Assurance Management Instructor’s Guide

INSTRUCTOR MATERIALS

Assurance Management

CERT Division

<http://www.sei.cmu.edu>

**http://www.cert.org**

Copyright 2014 Carnegie Mellon University  
  
This material has been approved for public release and unlimited distribution except as restricted below.  
  
This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.  
  
References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.  
  
NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.  
  
This material is distributed by the Software Engineering Institute (SEI) only to course attendees for their own individual study. Except for the U.S. government purposes described below, this material SHALL NOT be reproduced or used in any other manner without requesting formal permission from the Software Engineering Institute at permission@sei.cmu.edu.  
  
The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose this material are restricted by the Rights in Technical Data-Noncommercial Items clauses (DFAR 252-227.7013 and DFAR 252-227.7013 Alternate I) contained in the above identified contract. Any reproduction of this material or portions thereof marked with this legend must also reproduce the disclaimers contained on this slide.  
  
Although the rights granted by contract do not require course attendance to use this material for U.S. Government purposes, the SEI recommends attendance to ensure proper understanding.  
  
CERT® is a registered mark of Carnegie Mellon University.  
  
DM-0001287

Table of Contents

[Overview 5](#_Toc388442941)

[Project and Exercises 8](#_Toc388442942)

[Schedule 9](#_Toc388442943)

[1 Module: Introduction 11](#_Toc388442944)

[1.1 2003 Power Grid Blackout 11](#_Toc388442945)

[1.2 The Power Grid Failure 12](#_Toc388442946)

[1.2.1 Software Assurance Analysis 13](#_Toc388442947)

[2 Module: Risk Management 14](#_Toc388442948)

[3 Module: Risk Management Process 15](#_Toc388442949)

[4 Module: Assurance Case 17](#_Toc388442950)

[4.1 From Weakness Removal (Static Analysis) to Weakness Prevention 17](#_Toc388442951)

[4.2 Power Grid Assurance Case 19](#_Toc388442952)

[4.2.1 Other Observations 21](#_Toc388442953)

[4.3 Sustainment 21](#_Toc388442954)

[5 Module: Confidence 23](#_Toc388442955)

[5.1 Eliminative Induction 23](#_Toc388442956)

[5.2 Software Assurance Justification 23](#_Toc388442957)

[5.3 Reviews and Development as an Assurance Case 25](#_Toc388442958)

[5.4 Eliminative Induction 25](#_Toc388442959)

[5.5 Infusion Pump Assurance Case 27](#_Toc388442960)

[6 Module: Writing an Assurance Case 30](#_Toc388442961)

[7 Module: Building Security In Maturity Model (BSIMM) 31](#_Toc388442962)

[8 Module: Open Group Assurance Case 32](#_Toc388442963)

[9 Module: Systems and System of Systems 33](#_Toc388442964)

[Project 36](#_Toc388442965)

[References/Bibliography 37](#_Toc388442966)

Overview

The course design assumption is that instructors will draw material from specific modules to complement their own material as well as to better match the backgrounds of the students in a class. The instructor’s guide provides detailed information about the examples used in lectures and on areas such as assurance cases. Much of the material in this guide was drawn from two SEI reports that are available at <http://resources.sei.cmu.edu/library/>

* Assuring Reliability, Robert Ellison
* Quality and Software Assurance, Robert Ellison, William Nichols, and Carol Woody

The course is organized into content areas, called modules. In most cases, a module represents a week of lectures. Building competencies in the two key concepts, risk management and assurance case analysis, requires practice. Reports on team projects or in-class development of examples are recommended for the weeks following the lectures on risk management and assurance case analysis.

Scope of Software Assurance

The definition software assurance in the *Master of Software Assurance Reference Curriculum* published in 2010 is

**Software Assurance:** Application of technologies and processes to achieve a required level of confidence that software systems and services function in the intended manner, are free from accidental or intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures.

The scope of software assurance has evolved since 2010 when the curriculum was published. For example, many of the assurance analysis techniques are applied to reliability and safety. To avoid confusion, this course uses an attribute-independent definition of software assurance

**Software Assurance:** The application of technologies and processes to achieve a required level of confidence that software systems and services function in the intended manner.

and renames the reference curriculum definition as

**Security Assurance:** Application of technologies and processes to achieve a required level of confidence3 that software systems and services function in the intended manner, are free from accidental or intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures.

Key Concepts

Risk Management

Risk management is an essential aspect of assurance management but the analysis techniques depend on the context.

* What is the role of the risk assessor in the supply chain? Is the organization supplying or acquiring software?
* What is the scope of the analysis? Is the risk analysis being applied to a software application, to a system, to system of systems, or to an enterprise?

Module 2 provides an introduction to risk management and uses threat modeling as an example of incorporating risk management into a software development lifecycle. Module 3 introduces a general risk management framework.

Determining the scope of risk management depends on which risks can be realistically controlled. Large systems increasingly integrate commercial software with unknown software weaknesses. Threats and mitigations are increasingly difficult to identify as work processes flow among multiple organizations. Some of these issues are discussed in Module 9.

Assurance Case: Justify the Engineering

The outline for the Assurance Management Course in the Volume III: Master of Software Assurance Course Syllabi (Course Syllabi) refers several times to establishing and meeting “Targeted Level of Trust”. The same expression appears in the ISO 27034 standard on application security. The expression “Targeted Level of Trust can be interpreted as phrase “the desired level of confidence” that appears in the definition of software assurance.

The feasibility of achieving high assurance for a particular system is strongly influenced by early engineering choices. In particular, assessing assurance as a system is being developed had a high potential for improving the overall assurance of systems[[1]](#footnote-1).

Several studies of safety-critical systems show that while 70 percent of errors in embedded safety-critical software are introduced in the requirements and architecture design phases. 80 percent of all errors were only found at system integration or later [Feiler 2012]. The development model was a series of builds and tests. Defect removal was at the end of the lifecycle.

A key point is that an evaluation of engineering decisions is not an evaluation of how those engineering decisions are made, i.e. not an evaluation of the development practices and life cycle processes but rather an evaluation of the effectiveness of the engineering methods used to mitigate the identified risks, i.e. an evaluation of what was done and how well it was done. Module 4 introduces the assurance case which provides a concise way and understandable way to document and justify those engineering decisions. Guidance on writing an assurance case is provided by Module 5. The Open Group work real-time reliability for embedded systems is based on the concept of an assurance case.

Among the system attributes, security assurance may be one of the most challenging to achieve. Security has to deal with active agents who can dynamically adjust an attack based the defenses found. Assurance problems for most of the system attributes such as reliability arise in requirements or design. But security assurance is also affected by implementation defects in a detailed design or coding. The latter class of defects may not need the formalism provided by an assurance case and can be sufficiently mitigated with training, static analysis and coding standards, but as instances of the latter class of defects are reduced, assurance based on engineering analysis will increase in importance. A key aspect of Microsoft’s Security Development Lifecycle is using threat modeling to identify and mitigate defects during design.

Security assurance cases can be difficult to write. Limit the scope of any assurance case exercises and examples. The problem is not with the assurance case format but that writing an assurance case exposes the complexity of the security issues and the areas where our analysis techniques are weak.

Additional Sources

The Course Syllabi lists a number of sources for the course. Among those referenced, the following one provides coverage of assurance include assurance cases.

Mansourov, Nicolai & Campara, Djenana. *System Assurance: Beyond Detecting Vulnerabilities.* Elsevier, 2011.

The ISO 27034 standard on application security is underdevelopment. The content may be too general for use in the course.

The following standard applies assurance case analysis to real-time embedded systems.

Open Group, Dependability through Assuredness ™ Standard, <http://www.opengroup.org/news/press/open-group-releases-dependability-through-assuredness-%E2%84%A2-standar>.

Project and Exercises

Writing a business case is a very good topic area for the assurance management course. Projects could individually done or be group efforts. BSIMM can be a source for a wide-range of business cases targeting project and technical management to corporate management. For example, a business case could be written for obtaining an Open Group certification or for why a software development organization should add security capabilities beyond doing static analysis or why security analysis beyond that provided by using the CISQ Quality Rules for security is needed for a specific development.

Business cases can be introduced after the risk management lectures as doing a risk assessment typically requires some kind of business justification.

Students should consult sources such as the articles on “Making the Business Case for Software Assurance” on the DHS Build Security In Web site.

<https://buildsecurityin.us-cert.gov/articles/knowledge/business-case-models>

An SEI report is also available which combines the materials on the DHS web site.

“Making the Business Case for Software Assurance,” <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8831>

Schedule

The pace of this course is very dependent on the background of the students. The ability of a student to do risk analysis or to analyze/develop the engineering justifications described in an assurance case for a system attribute depends on the extent of the exposure to the kinds of security defects that can arise during development and operations, the potential consequences of such defects on business work processes, and the engineering options available.

The student needs vary. Some personnel with roles in assurance management will not have to do detailed risk assessments or develop specific assurance cases. On the other hand, those with technical management responsibilities may have to provide the technical leadership for those efforts and monitor the quality of the effort.

