a development process that leads to a product risk. For example, a software tester who writes poor

[1] In a previous publication [Layman et al. 2011], the PRI was named the Technical and Process Risk Measurement (TPRM) method. The steps of the methodology remain the same.

performance tests contributes to the product risk that the system may not meet performance requirements. All software development processes introduce additional risks that must also be controlled.

2.1. System and Software Risk Management

The specific sources of software risk are varied and well studied [Boehm 1991; Ropponen and Lyytinen 2000; Wallace and Keil 2004; Han and Huang 2007], and range from factors such as technical complexity [Boehm 1991] to management skills [Wallace et al. 2004]. In small-scale software development, the risk management process is often embedded in the software development process (e.g., defect tracking, burndown chats). In complex systems and systems-of-systems, the whole of project management is an instantiation of risk management [Charette 1996] with many risk management processes for different risk areas such as cost, schedule, hardware, software, technology, and personnel [Higuera and Haimes 1996]. These can be further broken down into areas such as software safety, software reliability, security, performance, quality, etc.

Numerous software risk management approaches exist (e.g., Boehm [1988], Higuera and Haimes [1996], and Kontio et al. [1998]) and commonly have some form of: (1) risk identification; (2) risk analysis; (3) risk planning; (4) risk mitigation; and (5) risk tracking. For large systems, different processes will be used to identify and analyze specific types of risk. *Hazard analysis* is one such process that can be used to identify, analyze, and design mitigations for safety risks (see Appendix A for a brief description of hazard analysis). Product risks associated with emergent properties such as safety are challenging to manage because such properties are a function of the system as a whole. Testing these properties with a high level of confidence before the system is completed is difficult, if not impossible. Furthermore, methods for assessing some properties (e.g., security, safety, performance) are often ill defined, immature, or nonrepeatable.

Software safety risks are an increasing concern in systems development as many traditionally hardware-centric systems become reliant on software for command and control. Catastrophic safety failures due to software are well documented in the literature, for example, Nuseibeh [1997] and Leveson and Turner [1993]. Much prior research has focused on software safety analysis methods (e.g., Leveson [1995] and Schneidewind [1997]) or certifying software safety (e.g., Lutz and Woodhouse [1999] and Panesar-Walawege et al. [2011]). To our knowledge, our work represents the first published methodology to quantify software safety process risks.

2.2. Process Risk and Process Conformance

A common challenge (risk) faced by project managers, particularly for large systems, is having confidence throughout the lifecycle that the risk analysis and mitigation processes are being followed and/or will achieve the desired property (e.g., safety). The purpose of our proposed process risk analysis methodology is to address this challenge.

Process risk mitigation often takes the form of quality assurance, process conformance evaluations, or the application of process improvement frameworks such as CMMi [Chrissis et al. 2011] and QIP [Basili and Rombach 1988]. Identifying the cause of process risk is where many 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). Indeed, research has shown that teams who follow the process consistently tend to perform better [Krishnan and Kellner 1999; Gibson et al. 2006]. But research has also shown that "even companies that use the same development process at the same level of maturity do not achieve the same levels of quality" [Pardo et al. 2011]. There are often reasons for process nonconformance, for example, the process is flawed or the context does not allow various steps to be performed (see *Assumptions 1* and *2)*. For example, developers may skip inspections because of tight project schedule deadlines, thus creating a process risk of not achieving reliability. An organization's performance testing guidelines may be ill specified and uninformative, and thus the performance testing process is ad hoc and ineffective. The goal of our PRI methodology is not to test conformance in the traditional sense, but instead to baseline process risk measures using process artifacts and, in the event that we cannot form these baselines, to understand why the process has failed to produce useful information.

Hazard and Operability analysis (HAZOP) is a technique for formally reviewing processes and studying process deviations that may contribute to safety hazards. HAZOP is complementary to hazard analysis that examines operational processes (e.g., the process for loading fuel into holding tank). HAZOP and variations of this technique have been applied to software models [McDermid et al. 1995; Reese and Leveson 1997]. While HAZOP analyzes process deviations, the focus of our methodology is to identify incorrect application or unsuitability of techniques such as HAZOP within the development context.

Our proposed methodology is most similar to the technique of process mining. "Process mining aims to discover, monitor, and improve real processes by extracting knowledge from event logs readily available in today's information systems" [van der Aalst 2012]. An IEEE Task Force on Process Mining has been established to foster this work (http://www.win.tue.nl/ieeetfpm/). Various methods have been proposed that use graphical analyses to ensure that the right process steps are being followed (e.g., Mendling et al. [2007] and Panesar-Walawege et al. [2011]) or measure changes in the product artifacts (i.e., source code) to understand the processes being followed (e.g., Nagappan et al. [2006]). But this assumes that with the ubiquity of terabyte stores, everything is being collected to permit relevant information to be extracted. We still cannot ensure that the right information is being collected as represented in these event logs. These still remains the need to understand what is being collected and how it is being used, not just that the correct steps are seemingly being followed.

3. A MODEL OF PROCESS RISK MEASUREMENT

Our approach to mitigating those risks associated with emergent properties focuses on *process risk management*. In this section, we present our model of process risk measurement.

3.1. The Process Risk Measurement Model

We assume that the properties of a product at time *t* are a function of at least four factors (Figure 1): (1) the functional and nonfunctional system requirements that describe the desired product properties; (2) the process or sequence of steps performed to achieve the desired properties; (3) the people who perform the process, for example, whether they are applying the process properly; and (4) the context in which the process is applied, which includes the influence of resource constraints, technologies, system type, etc. So we say that

$$\text{Property}_t(\text{Product}) = \varphi_t(\text{Requirements, Process, People, Context, ...}).$$

We assume that when the process is applied correctly to a set of requirements using the appropriate people and context, then the desired product property is more likely to be achieved. This is the same assumption the creators and implementers of the engineering processes make, that is, follow this safety analysis process to make the system safer. Conversely, we assert that the risk of not achieving the desired emergent property can occur for three process-related reasons[2].

[2] However, the properties may be achieved by other unintentional means. Furthermore, there are many nonprocess risks that may contribute to achieving a desired property [Boehm 1991].

