

# ***Pandora: An Approach to Analyzing Safety-Related Digital System Failures***

William S. Greenwell  
Department of Computer Science

A Dissertation Proposal Presented in Partial Fulfillment of the  
Requirements for the Doctor of Philosophy Degree

April 8, 2005

## **Abstract**

Accidents have occurred in which failures of digital systems have contributed to property damage, injury, and even loss of life. Most investigative agencies do not know how to investigate these failures comprehensively because the complexity and coupling of a digital system's functions can obfuscate its design, making it difficult for an investigator to understand the system, and because the uniqueness of each digital system prevents investigators from developing generic checklists for diagnosing failures. To address these problems, this thesis proposes the development of *Pandora*, a systematic approach to the analysis of safety-related digital system failures that is based upon the system safety case. As the complete argument that a system is safe to operate, the safety case contains the information needed to understand the safety-related aspects of a system and to diagnose and address the failure. Pandora audits a system's safety case to identify the fallacious reasoning or faulty evidence that allowed a failure to occur, and in doing so it drives the elicitation of background information and counter-evidence needed to diagnose the failure. The products of this audit are a set of lessons comprising the flaws identified in the original safety argument, a revised safety case that is free of those flaws, and a set of recommendations for repairing the system and its associated development and operational practices. Pandora is a rigorous approach because it systematically revisits each of the safety claims that, if unsatisfied, could have contributed to a failure, and it is efficient because it considers only those system details, evidence, and safety claims that pertain to the investigation. This proposal discusses Pandora and its supporting work, further development, and planned evaluation of the approach through case studies.

## 1. Introduction

Missing sections of a bridge deck are an obvious indicator that the bridge has collapsed. The severed fuselage of an airplane resting in a cornfield makes it readily apparent that the plane's last flight did not proceed as specified. We possess enough familiarity with civil and aerospace systems to know that, irrespective of the particular type of bridge or aircraft, a collapsed deck or damaged hull is always indicative of failure. The range of functions a civil engineering or aerospace system is expected to perform is limited enough that it is possible to develop generic requirements, designs, and procedures that are tailored to each system that is built, and from those materials a general model of system behavior may be inferred. Although precise requirements and design decisions may differ, these systems are similar enough that the majority of their failure modes are shared, allowing an investigator to diagnose a failure by running through a checklist of symptoms associated with possible causes.

When we turn our attention to software, and digital systems in general, our ability to diagnose failures or even to recognize that they have occurred rapidly diminishes. Unlike civil engineering and aerospace systems, software is not bound to any particular domain; instead it inherits the domain of the system comprising it. Although this property makes software amenable to a wide range of functions, it also causes the requirements, design, and operational semantics of each system produced to vary according to its operational context. The context determines what constitutes correct behavior for the system as well as the symptoms and consequences of failure, which in safety-critical applications may be catastrophic. Even extreme events such as runtime errors, which are considered to be failure modes in almost any context, might be undetectable until their effects are manifest in the operating environment unless the system has been specially instrumented. Because of the coupling between a software system and its context, there is no general model of system behavior from which a common set of failure modes or checklist of symptoms may be inferred. To detect that a software failure has occurred and to diagnose it, then, requires an understanding of the relationship between the system and its context. Just as each software or digital system is developed to a unique set of requirements, each failure analysis requires investigators to understand the unique features of a system's requirements, design, and semantics in order to diagnose and repair the system.

Failures of safety-critical software and other digital systems may be divided into three categories: (a) random failures that are within the scope of the system's safety requirements; (b) attempts to operate the system outside of its intended environment; and (c) failures resulting from defects that compromise the system's ability to meet its safety requirements [12]. Failures of the first form may be dismissed because they occur as part of the already accepted risk of operating the system. Failures of the second form may be remedied by operating systems within the constraints for which they were designed. It is failures of the third form that are the most severe because they are symptomatic of defects that undermine our confidence in the systems involved and could contribute to additional failures in the future. Typically one cannot discern to which of the three categories a failure belongs immediately, and so all failures should be investigated as if they resulted from systemic defects until proven otherwise.

Unfortunately, the complexity of modern digital systems makes them very difficult to analyze when they fail. Coupled with the informality of failure analysis, this complexity reduces the level of confidence that the lessons and recommendations obtained from an analysis are complete and correct [12]. Differences in design and development practices can further limit the scope of those lessons and recommendations significantly. Consequently, opportunities to correct faults in system designs, development practices, and operational and maintenance procedures might be missed. Comprehensive analysis of system failure requires that investigators understand not only the event sequence that triggered the failure but also the safety-related features of the system involved and the rationale behind the decision to accept the system as safe. Moreover, once they have determined the factors that contributed to a failure, investigators must be able to communicate their findings and recommendations in terms that are relevant to the system in question, practicable, and sufficiently general so that they may be applied to related systems.