Risk assessments and engineering assessments require practice. An assumption is that at least three weeks of the course will be devoted to discussions of student projects or presentations: Students could report on a group project or a number of lectures could be devoted to working through an assurance case or a risk assessment example.

| **Area** | **Topic** | **Module** | **Estimated time** |
| --- | --- | --- | --- |
|  | Course introduction | 1 | 1 week |
| Risk management | Introduction and threat modeling | 2 | The combination of modules 2 and 3 along with student exercises or lecture-based examples will take at least 2 weeks. |
| Security Engineering Risk Analysis  Method (SERA): | 3 |  |
| Assurance case analysis | Introduction | 4 | Modules 4 and 5 with in-class examples, should take two weeks. Another week could be devoted to discussions of student developed of assurance case segments. |
| Confidence Map | 5 |  |
| Constructing assurance cases | 6 | This module could be read by students outside of a lecture. |
| Developer capabilities | Building Security In Maturity Model (BSIMM): Module 7 | 7 | 1 week |
| Open Group’s Supply Chain Standard Accreditation as an assurance case | 8 | 1 week |
| System and systems of systems security assurance |  | 9 | 1 week |

# Module: Introduction

## 2003 Power Grid Blackout

The increased system complexity fundamentally changes how we should do causal analysis. Currently, a security compromise can often be traced to a root cause, but security compromises are increasingly caused by unanticipated interactions across multiple systems. A typical system is increasingly likely to be integrated from independently developed (COTS) components. Causal analysis for safety failures has changed to reflect the trend and security will eventually have to follow.

A MITRE technical staff member observed in a conversation with us that many of the users of the CWE make repeated visits to the web site seeking information for specific weakness. They do not appear to step back and take a higher-level, organizational or systems perspective on how to improve security.

The 2003 Power Grid black is a good example a failure whose causes spanned computing systems, operational practices, and operator failures. In particular, the analysis team implicitly used system assurance techniques.

An event-based causal analysis is often used for a hardware failure. Such an analysis identifies the sequence of events that preceded the failure, which are then analyzed to identify a root cause. But such an event-based analysis for failure with a complex system can be misleading. Leveson’s analysis of safety failures showed that the causes were frequently the concurrent occurrence of several events. The absence of any one of those events would have prevented the failure.

Many security failures have a root cause such as one of the CWE weaknesses. But with complex systems a security failure can involve multiple factors.

As example of the weaknesses of an event-based analysis for complex systems, consider the release of methyl isocyanate (MIC) from a Union Carbide chemical plant in Bhopal, India in 1984. A relatively new worker had to wash out some pipes and filters that were clogged. MIC produces large amounts of heat when in contact with water, and the worker did close the valves to isolate the MIC tanks from the pipes and filters being washed. However, a required safety disk which backed up the valves in case they leaked was not inserted. The valves did leak—which lead to 2,000 fatalities and 10,000 permanent injuries. The analysis identified the root cause as an operator error. Charles Perrow’s*[[2]](#footnote-2)* analysis of the Bhopal incident concluded that there was no root cause, and that given the design and operating conditions of the plant, an accident was just waiting to happen. His argument was

However [water] got in, it would not have caused the severe explosion

* had the refrigeration unit not been disconnected and drained of Freon,
* or had various steps been taken at the first smell of MIC instead of being put off until the tea break,
* or had the scrubber been in service,
* or had the water sprays been designed to go high enough to douse the emissions,
* or had the flare tower been working and been of sufficient capacity to handle a large excursion.

The 2003 power grid blackout was a reliability failure for the power grid control system for an Ohio utility. There had only been minor errors encountered with that control system, and hence model-based reliability analysis would have predicted high reliability. This section describes the engineering review that followed the blackout that came to the conclusion that the conditions were poor enough that a black-out was ready to happen. That review considered the blackout as a software assurance failure and implicitly developed an assurance case to identify the system weaknesses.

## The Power Grid Failure

On August 14, 2003, approximately 50 million electricity consumers in Canada and the northeastern U.S. were subject to a cascading blackout. The events preceding the blackout included a mistake by tree trimmers in Ohio that took three high-voltage lines out of service and a software failure (a race condition[[3]](#footnote-3)) that disabled the computing service that notified the power grid operators of changes in power grid conditions. With the alarm function disabled, the power grid operators did not notice a sequence of power grid failures that eventually lead to the blackout [NERC 2004].

The technical analysis of the blackout explicitly rejected tree-trimming practices and the software race condition as root causes. Instead if we phrase the conclusion like Perrow’s, it would be

However the alarm server failed the blackout would not have occurred

* if the operators had not been unaware of the alarm server failure,
* or if a regional power grid monitor had not failed,
* or if the recovery of the alarm service had not failed,
* or …... [NERC 2004]

A basic understanding power grid reliability requirements and monitoring capabilities is required to analyze the causes and mitigations for the blackout. Power grid operators typically have 30 to 60 minutes to respond to an alarm raised because a generator is out of service or adverse conditions have led to transmission lines being automatically disconnected from the power grid. The technical analysis sponsored by the North American Electric Reliability Corporation (NERC) provides the following summary of the reliability requirements and power grid monitoring activities.

**Reliability requirement:** The electricity industry has developed and codified a set of mutually reinforcing reliability standards and practices to ensure that system operators are prepared to deal with unexpected system events. The basic assumption underlying these standards and practices is that power system elements will fail or become unavailable in unpredictable ways. The basic principle of reliability management is that “operators must operate to maintain the safety of the system they have available.”

**Power grid monitoring:** It is common for reliability coordinators and control areas to use a state estimator to monitor the power system to improve the accuracy over raw telemetered data. The raw data are processed mathematically to make a “best fit” power flow model, which can then be used in other software applications, such as real-time contingency analysis, to simulate various conditions and outages to evaluate the reliability of the power system. Real-time contingency analysis is used to alert operators if the system is operating insecurely; it can be run either on a regular schedule (e.g., every five minutes), when triggered by some system event (e.g., the loss of a power plant or transmission line), or when initiated by an operator [NERC 2004].

### Software Assurance Analysis

The software subsystem that provided audible and visual indications when a significant piece of equipment changed from an acceptable to problematic status failed at 14:14.

* The data required to manage the utility’s power grid continued to be updated on a power grid operator’s control computer.
* After the server failure the power grid operator’s displays did not receive any further alarms, nor were any alarms being printed or posted on the alarm logging facilities.
* The power grid operators assumed that alarm service was operating and did not observe that system conditions were changing.

# Module: Risk Management

Threat modeling is used as an example of risk management as applied to application development. The article from Microsoft authors provides a quick refresh on threat modeling concepts.

“Uncover Security Design Flaws Using the STRIDE Approach” <http://msdn.microsoft.com/en-us/magazine/cc163519.aspx>

The Dan Geer tutorial on metrics is excellent. <http://geer.tinho.net/measuringsecurity.tutorial.pdf>. Other good Geer articles include

“Attack Surface Inflation”, Security and Privacy, August/September 2011

“Advanced persistent threat”, <http://www.networkworld.com/news/tech/2010/041210-tech-update.html>

**Exercises:** The examples in the lectures represent risk analysis done after the event. Students could be asked to do such an exercise for a recently publicized compromise or for one that they have encountered. Group exercises are a good option for risk assessments as then knowledge of known risks and mitigations can be shared among the team members.

# Module: Risk Management Process

The Security Engineering Risk Analysis (SERA) approach described in Module 3 provides a general risk management framework that is applicable to managing business risks.

NIST 800-39, “Managing Information Security Risk” describes the risk management framework used by the US Federal Government include the Department of Defense.

