CLASS ANALYSIS FRAMEWORK

John Knight and Jonathan Rowanhill
Dependable Computing LLC

Dependable Computing Technical Report TR-2014-1

12/1/2014

Dependable Computing LLC
2120 North Pantops Drive, Charlottesville, VA 22911-8648
www.dependablecomputing.com

CLASS Analysis Framework

Abstract

CLASS is designed to support the safety assurance of a system over the system's entire lifecycle. Major goals of CLASS are: (a) to provide sufficient structure to the process that faults arising in the target system from lifecycle process defects are reduced to the extent possible, and (b) to include within the lifecycle process techniques and methods that avoid or eliminate classes of defects. In this paper, we present the CLASS *Analysis Framework*, a framework designed to determine the extent to which CLASS achieves these goals. The theory upon which the Analysis Framework is based is a technique known as the Filter Model that was developed in separate research. A preliminary application of the technique to a hypothetical CLASS instantiation is described.

1 Introduction

Whenever a safety-critical system enters a hazardous state, with either: (a) no negative outcome, (b) an incident, or (c) an accident, the system can be said to have "failed". The basic approach that is used in safety engineering is the avoidance of hazardous states, and so a system entering a hazardous state has to be viewed as a failure of that system.

Investigation of failures that are detected often enables the determination of the series of events that led to the failure. From that series of events, it is often possible to determine means whereby the hazardous state could have been avoided. Safety engineering and the associated analyses attempt to minimize the possibility of event sequences that can lead to hazards.

The Comprehensive Lifecycle for Assuring System Safety (CLASS) is an evolving lifecycle concept that is designed to provide both support for and analyses of the safety engineering process. All aspects of CLASS are based on the development and manipulation of a rigorous safety case for the subject system. Since the safety case documents the rationale for belief that a system is adequately safe, analysis of the safety case should compel belief that the event sequences that lead to hazardous states have been avoided or found and eliminated.

CLASS embodies process elements that support the four major lifecycle phases of a safety-critical system: creation, certification, operation, and maintenance. Thus, if a system is deployed whose design includes the possibility of an event sequence that could lead to failure or if such a possibility arises during the lifetime of the system, then CLASS *itself* will have "failed".

A *critical* question at the center of the research on CLASS is how to determine its effectiveness. Effectiveness is only meaningful if there is a goal that is being sought, and for CLASS this goal is assurance that the subject system will avoid hazardous states adequately during operation. In this paper we present a preliminary *Analysis Framework* for CLASS designed to permit comprehensive assessment of its effectiveness.

In Section 0 we summarize the Filter Model, the technology upon which the Analysis Framework is based. Section 3 presents the approach developed to combine the two forms of evidence used in safety cases, direct and indirect evidence, and thereby eliminate indirect evidence. Sections 4 and 5 present a preliminary hazard identification and a preliminary hazard analysis of CLASS respectively, and Section 6 introduces preliminary ideas about a performance metric for CLASS.

2 The Analysis Framework

2.1 Analysis Framework Architecture

The Analysis Framework is shown in Figure 1. The Framework is based on the *Filter Model* [1]. The model has been extended to become the core of the Analysis Framework. This enables the use of the Filter Model on CLASS

as an entity. This analysis of CLASS as a whole is similar to but separate from the CLASS mechanism for certification.