The *system safety case*—a comprehensive argument for the safety of a particular system—offers a solution to both of these problems. As the container for a system's safety requirements, evidence for safety, and argument linking the evidence to the requirements, the safety case is the definitive reference for understanding the safety-related features of a system's design and how they achieve the stated safety objectives. And when the safety case is expressed in a structured format, flaws in the safety argument and recommendations for fixing them may be expressed in terms of the safety case itself, making it clear which aspects of the safety argument are affected by an investigation's findings and allowing investigators to extend their findings to other systems that use similar arguments. Thus, the safety case is not only a valuable reference for diagnosing failures of safety-critical systems but it is also a means by which the development practices that produce those systems may be iteratively refined, yielding systems that are more resilient to failure and greater confidence that the processes used to produce those systems will assure safety.

This work proposes *Pandora* as a systematic approach to the analysis of safety-related digital system failures that is grounded in the system safety case. Pandora is a set of processes for evaluating a system's safety argument in light of a failure, eliciting evidence from the failure that refutes the argument, documenting faults that are discovered, and recovering the argument so that it is free of the faults that contributed to the failure. As suggested above, by centering its analysis on the safety case Pandora enhances investigators' understanding of the contextual details of the systems they analyze and enables them to communicate their lessons and recommendations in the same context that the developer of the system used to argue its safety. Pandora also offers tools that may be applied to a safety argument prior to its acceptance to enhance confidence in its soundness, and proposed extensions to Pandora would allow it to be applied to systems for which no explicit safety case exists and to proactively apply lessons and recommendations to related systems that are susceptible.

## 2. Digital System Safety & Failure

Safety is a system property that we intuitively relate to a system's design, accident rates and risk, but we have trouble describing that relationship. As a result, various definitions of safety exist. As part of his dependability taxonomy, Laprie defines safety as "the absence of catastrophic consequences on the user(s) and the environment;" however this definition excludes systems such as commercial aircraft whose failure could have catastrophic consequences but whose probability of failure is very low [3]. Leveson advocates a similar definition: "safety is freedom from accidents or losses," but again showing that a system is safe in this absolute sense would require one to show that accidents are impossible, and so she states that safety is an ideal "that can only be approached asymptotically" [24]. Viewing safety as an absolute, unachievable ideal seems to contradict the intuitive judgements we make every day that systems such as airplanes, automobiles, and nuclear power plants are "safe enough" for us to use them or to not be worried about their affecting us. Thus, Lowrance defines safety in a relative sense as "a judgement of the acceptability of risk, and risk, in turn, as a measure of the probability and severity of harm to human health" [26]. Although Leveson and Lowrance raise the questions of how acceptability is to be determined, who will determine it, and who will be affected by the risk of loss, these questions must be answered even with an absolute definition since the best one can do is to arrive at an acceptable approximation of safety. Treating safety in a relative sense agrees with our intuition and allows us to label systems such as commercial air transport—one of the safest modes of transportation available—as safe despite the possibility of loss. It also provides an achievable safety goal from which one can define more specific system safety requirements.

### 2.1. Safety Assurance

Assuring the safety of digital systems, which include software, electronic, and programmable electronic systems, is a formidable task that cannot be completed through testing alone. Littlewood and Strigini identify three aspects of software systems that apply to other digital systems and present unique problems in their assurance versus that of mechanical and analog electrical systems:

- *Software failures are due to design faults, which are difficult both to avoid and to tolerate;*
- *Software is often used to implement radically new systems, which cannot benefit much from knowledge acquired from previous designs; and*
- *Digital systems in general implement discontinuous input-to-output mappings that are intractable by simple mathematical modeling. [25]*

The third point is especially important because it suggests that subtle changes in the inputs to a system may produce dramatic changes in its outputs. Butler and Finelli have demonstrated that statistical sampling is an infeasible approach to quantifying the reliability of ultra-reliable systems (those whose failure rates must be less than  $10^{-7}$  per hour) [5]. Since a safety-critical system is one whose failure could lead to hazards and possibly losses, most safety-critical systems are required to operate in the ultra-reliable region, and so sample-based testing alone cannot provide the required level of assurance that these systems will operate safely. Given the infeasibility of sample-based testing and the impracticability of exhaustive testing, Weaver argues that a combination of evidence from "testing, analysis, process, proof, and historical usage" should be used to argue that a system is safe [32].

In an effort to compensate for the inadequacy of testing alone, regulators and industry have developed standards that prescribe the development, verification, and validation activities that must be undertaken to produce safety-critical software and digital systems. Examples include DO-178B, which applies to software on board commercial aircraft; DS 00-55, which applies to software developed for the U.K. Ministry of Defense (MoD); and IEC 61508, which is a cross-domain standard for software, electronic, and programmable electronic systems. These standards

prescribe the development process that should be followed for a system based on the system's criticality level, and they further require evidence to be produced showing that the process has been followed and that the system has passed the prescribed verification activities. Software engineering standards in general, and especially prescriptive standards, tend to over-emphasize the importance of following a particular process instead of assuring the safety of the product, and they often impose process requirements whose effectiveness has not been validated [9]. They assume that by developing a system according to a set of best practices the system will be safe; in fact there is little evidence to support this assumption [27]. For these reasons and others, Weaver, McDermid, and Kelly advocate the use of the safety case as an "evidence-based approach" to safety assurance for software and other digital systems in which "explicit evidence of safety, directly linked to the requirements of the system" is provided [32, 27, 16].