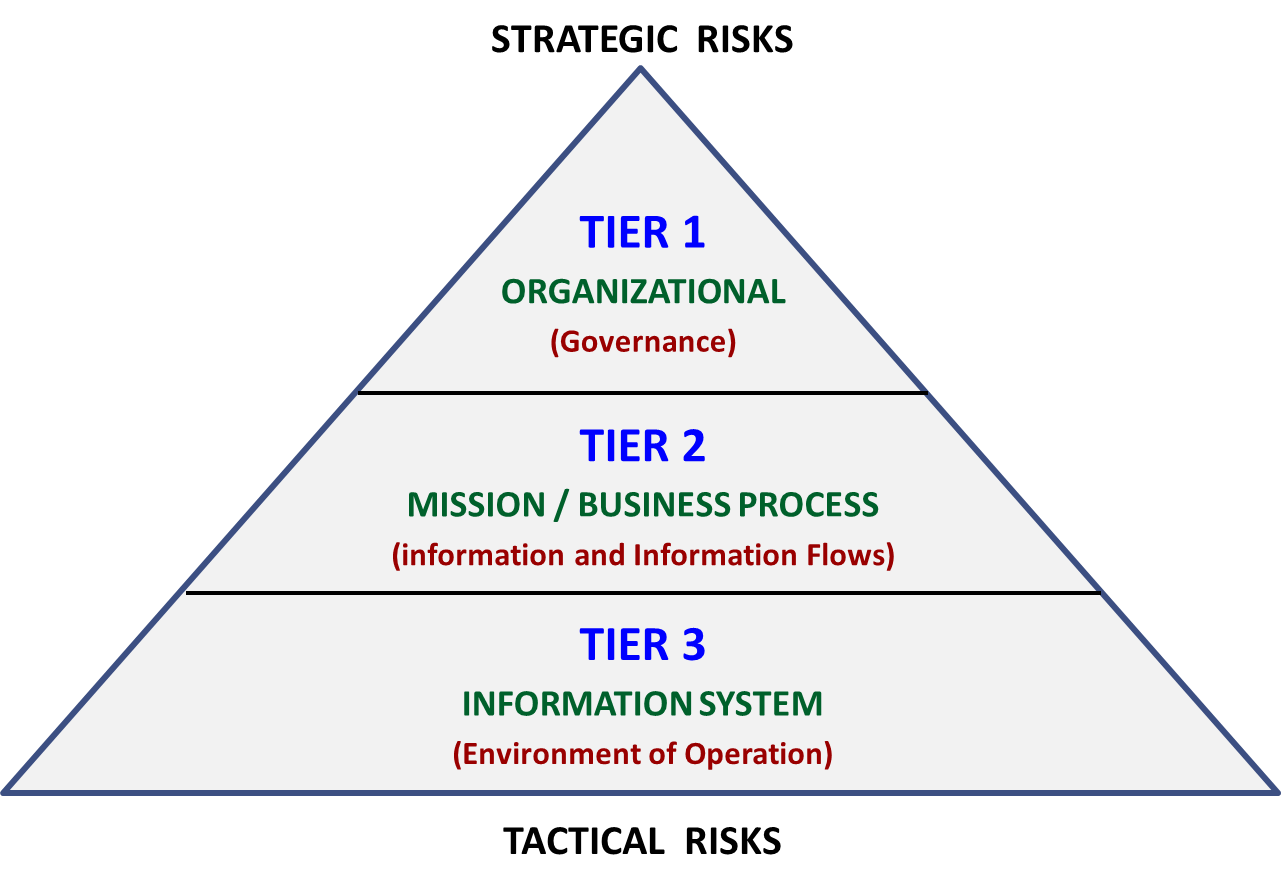


Figure : NIST Risk Management Framework

Sources for SERA

* Alberts, Christopher & Dorofee, Audrey. *Mission Risk Diagnostic (MRD) Method Description* (CMU/SEI-2012-TN-005). Software Engineering Institute, Carnegie Mellon University, 2012.   
  <http://www.sei.cmu.edu/reports/12tn005.pdf>
* Alberts, Christopher; Allen, Julia; & Stoddard, Robert. *Risk-Based Measurement and Analysis: Application to Software Security* (CMU/SEI-2012-TN-004), Software Engineering Institute, Carnegie Mellon University, 2012. <http://www.sei.cmu.edu/reports/12tn004.pdf>
* Alberts, Christopher & Dorofee, Audrey. *A Framework for Categorizing Key Drivers of Risk* (CMU/SEI-2009-TR-007). Software Engineering Institute, Carnegie Mellon University, 2009. <http://www.sei.cmu.edu/library/abstracts/reports/09tr007.cfm>
* SEI Mission Success in Complex Environments (CSE) Special Project  
  <http://www.sei.cmu.edu/risk/>

The following is a case study that could be assigned as a team-based exercise.

* Document the target of the analysis (i.e., software or system being developed) and describe how will be used during operations. (15%)
* Select one risk scenario that you intend to analyze. Document the following for the selected risk: threat, consequence, and enablers (35%)
* Document a risk statement (in if-then format) for the selected risk. (10%)
* Document mitigation actions for the selected risk. Identify actions that can be addressed during development. (35%)
* Describe what insights you gained (if any) by applying the method. (5%)

# Module: Assurance Case

## From Weakness Removal (Static Analysis) to Weakness Prevention

This section includes material developed after the slides for the assurance case were written. A recommendation would be to present this content in advance of the lectures on assurance cases.

The Building Security In Maturity Model (BSIMM) has Code Review as one of its twelve practice areas. Over 50 of the 67 firms whose security practices have been collected by BSIMM surveys have automated code reviews[[4]](#footnote-4). But as noted by McGraw, incorporating automated code analysis, into the development lifecycle encounters a number of issues of scale. The resource requirements have increased as these have matured to cover a broad range of vulnerabilities, and it may not be feasible to run them on a desktop computer. Scans can take two to three hours which makes it difficult to use them for agile development.

The Consortium for IT: Software Quality (CISQ) has developed specifications for automating measurement for the reliability, security, performance efficiency, and maintainability which complements static analysis [CISQ 2012][[5]](#footnote-5). The CISQ assessment of the structural quality of software is based on the analysis and measurements for violations of rules of good coding and practices that can be detected by analyzing the source code.

The good coding and practices rules are called Quality Rules. The Quality Rules for security are drawn from the CWE as those weaknesses are examples of violations of good practices. For example, CWE-129 is improper validation of an array index. CISQ selected CWE entries associated with vulnerabilities in the CWE/SANS 25 and the OWASP Top Ten lists.

Consider a subset of the CISQ security rules that appear in Table 1. The CWE describes multiple ways to mitigate many of the weaknesses. For example, consider Rule 2. A Standard Query Language (SQL) injection occurs when user input is used to construct a database query written in SQL. A poorly written input validation function can enable an attacker to access, change, or add entries to the database by including SQL expressions in the input provide. An experience programmer should be able to write an input routine that identifies SQL expressions in the input data. It can be difficult to automatically verify that a programmer-written routine prevents a SQL Injection. Instead CIAQ Quality Rule follows the CWE mitigation to use a vetted library or framework that is known to mitigate this vulnerability. Validating the using such a library can be automated and in addition the use of that library does not require the coder to have extensive security expertise.

Table : Subset of CISQ Security Rules

|  |  |  |
| --- | --- | --- |
| Issue | Quality Rule | Quality Measure Element |
| CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') | **Rule 1:** Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid, such as Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. | **Measure 1:** # of instances where output is not using library for neutralization |
| CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') | **Rule 2:** Use a vetted library or framework that does not allow SQL injection to occur or provides constructs that make this SQL injection easier to avoid or use persistence layers such as Hibernate or Enterprise Java Beans. | **Measure 2:** # of instances where data is included in SQL statements that is not passed through the neutralization routines. |
| CWE-129: Improper Validation of Array Index | **Rule 8:** Assume all input is malicious. When accessing a user-controlled array index, use a stringent range of values that are within the target array. Make sure that you do not allow negative values to be used. That is, verify the minimum as well as the maximum of the range of acceptable values. | **Measure 8:** # of array accesses with user input that IS NOT range checked |

The CISQ work provides is a good transition from simply finding defects by static analysis to incorporating prevention into development. The CISQ approach and static analysis are based on an analysis of developed source code, but there is a significant difference between the two approaches. The Quality Rules incorporate software engineering recommendations such as the use of vetted libraries that prevent specific vulnerabilities rather than depend on identifying defects after they have been created. Caper Jones[[6]](#footnote-6) in his analysis of quality practices noted that the combination of defect prevention and removal provided the best overall improvement in software quality.

Defect prevention typically depends on analysis that extends beyond a single software component. As noted by OMG [OMG 2013], detecting backdoors, cross-site scripting vulnerabilities (XSS) or unsecure dynamic SQL queries through multiple layers requires a deep understanding of all the data manipulation layers as well as the data structure itself. Overall, security experts Greg Hoglund and Gary McGraw believe cross‐layer security issues account for 50% of all the security issues [Hogood 2004].

The security analysis for XSS issues needs to consider what OMG refers to as System Level Analysis: the ability to analyze all the different code units and different layers of technology to get a holistic view of the entire integrated business application. System level analysis allows us to visualize complete transaction paths from user entries, through user authentication and business logic, down to sensitive data access.

The shift from components to systems requires that we apply quality measures to designs and engineering choices. Software reliability is a good example why such a shift is required.

Several studies of safety-critical systems show that while 70 percent of errors in embedded safety-critical software are introduced in the requirements and architecture design phases [Feiler 2012], 80 percent of all errors are only found at system integration or later. In particular, these errors were not found in unit testing. The rework effort to correct requirement and design problems in later phases can be as high as 300 to 1,000 times the cost of in-phase correction, and undiscovered errors likely remain after that rework.

Similar problems appeared with patient-controlled analgesia infusion pump that are used to infuse a pain killer at a prescribed basal ﬂow rate [FDA 2010]. The US Federal Drug Administration (FDA) uses a premarket assessment to certify the safety and reliability of medical infusion pumps before they are sold to the public. In spite of the FDA’s assessment, too many approved pumps exhibited hardware and software defects in the field, leading to death or injury of patients [FDA 2010]. From 2005 through 2009, 87 infusion pump recalls were conducted by ﬁrms to address identiﬁed safety problems. These defects had not been found during development by testing and other methods. After an analysis of pump recalls and adverse events the FDA concluded that many of the problems appeared to be related to deﬁciencies in device design and engineering.

The recommendations in the 2005 *Department of Defense Guide for Achieving Reliability, Availability, and Maintainability* (RAM) guide are to emphasize systems engineering design analysis and conduct formal design reviews for reliability. The FDA had to revise their remarket assessment to identify defects in device design and engineering. Both organizations had to consider how engineering decisions such as those made during design could reduce defects, i.e. improve quality.

## Power Grid Assurance Case

The 2003 electric power grid blackout is a good example of the kind of causal analysis that is required to improve the reliability of complex systems. A failure analysis based just on events could have concluded that the primary cause of the blackout was the failure of the alarm server caused by a race condition. The reliability problems were far more serious as the operational resiliency was so poor that a blackout was just ready to happen.

The Power Grid blackout is a good example of the role that management role that system assurance can play. The details of the failure are discussed in Module 1.

A key observation by the technical reviewers was that the blackout would not have occurred if the operators had known the alarm service had failed. Instead of analyzing the details of the alarm server failure, the reviewers asked why the following software assurance claim had not been met.

**Claim:** Power grid operators had sufficient situational awareness to be able to manage it in a manner that meets the reliability requirements.