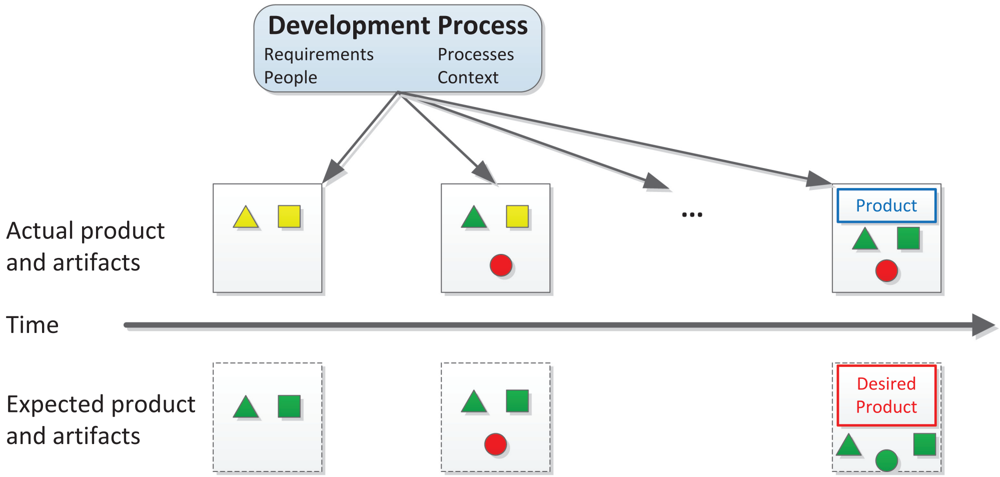

Fig. 1. A development model of the relation between process and outputs.

(1) The process is incorrectly applied, that is, the people do not follow the process.
(2) The process is intrinsically flawed, that is, the process would not achieve the property even in an optimal scenario. For example, a safety analysis process that lacks sufficient specification and thus cannot be followed correctly and consistently is intrinsically flawed.
(3) The process is not appropriate given the context. For example, a safety analysis process designed to analyze hardware is applied to software, or ad hoc testing is expected to find all defects in a safety-critical application.

The functional relationship between the requirements, processes applied, people, and context (the φ function) is often inscrutably complex, especially in large systems development. To gain insight into these relationships and to mitigate process risk, we must first determine what information can be gathered that will make the lack of process conformance visible, provide insights into the effectiveness of the process itself, and provide feedback for process and conformance improvement.

In general, a process is implemented to control some risk. A deviation in the process implies that the process may not succeed in controlling the risk—this assumption is the heart of quality assurance. The preceding model (and the PRI methodology) does not measure system risk directly, but rather indicates risk potential caused by the incorrect application or insufficiency of a process to mitigate a system risk.

3.2. Leveraging Process Artifacts to Identify Process Risk

We focus on the *process artifacts* that contain information reflective of both the process and the product property that we are trying to achieve.

The artifacts of a development process contain two types of information that provide insight into the process: syntactic information (i.e., a schema) and semantic information. An artifact's *syntactic information* is its 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 criticality of the defect. If these elements are missing, there is a risk that the defect tracking process is not being performed correctly. An artifact's *semantic information* is the meaning or interpretation of the syntactic data in the context of development. Interpreting semantic information frequently requires human expertise. For example, a computer cannot automatically infer the risks and mitigating actions required based on the contents of a defect report, whether the right information is in the defect report can only be determined by human expertise. Automated analysis of process artifact semantics has been demonstrated in research on bug triage [Anvik et al. 2006] and requirements analysis [Wilson et al. 1997].

We assert that the potential for 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 indicate whether or not steps have been taken to mitigate product risks and that these steps have been applied appropriately. Insufficient, incorrect, or missing syntactic information in process artifacts is an impediment to accurate, useful semantic analysis and is the indicator that we leverage to identify and measure potential risk during the development process.

Given the model in Figure 1, we evaluate the potential process risk by analyzing the process artifacts produced during development, such as source libraries, defect databases, resource expenditures, and other documentation. If the syntactic and semantic content of the intermediate outputs of the development process agree closely with the expected outputs, then we have reason to believe that the process is being followed closely and that it is appropriate for the development context. Alternatively, a divergence in values indicates a higher probability that process risk is contributing to product risk and a mitigation strategy needs to be employed. Note that there is no overhead on the project personnel to produce additional information, as PRI only looks at artifacts that are existing by-products of the lifecycle processes that are being followed.

Our process risk measurement methodology is founded on the following questions to identify and measure sources of process risk due to nonconformance, flaws in the process, or the applicability of the process to the context.

(1) Is there any information in the process artifacts?
(2) Is there enough information to perform a syntactic analysis of the data?
(3) Is there enough information to perform a 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 deeper understanding of the risks that may be present.

4. THE PROCESS RISK INDICATOR METHODOLOGY

In this section, we describe the six steps of our Process Risk Indicator (PRI) methodology. We explain each step in detail and provide examples of applying the methodology 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 these risks.

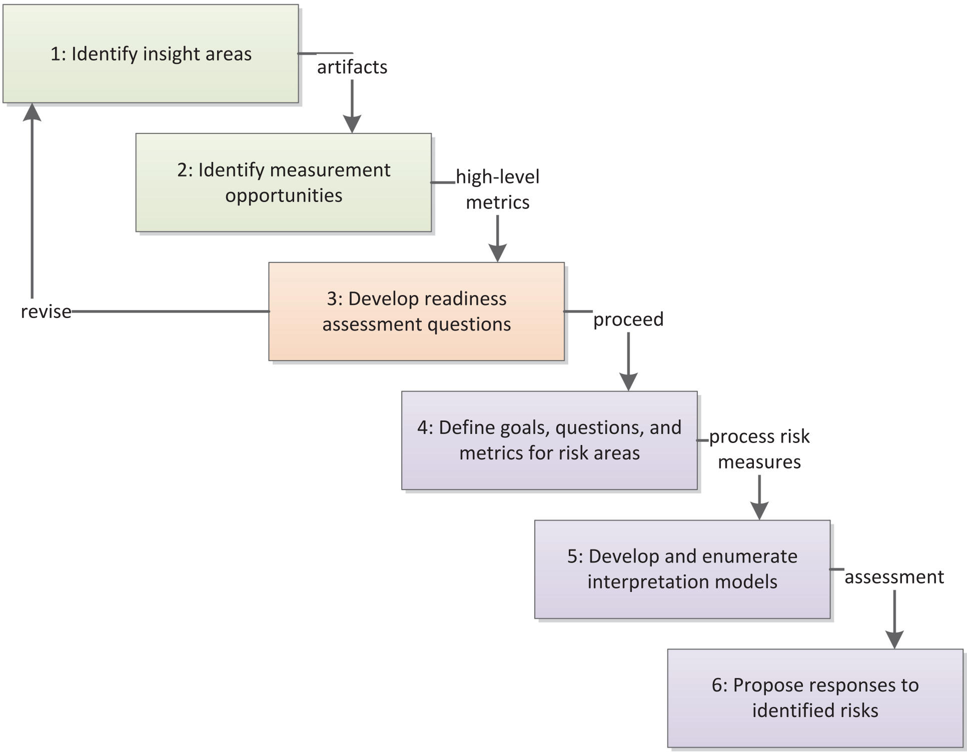
1: Identify insight areas
artifacts
2: Identify measurement opportunities
high-level metrics
3: Develop readiness assessment questions
revise
proceed
4: Define goals, questions, and metrics for risk areas
process risk measures
5: Develop and enumerate interpretation models
assessment
6: Propose responses to identified risks

Fig. 2. The PRI methodology.

