

A Methodology for Exposing Software Development Risk in Emergent System Properties

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

**Fraunhofer Center for Experimental Software Engineering
College Park, Maryland 20740**

Technical Report 11-101
April 21, 2011

Fraunhofer USA



Center for Experimental
Software Engineering, Maryland

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

A Methodology for Exposing Software Development Risk in Emergent System Properties

Victor R. Basili^{*}, Lucas Layman[†], Marvin V. Zelkowitz^{*}

^{*}The University of Maryland at College Park and the Fraunhofer Center for Experimental Software Engineering

[†]The Fraunhofer Center for Experimental Software Engineering

1 Introduction

Development of large complex systems in the aerospace, defense, energy, travel and related industries requires constant attention to building safe, secure, reliable systems. Failure to achieve such emergent properties during development is costly and may be life-threatening at worst. Eliminating such failures or mitigating the risk of not achieving such emergent properties is a critical part of the development process for any software system. By an emergent property we mean a property of the system that evolves over time during development and can only be fully tested when the system is completely operational.

Mitigating the development risk of achieving the desired emergent product properties (e.g. safety, reliability, security, performance) usually involves adhering to established processes for managing development to achieve those properties. To be effective in achieving the desired product properties, these processes must meet three, often unstated, assumptions:

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

Thus, if we are interested in creating a product that achieves the specified product properties, we need to develop a risk management approach that *ensures that we are using a process that can uncover all¹ risks for not achieving the desired system properties, is appropriate for the context in which it will be applied, and that the process is being correctly followed.* While each of these challenges has been investigated individually, research is needed on methods that address these issues simultaneously. It is the purpose of this paper to explore the multifaceted nature of risk management and to propose a measurement methodology that helps to address these concerns.

¹ Of course, no system will be defect free in practice, but an acceptable target for defect removal can be defined.

1.1 Mitigating product and process risk

We define *product risk*, or technical risk, as the risk that a system will not achieve a desired property, such as functionality, reliability, performance, safety or security. As a product is developed, we try to monitor properties of the product to determine if we are building the system correctly such that the desired properties are achieved. Testing is the primary mechanism used to evaluate product risk. As subsystems are completed we test them against known data to see if they operate correctly. If not, a defect database can be created, and the error is tracked until it is corrected. Engineers have long complemented testing with other processes, such as reviews, inspections, modeling and simulation to increase our confidence that product risks have been mitigated. Regardless of the techniques applied and despite the best efforts of the developers, product risks exist in nearly every system that makes it to production.

All software development processes, including processes for mitigating product risk, introduce additional risks that must also be controlled. We define *process risk* as the risk created by the (correct or incorrect) application of a process that leads to a product risk. For example, a tester who forgets to write performance tests on a component contributes to the risk that the system may not meet performance requirements. Monitoring process risk is analogous to the hardware concept of quality assurance. Quality assurance is the act of leading, teaching, auditing the process by which the product is produced to provide confidence that the product conforms to a specific design or specification. Quality assurance activities include defining and evaluating the processes, and providing feedback to projects, the organization, and the quality assurance organization itself. The major responsibility of project management is to perform process quality assurance. Quality Assurance personnel are tasked with ensuring that the development staff is appropriately following the development process and that the desired outcomes are being achieved. They commonly monitor (or measure) conformance to a process either formally or informally. For example, management may monitor the defect tracking database to ensure that developers and testers are correctly filling in all of the required information in defect reports, such as the reproduction steps. If the data is of poor quality, then the developers may not be able to fix the defects in a timely manner or they may be wasting too much effort trying to reproduce the bug. If the defect closure rate is too low, then the likelihood of missing the product deadline (or of shipping a product with an unacceptable amount of defects) is increased.

1.2 General limitations in product and process risk mitigation strategies related to emergent system properties

In general, mitigating product and process risks surrounding emergent system properties is more difficult than mitigating risks associated with product functionality, cost, and schedule. Often, these properties are evidenced only as a result of evaluating the entire system after development. Emergent system properties are highly influenced by the system requirements, the system design, the expertise and quality of the engineers, and the development processes

used, which are all decisions made early in the development lifecycle. However, evaluating properties such as safety and reliability using methods such as testing is only possible toward the end of the development process, when changes are difficult and expensive to make. Furthermore, methods for testing properties such as security, safety and performance are often ill-defined, immature and non-repeatable.

Quality assurance, as the primary means of mitigating process risk, also suffers from several limitations. Organizations use a variety of processes to achieve and manage reliability, safety, security, etc., which complicates the problem of measuring, monitoring, and collating the status of these emergent properties. Furthermore, quality assurance usually reduces to a comparison of the set of activities an organization is following with a predefined guideline of the activities it is supposed to follow. For example, the organization's safety standards may require the development group to perform code walkthroughs, but do these reviews find important defects or only superficial ones such as syntax errors? An organization may record defects in a defect-tracking database, but are developers entering *useful* data? An organization may use a requirements management system, but is the requirements database frequently updated to make sure that developers have the latest information available?

1.3 The challenge – improving process risk management to gain visibility into emergent system properties

Risks associated with emergent system properties are largely controlled by putting processes in place to mitigate those risks. As such, when we determine that the desired property is at risk of not being achieved, we must know *why not* in order to take the appropriate corrective action. We assert that processes create risk when one or more of the following assumptions are not met:

1. *The process is an effective way of mitigating risk and achieving the property;*
2. *The process is appropriate for the development context;*
3. *The process is followed correctly.*

Identifying the cause of process risk is where most projects struggle. This first instinct is to say “you’re not following the process” or “you’re not doing enough of the process.” While it certainly is the case that not following a process can lead to risk, there is often a hidden reason relating to assumptions 1 and 2 that is promoting process nonconformance, e.g., the process is flawed or the context does not allow various steps to be performed. Also, there may be conflicting goals, such as the achievement of schedule interfering with the application of the process. In reality, pressing a developer to apply more process may not be the solution. Perhaps an organization's performance testing guidelines are simply ill-specified and uninformative, and thus the performance testing process is ad-hoc and ineffective.

Project managers must understand if the process is not adequate for achieving the desired property, the process is not appropriate in the current context, or if the process is not being applied appropriately.

2 A Theory of Process Risk Measurement

In this section, we present our theory of process risk measurement. We then present the Process Risk Assessment (PRA) method, which we developed to address the challenges of identifying potential risks early in the life cycle. Given a development process for building a software system, how do you discover whether the process is achieving its goals and the development group is accurately following this process? We first describe our process, and then offer two case studies, one from the Defense Department and one from NASA, where we applied the method, uncovering problems in both the software safety process as defined and in the application of the process by the developers and were able to provide useful feedback to the projects.