The blackout analysis then identified multiple ways that the situational awareness claim could be satisfied. Figure 2 shows those possibilities as an assurance case where only one out of the six subclaims is required. For example, a 10-minute recovery time for the alarm server should be sufficient. Responses to adverse power grid conditions can often take an hour or longer. The level of confidence required for an electric utility requires concurrently available alternatives. An implementation of all six is realistic.

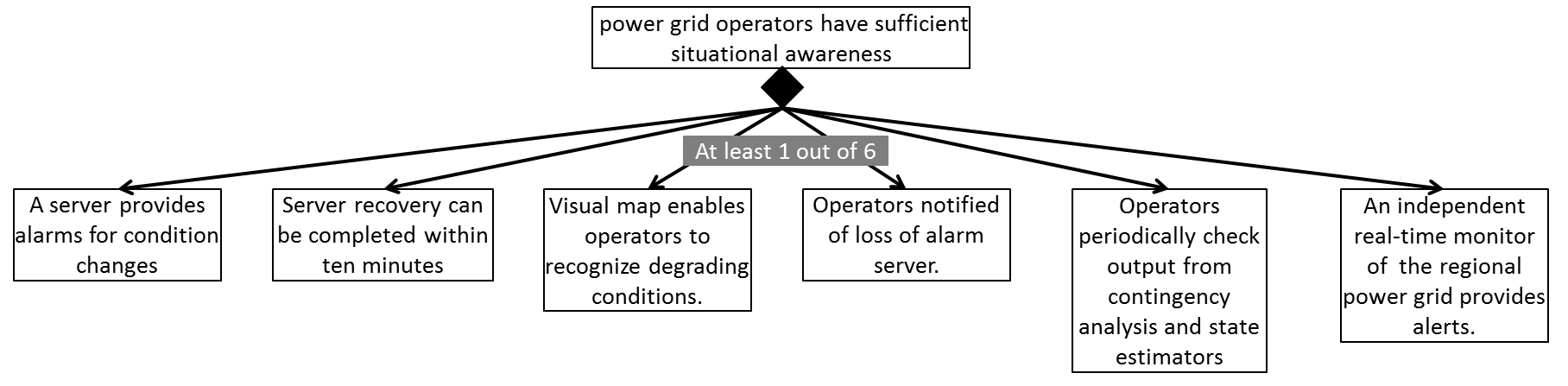


Figure : Alternate Ways to Provide Situational Awareness

The description of the alternatives and their status at the time of the blackout is as follows:

* The alarm server recovery service was designed for a hardware failure. The alarm service did fail over to the secondary server, but the primary server had stalled because of the race condition prevented it from accessing data. The secondary server was in the same stalled state. A warm reboot of the alarm service failed. The supplier of that control system told the computer support staff that a full restart of the energy management system was required which could take more than 30 minutes.
* A dynamic map board or other type of display that showed data relative to locations on the grid might have enabled the operators to recognize significant line and facility outages within the controlled area. Unlike many transmission system control centers, this utility power management center did not have a map board.
* The power grid operators could have been notified that the alarm service was not available by the computer support staff. But there was no formal procedure for such a notification. It appears that the operators only became aware of the lack of the alarm service about 90 minutes after its failure and only 20 minutes in advance of the final cascade of failures.
* The power grid operators could have learned of the change in conditions by looking at the output of the state estimators and real-time contingency analysis tools. But problems had been experienced with the automatic contingency analysis operation since the system was installed in 1995. As a result, the practice was for the operators or engineers to run contingency analysis manually as needed. Hence the operators did not access the contingency analysis results at any time that day.
* The government contracts with independent organizations to monitor regional sections of the national power grid. The state estimator and network analysis tools at that site for this segment of the power grid were still considered to be in development on August 14 and were not fully capable of automatically recognizing changes in the configuration of the modeled system. The state estimator at the independent monitor went out of service when it failed to deal correctly with the failure of two lines. An operator mistake after the state estimator had been fixed led to a second failure. It did not return to service until after the cascade of failures had started.

The analysis also said the lack of an automatic alarm failure system as one of the causes. An automatic notification existed, likely a heart-beat monitor that notified the secondary server when the primary one had failed. It should have been easy to use the same mechanism to also notify the computer support staff and the power grid operators in the event of a failure of one or both alert servers.

### Other Observations

The power grid blackout is an example of a number of the software reliability challenges listed in Table 1.

* A recovery from a failure mode corresponding to a software fault in the alarm server had not been considered. The IT support staff only determined on the day of the blackout that a full control system reboot would be required rather than just a restart of the alarm server.
* The guidance in Table 1 recommends that the design for system recovery should assume that failures could arise from currently unknown or rarely occurring faults. Software perfection was implicitly assumed by the utility from two perspectives. There was likely over confidence in the alarm server software and implicitly in the commercial organization that developed it. The IT support staff said they had encountered only minor problems with the alarm server. But the alarm software supported multi-tasking which should automatically raise reliability concerns. Careful engineering is required to avoid race conditions when accessing and modifying shared data. If the engineering choice is to use semaphores, experience shows that race conditions are likely. Race conditions had not been observed with this software. A fault such as a race condition is easy to ignore as it will occur only for a very specific set of operating condition which may never occur. The only safe assumption is that unanticipated failures will occur.
* The utility had not explicitly considered how to continue operations if the alarm server recovery failed. The assurance case shown in Figure 2 demonstrated that a design that was very resilient, i.e., could tolerate the failure of any six of the claims.

## Sustainment

The risk of a system failure can increase over time because of changes in operational conditions and work processes. An assurance case that documents the assumptions, argument, and evidence that justify a claim can be used to monitor how changes affect the confidence associated with a claim.

As an example, consider the assurance case for the utility shown in Figure 2. The utility did not provide a visual map, there was no requirement for the computer support staff to notify the operators of failures in the alarm service, and the alarm service recovery was designed only for a hardware failure. The three remaining alternatives, the alarm service, operators monitoring the contingency analysis, and the independent monitoring capability, were assumed by the utility to provide sufficient resiliency. But at some point after the installation of the control system, problems occurred with the automatic execution of the contingency analysis tool. The loss of that automatic analysis would leave the utility dependent on a single internal resource, the alarm service. But the resultant significant reduction in resiliency did not appear to be considered when the decision was made that the operators should manually run that analysis only when needed. Now only two of the alternatives listed in the assurance case remained. Both of those alternatives failed the day of the blackout. A simple analysis of the assurance case supports a conclusion that a blackout was just ready to happen. That conclusion was strengthened as the NERC analysis team found deficiencies other than those that caused the 2003 blackout that—under different circumstances—could also have led to a blackout.

# Module: Confidence

The participants in the recommended formal design reviews have to decide if they are confident that a proposed design will satisfy reliability requirements. An objective for incorporating software assurance into that review is for that judgment to be based on more than opinion. An assurance case provides a way to systematically do the analysis.

How can we determine the confidence that a system will behave as expected? As noted in the discussion of the power grid blackout, a combination of conditions is frequently the cause of a software system failure. It is impossible to examine every possible combination of conditions that could affect a system.

But achieving that confidence is important to those acquiring the system, those developing the system, and those using the system. Such confidence should be based on concrete evidence and not just on an opinion of the developers or reviewers. An assurance case provides the argument and evidence. Our level of confidence depends on understanding which evidence leads to an increase in the confidence that a system property holds and why specific evidence increases the confidence.

## Eliminative Induction

Reliability depends on identifying and mitigating potential faults. A design review should verify that faults associated with important business risks have been identified and mitigated by specific design features. Hardware reliability such as for a disk drive can draw on documented design rules based on actual usage. Software reliability has not matured to the same state. Software intensive systems are complex, and it should not be surprising that the analysis done by even an expert designer could be incomplete and has overlooked a risk or made simplifying but invalid development and operating assumptions.

A secure system must also be reliable. For reliability, the activities in the design phase of a custom developed system should identify possible failure modes and how they might affect operations. For security, an attacker compromises a system by exploiting the response to unanticipated conditions. The security failure modes are the exploitable conditions such as invalid input that can be created by an attacker.

## Software Assurance Justification

For example, developers might be asked during a review why they are sure that XSS vulnerabilities have been sufficiently mitigated, i.e. from a software assurance perspective what confidence is associated with the design.

A justification of an engineering decision has to be based on more than opinion. A review should be based on a documented body of *evidence* that provides a convincing and valid *argument* that a specified set of critical *claims* about a system’s properties are adequately justified for a given application in a given environment. The argument is that input routines use a vetted library to verify input data and the evidence that supports that argument includes library documentation that the vetting requirements are met and results from the CIAQ scans that show that the library has been used where required.

Such a justification is called an assurance case. An assurance case does not imply any kind of guarantee or certification. It is simply a way to document the rationale behind system design decisions.

**Assurance case**: a documented body of *evidence* that provides a convincing and valid *argument* that a specified set of critical *claims* about a system’s properties are adequately justified for a given application in a given environment.

A graphical representation of XSS justification using the Goal Structured Notation (GSN) is shown Figure 3. GSN is used extensively for safety assurance cases [Kelly 2004]. The graphical notation which is shown in Figure 4 provides a concise and understandable way to document the justification for a system property claim.