The PRI is a process risk measurement methodology that defines measures of the syntactic and semantic quality of process artifacts (Figure 2). The first three steps determine if the development process is being followed and if the process is producing intermediate outputs that will help to achieve the desired emergent system property. If insight areas or measurement opportunities cannot be identified (steps 1 and 2) and the artifact's syntactic and semantic information cannot be readily assessed (step 3), this is an indicator of development process risk. The Goal-Question-Metric (GQM) approach [Basili and Weiss 1984] is then applied (step 4) to create measures of the syntactic and semantic quality of the artifacts. Measurements are collected and evaluated through interpretation models (step 5) to assess the risk of not achieving the emergent property. Finally, responses to identified process risks are proposed (step 6).

The PRI methodology is descriptive rather than prescriptive. Our goal is to enable users to discover process risks particular to their projects, processes, and organizations. Artifacts and measurements based on artifact content will be particular to the processes used by an organization. Some measures may be reusable if similar processes are used. For example, the case studies that follow describe the use of hazard analysis for software safety in three different projects in two organizations, and thus some measures are reused. However, different projects and organizations have different thresholds for risk severity and different criteria for what constitutes a development risk. Instead of prescribing severities or risk interpretations, the methodology (in steps 5 and 6) describes how to define project- or organization-specific risk thresholds and responses.

4.1. Case Studies

To illustrate the PRI method, we provide examples of applying PRI to identify potential software safety risks to three systems: the NASA Constellation program[3], an unmanned NASA satellite, and a large, network-centric DoD system-of-systems. These systems involve numerous hardware and software safety features. Safety is an example of an emergent system property that cannot be fully tested until the system is operational. Safety analysis typically focuses on hazards, which are "real or potential condition[s] that could lead to an unplanned event or series of events (i.e., mishap) resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment" [Department of Defense 2012]. Safety risk is the combination of the severity of a mishap and the probability that it will occur [Department of Defense 2012].

We applied the PRI method to the *hazard analysis process* for each program. Hazard analysis is briefly described in Appendix A. In all case studies, hazard analysis was carried out by subcontractors to the government organization. In the case where hazards involved software concerns, the hazard analysis was either performed by software engineers or carried out by safety engineers in consultation with software experts.

DoD program. The DoD system was a very large, complex, network-centric system-of-systems with a large number of software safety risks that had to be tracked and mitigated before the system was deployed [Basili et al. 2008]. In this multibillion-dollar system, software was critical in assuring the safety of DoD personnel[4]. The development process was expected to follow the traditional defense acquisition V-model and safety process [Department of Defense 2012]. Safety risk management was integrated throughout the multiyear lifecycle. Our goal was to develop an approach that provided the lead 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 PRI early in the DoD program's lifecycle, when only requirements, preliminary designs, and the safety processes were available.

The DoD program was our first application of our risk measurement methodology. This case study was unique in that we performed all of the steps up-front, before looking at the data. At this time, step 3 was not included in the methodology. In this aspect, we learned firsthand the importance of applying steps 1–3 iteratively and in situ. Though measurements could not be completed, we include this case study to illustrate the importance of steps 1–3 in defining a measurement program and in highlighting process risks.

Constellation. The Constellation Program (CxP) is a complex system-of-systems to facilitate the next generation of human spaceflight at NASA. Software was safety critical in propulsion, navigation, maintaining a livable environment inside the crew capsule, and many other areas. In this case study, our goal was to provide insights into software safety during the *preliminary design phase*, when the first concrete designs were created after system requirements had been established.

We partnered with the software CxP Safety, Reliability and Quality Assurance (SR&QA) office to understand how software impacted system safety, either by causing hazardous conditions or preventing them. By understanding the role of software in system safety, SR&QA could identify systems and subsystems with the greatest potential software safety risk and take appropriate management actions. At each program milestone review, the development groups presented their safety analyses to the

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

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

Table I. Step 1: Identify Insight Areas

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?	

Constellation Safety and Engineering Review Panel (CSERP), who reviewed the risk mitigation design strategies, acting as gatekeeper for development milestones. The results of the Constellation case study are more fully documented in Layman et al. [2011].

NASA satellite project. The NASA satellite is a joint project with foreign space agencies. Because satellites are unmanned, the safety risks are fewer in number than on a manned mission. Most of the safety risks occur during the prelaunch and launch when explosions, structural collapses, or the release of toxic fuels can 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 on-board instruments. We applied the PRI method to the hazard analysis process during the *critical design phase* of development where the system design is reviewed and approved so that fabrication can proceed.

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

The first step of the PRI method is to identify the intermediate and final 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 high-level insight areas for further investigation (Table I).

Case study examples. On the NASA and DoD projects, software assurance personnel identified the hazard analysis process as readily accessible sources of software safety information early in the development lifecycle. The hazard analysis process is used to identify causes of hazardous safety conditions and to develop mechanisms to avoid, control, or otherwise mitigate those causes[5] [Federal Aviation Administration 2008]. The official hazard analysis processes for the three programs were described in system safety plans that required the software be analyzed for safety-critical procedures that

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

Table II. Step 1: Insight Areas for the Case Studies

	DoD	Constellation (CxP)	Satellite
Source of data	Hazard tracking system	Hazard tracking system	Hazard report documents
Insight areas	• The set of hazard, causes, controls, and verifications • The relations between hazard causes, controls, and verifications • The set of safety requirements resulting from hazard analysis		

could lead to a potential hazard. *Hazard reports* are the artifacts of the hazard analysis process and are iteratively refined throughout the lifecycle. The final hazard report contains a description of a hazard, the potential *causes* of that hazard, *controls* (design or operational features) for preventing or responding to the cause, and *verifications* with signatures that the controls have been implemented and tested. Preliminary hazard analysis is conducted in the requirements phase, and further details on causes, controls, and verifications are added throughout the lifecycle and evaluated at milestone reviews. In Constellation and the DoD system, all hazard reports were stored in a Hazard Tracking System (HTS), whereas the NASA satellite used a shared file system to collect hazard reports. The hazard reports contained the syntactic and semantic safety information that provided insight into potential software safety risks. Based on 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 into 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, that is, that the hazard controls are being implemented and verified.
—*The set of safety requirements resulting from hazard analysis*. The safety requirements must be implemented to avoid hazardous conditions, and must be tracked and verified to closure.

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

The second step of the PRI is to evaluate each insight area from step 1 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 (e.g., the number of defects found during test). A measurement opportunity is identified by creating a high-level, informal metric based on the information available. The purpose of this step is to identify measurable artifacts and to exclude processes or process artifacts that do not have measureable information. A process that does not have measureable output may represent a risk that the process or process artifacts do not contain useful information.