2.1 A model of process risk measurement

The Process Risk Assessment (PRA) approach is based upon analyzing the properties of a product over time. The properties of a product at time t are a function of at least four factors: (1) the functional and non-functional system requirements that describe the desired product property; (2) the process or sequence of steps performed to achieve the desired property; (3) the people who perform the process, e.g. 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:

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

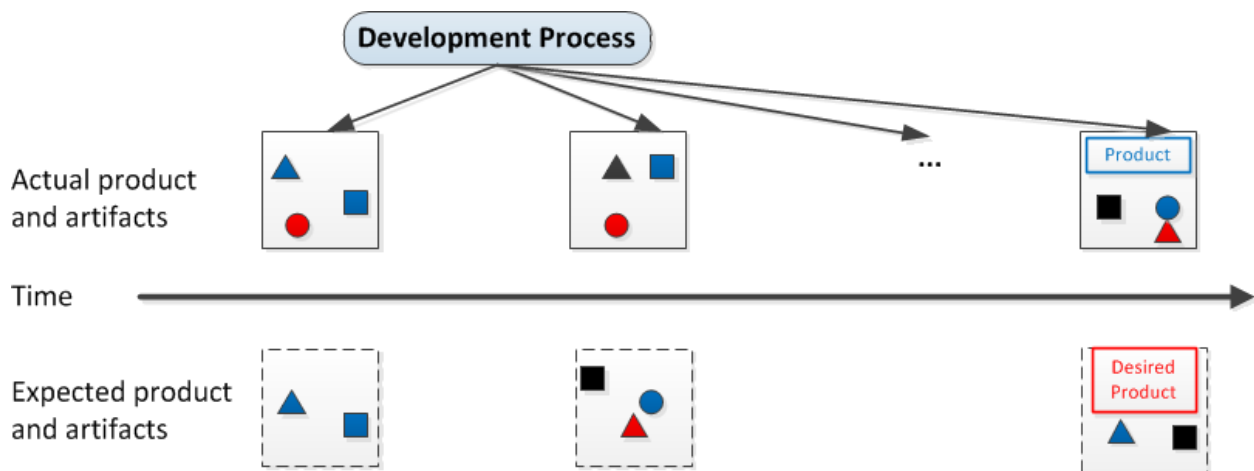


Figure 1. Development process

We use this relationship to reason that, if it is true 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, i.e. product risk will be minimized. If the process is incorrectly applied, then there is a risk of not achieving the proper product property, i.e. process risk contributes to product risk².

Given the above model, we can analyze the intermediate outputs from the development process to determine if there is potential risk by keeping track of the various intermediate artifacts produced during development (e.g., source libraries, defect databases, resource expenditures, other documentation) (See Figure 1). This analysis will help identify high risk issues for this system with respect to the anticipated property. If the intermediate outputs of a development process agree closely with the expected outputs, then we have reason to believe that the process is being followed closely and product risk is relatively low, assuming the process is effective in achieving the property. 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.

Based on this model, to mitigate process risk we must first determine what kind of 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. Thus, to maximize the chances that a process will achieve the required product property, we need to know:

1. *Is the development team following the process? Can we measure and evaluate process conformance to ensure in a quantitative way that the process is being followed and how well it is being followed?*
2. *Is the process yielding meaningful intermediate outputs that provide information about whether the desired property is being achieved? Project managers must be able to identify, in a quantitative manner, if the desired product property is at risk of not being met. Measuring these intermediate outputs means that issues can be uncovered earlier in the process.*
3. *Can we measure against a baseline (or establish a baseline) for the desired intermediate product properties. That is, do we have a mechanism that allows us to determine if our intermediate outputs are leading to our achieving the property or we are at risk of not achieving it?*

² However, the properties may be achieved by other unintentional means.

4. *Does the process work?* By comparing the intermediate results against baseline expectations, determine if the process is suitable for achieving the desired product property given the requirements, personnel and context.
5. *What improvements can be made to the process so that mitigations can be applied during development and in future applications of the process?*

The goal is to develop and implement a set of measures that provide risk management **visibility** into the system (and software) for the **purpose** of asking the right questions, identifying risks and monitoring the quality of the development process.

2.2 Levels of Obtainable Risk Insight

The artifacts of a development process contain two types of information that provide insight into the process: *syntactic* information and *semantic* information. An artifact's syntactic information concerns the important data elements of the artifact and how they are composed. For example, the syntactic information of a defect report might include specific sections for a description of the defect, reproduction steps, and a criticality of the defect. Using a predefined grammar, set of terms, or pull-down menus in a computerized defect library, library items in the library can be made syntactically correct and also provide clues as to the semantic information that is needed.

An artifact's semantic information is its meaning in the context of development. Interpreting semantic information almost always requires human expertise. For example, a computer cannot automatically infer the risks and necessary mitigating actions required based on the contents of a defect report. 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, impaired or mislead, then there is an increased product risk. Insufficient, incorrect or missing syntactic information in process artifacts is a strong impediment to accurate useful semantic analysis and is the crucial indicator that we leverage to identify and measure potential risk during the development process.

There are levels of insight that can be obtained from examining intermediate products during development. Our risk measurement methodology is based on answers to the following five questions about processes and process artifacts, starting from the most basic to the most sophisticated:

1. *Do we have any information?* What artifacts are being developed and saved during the development process? Is the information relevant to understanding the development process? Can we look at requirements documents, source code, design documents, defect reports, expenditures over time, etc. in order to understand development?
2. *Assuming we have information, is it useful?* Can we use it to perform a syntactic or semantic analysis of the development? For example, do defect reports list source, effects and impact of each defect? Is data being collected at regular and appropriate time intervals? Are all

groups generating data in a consistent manner to allow them to be compared? Do the reports contain the information necessary for understanding, reproducing and correcting the defect?

3. *Do we have enough information to perform a syntactic analysis of the data?* Do all reports use a consistent style and terminology? Does the data (or a subset of the data) allow for automatic parsing and report generation?
4. *Do we have enough information to perform some form of semantic analysis of the data?* Does the semantic information follow syntactic rules allowing automatic processing of the terminology used or data bounds? E.g., are certain terms expected to be found? Are certain data items outside the acceptable range? Can pull-down menus be used?? If not, the information may still be useful, but requires domain experts to evaluate.
5. *Is the data semantically correct?* That seems to be outside of the scope of automation given today's technology. It requires domain experts to fully examine the intermediate artifacts.

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

3 The risk measurement methodology, PRA

In this section, we introduce the steps of our risk measurement methodology. We then explain the methodology steps in detail and provide an example of applying the methodology to a large, network-centric system-of-systems to evaluate software safety risk.

3.1 Methodology overview

The PRA approach evaluates the ability of the syntactic and semantic information in the intermediate outputs of software development processes (i.e. processes to improve reliability, safety, etc.) to expose potential risks that desired product properties may not be achieved. This provides the opportunity for feedback to project personnel early so that corrective actions can be made.

PRA is comprised of six steps, which are applied iteratively to form a risk measurement plan. The six steps are grouped into three stages, as follows:

<p><i>I. Identifying insight opportunities</i></p>	<ol style="list-style-type: none"> 1. Identify insight areas from the development process that provide insight into risk areas. 2. Identify measurement opportunities that provide insight into each risk area.
<p><i>II. Evaluating the quality of information</i></p>	<ol style="list-style-type: none"> 3. Develop readiness assessment questions to identify if it is possible to delve deeper into the area.
<p><i>III. Measuring, interpreting, and providing advice</i></p>	<ol style="list-style-type: none"> 4. Define goals, questions, and measures for each risk area to expose risks associated with process artifacts. In our examples, our goal is to identify areas of greatest software risk so that we can redirect work effort as needed to mitigate that risk. 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.

The first three steps of this methodology determine if a process risk analysis has any hope of succeeding. When readiness assessment questions cannot be answered, measurement opportunities do not exist, or insight areas cannot be identified, these all indicate that the development process itself is a source of risk. A development process that does not have any insight areas into the risk it is meant to mitigate is a strong indicator that the process is immature or not appropriate for the current context. A process that does not have useful measurement opportunities is an indicator that the process or process artifacts do not contain meaningful information, in which case one must question the suitability of the process for the current problem. Processes and process artifacts that do not pass a readiness assessment are indicators that the process is not being followed, or that the processes is not well-defined for the current context.