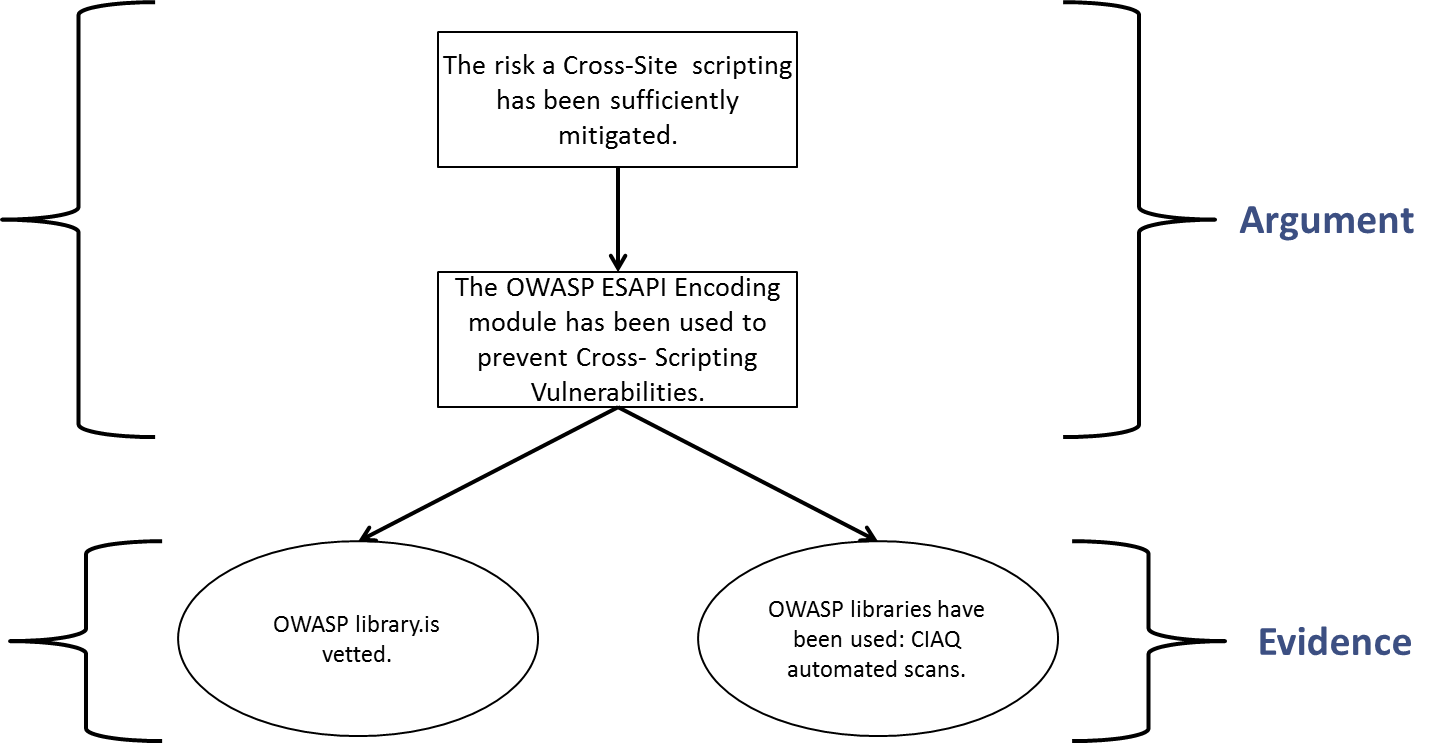


Figure : Justify XSS Mitigation

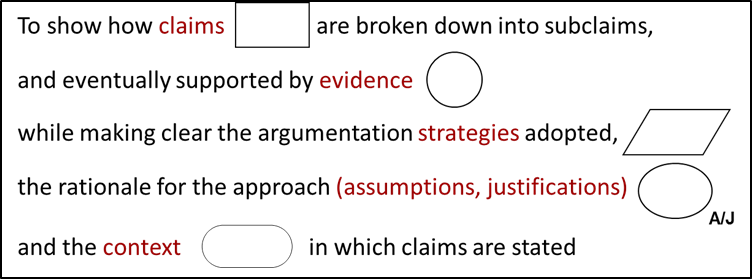


Figure : Goal Structured Notation

## Reviews and Development as an Assurance Case

An implicit assurance case is typically created during a system review. Instead of trying to estimate the likelihood that system claims made by a developer are true, a reviewer is more likely to look for reasons the claim may be false. For example, a reviewer might doubt the evidence provided for any of the following reasons.

* The test plans include all of the hazards identified during design.
* The acquirer did provide sufficient data to validate the modeling and simulations.
* The integration testing did adequately test recovery following component failures.

The existence of any of the practices in the Poor Result column of **Error! Reference source not found.** such as testing being the primary form of defect removal should raise doubts about a claim.

It is not at all obvious, but such an approach is constructing an assurance case for a claim. Instead of constructing an argument for the validity of a claim such shown in Figure 3 , we identify the various possibilities for why the claim could be false. An assurance case now consists of gathering evidence or performing analysis that removes those possibilities. The SEI refers to the graphical representation of the assurance case created by eliminating doubts as a confidence map. Each eliminated possibility removes a reason for doubt and thereby increases our confidence in the claim. The expectation during a review is that the developer is able to show how to eliminate the doubts that are raised.

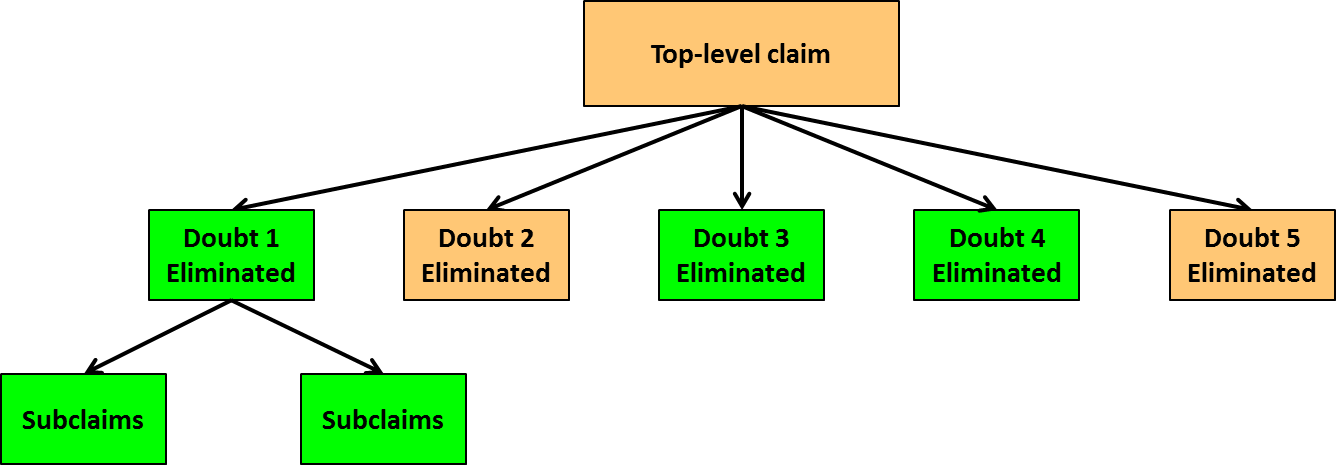


Figure : Confidence Map

Similar reasoning supports a security risk assessment such as threat modeling. As noted earlier, defects such as XSS vulnerabilities or unsecure dynamic SQL queries typically involve multiple layers of a system. Threat modeling analyzes the data flows across those layers and well as any external dependencies to identify how a design could be compromised, an explicit search for potential defects. One objective of threat modeling is to find exceptions to a claim of security.

## Eliminative Induction

A claim is tentative. We cannot deductively prove using the argument in an assurance case that the evidence **E** proves the claim **C** is true. Additional information could show that it is false. Rather our logic looks like

if **E** then (usually) **C** unless **R, S, T, …**

where **R,S,T…** are exceptions. The doubts identified during design review are the potential exceptions to the claim. Removing doubts about a claim is called eliminative induction. These exceptions are called defeaters. Each defeater is a source of doubt about the truth of a claim. The research done for confidence has identified three classes of defeaters that are applicable for a justification of an assurance claim. An assurance justification has the form

*Argument* shows *Evidence* confirms *Claim*

The three classes of defeaters are

1. **Doubt the claim:** There is information that contradicts or rebuts a claim. The cause can be a combination of a poor argument and insufficient evidence. Referred to as rebutters in the literature.
2. **Doubt the argument:** There are specific conditions under which the claim is not necessarily true even though the premises (i.e., evidence) are true. Such conditions create doubts or undercut the validity of the argument. We can doubt the inference among claims or between a claim and its supporting evidence. Referred to as undercutters in the literature.
3. **Doubt the evidence:** There are conditions that invalidate one or more of the premises. The argument is valid, but insufficient evidence weakens or undermines our confidence in the claim. Referred to as underminers in the literature.

Removing doubts just inverts what might be thought of as a constructive assurance case. Consider the assurance case shown in Figure 6 for the claim that flipping a switch will turn the light on[[7]](#footnote-7). An alternative assurance argument if those failures are eliminated then the light will turn on.