Case study examples. We identified measurement opportunities that focused on syntactic information in the hazard reports 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).

These measurement opportunities included the following.

—Evaluate hazard causes, controls, and verifications to determine if they contain the syntactic components that are required for a "meaningful" hazard analysis.
—count the number of hazards, causes, controls, and verifications that involve software to quantify software involvement in hazardous conditions.

Table III. Step 2: Identify Measurement Opportunities

Step 2: Identify measurement opportunities	
Inputs • Process outputs/artifacts identified in step one	Outputs • 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?	

Table IV. Measurement Opportunities Identified for the Case Studies

Measurement opportunity	DoD	CxP	Satellite
Hazard cause, control, verification syntax	X	X	X
Count causes, controls, verifications involving software	X	X	X
Cause → control → verification mapping		X	
Count the number of "transfers"		X	X
Safety requirement → hazard mapping	X	X	X
Safety requirement syntax	X	X	X

—verify that each cause is mitigated 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); for example, a cause in Hazard A is indicated as addressed by the controls in Hazard B; all transfers must be traced and verified as completed for a hazard report to be considered "closed."

—verify that each hazard cause is mapped to one or more safety requirements.

—evaluate the safety requirements to determine if they contain the syntactic components to sufficiently describe the required software safety feature.

For the three programs, the hazard analysis processes and organizational standards described the syntactic components that were elements of a semantically "meaningful" description of hazard causes, controls, and verifications. For example, official guidance documents indicated that language such as "Component X issues a command that prematurely activates Subsystem Y, resulting in premature thrust" is meaningful, whereas "Component X fails" is not helpful from a safety analysis perspective. We also assumed that we could easily search for keywords such as "software" or "computer" to help identify software-related causes, controls, and verifications. For Constellation, the mapping and transfer traceability requirements were derived from requirements in the Constellation program hazard analysis methodology [National Aeronautics and Space Administration 2009], but no such requirements were defined for the DoD program. Traceability between safety requirements and hazard reports were process requirements of all three projects.

The purpose of the first two PRI steps is to understand the development process we wish to measure and to understand the process artifacts. In these steps, we have not gathered any actual data; we are only looking at the process outputs and their potential to be measured. 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.

Table V. Develop Readiness Assessment Questions

<table>
<tr><td colspan="2">Step 3: Develop readiness assessment questions</td></tr>
<tr><td>Inputs
• Proposed measurement opportunities and the associated risks they measure</td><td>Outputs
• 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?</td></tr>
<tr><td colspan="2">Activities or Questions to ask
• Examine the process artifacts and to determine if the proposed metrics can be applied:
◦ Is the information accessible and available?
◦ Is the information in a 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?</td></tr>
</table>

4.4. Step 3: Develop Readiness Assessment Questions to Provide Risk Status and Identify if the Insight Area Can Be Evaluated

For step three, we develop a set of *readiness assessment questions* to determine whether the syntactic and semantic information in the process artifacts contain sufficient information to investigate the risk further. These questions focus on the ability to conduct measurement on process artifacts and, if not, why not. For example, we might ask "are software safety-related requirements identified in the requirements repository?" If the artifact is not available, the data is incomplete, or the data's syntax does not meet the specification, there is a risk that the process is not being followed or the process is not appropriate for the project. There may be insufficient information to perform even simple measures. Failing to pass readiness assessment is an indicator that the process may not achieve the desired emergent property, and the project should take steps to correct the development process.

The readiness assessment questions should be specific instantiations of the questions in Table V for the measurement opportunities from step 2. Answering readiness assessment questions is an iterative process, as answering one question may lead to others. The questions are specific to the process, organization, and context of each project as the process artifacts will vary from project to project.

Case study examples. We developed many readiness assessment questions while exploring the measurement opportunities in our case studies. Next we provide examples of readiness assessment questions that were applied to each study. These questions were applied to both the hazard reports and safety requirements.

—*Can we access the hazard report and safety requirement repositories?* Access to the IT systems used to store the hazard reports and safety requirements is a necessary first step to analyzing the artifacts. Often, such systems are the property of a contractor and not available to the developing organization.

—*Can we access the hazard reports and safety requirements for the entire system?* Access to the process artifacts is a necessary precondition to formal measurement.

—*Is the content of the repositories up to date?* If the hazard reports or safety requirements are not up to date, then measurements will not reflect the current state of the system.

—*Are hazard reports and safety requirements complete enough for analysis?* If the artifacts are incomplete, then an evaluation of the syntactic and semantic information in the hazards will be limited.

For the DoD system and Constellation, we were granted access to the projects' hazard tracking systems, whereas the NASA satellite project did not have a centralized IT system to manage hazards. While the satellite hazard reports were accessible, hazard reports for some systems in the DoD and Constellation projects were not available. However, we could not gain access to the requirements repository or obtain copies of the safety requirements for the Constellation project, thus eliminating the possibility of examining these artifacts. In this case, our lack of access was not due to a process-related risk (such as not having safety requirements), but rather because of contract specifics granting us access to this repository. We did not examine safety requirements on the Satellite project due to time constraints.

The NASA satellite project was more mature in its development lifecycle than the other systems and had complete, up-to-date hazard reports. In Constellation, the hazard tracking system contained up-to-date hazard reports for only one of the three major spaceflight systems we examined. One system's hazard reports were in the HTS but not visible because the development contractor did not want to release "intermediate" versions of the hazard reports and were not required to by the contract. Another system's hazard reports were created prior to the development of the hazard tracking system itself. We spent significant effort collecting the remaining hazard reports from a Web site containing CSERP review meeting materials.

The hazard reports were not complete for the Constellation and the DoD systems. For Constellation, causes and controls were specified, but verifications were not yet entered in the HTS. However, there was an acceptable process justification: the goal of hazard analysis in the preliminary design phase of Constellation was to identify and describe all potential failure causes and to develop preliminary controls. Verifications were not yet specified nor required [National Aeronautics and Space Administration 2009]. As such, we could not measure them and consequently adjusted our measurement opportunities.

In the DoD system, we found that the HTS contained little usable data. The HTS was viewed as a storage repository rather than an analysis tool by the engineers, that is, something to be used as a library after the fact rather than a tool to support safety understanding. Hazard data was missing, difficult to extract because of a multitude of formats, and no automated syntactic analysis was available. The DoD safety requirements repository had similar issues, preventing us from being able to even identify software-related hazards and requirements analysis, much less present a system-wide picture of software safety process risk. There was not sufficient information to identify a sample set software hazards and safety requirements, for example, at the time we were examining the repository, very few hazards were reported as software related even though there were very many causes or controls that required software. The entire collection of hazards and requirements was simply growing too large (already several hundred hazards) for us to perform a manual analysis given our available resources.