3.2 Applying the PRA methodology: examples from a DoD program

To illustrate this process, we provide an example which was the initial motivation for the approach, the identification of potential software safety risks during the development of a Department of Defense (DoD) safety critical complex system of systems [Ba08]. Safety is an

ideal example of an emergent product property that cannot be fully tested until the system is operational. A safety risk is a risk whose effect can be injury or the loss of life either directly or through a chain of events. Safety risks caused or contributed to by software have become a greater concern in systems development as many traditionally hardware-centric systems become more reliant on software. The DoD system was a large, network-centric system of systems with a potentially large number of software safety risks that safety engineers needed to track and verify before the system was deployed. System development would be a multi-year, multi-million dollar effort that would involve hundreds of engineers from over 50 subcontractors dispersed across the country. The development process was expected to follow the traditional Defense Acquisition V-Model and the safety process MIL-STD-882, DoD Standard Practice for System Safety [MIL-STD-882, 2000]. Because of the cost and complexity of developing this system, safety risk management was integrated throughout the lifecycle, since design and architecture changes late in development would be prohibitively expensive. Our goal was to help the software safety engineer develop an approach that provided early warning signs of safety problems during its development. More details on the planning and collection of safety visibility data can be found in [Ba08]. The problem led to the decision that tracking process conformance would provide the best insights into what and where problems might arise.

3.2.1 Step 1: Identify insight areas from the development processes that provide insight into risk areas

We first identify which of the intermediate outputs of the process can provide insights into process conformance and the process effectiveness needed to achieve the desired product property. That is, is there potential syntactic and semantic information that can be gathered from process artifacts that can provide insight into software development risk? If such information cannot be identified, then it is likely that risk will be present in the system because there is no process or artifact capturing that risk. From a methodological perspective, this step allows us to determine whether sufficient information exists that might be measured and to understand the level of detail in that information.

Keys to this step

Inputs <ul style="list-style-type: none">• The property you want to measure• The processes associated with achieving that property• The intermediate outputs of each step for each process	Outputs <ul style="list-style-type: none">• 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:<ul style="list-style-type: none">○ 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 <ul style="list-style-type: none">• What are the process outputs created during application of the process?• How much 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 in terms of cost, schedule, and timing to analyze the insight area?	

In the DoD example, the safety process outlined the steps for identifying and tracking safety hazards and verifying that those hazards were mitigated in design, implementation and test (see Appendix A for an overview of the hazard analysis process). Among the tasks whose outputs from the software safety process that provided insight into potential risks were:

- The set of safety-related software requirements in the Requirements document identified. They can be used to confirm that system and software requirements and development practices are in compliance with safety processes;
- The set of hazards, with its causes, controls, and verifications in the Hazard Tracking System. They can be used to ensure that the program is adequately identifying and documenting the appropriate information about a hazard;
- The relationship between the hazard causes, controls, and verifications in the Hazard Tracking system. This can be used to ensure that sufficient actions are taken over time, i.e., the hazard controls are being implemented, and verified;

- The set of safety related defects in the Problem Tracking System. They can provide insight into whether any safety problems remain in the system.

3.2.2 Step 2: Identify the measurement opportunities that provide insight into each risk area

Each insight area is evaluated for its potential to identify syntactic and semantic information in the process artifacts that could be measureable indicators of process risk. A measurement opportunity can be an opportunity to measure process conformance (e.g. are all of the fields in the test results filled in correctly?) or an opportunity to measure the software risk directly (the number of defects found during test). A measurement opportunity is a high-level metric based on the information available; elaborate, detailed metrics based on complicated computations are not required here.

Keys to this step

<p>Inputs</p> <ul style="list-style-type: none"> • Process outputs/artifacts identified in step one 	<p>Outputs</p> <ul style="list-style-type: none"> • Potential metrics based on process outputs/artifacts
<p>Activities or Questions to ask</p> <ul style="list-style-type: none"> • 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? • Is it feasible in terms of cost, schedule, and timing to pursue this measurement opportunity? 	

In these first two steps, we have not looked at any actual data; we are looking at the process outputs and their potential to be measured. The purpose of these first two steps to understand the software development process we wish to measure (e.g. the software safety process), and to understand the process artifacts. These steps may seem superfluous, but are necessary. Too often we assume that a process is producing meaningful, insightful information, but this information may not actually be available or useful. This step directs the creation of specific goals and measurements, and identifies (and thereby excludes) processes or process artifacts

that do *not* have measureable information. A process that does not have measureable output may represent a risk itself, and is a candidate for process improvement.

In our DoD example, we identified several measurement opportunities, including:

- Counting the number of software safety requirements to recognize whether a reasonable number are being identified.
- Counting the number of hazards, causes, controls, and verifications that involve software to recognize whether the numbers are growing appropriately over time and that each cause is covered by at least one control and each control is verified.
- Measuring the number of controls that are and are not mapped to requirements in the Requirements Document.
- Measuring whether the number of open software causes and controls is shrinking over time, i.e., risk mitigation is actively occurring.
- Measuring the number of software safety related defects being closed at a reasonable rate over time.

3.2.3 Step 3: Develop readiness assessment questions to provide risk status and identify if the insight area can be evaluated

In the third step, we determine if the syntactic and semantic information in the process artifacts contains sufficient information to investigate the risk further. Before developing models and measures and analyzing data, we propose a set of questions that allow us to gain initial insight into the areas of interest, to get a quick and easy status report of the area, and to identify whether it is possible to go deeper. It is possible that the process is poorly applied or misunderstood, and the project needs to correct these risks first.

What is learned from these questions helps tailor the models, measures, and responses applied at a deeper level. For example, in measuring software safety risk, we might ask if there is a documented software safety process that identifies requirements as safety-related and if safety-related requirements are marked as such in the requirements repository. Otherwise, there may not be sufficient information to evaluate safety risks. Answering these readiness assessment questions is an iterative process, as answering one question may lead to others.

Keys to this step

Inputs	Outputs
<ul style="list-style-type: none">Proposed measurement opportunities and the associated risks they measure	<ul style="list-style-type: none">Advice on whether the intermediate outputs and metrics can be used to identify process riskA high-level assessment of process conformance risk, i.e. are the processes producing meaningful outputs?
Activities or Questions to ask	
<ul style="list-style-type: none">Examine the process artifacts and try to apply the proposed metrics. Can I apply the metric?<ul style="list-style-type: none">Is the information accessible and available?Is the information in good enough form that it can be measured?Is the information complete?If I can apply a metric, then it will be a candidate for future measurement.If I cannot apply a metric, why not?<ul style="list-style-type: none">Why is the information inaccessible?Why is the information in such a poor state?Why is the information incomplete?Is it feasible in terms of cost, schedule, and timing to pursue this insight area?	

Readiness assessment questions in our DoD example included:

- Are the safety-related software requirements marked as such? If not, it will be difficult to measure the number of safety-related software requirements.
- Is there an automated HTS where software-related hazards, causes, controls, and verifications are recorded (and can be counted)? If not, identifying software-related hazards, causes, controls and verifications could be an effort-intensive process.
- Are hazards mapped back to their source (requirements) and controls mapped to requirements? If not, there is a risk that the process is not producing the information needed to verify that all safety requirements are being implemented.
- Are all fields being entered into the HTS? If not there, there is a risk that there is insufficient data necessary to perform a safety evaluation.