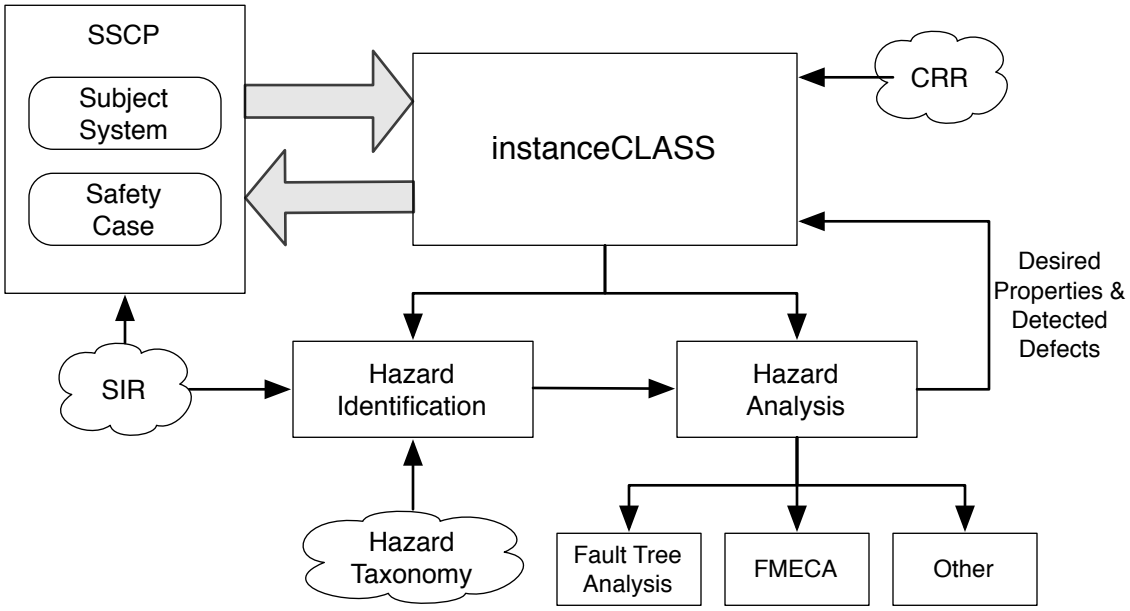


Figure 1: CLASS Analysis Framework

The Analysis Framework enables both design and analysis of a process so as to ensure that the product of that process has desired properties. The initial application of the Filter Model, and the original motivation for its development, was to analyze approval/certification processes. The goal was to model approval so as to determine the properties that a system needs to have in order to be approved, i.e., the effectiveness of approval. By approval, all forms of examination and certification were included, although the primary area of interest was certification of aviation systems. Of particular interest was the goal of determining whether a given approval process might allow a system with specific, perhaps significant deficiencies to be approved unexpectedly.

The introduction of the Filter Model to the analysis of CLASS enables precise statements to be made about properties of the subject system that result from various process elements and their combination. In essence, the Analysis Framework enables both design and analysis of a process so as to ensure that the product of that process has desired properties. Thus, assurance analysis of CLASS using the Analysis Framework provides rigorous but informal guarantees about CLASS' safety performance. Although not formal, the analysis allows rigorous analysis about the possible deviation from safe performance of a target system.

An important consequence of the CLASS Analysis Framework and the Framework's capability to allow rigorous guarantees about CLASS' performance is that the two major forms of evidence used in assurance cases, *direct* and *indirect* evidence have been combined so that the notion of indirect evidence is no longer required. The reason that indirect evidence can be eliminated is that such evidence relies upon an assumption about the effect of a process on system quality. That assumption is no longer needed.

2.2 Filter Model Concept

The *Filter Model* [1] was developed by Steele and Knight. The model is summarized as follows:

> *The Filter Model characterizes the certification process as a safety-critical system in which incorrectly certifying a system that should be rejected is an* accident*. The role of certification is modeled as the identification of faults in the system being certified, i.e., a* Filter *for system faults.*

Application of safety-engineering techniques to the certification process allows assessment of certification and offers the possibility of correction of weaknesses.

Although intended originally for modeling and assessment of approval (including any modern concept of certification) processes, the Filter Model has been extended to allow it to be applied to other process issues, including the assessment of CLASS itself. For CLASS assessment, therefore, the approach we have begun to develop is a Filter Model in which CLASS is a safety-critical system that is vulnerable to accidents, i.e., loss of synchrony is treated as a "synchronization accident" in this model.

2.3 Analysis Techniques

Since the Analysis Framework treats processes within entities such as instanceCLASS as safety-critical systems in operation, all of the analysis techniques developed for analysis of safety critical systems can be applied to the subject process. Techniques that apply include:

- Fault Tree Analysis (FTA)
- Failure Modes Effects and Criticality Analysis (FMECA)
- Hazard and Operability Analysis (HazOp)
- Functional Hazard Analysis (FHA)

2.4 Analysis Terminology

Since the basic model of CLASS that is being used is that of a safety-critical system, care must be exercised in using terms such as "hazard" and "fault". When referring to a term that is applied to CLASS, the subscript *CL* will be used to indicate that the term is being applied to CLASS itself. Thus, for example, a hazard in CLASS will be referred to as a $hazard_{CL}$. Correspondingly, a hazard in the subject system or its associated safety case will be referred to as a $hazard_{SS}$.

2.5 Interpretation of The Analysis

The modeling of a process as a safety critical system and subsequently analyzing this system using safety-engineering technology enables a useful interpretation of the results. By definition of the model, an approval process is a filter and, to the extent that the approval process is free of defects, any system that passes through can claim to be absent whatever defects are the subject of the filter.