While these readiness assessment questions helped determine if we should go further in our analysis, they also highlight some hazard analysis process risks in these three programs. Keeping the HTS up to date is an easy first step to increase the effectiveness of the hazard analysis process and mitigate process risks. 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, we found that one contractor withheld intermediate hazard information from the HTS on Constellation, thus preventing SR&QA

Table VI. Examples of Readiness Assessment Questions Used in All Three Case Studies

Readiness assessment question	DoD	CxP	Satellite
Can we access the hazard tracking system?	Yes	Yes	N/A
Can we access the hazard reports?	Partially	Partially	Yes
Is the content of the hazard tracking system up to date?	No	Partially	Yes
Are the hazard report data complete enough for analysis?	No	Partially	Yes
Can we access the safety requirements repository?	Yes	No	-
Can we access the safety requirements?	Partially	No	-
Is the content of the requirements repository up to date?	No	-	-
Are the safety requirements complete enough for analysis?	No	-	-

from easily assessing the progress of the hazard analysis process in a timely way. As previously mentioned, we do not prescribe severities or other risk categorizations of these risks, instead leaving it up to the development organization to determine their severity. In the Constellation and DoD systems, these issues were raised with program management, who agreed to take steps to solve the issues (see Basili et al. [2010, 2008] for more details on responding to these risks).

When applying step 3 to the DoD case study, we found that we could not continue further analysis. At the time, however, we first went through all of the steps of the PRI *before* we had the opportunity to look at the data. As a result of this application of PRI, we modified the PRI method to be an iterative process where steps 1–3 were performed (sometimes repeatedly) prior to steps 4–6. Because no actual data is available from steps 4–6 for the DoD system, it is omitted from further examples.

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

Having determined in steps 1–3 that we have sufficient information to make deeper risk assessments, in the fourth step we apply the Goal-Question-Metric (GQM) approach [Basili and Weiss 1984; Basili and Rombach 1988] to identify specific software safety process risks based on the syntactic and semantic information available in the development process artifacts. Goals and questions should focus on the application of the development process and the expected information in the process artifacts. 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?” The purpose of this step is to determine if the process artifacts meet expectations (i.e., process conformance) and to help evaluate the quality of the artifacts (i.e., the information is useful). If not, there is risk that the process is not appropriate for achieving its goals or that the process is not being applied appropriately. 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.

Case study examples. On the Constellation program, one of the main objectives was to identify systems and subsystems with the greatest potential software safety risk. After applying PRI steps 1–3, we worked with SR&QA to specify two GQM goals that, once again, focused on identifying deviations from the prescribed safety analysis process that lead to residual safety risk due to inadequate analysis. The first goal was to quantify the prevalence of software-related hazards, causes, and controls (Table VIII). The second goal was to evaluate whether the syntactic and semantic information in software causes of hazards adhered to established NASA guidelines (Table IX). If

Table VII. Step 4: Define Goals, Questions, and Metrics

<table>
<tr><td colspan="2">Step 4: Define goals, questions, and metrics</td></tr>
<tr><td>Inputs
• A set of proposed metrics that have passed the readiness assessment check</td><td>Outputs
• A GQM structure with specific goals, questions and metrics</td></tr>
<tr><td colspan="2">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?</td></tr>
</table>

Table VIII. Goal 1 - Prevalence of Software

Goal 1 - Prevalence of software Analyze the available set of hazards reported for the 3 CxP projects 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 CxP 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. Q1-5. What percentage of causes is transferred? Q1-6. What percentage of controls is transferred? Q1-7. What percentage of controls are references to “generic” software hazard reports?
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 M1-6. The number and percentage of transferred causes M1-7. The number and percentage of transferred controls

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.

The objectives for measurement on the NASA satellite project were the same as with Constellation, but with the added research goal of assessing the applicability of the measures to a more mature project with vastly different safety requirements. Whereas the Constellation program was concerned with safety issues in all phases of the mission, the unmanned satellite focused mostly on ground safety issues (e.g., premature deployment of solar panels on the ground could injure personnel). The safety analysts themselves brought a perspective focused on robotic mission safety issues, whereas the safety analysts for Constellation had broader expertise in human

Table IX. Goal 2 - Specificity of Software Causes

Goal 2 - Specificity of software causes: Analyze the *software causes* in a sample set of hazard reports for the 3 CxP projects 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 CxP 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 where: • L1: a specific software cause or sub-cause[6] 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.

spaceflight safety. Due to these factors, the satellite hazard reports were written using different syntax and semantics language that represented a different perspective on system safety. A reusable set of metrics applicable to a variety of missions would be more useful to NASA.

In Goal 1, the intent from the perspective of software safety risk is to understand the extent to which software is involved in system hazards.

One risk metric is counting the *transferred* causes, controls, and verifications. During system implementation and test, all transfers must be verified for hazard reports to be considered "closed." Verifying transfers is a manual, labor-intensive process that was considered error prone and costly to assure. Furthermore, too many transfers were considered a bad application of process, whereas too few was an indicator of process inefficiency.

The so-called *generic software hazard report* is a more direct indicator of potential software risk. Each Constellation system had a small number of "generic" software hazard reports, which describe only the procedures for how software should be developed, but do not describe a mitigating design or operational control. Instead, the control was, "we will mitigate this risk by building the software correctly", which is not a verifiable statement. Some software causes had a single control referring only to these "generic" reports rather than a specific design attribute. These controls represent risk in that there is no objective verification that a software cause has been controlled by adhering to the software process.

[6] Many software causes contained a number of "subcauses." Subcauses 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 subcauses 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 is inferred from NASA standards and enables meaningful semantic information though it does not guarantee it.

4.6. Step 5: Develop Interpretation Models and Threshold Values of Measures

Using the measures developed in step 4, we need to interpret the data. What values indicate low risk for this process and which values represent risk? For example, for Goal 2 given earlier, 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, for example, what has been the normal value (e.g., mean, median) for projects of this type and what is the range around this norm that has been considered safe (e.g., one standard deviation). If no such data exists, we must make an assumption about the expected value and range, for example, experts can provide proxies projects or informed estimates for the thresholds. In this case we note that the relevance of these models is only an assumption and must be checked as the measurement process continues. The estimate of any expected value or range should be improved over time based upon new information. One side-effect of the PRI method is that the collected data can be used to create baselines that can be used as thresholds in future projects.