- Are software safety-related failures/faults identified as such in the Problem Tracking System? If not, then identifying software safety-related failures will be an effort-intensive process.

3.2.4 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 in which to make deeper risk assessments, in this fourth step we set goals for identifying specific software risk based on the syntactic and semantic information available in the development process artifacts. We develop a set of questions that will help determine if those goals are being met, and we identify formally the metrics we will use. This step is an application of deriving goals, questions, and metrics in the GQM approach [Ba84]. Goals and questions should focus on the application of the development process and the expected information in the process artifacts. The premise of our methodology is that the syntactic and semantic information in process artifacts should meet the expectations of the processes, and, if not, there is risk that the process is not appropriate for achieving its goals or that the process is not being applied appropriately.

We can ask questions 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?”

Keys to this step

<p>Inputs</p> <ul style="list-style-type: none"> • A set of proposed metrics that have passed the readiness assessment check 	<p>Outputs</p> <ul style="list-style-type: none"> • A GQM structure with specific goals, questions and metrics
<p>Activities or Questions to ask</p> <ul style="list-style-type: none"> • Apply the GQM method to derive a goal template, the questions, and what measures are needed. <ul style="list-style-type: none"> ○ 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? 	

For our DoD example, using the goal template of the GQM [Ba88], we derived a set of software safety visibility goals and questions regarding the hazard process. A subset of the goals, questions and metrics are presented below:

- **Goal 1. (Number of hazards):** Analyze the set of software-related safety requirements in the Requirements Document to evaluate if a reasonable number of software-related safety requirements have been identified from the point of view of the safety engineer in the context of the DoD system.

Expanding on this goal, we need to address the following questions:

- What is the total number of software requirements and how many of them and what percent of them are safety requirements?
- For comparable systems, what is the expectation of the number of software-related safety requirements that should have been identified?

Metrics, which can be used to answer these questions, include:

- Number of software requirements and the number of software-related safety requirements, which have been identified
 - Percent of those software-related requirements that are safety related (Call this PSSR.)
 - What is the expected range for PSSR for similar systems? (Call this EPSSR.)
- **Goal 2. (Mitigating hazard risks):** Are the hazards in the HTS being worked on effectively? That is, analyze the set of software-related safety hazards, causes, controls, and verifications in the HTS to evaluate if they are being closed at an appropriate rate from the point of view of the safety engineer in the context of the DoD system.

Expanding on this goal, we need to address the following question:

- At specified points in time for any hazard, what is the number of open hazards and their associated causes (and controls)?

A metric, which can be used to answer this, could be:

- Compute the moving average over several months (e.g., 3) of the number of open causes for which controls have not been fully specified. Let $HCCE_{i,3} = MA_{i+1,3}$

$/ MA_{i,3}$ where $MA_{i,3} = (X_{i-2} + X_{i-1} + X_i) / 3$ is the average number of open *causes* at three consecutive time intervals., e.g., monthly.

Note that all the goals here are syntactic in nature, rather than offering any semantic insights. They focus on the question: Is sufficient material there? Where are the potential risks based upon missing information? Such syntactic measures, provided that the necessary information is available in the artifacts, could be calculated automatically.

The more interesting question is, “Is the right material there?” This question focuses on a semantic analysis and requires domain expertise to answer. For this step, the goal is to generate statistical samples, based on the lack of sufficient syntactic data, to be analyzed manually by experts for process risks.

The measures collected by a program should be saved and stored in an Experience Base to create an historical base that can be used to extract model criteria and provide bounds for the interpretation of the measures in the next step.

3.2.5 Step 5: Develop interpretation models and threshold values of measures

Given the metrics developed in Step 4, we need to interpret the data by using those metrics. We define countable measures that help answer the questions and meet the goals of Step 4, and define interpretation models of those measures. What are good values and which values represent risk? For example, for Goal 1 above, what is the range in value for EPSSR (percent of software-related hazards), which indicates risk that an insufficient number of software safety requirements have been captured?

The measures and interpretive models are based on syntactic (and possibly) semantic information extracted from the software process artifacts or from experts. By collecting measureable information, we enable quantifiable comparison, measurement repeatability, a baseline for measuring future changes, and well-defined targets for future efforts.

Keys to this step

Inputs <ul style="list-style-type: none">• A set of goals, questions and metrics to be collected	Outputs <ul style="list-style-type: none">• A set of models that provides indication that there may be a risk
Activities or Questions to ask <ul style="list-style-type: none">• 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.<ul style="list-style-type: none">○ What is the expected value of that metric and possible margin of error, i.e. what is the range of values that would be acceptable?○ Do historical data exist for any of the metrics?○ Are there proxies for the bounds on these metrics?○ Can we gather any expert opinion on the bounds?	

The definition of measures and models follows a template. First, we take a **question** posed in the previous step along with the metric that captures the concept addressed by the question. The **metric** can be any basic measure or derived metric, as well as evidence of some sort (e.g., existence of documents or processes). We then select a **model** that defines the expected mathematical bounds on the metric and the **interpretation** of the values of the metric(s), in order to answer the target questions. For each model we make assumptions about how the metric values should be interpreted. This involves the selection of an **expected value** and a **range** for that expected value. We can use several approaches for estimating the **expected value** (in approximate order of quality): historical data from past projects, data from current related projects, a proxy estimate, or an experienced expert's estimation

The **range** of the expected value can be based upon a known distribution, based upon the distribution determined by the values used to make the proxy estimate (such as a running average over several points on a curve), or an expert estimate of the range. To this we add a **scope** of application of the metric, e.g., we can assume these metrics are taken for certain suppliers or certain types of systems.

In our DoD example, consider the following metric and interpretation model for Goal 1:

Questions: What percent of the total number of software requirements are safety requirements? For comparable systems, what is the expectation of the number of software-related safety requirements that should have been identified?

Metric: PSSR is the percent of software-related safety requirements, or the number of safety-related software requirements / total number of software requirements.

Interpretation model: We want PSSR to be within a certain percent (e , the precision of our estimate) of an expected percent ($EPSSR$), we compute

if $|PSSR - EPSSR| < e$ *then* a reasonable number of software safety requirements have been identified; *otherwise* we have either too many or too few.

To calculate EPSSR, if we have historical data for similar systems, we can let **EPSSR** = the average of the PSSRs for all similar systems and $e = \sigma(EPSSR)$. Alternatively, we can define a proxy, e.g., assume the relationship is in line with system safety in general, then $EPSSR = \#system\ safety\ requirements / \#system\ requirements$. For the precision of our PSSR e we can base it upon expert opinion or historical data, e.g., $e = 20\%$ of $EPSSR$. (E.g., if historically there was 1 safety-related software requirements for every 5 software requirements, then $EPSSR$ would be .2 and e might be .05 allowing for a 5% range from 15% to 25% in allowable percentage of safety requirements.) If we are outside of that range, then, as we shall show in Step 6, there may be increased risk in the process.

3.2.6 Step 6: Propose responses to identified risks

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

Not identifying any risks at this step does not mean that there are no software-related risks. It only means that the development team seems to be following the process in a reasonable manner. Risks may still be present in the details of the design or implementation that is being worked on. However, as we describe later, the PRA method has uncovered process risks in all of the projects in our case studies to date.