## 2.2. The System Safety Case

"A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context" [20]. The system safety case documents the safety claims for a system, the evidence that the requirements have been met, and the safety argument linking the evidence to the requirements. Bishop and Bloomfield decompose the safety case into claims, evidence, argument, and inferences. *Claims* are simply propositions about properties of the system supported by evidence. *Evidence* may either be factual findings from prior research or scientific literature or sub-claims that are supported by lower-level arguments. The safety *argument* is the set of *inferences* between claims and evidence that leads the reader from the evidence forming the basis of the argument to the top-level claim—typically that the system is safe to operate in its intended environment [4]. Thus, a safety argument may be viewed as a directed acyclic graph (DAG) in which the root node is the top-level claim, the internal nodes are sub-claims, and the leaves are the basis of the argument. Safety cases differ from prescriptive approaches to safety assurance in that they do not impose specific process requirements on the developer. Rather, the developer is free to produce a system as he chooses provided he can submit evidence accompanied by a compelling argument that the system meets its safety requirements. System safety is argued through satisfaction of the requirements, which are then broken down further into more specific goals that can be satisfied directly by evidence.

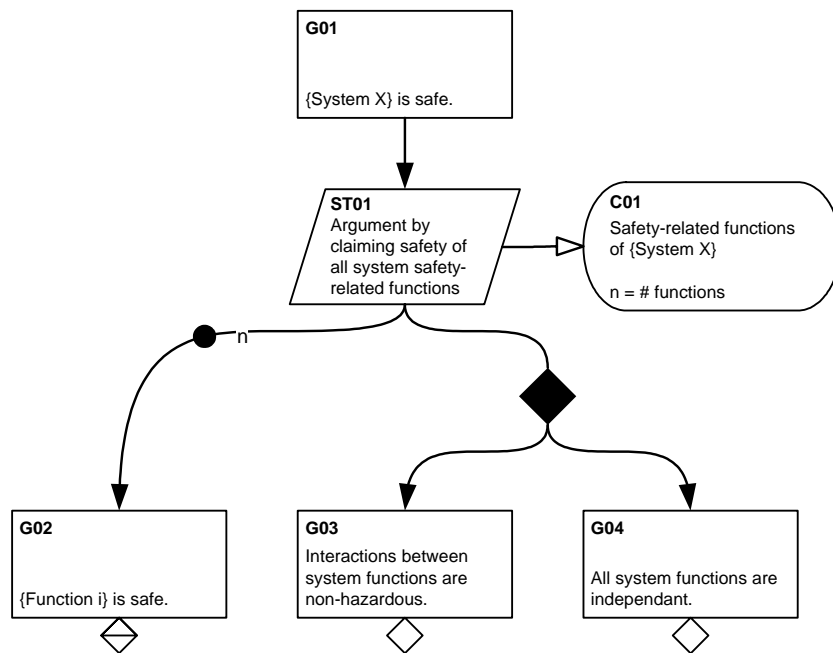
Most developers write their safety cases in natural language; however graphical approaches are becoming more common. Adelard's Claims-Argument-Evidence (ASCAD) and Kelly's Goal Structuring Notation (GSN) are two popular graphical notations for depicting safety arguments [2, 20]. Graphical approaches like ASCAD and GSN formalize the structure of a safety argument—that is, the relationships between claims, evidence, and accompanying contextual information—and present it diagrammatically to make the argument clearer. Formalizing the structure of an argument also reinforces the DAG structure and makes the argument more amenable to analysis. Tool support for constructing and analyzing safety arguments is available for both notations [1, 6].

An additional advantage of representing the structure of safety arguments formally is that successful arguments may be reused easily provided the context is compatible. This observation leads to the notion of *safety-case patterns*, which capture solutions to common problems in safety argumentation much as design patterns do for software development [18]. As an example, Figure 1 presents the functional decomposition pattern in GSN, which could be instantiated as a top-level argument for the safety of a system based upon safety of its component functions. The argument states that the system is safe because each of the  $N$  functions in the system is safe and either that interactions between functions are non-hazardous or the functions are independent. To instantiate the argument, one would have to provide a separate safety argument for each function of the system as well as an argument showing that interactions are non-hazardous or that the functions are independent. Following a pattern can help a developer ensure that all the required evidence for an argument has been presented, which makes patterns an attractive method of storing knowledge gained through experience; however pattern use alone does not make an argument correct.