<table>
<tr><td colspan="2">Step 5: Develop interpretation models and define threshold values</td></tr>
<tr><td>Inputs
• A set of goals, questions and metrics to be collected</td><td>Outputs
• A set of models that provides indication that there may be a risk</td></tr>
<tr><td colspan="2">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 expected value or bounds?
∘ Are there proxies for the expected values and bounds on these metrics?
∘ Can we gather expert opinions for the expected values and bounds?</td></tr>
</table>

Case study examples. We analyzed a total of 154 hazard reports for the three Constellation systems: 77 in System A, 57 in System B, and 20 in System 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. The NASA satellite had 23 hazard reports with a total of 93 causes, 250 controls, and 445 verifications. These hazard reports differed from Constellation in that they were from later in the lifecycle and provided more detailed descriptions of hazard controls and verifications.

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 X provides the statistics that answer the questions in Goal 1.

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

Table X. Measuring the Prevalence of Software

		Constellation			Satellite
	Question	Sys A	Sys B	Sys C	
Q1-1	What percentage of the hazard causes are software causes?	15%	12%	17%	5%
Q1-2	What percentage of the hazards is software-related?	49%	67%	70%	26%
Q1-3	What percentage of the causes has software controls?	29%	23%	n/a	20%
Q1-4	What percentage of hazard causes are hardware causes with software controls?	14%	11%	n/a	15%
Q1-5	What percentage of causes is transferred?	31%	23%	36%	0%
Q1-6	What percentage of controls is transferred?	22%	11%	n/a	0%
Q1-7	What percentage of controls are references to “generic” software controls?	5%	2%	n/a	0%

Table XI. Interpretation Models from the Case Study Projects

	% of software causes				
Rating	Sys A	Sys B	Sys C	Satellite	Interpretation model
L1	50%	38%	0%	40%	*if % of L1 = 100%* ***then*** the software causes contain the necessary semantic information to evaluate the safety risk and no further specificity evaluation is required; ***otherwise*** domain experts must determine if the causes are adequately specified or if more work is required.
L2	20%	40%	71%	60%	*if L2 > 0* ***then*** some software causes are missing the syntactic information required by the process and the evaluation of safety risk by domain experts may be inaccurate; ***therefore*** domain experts must determine if the causes are adequately specified or if more work is required.
L3	30%	22%	29%	0%	*if L3 > 0* ***then*** additional syntactic information is required to accurately identify and understand software safety risk; ***otherwise*** the causes are ready to be evaluated for quality by domain experts.

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 controls and verifications. For the satellite project, only 22% of hazards were software related. These numbers are more in line with the expectations of NASA personnel.

Goal 2 provided insight into the execution of the hazard analysis process itself. In this example, we create an interpretation model for the questions Q2-1 – Q2-3.

The threshold of $L3 > 0$ was selected by the local experts because there should be no causes or subcauses that are generically defined and do not specify the origin, erratum, or receiver at the CSCI/hardware subsystem level. A “generically defined” cause is not specific enough to specify a design feature or operational procedure that could function as a control strategy and so cannot be verified. Thus, all generically defined causes imply risk and should be improved to be better specified. Conversely, if all causes are L1 causes, then we have confidence that sufficient syntactic information is present in the description of the safety risk for domain experts to evaluate the risk and implement appropriate mitigation strategies.

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 earlier, the initial reaction may be to say "you're not following the process correctly." However, SR&QA personnel acknowledge that nonconformance 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 being developed. 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.

4.7. 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.

If not risks are identified in this step, this 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, product 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.

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	

Case study examples. First, a note on the satellite project: we received feedback that the metrics analysis provided an accurate picture of software safety concerns on the project. However, due to the late phase of system development and the nature of the contract (the satellite involved international cooperation for analysis and construction), no changes to the system engineering process could be made at that point in time. While our analysis suggested no significant risks in the project, this situation underscores the importance of integrating the PRI method early in system development when the cost of change is relatively low.

On Constellation, SR&QA integrated the percentage of software-related hazards from Goal 1 and the specificity of hazard causes into a risk scorecard of the systems and subsystems with greatest potential software safety risk (Table XII). The risk scorecard was to be used to track the evolution and improvement of software safety risk over the course of the project. Low-(green), medium-(yellow), and high-risk (red) colors gave a quick assessment of an overall risk metric for project systems. As the project matured, the metrics on the scorecard would, hopefully, change for the better, and the thresholds for what constituted a low-, medium-, or high-risk value would also

Table XII. Excerpt from Constellation's SR&QA Risk Communication Scorecard – Values are Notional

Constellation – Preliminary Design Phase development						
(Sub)System	Software Safety Hazard Analysis			Reliability FMEA	PRI	Quality Audits
	# of s/w Controls	# of s/w causes	Median specificity of causes	# of s/w failures	S/W high risk areas	Major findings
System A	243	58	L1	256	14	Missing traceability
Propulsion	35	12	L1	33	3	None
Avionics	59	13	L2	42	N/A	Incomplete HA, PRI

change. For example, the L2 median specificity of causes was deemed a medium risk during Phase B, but during Phase C (detailed design and implementation), this would be considered a high risk because all hazard causes must be well defined according to the Constellation development process. In the scorecard example, the thresholds from step 5 were used to guide the risk value for Goals 1 and 2, whereas the values for other items in the scorecard were determined using other methods appropriate for the process (e.g., guidance documents, expert opinion, statistical analysis, probability of failure).

The data related to Goal 1 were used to inform other members of Constellation program management of the importance of software in overall system safety. The results were surprising to many. 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 how project responses might look.

—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 how to integrate software concerns into the hazard analysis 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 the following.

—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 these causes rewrite them using the "User Guide for Specifying Software Causes" we produced with SR&QA as guidance and re-evaluate at the next Technical Interchange Meeting.

5. UNCOVERING PROCESS RISKS USING THE PRI METHOD – CASE STUDY RESULTS AND LESSONS LEARNED

We applied PRI to the hazard analysis process to assess process risks that could lead to not achieving software safety during the early stages of the development lifecycle of three projects: (1) the Constellation study in the previous section; (2) a large, network-centric U.S. Department of Defense system-of-systems; and (3) a NASA-developed Earth-observation satellite. We uncovered a number of process risks in each project with respect to the application of hazard analysis to software safety[7]. In this section, we identify four process risk "themes" that were common across the projects and offer guidance to other programs implementing software safety analyses to help avoid these risks.

5.1. Common Process Risk Themes

These themes may be indicative of large engineering projects being developed by many organizations. These risks indicate that defining a development process is not sufficient to identify software safety (or any other kind of) emergent risk. Management, measurement, and feedback of the process actually being used is equally important to risk management.