Keys to this step

Inputs <ul style="list-style-type: none">• Metrics and an interpretation model• Data from intermediate project artifacts	Outputs <ul style="list-style-type: none">• Advise on what the project should do if we are outside the acceptable bounds and there is a risk
Activities or Questions to ask <ul style="list-style-type: none">• Provide expert safety engineer advice on what to do under the circumstances	

For *PSSR* (percentage of software safety requirement) for Goal 1 we provide the following response: If *PSSR* is not within the range of *EPSSR* then there is a need for management action.

- If *PSSR* is too large then what are the cost and schedule implications of so many safety requirements?
- If *PSSR* is too small then are a significant number of safety issues not being identified?

We need to check if the safety analysis process is being applied correctly, develop a “get well” plan, or investigate the reason why the system under consideration has such a small (or large) number of software safety requirements.

For *HCCE* (hazard cause closure evolution) for Goal 2, we provide the following response: If the number is ≥ 1 and it is not the beginning phases of development, more effort should go into closing the hazard software causes.

- If it is because the opens are increasing too fast (new hazards are being introduced, new causes for existing hazards), then investigate the reasons.
- If it is because the closes are not increasing fast enough, then investigate the reasons.

Graphing the cumulative identified, open, and closed causes and the rate that they are opened and closed provides good insight into the trends of these variables.

For more goals, questions, measures, interpretation models and responses, see [Ba08]

3.3 Results of Applying the PRA to the DoD program

The illustrative examples in the previous section were taken from a case study in which we applied and evaluated the PRA method to gain visibility into software safety risks in a large DoD program. We began our effort early in the lifecycle, when only requirements, preliminary designs, and the safety processes were available to help gauge software safety risk. We selected the insight areas, identified measurement opportunities, readiness questions, measures, models and responses before we had the opportunity to look at the data. In other words, we attempted to follow the entire method up-front instead of iteratively applying the steps to the project data (as it should be).

However, in asking the readiness questions, we found that measuring HTS problems were difficult due to a limited vision for the HTS, i.e., it was viewed as a storage repository rather than an analysis tool. Extracting the information we needed to apply our measurements was difficult or non-existent in some cases. Furthermore, we were not able to perform an automated syntactic analysis of the data. There was insufficient information to identify even sample set of *software* hazards or requirements for more detailed semantic analysis as the HTS and requirements management systems did not distinguish between software and non-software hazards and requirements. The entire collection of hazards and requirements was simply growing too large (already several hundred hazards) for us to perform our analysis given our available resources.

The readiness assessment questions exposed some common problems with the overall hazard tracking approach and in implementing a useable, cost-effective HTS. These included:

D-1: Software Hazard Identification: Safety-related requirements were not identified as such and many hazard controls were not identified as software-related, even if they were. This demonstrated inadequate attention to software safety. Recommendations were made to reanalyze and update the data in the HTS to correctly identify hazards and causes as software related and that guidance be added to the HTS user guide to more clearly define a software hazard, cause, and control i.e., a hazard is a software hazard "if it has at least one software cause or one software control" .

D-2: Hazard Traceability: Hazards were not fully traceable to the source of the hazard. The HTS did not provide sufficient linkages among the requirements documentation system, the test plan, or to the defect tracking system. Hazards must be bi-directionally traceable to requirements, tests, and defects in order to verify complete coverage, determine comprehensiveness of the hazard analysis, and ensure that the hazard data represents the system accurately over time. Among the recommendations made were that safety engineers perform traceability activities as part of the Safety Analysis process and that fields be added to the HTS to support traceability.

D-3: Data Integrity: Hazards, causes, and controls were not described in sufficient detail to be understood and verified. The wording in the HTS to describe a cause or control was often too general or cryptic. It was recommended, for example, that controls be worded so they can generate or relate directly to requirements.

D-4. Inadequate vision for the use of the HTS: It was viewed as a storage repository rather than an analysis tool. It is important to make sure that (1) the HTS has adequate functionality, quality checks, and documentation, (2) there is traceability and synchronization among the various support systems, e.g., the HTS and the requirements management system and the defect tracking system, and (3) the quality of the data is monitored to minimize the need to scrub the data at a later date. The cost of not adhering to this advice is high rework costs and lower than desired safety of the system. The HTS even violated the simplest goals for syntactic analysis.

What we learned from this application was that PRA can work; in fact it did provided insight into the software safety of the DoD system from just the readiness questions alone. While the readiness questions seem simplistic, they uncovered issues that may not be obvious unless the questions are asked explicitly. Even though we were unable to generate quantitative measurements from our interpretation model, we still uncovered many potential development process risks and were able to provide input to the safety engineer as to what recommendations to make to the project.

One lesson learned from this application of the approach was that before applying steps 4 through 6, the readiness questions should be evaluated to check that there is sufficient data to perform the detailed measurement. This lesson was applied in the next application of the approach to the NASA Constellation program.

4 Applying the PRA method to Gaining Visibility into Software Safety to the NASA Constellation program

After our initial experience on the DoD program, we refined the PRA method and applied it to the software safety process for the NASA Constellation Program. In the NASA case study, we applied all six steps of the methodology, the results of which are more fully documented in [La11]. As with the DoD program, our goal was to assist system safety engineers in identifying potential software safety risks early in the development lifecycle.

4.1 Constellation case study

The Constellation program was a complex *system of systems* to support the next generation of human spaceflight at NASA. We worked with the Software Reliability and Quality Assurance (SR&QA) group for Constellation to provide visibility into the software safety process to enable a meaningful evaluation of software related hazards by program management, to help software quality engineers to implement software safety processes, and to demonstrate conformance to the software safety processes.

We examine three spaceflight systems in the preliminary design stage that involved a significant number of hardware and software features. As in the DoD program, we focused on the hazard analysis process [CxP70038]. The official hazard analysis process (see Appendix A) for the Constellation program contained provisions and guidelines for examining potential hazards, causes and controls with respect to software. Project safety engineers identified potential hazards in system operation and design and created strategies (i.e., controls) for mitigating those risks, recorded results in the hazard tracking system, and presented their findings to the Constellation Safety & Engineering Review Panel (CSERP). At each milestone, the development groups identified safety risks in system operation and design and created strategies (i.e., controls) for mitigating those risks. The CSERP reviewed the risks and the operational or design strategies for mitigating these risks, acting as gatekeeper for development milestones³.

4.1.1 Step 1: Insight areas – the hazard analysis process and hazard reports

Safety analysis in Constellation is vested in the CSERP and its use of hazard analysis to drive actions, and in the Constellation SR&QA manager who identified the hazard analysis process as the best potential source of software safety information early in the development process. All hazards, including software-related hazards, are stored in the Constellation HTS. As in the DoD study, the hazard analysis process generated intermediate outputs containing syntactic and semantic safety information that provided insight into potential process risks. We identified the following potential insight areas:

- The set of hazards, with its causes, controls, and verifications in the Hazard Tracking System. They can be used to ensure that the program is adequately identifying and documenting the appropriate information about a hazard;
- The relationship between the hazard causes, controls, and verifications in the Hazard Tracking system. This can be used to ensure that sufficient actions are taken over time, i.e., the hazard controls are being implemented, and verified;

4.1.2 Step 2: Measurement opportunities in hazard reports

We next identified measurement opportunities to help quantify software safety risk based on information captured in the hazard reports. Our measurement opportunities focused on syntactic information to help evaluate conformance to the prescribed hazard analysis process. Furthermore, the hazard analysis process, together with other NASA standards, described what constituted a “meaningful” description of a hazard cause. Thus, we were able to identify measurement opportunities that were related to the quality of the semantic information in hazard causes in addition to syntactic data.