Similar to the concept of safety-case patterns, *AntiPatterns* "communicate weak and flawed arguments – such that they may be recognized and avoided in future developments" [21]. They are "the antithesis of patterns, but also include an approach for re-factoring the problem, which has negative consequences, into an acceptable solution" [32]. AntiPatterns can assist detection of fallacious inferences in system safety arguments, and analysis of failed systems may reveal new AntiPatterns.

## 2.3. Failure

Leveson defines *failure* in the context of system safety as "the nonperformance or inability of the system or component to perform its intended function for a specific time under specified environmental conditions" [24]. Because failures concern deviations from *intended* behavior, any system whose behavior is inconsistent with its goals



**Figure 1: Functional Decomposition Safety-Case Pattern [19]**

may be considered to have failed, even if the system behaved as specified. Failures of digital systems are caused by faults in a system’s specification, design, or implementation [30]. In the case of control systems, these systemic faults may cause a system to take actions that directly compromise safety; for example, the series of radiation overdoses administered by the Therac-25 treatment apparatus that caused the deaths of six patients [24]. Alternatively, if the system monitors potentially hazardous operations, systemic faults may cause the system to display false information, issue erroneous advice, or not take any action when a hazardous situation arises; such was the case in the failure of the Minimum Safe Altitude Warning (MSAW) system that contributed to the August 1997 Korean Air flight 801 crash into Guam [31]. The severity of a failure depends upon the context in which it occurs; that is, the range of hazards the failure exposes and additional barriers in place to prevent those hazards from occurring. A failure might be harmless if an operator notices it and is able to compensate for it, but some failures may lead to hazards too quickly to be mitigated and others may go undetected until mitigation is no longer possible. Even if the effects of a failure can be mitigated, given enough recurrences it will eventually create a hazard that leads to an accident.

Perrow introduces the dimensions of *complexity* and *coupling* in describing our ability to comprehend various classes of systems. Digital systems, and especially software, engage in complex interactions both with external systems and throughout their internal designs. Control branches, feedback loops, and jumps between routines are common in software-based control systems. Hidden interactions might also exist, especially with regard to shared resources where contention between processes can create race conditions and deadlock. Moreover, digital systems are tightly coupled because components frequently depend on other components within the system to complete their tasks. Perrow argues that systems exhibiting interactive complexity and tight coupling are prone to accidents involving “the unanticipated interaction of multiple failures” of components, which he calls *system accidents* [28]. Not only are system accidents nearly inevitable, but the interactions between failures may hinder attempts to diagnose them.

## 2.4. Analysis of Digital System Failures

Existing software engineering techniques are not well-suited to the analysis of digital system failures. As the previous section concluded, the complexity and coupling typically seen in digital systems can lead to accidents caused by interacting failures of multiple components; however software engineering techniques tend to treat components in isolation according to the principle of functional decomposition [13]. Debugging is a familiar technique for analyzing undesired behavior, but it is an art that is difficult to master and quickly becomes infeasible as system complexity grows [22]. More sophisticated tools such as UML and theorem provers can be used to represent or confirm a

theory of system failure, but they cannot attest to the completeness of that theory [14]. Model checking is a notable exception that is effective at detecting potentially hazardous states in complex systems, but it has not been applied to forensic analysis of failure [29]. Aside from the inapplicability of existing software engineering techniques, Johnson identifies the following additional factors that complicate the forensic analysis of digital system failures:

- *the lack of an accepted stopping rule for framing the analysis;*
- *the assessment of the developer's intentions;*
- *the influence of contextual factors on development; and*
- *the challenge of issuing relevant and practicable recommendations [14].*

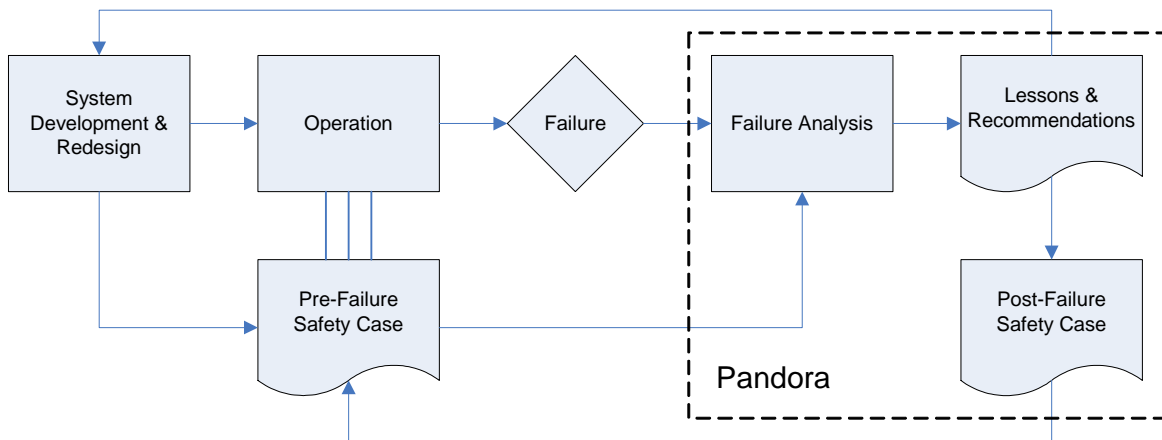
Framing an analysis refers to determining which aspects of a system's development and maintenance history might have contributed to a failure as well as the evidence that would be required to confirm or exclude them as contributory. If the scope of an investigation is too shallow, it might only address superficial issues that are symptomatic of deeper, more fundamental problems. Investigators may also wish to consider the intentions behind development decisions—why those decisions were believed to enhance or not adversely affect safety—as well as contextual factors that might have influenced the developer's actions. Finally, investigators should be confident that the recommendations they issue are both feasible and pertinent to the problems discovered during the analysis.