Risk 1: Difficulty tracking software hazards and software safety requirements against the backdrop of system hazards and system safety requirements. In all three programs, software safety risks were not specifically or correctly marked in the hazard reports. Identifying which hazard reports described software was a time-consuming, manual process: hazard reports had different writing styles and used different terminology, and were often lengthy. The Constellation hazard reports, for example, had a median length of 36.5 pages (min: 8, max: 152, mean: 43.2). This introduced significant overhead in tracking software safety concerns and introduced the risk that some software safety concerns would become lost or obfuscated in the hazard tracking system, as happened on the DoD program. Even though there was a checkbox in the DoD hazard tracking system to identify a hazard as "software related", this identifier was not used correctly even when it was clear that software was the cause or control of the hazard. Several hundred hours were spent by us merely to identify if software was involved in the Constellation hazard causes and controls.

Risk 2: Potentially hidden software risk in software controls. On all three programs, many hazard controls were not identified as software related even though the mitigation mechanism (i.e., the control) included software, and the corresponding software requirements for implementing the control were not marked "safety related" as required. Contrastingly, if a hardware control was specified, then there would be an additional control to ensure that the new hardware control did not introduce a new hazard. But for software controls to hardware causes, no such control was included to ensure that the added software did not introduce new risk.

Risk 3: Inadequate or incorrect traceability exists between safety requirements, hazards, causes, and controls. In all programs, bidirectional traceability was required between safety requirements, associated hazards, causes, controls, and verifications to ensure that safety requirements have been implemented in the delivered systems. On Constellation and the DoD programs, bidirectional 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 all three programs, the hazard reports contained sections for safety requirement references, though these sections were not completely filled in or were marked as "to be determined".

On Constellation, we observed a number of hazard reports where references (transfers) to other hazard reports' causes and controls were missing or incorrect. 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 can inhibit traceability due

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

to incorrectness and require more effort to understand and maintain. On the NASA satellite project, an external document was used to keep track of software causes and controls across hazard reports, somewhat lessening the traceability burden.

Risk 4: Inconsistent scope and syntax when describing hazards, causes, and controls. The safety engineers wrote their hazards, causes, and controls in unique ways, making consistent evaluation on the part of safety management difficult. For example, all three of the Constellation 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. This was also true among the various organizations that wrote hazard reports in the DoD project. The nonuniformity of the syntactic content in the hazard reports became apparent when applying step 4 of the PRI. Goal 2 in the Constellation study was created in response to this observation.

A lack of consistent understanding of how to apply hazard analysis to software led to both overspecified and underspecified software cause descriptions. The Constellation Safety Engineering Review Panel 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) began. Furthermore, inconsistent structure and vocabulary precluded an automated syntactic analysis of the hazard reports.

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.

The inconsistency in scope and syntax was not a matter of nonconformance to the safety analysis processes, but instead was the result of different interpretations of processes that were not concretely defined.

5.2. 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. Defining how software should be incorporated into traditionally hardware-oriented analyses such as hazard analysis is still 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], and 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 (e.g., the Ariane 5 where duplexed computers both failed on the same software bug [Nuseibeh 1997]). Furthermore, software controls are difficult to specify because software properties and constraints are not well understood early on. The interaction of hardware and software is also a source of risk. 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 lessons learned from applying the PRI method on the three programs, we offer the following recommendations for future software safety-critical programs.

Recommendation 1: Provide explicit guidance for applying safety analyses to software. Assuring software safety is not as simple as saying “apply the hazard analysis process to software”. 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 allowed reviewers to more readily monitor process risks across multiple developments.

Recommendation 2: Plan for automated analysis and traceability and to promote usage of the HTS capabilities. 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; and (3) to provide automated 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. We demonstrated a prototype HTS that 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.

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 them in the HTS. A properly used 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.

Recommendation 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 risks. 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.

In another example, the predominant reason for safety engineers using the HTSes as a post hoc document repository was that the HTSes were not integrated into the contractor's internal safety analysis processes up-front. The contractors working on these systems followed their own processes internal to the organization, producing reviewable output for the government agencies at milestones or upon request. Consequently, there was significant pushback in incorporating a new technology (the HTSes) that might disrupt the internal process. However, if the use of the HTSes had been more carefully specified during contract acquisition, the integration of such systems should already be detailed in the contractor's proposal response.

6. CONCLUSION

We have seen that the PRI 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 PRI method over the course of these three studies.

6.1. Lessons Learned about PRI

The PRI method is both an evaluative method of a process and a method to improve the process while it is being applied. PRI 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 PRI method is that there is a process, whether implicit or explicit, that can be analyzed. Ad hoc methods of achieving a desired product property are not candidates for the PRI method of detecting process risk.

If the process one wishes 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 these 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 PRI methodology is iterative. For a given application, readiness assessment questions may not pass, thus requiring one to start over in identifying insight areas. The metrics one interprets may raise further questions (e.g., after counting the number of software defects, what is really needed is how severe they are), causing one to reformulate goals and questions. It is very likely that the process artifacts that one plans to examine (e.g., safety requirements) do not contain the information one desires or the quality of information is poor (both of which are indicators of process risk), and thus one 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 and 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 one to reevaluate steps 1 and 2 regarding where one 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 not available when needed. 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, and models will vary according to when the measurement takes place as the process can have different expectations/outputs at different points in time. 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).

6.2. Summary

The PRI 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 PRI 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 PRI method does not and cannot provide any assurance that the emergent property exists, for example, that the system is safe. It only provides indicators that there is a risk that the product will not satisfy the emergent property, that is, 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 PRI 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 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 looking at applying the PRI method to identify process risks in achieving other emergent system properties, such as security and reliability.

APPENDIX – THE HAZARD ANALYSIS PROCESS

Because 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.

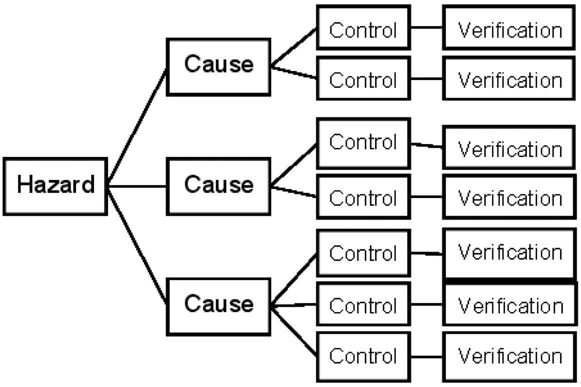

Fig. 3. Hazard structure.

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 focus on the identification of several important properties.

—*Causes*. A cause is the root or symptomatic reason for the occurrence of the hazardous condition.

—*Controls*. A control is an attribute of the design or operational constraint of the hardware/or software that prevents the cause from occurring or reduces the residual risk to an acceptable level.

—*Verifications*. A verification is 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., rupture of 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.

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).

Hazard analysis was typically conducted by a project engineer, who may or may not have software expertise. Hazard analysis was performed concurrently with system analysis and design, with each informing the other through iterative refinement. Hazard analysis review meetings are typically driven by a hazard and focus on the controls the teams propose and eventually verify to mitigate the hazard causes.