³ Although the Constellation program was cancelled, some of the projects are expected to be continued. Our results are applicable to other NASA activities as well.

We identified the following measurement opportunities:

- Counting the number of hazards, causes, controls, and verifications that involve software to quantify software involvement in hazardous conditions.
- Verifying that each cause is covered by at least one control and that each control is covered by at least one verification.
- Evaluating hazard causes to determine if they contain the required syntactic components that are a prerequisite for a “meaningful” hazard cause description.
- Counting the number of causes, controls and verifications that are “transfers”. Transfers are a reference to a cause, control or verification in another hazard report. Transfers imply that the cause, control or verification is fully described in the other hazard report and does not need to be repeated. During system implementation and test, all transfers must be verified for a hazard reports to be considered “closed.” Verifying transfers is a manual, labor intensive process and is at risk when transfer references are not kept up to date. Thus, transfers themselves are a measurement opportunity as they can represent both technical and a process risk.

4.1.3 Step 3: Readiness assessment questions

As we learned in the DoD application, asking readiness assessment questions is important to determine if analysis can proceed. If the readiness assessment questions cannot be answered satisfactorily, this may also be an indicator of process risk. It is important to note that answering these questions is an iterative process, as answering one question may lead to others. We developed a number of readiness assessment questions while exploring the measurement opportunities and offer a sample below.

- ***Can we access the hazard tracking system and hazard reports?*** Access to the process artifacts was a necessary precondition to any measurement. The HTS was made available to us by NASA personnel, but the HTS contained hazard reports for only one of the three spaceflight hardware systems we examined (see next bullet).
- ***Is the content of the hazard tracking system up to date?*** The hazard tracking system only contained up-to-date hazard reports for one project (Project A). The hazard reports from other projects were obtained from a database containing materials from CSERP review meetings. Project B’s hazard reports were in the HTS but not visible as the development contractor did not want to release “intermediate” versions of the hazard reports. Project C’s hazard reports were created prior to the development of the hazard tracking system itself. Since Project B and C’s hazard report were not in the HTS, we could not leverage the querying capabilities of the HTS and had to spend additional effort collecting all of the hazard reports for the two systems.

- **Are the cause, control and verification data complete enough for analysis?** For the NASA engineers writing the hazard reports, the goal of hazard analysis in the preliminary design phase was to identify and describe all causes and to develop preliminary controls. Verifications were not yet specified nor required. As such, we could not measure software verifications. Most causes and controls were specified and thus could be analyzed, though some were still works-in-progress and had a “To Be Determined” placeholder.

In addressing the readiness assessment questions, our first observation was that few of the hazards were actually in the HTS. CSERP receives those hazards as part of its assessment of progress, so we were able to obtain the reports. However, keeping the HTS more up to date is an easy first step in mitigating hazard risks. The HTS had the capability to automatically keep track of the “transfers” hazard reports, making traceability maintenance a much lower cost than the manual effort required of engineers working only with word processor documents. Furthermore, the contractor withholding intermediate hazard information from the HTS prevented SR&QA management from easily assessing the progress of the hazard analysis process.

4.1.4 Step 4: Define goals, questions and metrics

When presented with the analysis possibilities, SR&QA selected several goals, among them are the following two. For additional analysis, see [La11].

SR&QA management wanted to understand the role of software in possible system hazards. By understanding the role of software in system hazards, SR&QA could identify systems and subsystems with the greatest potential software safety risk. To this end, we identified the following GQM goal:

- **Goal 1: (Role of Software)** Analyze the available set of the hazards reported for Projects A, B and C in order to characterize them with respect to the prevalence of software in *hazards*, *causes*, and *controls* from the point of view of NASA quality assurance personnel in the context of the Constellation program.

Expanding on this goal, we identified the following example questions. (A complete list of questions and metrics may be found in [La11].)

1. What percentage of hazards are software hazards? We define a *software hazard* as a hazard that contains one or more software causes.
2. What percentage of the hazards is software-related? As described earlier, a software-related hazard has at least one software cause or software control.
3. What percentage of hazard causes are software causes?
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, the software can again be the cause of a hazardous condition.

The following basic **metrics** were used to answer these questions:

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

SR&QA also wanted to evaluate the syntactic and semantic information in software causes of hazards to established NASA guidelines. If software cause descriptions that did not conform to process standards, then there was a risk that there was not enough semantic information to adequately describe the hazard cause. A lack of conformance also indicated that the hazard analysis process could be improved to provide better guidance for describing software-related causes.

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

We asked the following questions to help meet this goal:

1. What is the number and percentage of software causes is *well-specified* according to the Constellation hazard analysis methodology requirements?
2. What is the number and percentage of software causes is *partially-specified*? These causes lack certain pieces of information needed to evaluate their quality.
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.

The metrics used to answer these questions were derived from requirements in the Constellation hazard analysis process, the NASA Software Assurance Standard, the NASA Software Safety Standard, and the NASA Software Safety Guidebook.

- For each hazard report, what are the number and percentages of L1, L2 and L3 causes where L1, L2 and L3 are defined as:

- **L1:** a specific software cause or sub-cause⁴ for a hazard, where a specific software cause *must* include all of the following:
 - o Origin – the CSCI that fails to perform its operation correctly
 - o Erratum – a description of the erroneous command, command sequence or failed operation of the CSCI
 - o Impact – the effect of the erratum where failure to control results in the hazardous condition, and if known, the specific CSCI(s) or hardware subsystem(s) affected
- **L2:** a partially-specified software cause or sub-cause for a hazard, where a partially-specified software cause specifies one or two of the origin, erratum or receiver at the CSCI/hardware subsystem level.
- **L3:** a generically defined software cause or sub-cause for a hazard, where a generically-defined software cause does not specify the origin, erratum or receiver at the CSCI/hardware subsystem level.

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

4.1.5 Step 5: Develop interpretation models and threshold values of measures

We analyzed a total of 154 hazard reports for the three Constellation systems: 77 in Project A, 57 in Project B, and 20 in Project C. The analysis of each hazard report was performed manually by reading the text of the causes and controls. In total, over 2000 causes and nearly 5000 controls were examined. Details of the analysis procedure and additional measures and models are provided in [La11]).

The following interpretation model is related to Goal 1 – understanding the role of software in system safety:

Question: What percentage of the hazards is software-related?

Metric: The number of hazards having a software cause or software control / the total number of hazards.

Interpretation model: While no specific thresholds were set by SR&QA personnel regarding this metric (i.e. “what percentage of software-related hazards is too high?”), the metrics were used to identify subsystems with the most software risk that may require more software verification

⁴ Many software causes contained a number of “sub-causes.” Sub-causes were identifiable by either: 1) explicit enumeration in the cause description by the hazard report author; or 2) separate paragraphs describing errors by different CSCIs. Because sub-causes described different software behaviors, each was measured for its specificity.

effort. Furthermore, these measurements were used to help establish a baseline for these measures to inform future projects.

if % software-related hazards > e then additional software safety design and verification effort is needed; *otherwise* the current level of effort is acceptable.

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

Goal 2 was also interesting as it provided insight into the execution of the hazard analysis process itself.

Question: What percentage of software causes is *generically-defined*? A generically-defined software cause does not specify the origin, erratum or receiver at the CSCI/hardware subsystem level.