Several models and processes have been developed to facilitate failure analysis of digital systems, including Ladkin and Loer's Why-Because Analysis (WBA), Leveson's Systems-Theoretic Accident Model and Processes (STAMP), and Johnson's Programmable Electronic Systems Analysis for Root Causes and Experience-based Learning (PARCEL) among others, each of which is discussed further in section 6 [22, 23, 15]. Despite the availability of these techniques, the issues Johnson raises still remain. The system safety case, as the complete reference for the safety objectives of a system, the evidence that the system is safe, and the developer's rationale for claiming safety, offers a promising solution to some of these issues. Employing the safety case to facilitate the analysis of digital system failures is the topic of this research.

### 3. Proposed Research

An important feedback loop exists between safety-critical system development and failure analysis in which systems are deployed with unknown faults, those faults are discovered through observed failures, and lessons are obtained that are ultimately incorporated into development practices for future systems. The system safety case plays a central role in this feedback loop because it attests to the safety of a system before the system is deployed, and if it is undermined by a failure the safety case must be recovered in order to determine what design or procedural changes should be made to prevent the failure from recurring. Kelly and McDermid have developed a process for updating the system safety case in response to a recognized challenge to its validity that they refer to as the *safety-case lifecycle*. Challenges may arise in the form of changes to regulatory requirements, system design changes, modified assumptions, and counter-evidence obtained from operational experience including mishaps [4, 17]. They assume that the challenges to a safety argument have been recognized prior to the application of their process, and they do not address the problem of deriving challenges from an observed failure. To complete their lifecycle, we added an additional step in which failure analysis is performed on the safety case to discover the flaws in the safety argument that contributed to a failure so that the argument may then be recovered. This refined process is the *enhanced safety-case lifecycle*.

The enhanced safety-case lifecycle, illustrated in Figure 2, augments the core lifecycle so that the safety case guides failure analysis and serves as the medium through which lessons and recommendations from the analysis are communicated. The lifecycle begins when a system is initially developed according to relevant standards and industrial best practices. At the same time the system is produced, a safety argument is prepared that shows why confidence should be placed in the system's ability to meet its safety requirements and why those requirements are sound. Once the system and its safety argument are accepted, the system is put into operation; however faults may still exist in the system or its operational or maintenance procedures if the safety argument is incomplete. If triggered, these faults may lead to a failure, in which case an analysis will be conducted to discern the nature of the failure. The *pre-failure safety case*—that is, the safety case for the system as it existed just prior to the failure—can guide the analysis by leading investigators through the original argument describing why the system was thought to be safe. Using the evidence obtained from the failure, investigators correct flaws in the argument as they discover them, which are documented as *lessons and recommendations*, and ultimately produce a revised safety case for the system called the *post-failure safety case*. Implementing the post-failure safety case may require developers to make changes to the system or its associated procedures as well as changes to development practices so that future systems will not exhibit similar