In both the DoD and Constellation program 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.

ACKNOWLEDGMENTS

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

J. Anvik, L. Hiew, and G. C. Murphy. 2006. Who should fix this bug? In *Proceeding of the 28th International Conference on Software Engineering (ICSE'06).* ACM Press, New York, 361–370.

V. Basili, F. Marotta, K. Dangle, L. Esker, and I. Rus. 2008. Measures and risk indicators for early insight into software safety. http://www.cs.umd.edu/~basili/publications/journals/J112.pdf.

V. R. Basili and H. D. Rombach. 1988. The tame project: Towards improvement-oriented software environments. *IEEE Trans. Softw. Engin.* 14, 6, 758–773.

V. R. Basili, M. V. Zelkowitz, L. Layman, and K. Dangle. 2010. Obtaining valid safety data for software safety measurement and process improvement. In *Proceedings of the 4th ACM/IEEE International Symposium on Empirical Soiftware Engineering and Measurement (ESEM'10).* 1–4.

V. R. Basili and D. M. Weiss. 1984. A methodology for collecting valid software engineering data. *IEEE Trans. Softw. Engin.* 10, 6, 728–738.

B. W. Boehm. 1988. A spiral model of software development and enhancement. *Comput.* 21, 5, 61–72.

B. W. Boehm. 1991. Software risk management: Principles and practices. *IEEE Softw.* 8, 1, 32–41.

R. N. Charette. 1996. Large-scale project management is risk management. *IEEE Softw.* 13, 4, 110–117.

M. B. Chrissis, M. Konrad, and S. Shrum. 2011. *CMMI for Development: Guidelines for Process Integration and Product Improvement, 3rd* Ed. Addison-Wesley Professional.

Department of Defense. 2012. Standard practice: System safety. MIL-STD-882E. http://www.system-safety.org/Documents/MIL-STD-882E.pdf.

Federal Aviation Administration. 2008. System Safety Handbook. http://www.faa.gov/library/manuals/aviation/risk_management/ss_handbook/.

D. L. Gibson, D. R. Goldenson, and K. Kost. 2006. Performance results of cmmi®-based process improvement. Tech. rep. CMU/SEI-2006-TR-004. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8065.

W.-M. Han and S.-J. Huang. 2007. An empirical analysis of risk components and performance on software projects. *J. Syst. Softw.* 80, 1, 42–50.

R. P. Higuera and Y. Y. Haimes. 1996. Software risk management. Tech. rep. CMU/SEI-96-TR-012. http://www.sei.cmu.edu/reports/96tr012.pdf.

J. Kontio, G. Getto, and D. Landes. 1998. Experiences in improving risk management processes using the concepts of the riskit method. In *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'98/FSE'98).* ACM Press, New York, 163–174.

M. S. Krishnan and M. I. Kellner. 1999. Measuring process consistency: Implications for reducing software defects. *IEEE Trans. Softw. Engin.* 25, 6, 800–815.

L. Layman, V. R. Basili, M. V. Zelkowitz, and K. L. Fisher. 2011. A case study of measuring process risk for early insights into software safety. In *Proceedings of the 33rd ACM/IEEE International Conference on Software Engineering (ICSE'11).* 623–632.

N. G. Leveson and C. S. Turner. 1993. An investigation of the therac-25 accidents. *Comput.* 26, 7, 18–41.

N. G. Leveson. 1995. *Safeware: System Safety and Computer.* Addison-Wesley Professional.

R. Lutz and R. Woodhouse. 1999. Bi-directional analysis for certification of safety-critical software. In *Proceedings of the 1st International Software Assurance Certification Conference (ISACC'99).*

R. R. Lutz and H.-T. Shaw. 1999. Applying adaptive safety analysis techniques. In *Proceedings of the 10th International Symposium on Software Reliability Engineering.* IEEE Computer Society, 42–49.

T. Maier. 1995. FMEA and fta to support safe design of embedded software in safety-critical systems. In *Proceedings of the 12th Annual CSR Workshop on Safety and Reliability of Software Based Systems.*

J. A. Mcdermid, M. Nicholson, D. J. Pumfrey, and P. Fenelon. 1995. Experience with the application of hazop to computer-based systems. In *Proceedings of the 10th Annual Conference on Computer Assurance Systems Integrity, Software Safety and Process Security.* 37–48.

J. Mendling, G. Neumann, and W. Van Der Aalst. 2007. Understanding the occurrence of errors in process models based on metrics. In *Proceedings of the OTM Confederated International Conference on ON the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS.* F. Curbera, F. Leymann, and M. Weske, Eds., Lecture Notes in Computer Science, vol. 4803, Springer, 113–130.

N. Nagappan, T. Ball, and B. Murphy. 2006. Using historical in-process and product metrics for early estimation of software failures. In *Proceedings of the 17th International Symposium on Software Reliability Engineering*. 62–74.

National Aeronautics and Space Administration. 2009. Constellation program hazard analyses methodology. Tech. rep. CXP-70038, rev. b.

B. Nuseibeh, 1997. Ariane 5: Who dunnit? *IEEE Softw.* 14, 3, 15–16.

R. K. Panesar-Walawege, M. Sabetzadeh, and L. Briand. 2011. A model-driven engineering approach to support the verification of compliance to safety standards. In *Proceedings of the 22nd IEEE International Symposium on Software Reliability Engineering*. 30–39.

C. Pardo, F. J. Pino, F. García, and M. Piattini. 2011. Harmonizing quality assurance processes and product characteristics. *Comput.* 44, 6, 94–96.

J. D. Reese and N. G. Leveson. 1997. Software deviation analysis. In *Proceedings of the 19th International Conference on Software Engineering (ICSE'97).* 250–260.

J. Ropponen and K. Lyytinen. 2000. Components of software development risk: How to address them? A project manager survey. *IEEE Trans. Softw. Engin.* 26, 2, 98–112.

N. F. Schneidewind. 1997. Reliability modeling for safety-critical software. *IEEE Trans. Reliab.* 46, 1, 88–98.

P. Torr. 2005. Demystifying the threat-modeling process. *IEEE Secur. Privacy Mag.* 3, 5, 66–70.

W. Van Der Aalst. 2012. Process mining. *Comm. ACM* 55, 8, 76–83.

L. Wallace and M. Keil. 2004. Software project risks and their effect on outcomes. *Comm. ACM* 47, 4, 68–73.

L. Wallace, M. Keil, and A. Rai. 2004. How software project risk affects project performance: An investigation of the dimensions of risk and an exploratory model*. *Decision Sci.* 35, 2, 289–321.

W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt. 1997. Automated analysis of requirement specifications. In *Proceedings of the 19th International Conference on Software Engineering (ICSE'97).* ACM Press, New York. 161–171.

Received March 2012; revised November 2013; accepted December 2013