Metric: $L3_{total} = \sum_i^{n=\# \text{ hazards}} |L3 \text{ causes and subcauses}|$ = The total number of software causes and sub-causes that do not specify the origin, erratum or receiver at the CSCI/hardware subsystem level

Interpretation model: A “generically-defined” cause is not specific enough to specify a design feature or operational procedure that could function as a control strategy. Thus, all generically-defined causes should be improved to be better specified ($e = 0$):

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

In the three Constellation projects, Project A had 38 (29%) L3 causes, Project B had 37 (22%) L3 causes, and Project C had 16 (29%) L3 causes. None of the projects met the criteria of the interpretation model. At its core, Goal 2 examines process conformance by looking for syntactic information in the cause description. Many causes did not follow the prescribed syntax, and the resulting semantic information was insufficient for identifying controls. As discussed early in the paper, the initial reaction may be to say “you’re not following the process correctly.” However, SR&QA personnel acknowledge that potential non-conformance in this case was not necessarily a failure on the part of the safety engineers; the integration of software safety with traditional hardware-centric system safety process (such as hazard analysis) was still a work in progress. The guidelines for specifying software causes were

scattered over five different standards and process documents, and each project reported and scoped software causes differently. In this case, the lack of process conformance was partially attributed to needing process guidance for the current context.

4.1.6 Step 6: Responses to identified risks

SR&QA integrated the percentage of software-related hazards (along with other measures related to Goal 1) into a risk scorecard of the systems and subsystems with greatest potential software safety risk. SR&QA then used this scorecard to help manage safety support effort. Example responses include:

- If the percentage of software-related hazards for a subsystem is in the top ten of all subsystems for that project, then additional software safety data (e.g. software safety defects, software safety requirements metrics) must be gathered for that subsystem.
- If the percentage of software-related hazards for a subsystem is less than 10%, software safety personnel only need to attend a safety review if the project indicates that software safety concerns have changed.

The data related to Goal 1 were largely used to inform other members of Constellation program management as to the importance of software in overall system safety. These metrics results were surprising to many. The risk scorecard was to be used to track the evolution and improvement of software safety risk over the course of the project. Unfortunately, the program was canceled shortly after this analysis, and thus we could not observe how the data would be used as part of proactive program management.

The responses to risks evaluated in Goal 2 focused on improving the quality of the software cause descriptions. Example responses include:

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

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

4.2 Summary of software safety process risks discovered in the Constellation hazard analysis process

We discovered a number of potential process risks for the program with regard to how software-related hazards are reported. Some of these risks were interpreted from the metrics, and others were findings from analysis of other goals on the project.

N-1: Lack of consistency in structuring hazard report content, causes and control descriptions impairs understanding.

N-2: Lack of consistent scope in causes and controls impairs risk assessment.

N-3: “Lumped” software causes and controls impede verification.

In order to address these three risks, 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 any clear indication of the level of detail or the format of this information. Our guideline, if followed, provides consistency across development teams, which allows CSERP to more readily monitor process risks across multiple developments.

N-4: Incorrect references to hazard reports, causes and controls impair traceability.

N-5: Sub-controls dissuade independent verification and add overhead.

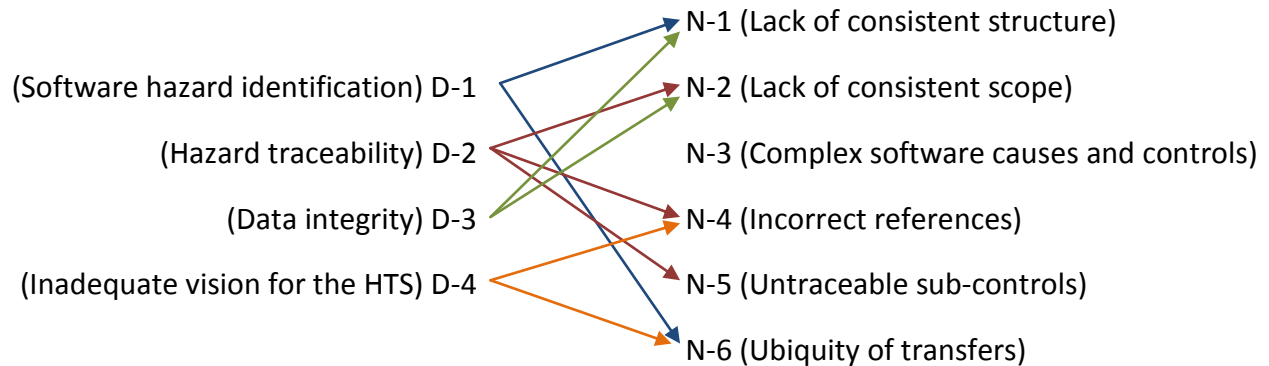
N-6: Ubiquity of transferred causes and controls may mask software risk. A transferred cause or control indicates that the same cause or mitigation strategy is given in another hazard report. This inhibits traceability in verifying the controls to mitigate a cause.

We have proposed a minor restructuring of the format of the HR to allow for better traceability. In addition, we developed a prototype front end for the HTS that provides clear visibility of each hazard, cause and control. Additional information on the risks uncovered and proposed and implemented program responses can be found in [Layman, 2011].

5 Case Study Results and Lessons Learned

5.1 Process risks common to the DoD and Constellation programs

We applied the PRA method to the DoD system and NASA Constellation program to improve visibility into the state of software safety on both programs. The risks we uncovered in the DoD system (Section 3.3) and the Constellation program (Section 4.2) were similar in a number of cases:



Three themes emerge from these risks:

- 1) **The inability to track software hazards and software safety** requirements against the backdrop of system hazards and system safety requirements. In both case studies, software safety leadership had no easy way to assess the current state of software safety early in the development lifecycle. The safety analysis processes did not adequately distinguish between software and non-software safety concerns. This forced software safety management to manually collate and track this information, which became an expensive, but necessary, requirement for proactive software safety management across these large projects. Tracking software safety concerns would be easily enabled if safety requirements, hazards, and other safety analysis artifacts contained meta-data (e.g. a checkbox) indicating whether or not those items were software related.
- 2) **Inadequate traceability** exists between safety requirements, hazards, causes and controls. In both programs, bi-directional traceability is required between safety requirements, associated hazards, causes, controls and verifications. This is so that all safety requirements can be traced to their implementation in the completed system. In both programs, bi-directional traceability was not present in the artifacts we examined. Retroactively inserting this traceability as the project nears completion (as is the historical practice) is extraordinarily expensive.
- 3) **Inconsistent scope and unstructured details** when describing hazards, causes and controls. In both programs, safety engineers wrote their hazards, causes and controls in unique ways. This made consistent evaluation on the part of safety management and expert engineering teams difficult. Furthermore, the inconsistent structure prevented an automated syntactic

analysis of the safety artifacts. This was not simply a matter of non-conformance to the safety analysis processes, but was the result of different interpretations of processes that were not concretely defined. For both programs, elevating software safety to a level of importance equivalent to hardware and system safety was challenging enough. Defining how software should be incorporated into traditionally hardware-oriented analyses (such as hazard analysis) was still very much a work in progress for both programs.

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

5.2 Lessons learned for implementing software safety analysis processes on future programs

In both case studies, the programs were attempting to integrate software safety with traditional safety processes that originated in hardware and system reliability. These programs were both learning the best ways to incorporate software into the existing processes. Indeed, it can be a struggle to convey the importance of software safety analysis in a systems engineering environment. 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 [Nu97], THERAC-25 [Le93]) 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: it requires different expertise, software controls are more difficult to specify because software properties and constraints are not well understood early on, and the software developers rather than the software design are the greatest source of risk.