**Figure 2: The Enhanced Safety-Case Lifecycle**

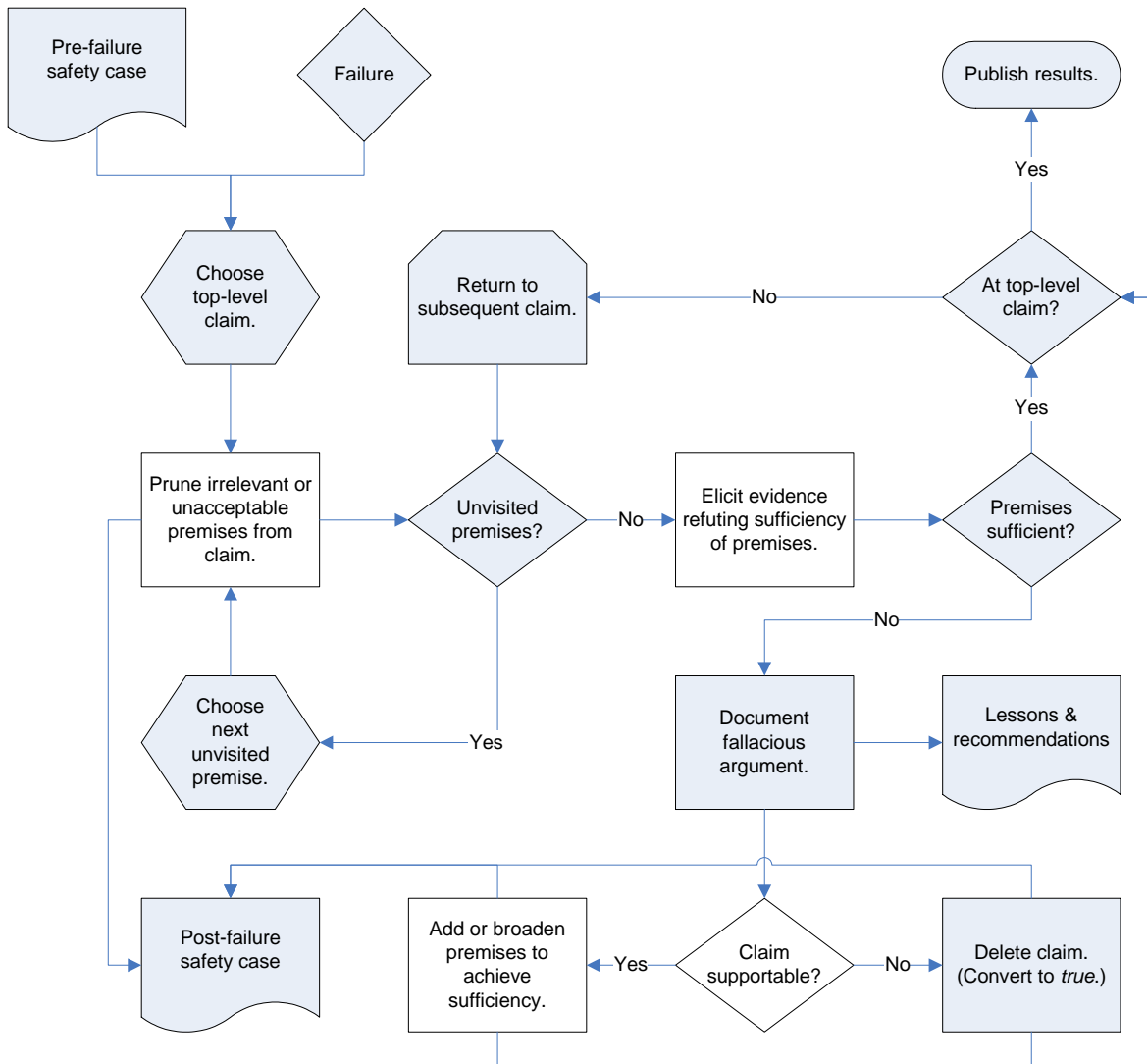
failures. Through this feedback loop in which the safety case plays a central role, system development process changes over time to reflect the accumulated experience gained from observing operational systems.

### 3.1. Pandora Overview

The proposed research concerns the development and evaluation of *Pandora*: a systematic approach to analyzing digital system failures that is grounded in the system safety case and implements the failure analysis process described in the enhanced safety-case lifecycle. It assumes that safety-related failures result from systemic defects, which exist in an operational system due to flaws in the system’s safety case. Based upon counter-evidence obtained from a failure, Pandora identifies the fallacious inferences and faulty evidence in the system’s pre-failure safety case that allowed the failure to occur. It then develops lessons and recommendations from which a post-failure safety case is produced that is free of the defects discovered earlier. Through its processes of eliciting evidence, detecting fallacious inferences, and developing lessons and recommendations—all of which are centered on the system safety case—Pandora seeks to systematize the analysis of safety-related system failures so that greater confidence may be placed in both the rigor of investigations and the effectiveness of their findings.

The high-level Pandora process is presented in Figure 3. Pandora begins with a pre-failure safety case—the safety case of the system as it existed just prior to the failure—and an observed system failure. At the heart of Pandora is a depth-first traversal of the pre-failure safety case to evaluate the cogency of claims in the safety argument and elicit refutatory evidence from the observed failure. The method of detecting fallacious reasoning in the safety argument is built on a taxonomy of safety-argument fallacies adapted from Damer and Govier and further refined through empirical study [7, 10, 11]. Pandora evaluates inferences in the safety argument with respect to three broad criteria: their *relevance* to the claims being established, their *acceptability* as plausible inferences, and, taken together, their *sufficiency* in establishing the truth of the claims they support. These three criteria constitute the basic conditions for a successful argument [10].