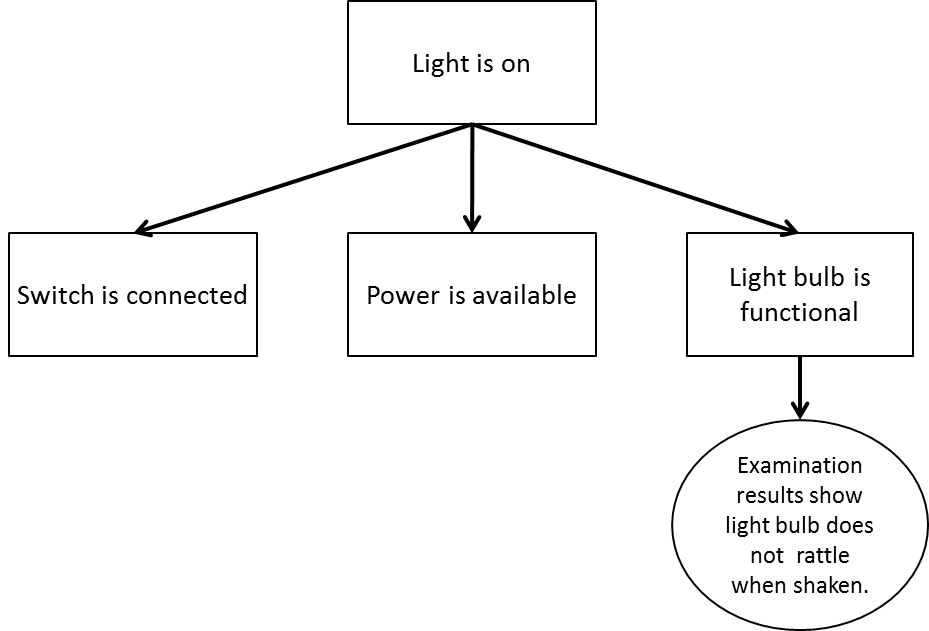


Figure : Light Bulb Example

The claim is invalid if there is no power, if the switch is not connected, or if the bulb is defective. Those conditions rebut the claim as shown Figure 7. The assurance argument now is if those failures are eliminated then the light will turn on. The confidence of the assurance now depends on our confidence in the statement that there are no other reasons for a failure.

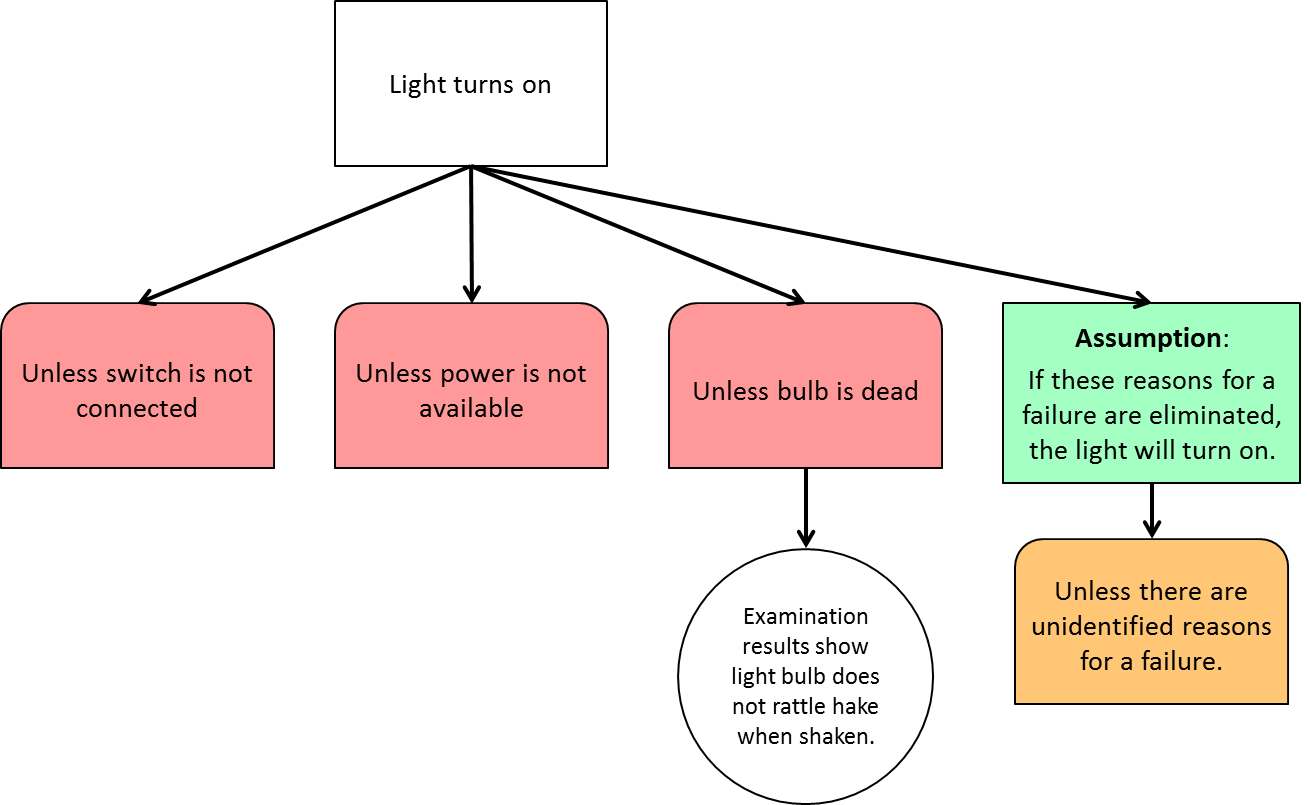


Figure : Light Bulb Rebutters

Doubts can also be raised on the evidence that a light bulb is functional as shown in Figure 8. The evidence given for a functional light bulb is that that it does not rattle when shaken, i.e., the filament is intact. The validity of that evidence can be undermined by an examiner who is hard of hearing or has headphones on. The validity of the evidence argument is undercut by an LED bulb as such does not rattle when defective.

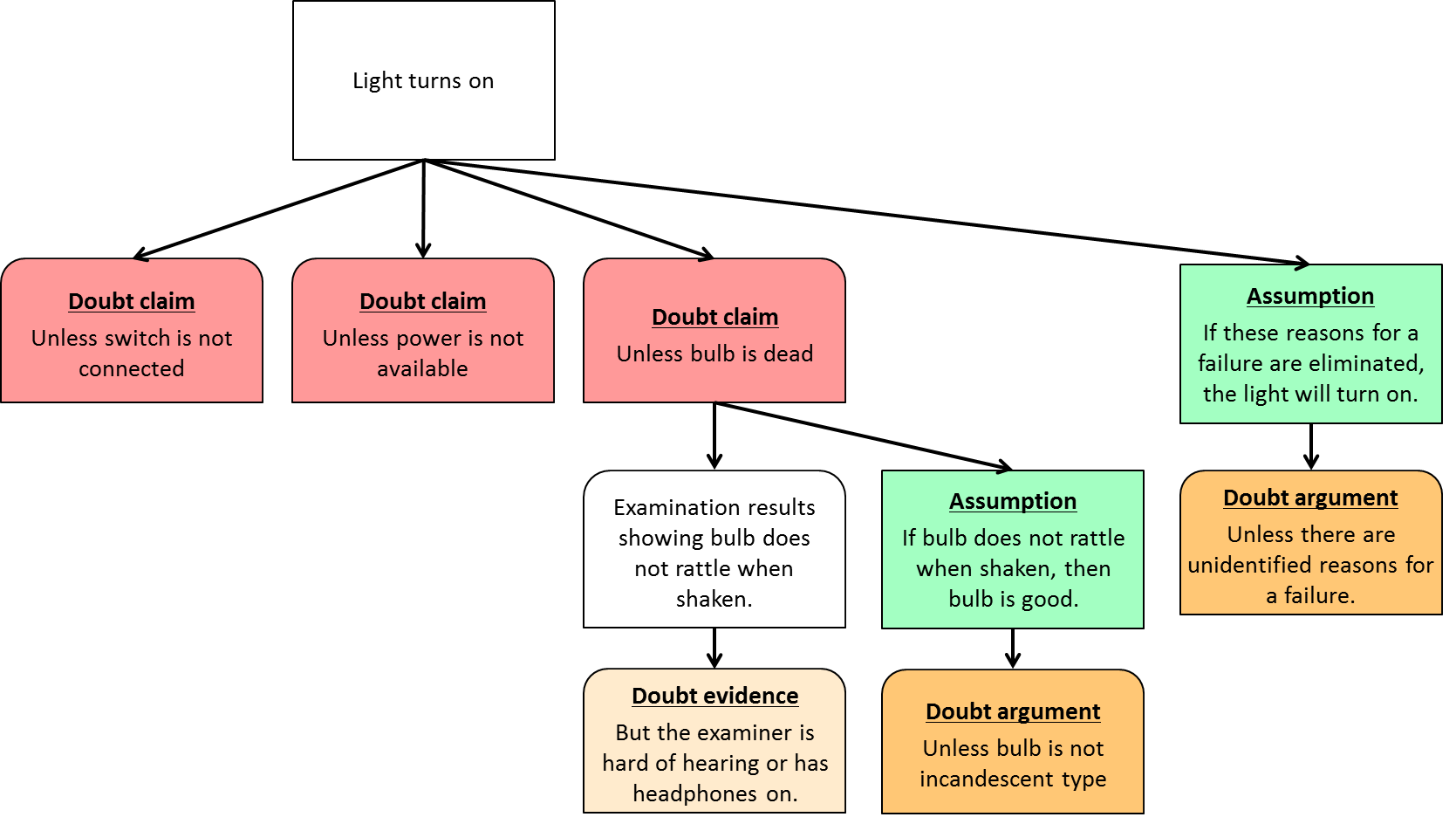


Figure : Expanded Confidence Map

## Infusion Pump Assurance Case

A sketch of an infusion pump assurance case is shown in Figure 9. The strategy box says that safety of the pump will be shown by demonstrating that each of the nine health risks has been mitigated, i.e., divide the initial claim into—it is hoped—simpler subclaims. The overdose subclaim is then divided into subclaims based on the hazards that could result in an overdose. For example, an overdose could be caused by environmental hazards such as a high temperature which led to a pump malfunction. It would also result from a patient tampering with unit or from cell phone interference.