This property is exploited in the CLASS Analysis Framework to allow claims to be made about the subject system being free of defects in various categories. This property is also the basis for the elimination of the concept of indirect evidence in assurance cases. Finally, this property is the basis for assurance of CLASS itself. Using the Analysis Framework, we are able to establish that CLASS will either avoid or eliminate faults in certain explicit categories from the subject system.

3 Combining Direct and Indirect Evidence

An important advance that CLASS brings is the combination of the two types of evidence used in assurance arguments that have been treated separately in the past. These types of evidence are:

- **Direct evidence**. Direct evidence is evidence obtained about the subject system by some form of analysis.
- **Indirect evidence**. Indirect evidence is evidence obtained about the development activities undertaken for the subject system.

The difficulty with *indirect* evidence is that conclusions about specific properties of the subject system are difficult to draw. For example, the use of a particular process in the development of software (e.g., agile, spiral, waterfall, etc.) does not imply a specific property about software developed with the process. Indirect evidence is used along

with human judgment to draw conclusions (and hence to associate belief with a claim) in assurance arguments. Confidence is conclusions that are based on indirect evidence is usually weak.

An instanceCLASS for a specific system is the result of CLASS instantiation can include any assets, mechanisms, process elements, etc. from the CRR. In addition, development technology can be incorporated from any source and with any form that will provide value to the system and the associated safety case. CLASS is not tied to any standard although technology from any standard can be included.

In the design of an instanceCLASS, the developers have the opportunity to create the instantiation with technologies that ensure that a specific type of fault is either avoided or eliminated in a subject system. The CLASS Analysis Framework begins by defining the $hazard_{CL}$ of concern, i.e., the hazard in instanceCLASS that would lead to $hazard_{SS}$ in the subject system or its safety case. With that $hazard_{SS}$ identified, safety-engineering techniques, such as fault-tree analysis, can be applied and changes made to instanceCLASS to reduce the risk associated with the hazard to an acceptable level. This approach might lead to significant changes to instanceCLASS including replacement of the process technology being used.

As an example, consider a subject system that relies upon software. This reliance will require that a software assurance case be included in the overall system safety case. The top-level goal of the assurance case will be derived from the goals of the safety case. In a traditional development activity, a software assurance standard such as RTCA DO-178B/C might be used to provide indirect evidence of software quality. Belief in the necessary claim about software quality relies upon indirect evidence, i.e., conformance with the standard.

A standard could be used in CLASS if CLASS were used in this example scenario. However, the Assurance Framework would identify a deficiency in software quality as a hazard and analyze the hazard using safety-engineering techniques. This analysis would identify the process steps needed to acquire the desired degree of certainty that the assurance goal was met. Thus, specific properties of the software would be obtained by a mechanism similar to a standard, but the resulting evidence would be direct.

4 Preliminary CLASS Hazard Identification

The first step in applying the Analysis Framework is to undertake a hazard identification of the subject system. For CLASS, hazard identification yields the following $hazards_{CL}$:

- **Loss of Synchrony**: The basic model for CLASS is that the safety case for the subject system is complete, valid, and compelling. Were the safety case to fail to represent the *actual* subject system, i.e., the subject system and the safety case became unsynchronized, then the safety case would not be complete or valid. Thus the first hazard identified for CLASS, i.e., a $hazard_{CL}$, is *loss of synchrony* between the subject system and the safety case.
- **Process Conformance Failure**: CLASS describes a detailed process for building the safety case for a system. Each element of the process is present to prevent a $fault_{SS}$ from either being introduced or allowed to remain in the subject system. Failure to follow the process, i.e., a conformance failure, is crucial, therefore, and so *failure to conform* to the process is a $hazard_{CL}$.
- **Loss of Comprehension**: The semantics of a typical safety case and the associated subject system can be extremely complex. Despite the best of intentions and a process that is free of $faults_{CL}$, $faults_{SS}$ can still enter the subject system because of a *loss of comprehension* by those effecting the process.

In the remainder of this section, we examine each of these hazards and continue the process of applying the Analysis Framework. This continuation begins with development of the fault trees for the various $hazards_{CL}$.