For each claim in the safety argument and beginning with the top-level claim, Pandora first considers the relevance and acceptability of that claim’s premises. Premises that do not support the claim or invoke inherently faulty reasoning are pruned from the argument and not considered further. This pruning process is repeated for each of the remaining premises, leading to a depth-first recursion through the safety argument. Once the relevance and acceptability of the supporting premises have been established, they are taken together and their sufficiency is evaluated. If premises are missing or if there exists evidence from the failure refuting the truth of the claim, then the premises are insufficient. When an insufficient argument is discovered in the safety case, Pandora repairs the argument by either amending the premises or, if the claim is unsupported, by deleting the claim from the argument and making reparations elsewhere. It then resumes consideration of the consequent (parent) claim in the argument. In its final step, Pandora considers the sufficiency of the top-level argument in the safety case after it has pruned from the entire argument irrelevant or unacceptable inferences and either affirmed or restored the sufficiency of those that remain.



**Figure 3: Pandora High-Level Analysis Process**

Upon concluding its traversal through the safety argument, Pandora has produced a set of *lessons* documenting the flaws in the pre-failure safety case that might have contributed to the failure, a set of *recommendations* for removing those flaws from the argument, and the *post-failure safety case*: a prototype of the pre-failure case into which the recommendations have been incorporated. With some minor revisions, the post-failure safety case is intended to become the new operational safety case for the system in light of the observed failure. Implementing the post-failure safety case may impose new requirements on the system or otherwise entail changes to the system’s design or associated operational or maintenance procedures. To further enhance the preventative value of Pandora, lessons and recommendations may be applied proactively to related systems, especially those with similar safety arguments; this topic is discussed further in section 3.4.

### 3.2. Detecting Fallacious Reasoning

Safety cases may contain premises that are irrelevant to the claims they support or that are inherently unacceptable. *Irrelevant* premises are those from which the truth of the claim does not follow from the truth of the premise—commonly known as *non sequiturs* [7]. *Unacceptable* premises invoke reasoning that is unbelievable or otherwise known to be false irrespective of the claim they attempt to establish. To use an irrelevant or unacceptable

premise in an argument is to commit a *relevance* or *acceptability* fallacy, respectively. The mere existence of a relevance or acceptability fallacy in a safety argument cannot contribute to a failure. Rather, these fallacies can mislead a developer or certification official into accepting an incomplete argument, which, in turn, may contribute to a system failure. Since irrelevant and unacceptable premises cannot themselves contribute to a failure, however, there is no need to consider them as part of an investigation, and so Pandora prunes them from the safety argument early in the process. Even if the arguments supporting the pruned claims are further flawed, repairing them is futile because they ultimately support inconsequential conclusions. Pruning irrelevant or unacceptable premises from the safety argument improves the efficiency of Pandora because it reduces the amount of evidence investigators must collect.

To assist investigators in detecting irrelevant and unacceptable fallacies in safety arguments, Pandora provides a taxonomy of common relevance and acceptability fallacies adapted from philosophical literature and refined through case studies [11]. Table 1 presents a summary of the fallacies in the taxonomy. Each entry in the taxonomy contains the name of the fallacy, a brief definition, and a more detailed explanation of how the fallacy is likely to appear in a safety argument accompanied by examples. As part of the planned research, AntiPatterns will be developed for each of the fallacies to provide investigators with graphical templates that they can compare against the arguments they evaluate. Once the pruning process has been fully developed, it will be evaluated through case studies as described in section 3.5 to determine its ease-of-use and the efficiency benefit it provides by reducing the scope of the investigation.

**Table 1: Taxonomy of Safety Argument Fallacies**

Relevance Fallacies	Acceptability Fallacies	Sufficiency Fallacies
Appeal to Improper Authority Red Herring Drawing the Wrong Conclusion Using the Wrong Reasons	Fallacious Use of Language Arguing in a Circle Fallacy of Composition Fallacy of Division False Dichotomy Faulty Analogy Distinction without a Difference Pseudo-precision	Hasty Inductive Generalization Arguing from Ignorance Omission of Key Evidence Ignoring the Counter-Evidence Confusion of Necessary & Sufficient Conditions Gambler’s Fallacy

While relevance and acceptability fallacies may hinder one’s ability to evaluate an argument, safety-related failures ultimately arise from *insufficient* arguments in a system’s safety case. An argument is insufficient if it provides too little evidence, if the evidence is biased, or if it fails to provide evidence that would be expected for the particular claim it supports [7]. To see why the former statement is true, consider the typical top-level claim of a safety case that a system is acceptably safe to operate in its intended context. If the system fails, and the failure is assumed to be systemic, then clearly the top-level claim has been violated. It is then the case that at least one of the premises supporting the top-level claim was not satisfied or that the premises together do not provide adequate support for the claim. The same can be said of each premise for which contradictory evidence is obtained until eventually one or more inadequately supported premises are reached.