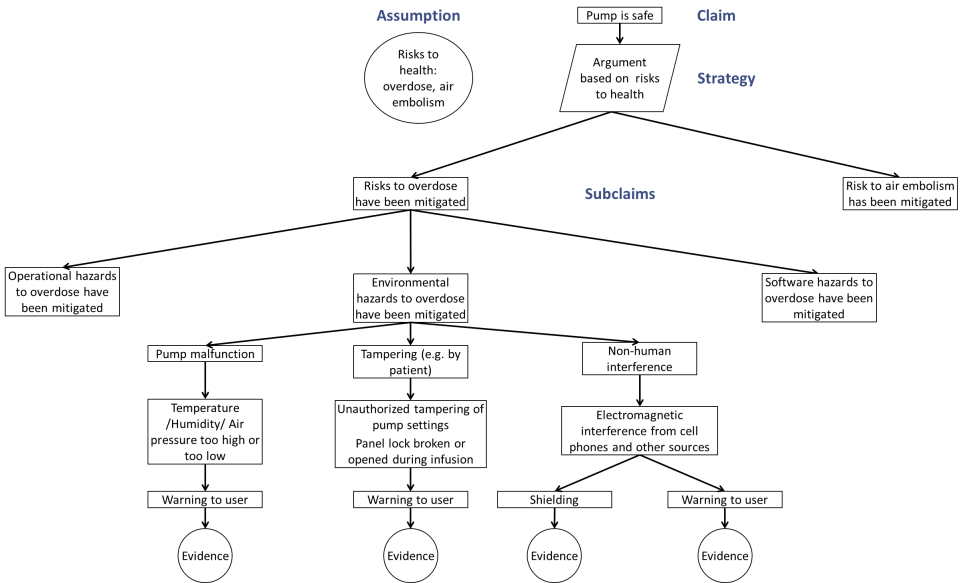


Figure : Using GSN for Infusion Pump Assurance Case

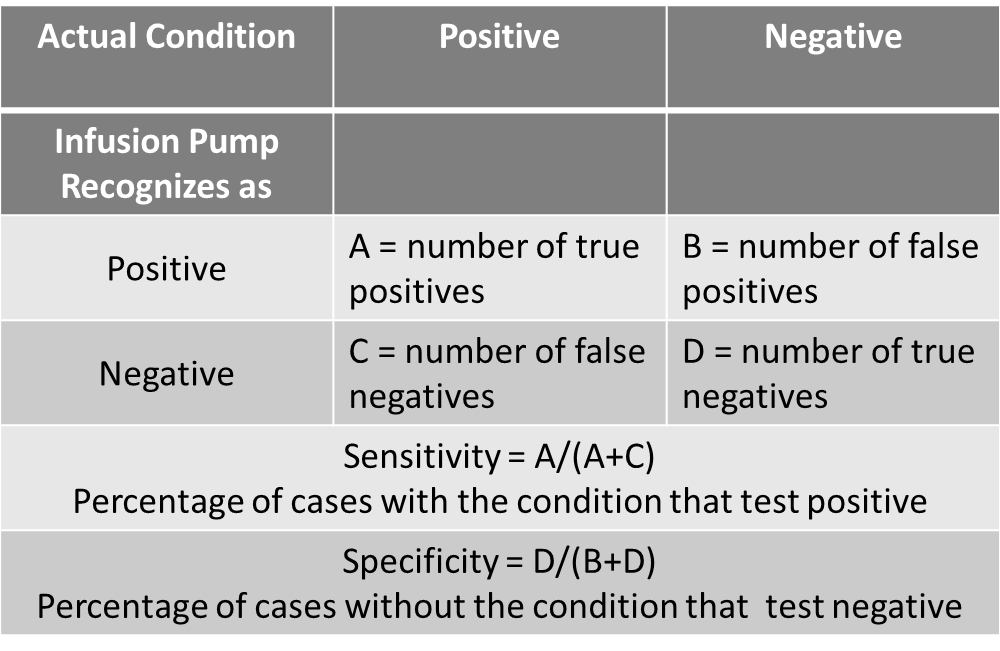
A manufacturer needs to show that the sensors have been engineered to monitor the state of physical configuration and the flow of the medicine and that the software has been designed to recognize hazards based on the aggregate of sensor values and take the action as defined by the safety requirements.

For example, if an overdose has a high health risk, then an alarm should sound. An overdose can be the result of a free flow of the medication because the valves in the delivery paths are broken or the delivery path is damaged, creating a vent in a line. The high health risk requires that an infusion pump be designed to be very sensitive to hazards that could cause an overdose, i.e., the percentage of such hazardous condition that are not recognized has to be low (see Table 2).

There are other cases where the recognition has to be what is called specific, i.e., there is a high probability that the condition recognized by the pump actually exists. A warning light that comes on too often when there are no problems will soon be ignored.

It should not be surprising that testing alone had not provided sufficient evidence to demonstrate acceptably safe infusion pump behavior. The justification had to consider any safety issues raised by how the infusion pump software responded to the interactions of the data from multiple sensors. The sensitivity and specificity of a pump’s hazard tests had to be consistent with the associated health risks, but such analysis requires the data shown in Table 2 for each hazard considered. That analysis depends on applying techniques such as simulations, static analysis, and state machine modeling. The lack of that kind of engineering analysis during a design is consistent with the number of engineering and manufacturing defects identified by the FDA.

Table : Evaluating Pump Hazard Recognition



Activities in the design phase of a custom developed system should identify possible failure modes and how they might affect operations. The causes of such failure modes can be a combination of those from the operational environment and those associated with software defects.

The design for a software-intensive system is usually an incremental process. A proposed design that mitigates a particular failure mode can introduce new ones. The expense of a high-assurance mitigation may lead to requirement changes or require additional time to evaluate less-expensive alternatives.

The conclusion of a formal design review should be based on more than opinion. An assurance case provides a way to systematically do the analysis. Software intensive systems are complex, and it should not be surprising that the analysis done by even an expert designer could be incomplete and has overlooked a hazard or made simplifying but invalid development and operating assumptions. The use of eliminative induction as described in Section 5.1 provides a systematic way to look for exceptions for the claim, the evidence, or the arguments used to justify the engineering decisions incorporated in a design.

# Module: Writing an Assurance Case

This material was written for a tutorial on assurance cases and should be covered in a lecture or by a reading assignment before students are asked to write an assurance case.

# Module: Building Security In Maturity Model (BSIMM)

The BSIMM website provides extensive documentation. The primary point is that security depends on more than specific development practices. It is also important to note, that BSIMM surveyed very large organizations. We do not know what is realistic and effective for small to medium size developers.

# Module: Open Group Assurance Case

Managing the risks associated with supply chains that provide software and computing components is an important role for assurance management. One aspect of supply chain risk management is a vendor risk assessment. There are frequent requests for something like an Underwriter’s Lab certification that could be applied to software. The Open Group Trusted-Technology Forum considered such a vendor certification for supply chain risk management. The Open Group certification process is an assurance case with claims and the evidence specified in a way that the assessment can be done by an independent laboratory.

A good topic for an assurance management discussion is described in the slide “Will anyone use it?” Creating a standard is not an academic exercise. Participating commercial organizations want a technical standard that favors their technology. Equivalent circumstances can arise in standards where the participants are governments – the Internet is a good example. The Open Group has to “market” their standard to governments and to commercial organizations for it to be widely used.

The Common Criteria is being revised. The drafts of the new version are available on the Internet and could be discussed in a lecture. <http://www.commoncriteriaportal.org/>

# Module: Systems and System of Systems

This section concludes with a discussion of SEI experience applying assurance case techniques to the early phases of the system development lifecycle of a DoD system [Blanchette 2009]. That experience suggests that the assurance case technique is a powerful tool for analyzing systems. Assurance cases give managers answers about design progress that are demonstrably rooted in facts and data instead of opinions based on hope and best intentions. Techniques such as the confidence map described in Section 5 provide a concise and understandable way to show the effects of a specific development shortfall and to track progress between reviews.

The SEI has applied assurance case techniques in the early phases of system development life cycle for a large DoD system of systems (SoS) as described in Blanchette [Blanchette 2009]. The general approach is applicable for less complex systems. The SEI team analyzed the software contributions to the definitive characterization of operational needs – the SoS key performance parameters (KPPs). Within the DoD, KPPs are the system characteristics essential for delivery of an effective military capability. All DoD projects have some number of KPPs to satisfy in order to be considered acceptable from an operational perspective. For example, any DoD system that must send or receive information externally is required to fulfill the Net-Ready KPP (NR-KPP). The top claim is that the SoS supports Net-Centric military operations. The subclaims of that node are

* The SoS is able to enter and be managed in the network.
* The SoS is able to exchange data in a secure manner to enhance mission effectiveness.
* The SoS continuously provides survivable, interoperable, secure, and operationally effective information exchanges.