4.1 Loss of Synchrony

Synchrony is defined as:

> *Assurance synchrony is the property that an assurance case is in synchrony with the system about which it argues. For example, a safety case is in synchrony with its system when all of its safety*

claims and evidence are true for the current implementation or furthest so far refinement of the system model. This further requires that all analytic products are coherent. For example, evidence and hazard analysis cannot be allowed to ignore or contradict one another.

Synchrony would be broken if, for example any of the following occurred:

- The safety case was updated but the system was not.
- The safety case was not synchronized with the system when the system was initially approved for use.
- The system was updated but the system was not.
- An assumption included in the safety case was false either initially or became false at some point.

In practice, synchrony *has* to be broken regularly, because of the inherent need to be able to modify the system and the associated safety case independently. During system creation, for example, modifications to the system will not necessarily be matched exactly by changes in the safety case, and there would be no point in trying to achieve this. What matters is that the system and the safety case be synchronized when they need to be synchronized, i.e., during system development when *analysis* is conducted, during *operation*, and after *maintenance* or *enhancement*.

The situation with system/safety case synchrony in CLASS is exactly the same as the situation that arises in modern software development. Developing a large modern software system usually employs a process whereby the assets associated with the system are maintained in a consistent state as part of a "trunk" development sequence with "branches" used to permit sequences in which the elements of the system become inconsistent. When the activities of a branch are completed, the changes are made consistent and merged with the trunk. The same basic process is required for the safety case and subject system in CLASS.

The lifecycle phases are sufficiently different that analysis of the loss-of-synchrony $hazard_{CL}$ as a single hazard is not appropriate. Rather, we break loss of synchrony into separate hazards for each lifecycle phase:

- Safety case development.
- Safety case certification.
- System operation.
- System maintenance.

4.2 CLASS Conformance

The CLASS process is quite detailed, and every element of the process is designed to ensure that $faults_{SS}$ are avoided or eliminated. The notion of conformance requires that either:

- tools or some other technique ensure that those effecting the process do so entirely as intended, or
- a check of some sort is conducted at every step of every phase of the process to ensure that the prescribed process is followed.

The first option, ensured conformance, is unrealistic, and so we concentrate on the second option, a monitored process. This notion of a means whereby conformance is checked is a generalization of the previous notion of monitoring of safety-case assumptions. The result is a general monitoring mechanism that is part of CLASS and that provides monitoring of everything about CLASS.

4.3 CLASS Comprehension

The complex semantics in safety cases and their relationship to artifacts such as system functional and safety requirements is a common cause of system failure, including safety failures. The primary source of difficulty is the real-world meanings that have to be communicated in creating system development artifacts.

Applying the Analysis Framework using experience and documented problems to identify the aspects of development artifacts that might be misunderstood and where with the overall lifecycle misunderstandings might

occur can be accomplished by functional hazard analysis. With that analysis completed, fault trees can be developed to identify specific communication paths that might fail with serious consequences.

With faults identified, reduction of the risk associated with the hazard can be undertaken by determining which techniques can be used to reduce risk. Applicable techniques include:

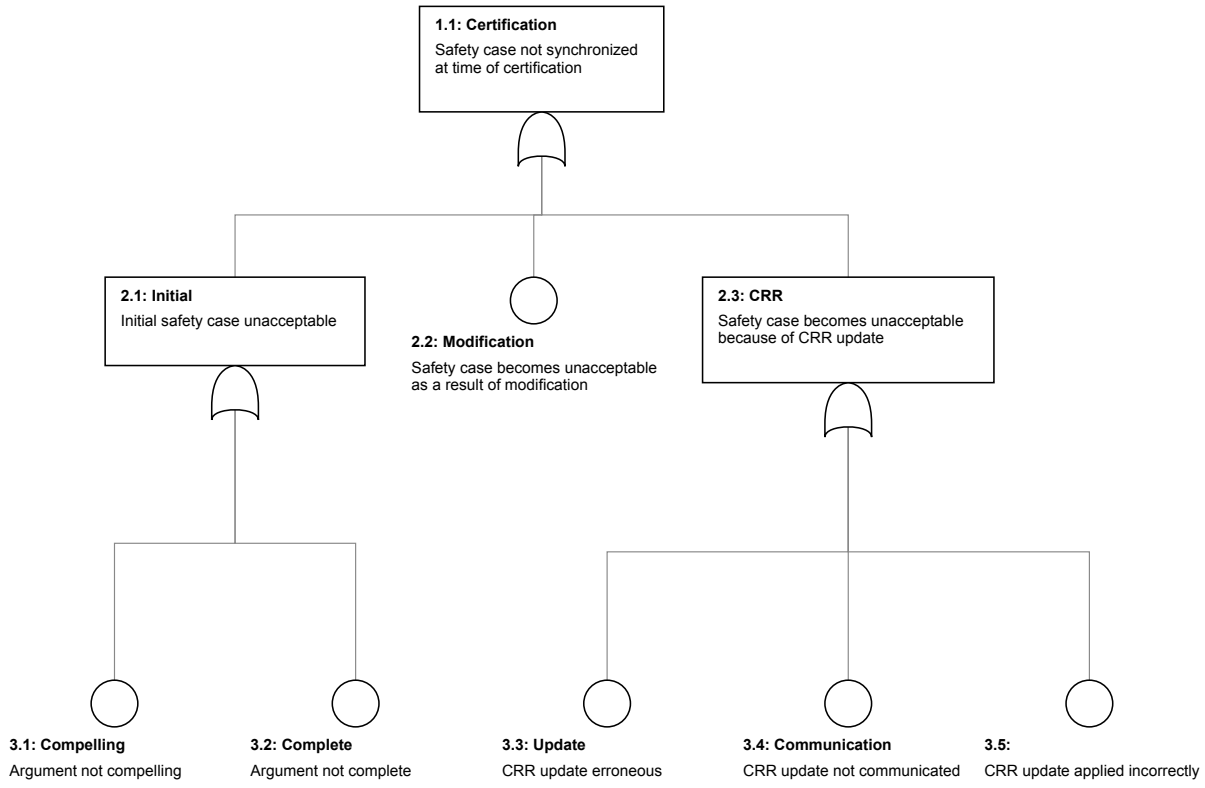
- Ontologies that provide domain-specific definitions that can be adopted.
- Rigorous inspections in which specific communications failures are identified using precisely defined check lists.

5 Preliminary CLASS Hazard Analysis – Fault Tree Analysis

Step one in hazard analysis is to apply fault-tree analysis. Irrespective of the details of CLASS implementation, some of the faults that can lead to hazards in CLASS include:

- InstanceCLASS does not receive details of a known flaw in materials from the CLASS Resource Repository (CRR) and subsequently makes the same mistake as the source of the flaw as a result.
- A CRR item is defective when installed.
- A CRR item become defective when updated.

An initial sample of a fault tree for a CLASS hazard is shown below:



With fully developed fault trees, the effects of the identified events and determination of means to deal with these circumstances by modifying CLASS can be determined.

6 CLASS Performance Metric

Analyses such as those described above can yield substantial evidence about the efficacy of an instanceCLASS. All of this evidence is direct, because the concept of and need for indirect evidence has been eliminated. With the Analysis Framework in place, the possibility of a CLASS performance metric can be considered.

Existing safety assessment techniques, including quantitative models, could be applied to the analysis of CLASS. Very little confidence remains in any general form of probabilistic metric. Recent work by Rodes, Knight and Wasson [2] suggest a different direction. Although that work developed a security metric, the basic mechanism can be applied immediately to the safety case in CLASS.

The underlying principle of the work by Rodes et al is to measure a dependability property such as security or safety by:

- Stating a high-level goal for dependability property in the exact form employed in safety cases as they are used in CLASS.
- Define confidence in that claim as the basic metric used for system evaluation.

The important distinction that this approach provides is as follows. The high-level goal in a security or safety case defines a dependability property that is desired of the subject system. There is no need to measure this property. The property as stated was defined by the system requirements. The metric of importance is the extent to which one can believe that the stated goal is met.

With no additional information, belief in whether a system has a specific property is a vague notion. However, with a safety case (as in CLASS), the rationale for belief in the top-level goal is explicit. By definition, that is the intent of the safety argument. Thus, the only issue is whether the safety argument can be believed, and this need for belief is the motivation for defining confidence in the safety argument as a suitable metric.

By using CLASS, we expect that belief in a system's safety should be high. Thus, performance of CLASS is essentially identical to the metric concept introduced by Rodes et al.

References

1. Steele, P. and J. Knight, "Analysis of Critical System Certification", HASE 2014: 15th IEEE International Symposium on High Assurance Systems Engineering, Miami FL (2014)
2. Rodes, B., J. Knight and K. Wasson, "A Security Metric Based on Security Arguments", WETSoM 2014: 5th International Workshop on Emerging Trends in Software Metrics, International Conference on Software Engineering, Hyderabad, India (2014)