Software safety risk management should not be isolated, however. The interaction of hardware and software is a source of risk, and software failures will almost always manifest in some loss of system function. Software safety risk management must be integrated in the overarching system safety process in such a way to account for the unique ways that software risk must be managed, while recognizing that software safety risk is of equal importance in the overarching system safety process. It is not as simple as saying “apply the hazard analysis process to software,” however. As we observed in the Constellation case study in particular, a lack of consistent understanding of how to apply hazard analysis to software led to both over specified and underspecified software cause descriptions. The CSERP review was not interested in reading about state-transitions in software that would undoubtedly change, nor was a description of “the software fails” good enough. The safety engineers often spent several iterations simply getting the *description* of the cause to be acceptable before an analysis of the actual design features could even begin. Providing guidance to the safety engineers on how to specify software causes (e.g. through the “user guide” mentioned in Section 4.2) 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 semantic meaningful.

In our case studies, we observed inadequate planning in the use of the hazard tracking systems, which impaired our ability to track and measure software safety risks. The first was the use of the hazard tracking systems. Ostensibly, these systems were designed for three purposes: 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 automatic support for traceability between hazards and hazard attributes. In both case studies, it was clear that goals for searching, filtering and analyzing hazard reports to support risk management were not fully considered. For example, there was no way to identify software-related hazards among the set of system hazards, despite the fact that software safety management was handled by special groups in each program. Several hundred person-hours were spent to analyze the Constellation hazard causes and controls to identify merely if software was involved. However, as the authors were able to demonstrate in a prototype HTS, adding a checkbox next to each cause or control denoting “this is software-related” enabled an automated search for software causes and controls that took mere seconds instead of hundreds of hours. Also in both systems, the traceability features were not used by the safety engineers, who preferred to author the hazard reports in a word processor and then copy and paste in the HTS. The HTSes enabled a number of useful features regarding traceability, such as verifying that references and links are still valid, automatically updating traceability links, and detecting dependencies. However, these capabilities are only useful when the users do their part to enter and maintain accurate traceability information. Process risk and unnecessary effort could be avoided if more consideration is given to merging the powerful capabilities of a database storage solution with user’s preferred methods of working and management’s need to track and analysis program risk.

We also learned that the vendor acquisition process must promote the importance and iterative measurement of software safety. The sources of information that can be used to measure software safety is limited to the process artifacts that are provided by the vendor. 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. Similarly, 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 unmeasurable.

5.3 Lessons learned on applying the PRA methodology

We have seen that the approach has been effective in uncovering potential risks and making visible early in the development cycle concerns that need to be addressed right away to support the achievement of the emergent property, safety in this case. We have also uncovered

several lessons learned and observed some limitations in applying the PRA method over the course of these two case studies.

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

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

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

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

Models and interpretations help shape goals. Goals, measures, models will vary according to when the measurement takes place as the process can have different expectations/outputs at

different points in time. For example, both projects we studied were in the preliminary design phase when only requirements and high-level designs were available for safety analysis. The responses to the identified process risks were to improve the process and provide process support, and the response to product risk is to change the design of the system. At a later stage of development, say testing and verification, one could obtain more concrete metrics on the state of system safety with respect to actual system behavior. At that point, the responses are more limited in that changing design and implementation will be extraordinarily expensive (especially in a large system).

6 Conclusion

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

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

The PRA method provides visibility into process risks throughout the development lifecycle by measuring process conformance through the analysis of syntactic and, possibly, semantic information contained in intermediate process artifacts. *It is important to note that the PRA method does not and cannot provide any assurance that the system is safe.* It only provides indicators that there is a risk that the system will not be safe and provides insights into why and how the exposed issues might be fixed while the system is still under development.

We have provided two examples of applying the PRA method in case studies of software safety processes on the NASA Constellation program and a large, network-centric Department of Defense system of systems. We uncovered several risks in the software safety processes of both 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 will apply the PRA method to identify process risks in achieving other emergent system properties, such as reliability and security.

Acknowledgements

This research was supported by NASA OSMA SARP grant NNX08AZ60G to the Fraunhofer CESE. We would like to acknowledge the help of Frank Marotta at the US Army Aberdeen Test Center 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 Program.

References

- [BA84] Basili, V. R. and Weiss, D., "A Methodology for Collecting Valid Software Engineering Data", *IEEE Transactions On Software Engineering*, Nov. 1984, 728-738.
- [Ba88] V.R. Basili, H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Transactions on Software Engineering*, 14(6):758-773, June 1988.
- [Ba08] V. R. Basili, K. Dangle, L. Esker, F. Marotta, and I. Rus, Measures and Risk Indicators for Early Insight Into Software Safety, *CrossTalk: The Journal of Defense Software Engineering*, 21(10): 4-8, October 2008.[MIL] MIL-STD-882, DoD Standard Practice for System Safety, 10 February 2000.
- [CxP70038] National Aeronautics and Space Administration, "Constellation Program Hazard Analyses Methodology", CXP-70038, Revision B, April 22, 2009.
- [FAA08] Federal Aviation Administration, "System Safety Handbook", accessed October 15, 2010, http://www.faa.gov/library/manuals/aviation/risk_management/ss_handbook/, updated May 21, 2008.
- [LA11] L. Layman, V. R. Basili, M. V. Zelkowitz, "A Case Study of Measuring Process Risk for Early Insights into Software Safety," Proceedings of the 33rd Int'l. Conf. on Software Engineering (ICSE '11), Honolulu, HI, to appear.
- [Le93] N.G. Leveson, C.S. Turner, "An investigation of the Therac-25 accidents", *IEEE Computer*, 26(7):18-41, July 1993.
- [MIL-STD-882] Department of Defense, "Standard Practice for System Safety", MIL-STD-882D, February 10, 2000.
- [Nu97] B. Nuseibeh, "Ariane 5: Who Dunnit?", *IEEE Software*, 14(3):15-16, May/June 1997.

APPENDIX A – The Hazard Tracking Process

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

Hazard analysis is a top-down approach to system safety analysis [FAA08]. In hazard analysis, a hazard is any real or potential condition that can cause: injury, illness, or death to personnel; damage to or loss of a system, equipment, or property; or damage to the environment. An example of a hazard might be “Avionics 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 are documented and require the addition of several important properties:

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

Figure A.1 illustrates the conceptual organization of a hazard. Each hazard (e.g., engine failure) has one or more causes (e.g., failure with fuel line, software turns off the engine). Each cause has one or more controls that reduce the likelihood that a cause will occur or mitigates the impact should the cause be realized; controls often represent new requirements for the system (e.g., backup computers to account for software failures, redundant hardware). Each control has one or more verifications (e.g. test, inspection, simulation or demonstration) to ensure that the control is appropriately implemented.

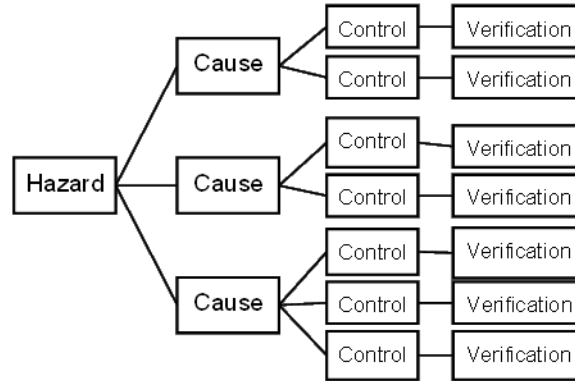


Figure A-1. Hazard structure

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

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