When performing an assurance case analysis of a completed design, the outcome is rather black-and-white: either design artifacts are complete and sufficient, or they are not. Reviewing an in-progress design requires a more nuanced approach, one that reflects relative risk, since the design artifacts will necessarily be in different stages of completion. For this example, the SEI used a simple and familiar stoplight approach to scoring (so named for the red-yellow-green coloring), where the color red designates a relatively high risk area, the color yellow designates a relatively medium risk area, and the color green indicates a relatively low risk area. The rules for assigning colors are slightly different at the evidence level than they are at the level of the claims, as is shown in Figure 10.

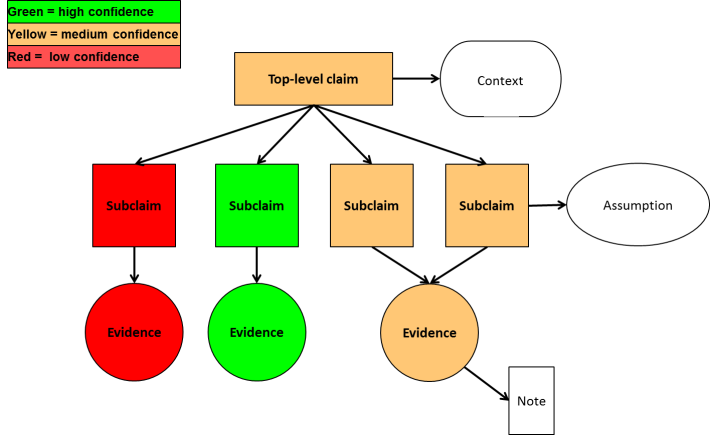


Figure : Scoring Legend

When the subclaims are not all uniformly the same color, an analyst must make a subjective decision on the risk to assign to a node. For example, an analyst might conclude a medium risk given the following doubts raised about the evidence and arguments.

1. Only a subset of information exchanges has been implemented to date.
2. The noted risks are, at best, medium at this time.
3. The security architecture has not been completely propagated across the SoS.
4. An evaluation of the security architecture revealed some design choices that will prevent system accreditation.
5. Preliminary field tests indicate some information exchanges are exceeding prescribed timelines for completion.

The overall analysis tree might appear as shown in Figure 11 in a confidence map. The color assigned represents an analyst’s judgment on the seriousness of the doubts identified for a specific claim. It can provide both program and developer managers a sort of roadmap for prioritizing and addressing the issues.

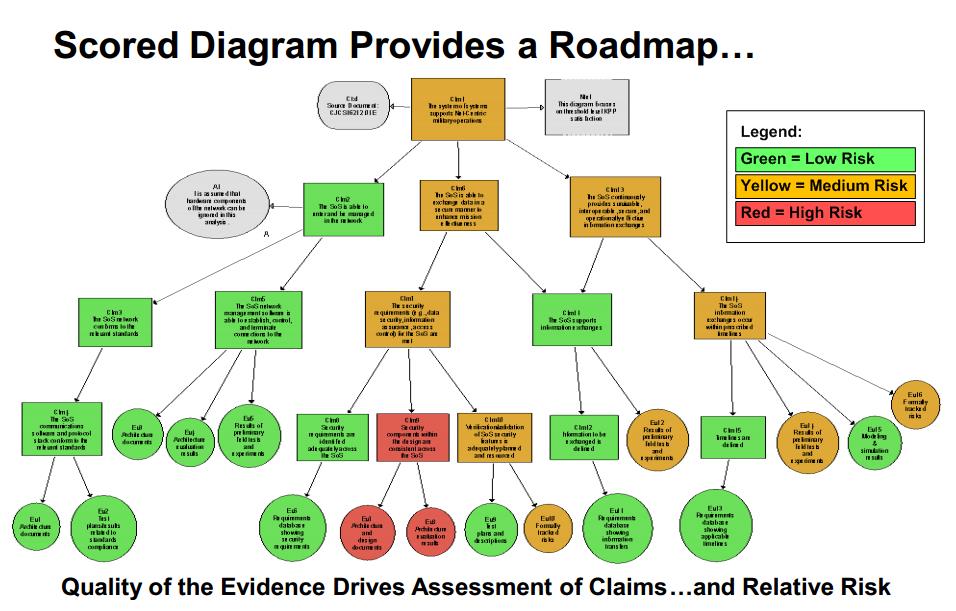


Figure : KPP Scored Diagram

For example, an item may be red because

* It is scheduled to be addressed at a later date.
* The contractor is significantly behind schedule.
* A redesign is required because of changes in requirements.
* The problem is harder than anticipated.
* There is a significant risk that the current approach will not meet requirements: The reason can be a poor design or unrealistic requirements.

Such a confidence map provides a concise and understandable way to show the effects of a specific development shortfall and to track progress between reviews.

Experience with actual projects suggests that the assurance case technique is a powerful tool for analyzing a large and complex SoS software design. It provides a means of taking a crosscutting look at a SoS, a perspective often achieved only with great effort even in less complex development projects. Assurance cases give managers answers about design progress that are demonstrably rooted in facts and data instead of opinions based on hope and best intentions.

Project

Writing a business case is a very good topic area for the assurance management course. Projects could individually done or be group efforts. BSIMM can be a source for a wide-range of business cases targeting project management to corporate management. A business case could be written for obtaining an Open Group certification or for why a software development organization needs to add security capabilities beyond doing static analysis.

Business cases could be discussed after the lectures on risk management. Students could consult sources such as the articles on “Making the Business Case for Software Assurance” on the DHS Build Security In Web site.

<https://buildsecurityin.us-cert.gov/articles/knowledge/business-case-models>

An SEI report is also available.

“Making the Business Case for Software Assurance,” <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8831>

References/Bibliography

*URLs are valid as of the publication date of this document.*

[Blanchette 2009]

Blanchette, S. *Assurance Cases for Design Analysis of Complex System of Systems Software.* American Institute for Aeronautics and Astronautics (AIAA) Infotech@Aerospace Conference. Seattle, Washington, U.S.A., April 2009.

<http://resources.sei.cmu.edu/asset_files/WhitePaper/2009_019_001_29066.pdf>

[BSIMM 2013]

Building Security In Maturity Model, <http://bsimm.org>

[CISQ 2012]

CISQ Specifications for Automated Quality Characteristic Measures, CISQ–TR–2012–01, <http://it-cisq.org/wp-content/uploads/2012/09/CISQ-Specification-for-Automated-Quality-Characteristic-Measures.pdf>

[FDA 2010]

U.S. Food and Drug Administration. *Guidance for Industry and FDA Staff Total Product Life Cycle: Infusion Pump Premarket Notification [510(k)] Submissions (Draft Guidance).* <http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm206153.htm>

[Feiler 2012]

Feiler, Peter; Goodenough, John; Gurfinkel, Arie; Weinstock, Charles; & Wrage, Lutz. *Reliability Improvement and Validation Framework* (CMU/SEI-2012-SR-013). Software Engineering Institute, Carnegie Mellon University, 2012. <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=34069>

[Hogood 2004]

Hoglund, G. & McGraw, G. (2004). Exploiting Software: How to Break Code. Boston: Addison‐Wesley.

[Kelly 2004]

Kelly, T. & Weaver, R. “The Goal Structuring Notation: A Safety Argument Notation.” *Proceedings of International Workshop on Models and Processes for the Evaluation of COTS Components (MPEC 2004).* Edinburgh, Scotland*,* May 2004. IEEE Computer Society, 2004.

[Nichols 2012]

Nichols, William R., “Plan for Success, Model the Cost of Quality”, Software Quality Professional, vol. 14, 2., (March 2012), p4-11.

[OMG 2013]

OMG, *How to Deliver Resilient, Secure, Efficient, and Easily Changed IT Systems in Line with CISQ Recommendations,* <http://www.omg.org/CISQ_compliant_IT_Systemsv.4-3.pdf>

[Open Group 2014]

Open Trusted Technology Provider Standard (O-TTPS)™, Version 1.0, *Mitigating Maliciously Tainted and Counterfeit Products*, <https://www2.opengroup.org/ogsys/catalog/C139>

1. The National Research Council report, Critical Code: Software Producibility for Defense. [↑](#footnote-ref-1)
2. The Habit of Courting Disaster, Charles Perrow, The Nation (October 1986) 346-356. [↑](#footnote-ref-2)
3. The software failure was caused by a race condition. An error in the implementation of the software controls that managed access to the data by multiple processes caused the alarm system to stall while processing an event. With that software unable to complete that alarm event and move to the next one, the alarm processor buffer filled and eventually overflowed. [↑](#footnote-ref-3)
4. Gary McGraw, Software [In]security and Scaling Automated Code Review http://searchsecurity.techtarget.com/opinion/McGraw-Software-insecurity-and-scaling-automated-code-review [↑](#footnote-ref-4)
5. CISQ Specifications for Automated Quality Characteristic Measures, CISQ–TR–2012–01, http://it-cisq.org/wp-content/uploads/2012/09/CISQ-Specification-for-Automated-Quality-Characteristic-Measures.pdf [↑](#footnote-ref-5)
6. SOFTWARE QUALITY IN 2012: A SURVEY OF THE STATE OF THE ART, http://sqgne.org/presentations/2012-13/Jones-Sep-2012.pdf [↑](#footnote-ref-6)
7. http://blog.sei.cmu.edu/post.cfm/eliminative-argumentation-a-means-for-assuring-confidence-in-safety-critical-systems [↑](#footnote-ref-7)