Pandora considers the sufficiency of the argument supporting a claim after it has established the relevance and acceptability of the claim and its premises and after it has either confirmed or restored the sufficiency of the arguments supporting the premises. Thus, by the time investigators consider a particular argument within the safety case they have already convinced themselves of the truth of its premises and their relevance to the claim. Sufficiency is evaluated in two ways: first by checking whether the argument fits a known pattern (or AntiPattern) and has omitted expected evidence; and second by eliciting evidence from the failure suggesting that the claim was not satisfied despite the truth of the premises and then developing an AntiPattern depicting how the argument failed to address the counter-evidence. Investigators are asked to consider, for a given claim, whether there exists evidence that the claim was unsatisfied despite the premises supporting it and, if so, whether one or more the premises was unsatisfied or whether the premises, taken as a whole, provided inadequate support of the claim. If the argument fails either of these checks, then it is insufficient and must be recovered; section 3.3 discusses recovery.

If the argument being considered fits a known pattern, then investigators can use the pattern to determine whether the argument has provided the appropriate kinds of evidence and to some extent whether the evidence carries

enough weight to satisfy the claim. The fallacy taxonomy summarized in Table 1 provides broad descriptions of known sufficiency fallacies an argument might commit. Like the relevance and acceptability fallacies, these will be refined into AntiPatterns to facilitate their application to graphical safety arguments; however it is expected that specific patterns depicting common, sufficient arguments as well as AntiPatterns illustrating specific instances of the fallacies will need to be developed to make this method practical.

Even if an argument properly instantiates a pattern or if it does not fit any known patterns, it is still possible that the argument is insufficient. To determine this, investigators must search for counter-evidence indicating that the claim of the argument was not satisfied even though its premises were. If counter-evidence is discovered, then there must exist additional premises that were unsatisfied because a true antecedent cannot imply a false consequent. The counter-evidence is documented as an AntiPattern showing how the argument failed to satisfy its claim from which a pattern will later be developed that may then be used to evaluate future arguments as part of the first sufficiency check. This second method of sufficiency evaluation is key to Pandora's ability to learn from incidents because it allows investigators to discover new forms of fallacious reasoning in safety arguments and model them as patterns to ease future detection.

Once it is fully developed, the sufficiency evaluation process of Pandora will be evaluated through case studies as with the pruning process. The objectives of the evaluation will be to determine how well the process matches actual arguments against patterned ones as well as how effective it is at developing new patterns based on counter-evidence obtained from a failure.

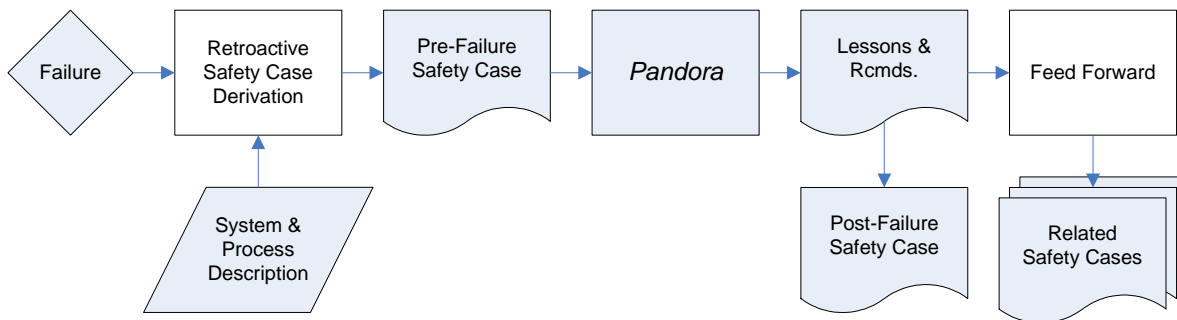
### 3.3. Repairing the Safety Argument

Once a flawed argument has been discovered in a safety case, it must be repaired. Pandora performs recovery in two phases: it first documents the flawed argument as a *lesson* and then develops a *recommendation* as to how the argument should be amended in order to satisfy its claim. Alternatively, if a claim is found to be unsupported, then the recommendation would be to strike it from the safety case. Pandora derives lessons and recommendations systematically by requiring that each lesson describe an actual fallacy in the safety case and that each recommendation correspond to and address one or more lessons, greatly reducing if not eliminating the number of superfluous lessons and recommendations produced. If the lessons and recommendations are sufficiently general, they may be applied to other Pandora investigations or even to similar safety cases for systems that have not yet failed; this topic is discussed further in section 3.4.

A lesson in Pandora is an AntiPattern describing how an insufficient argument in the safety case either failed to provide expected evidence in support of its claim or failed to address counter-evidence obtained from the failure. Lessons are generated whenever insufficient arguments are discovered. If the argument was discovered by matching it against a known pattern (or AntiPattern), then the lesson will be an AntiPattern showing how the argument fails to meet that pattern (or conforms to the AntiPattern); if it was discovered based upon counter-evidence obtained from the failure, then the lesson will be a new AntiPattern altogether. Documenting lessons as AntiPatterns helps investigators to describe precisely the nature of the fallacy that was committed and where the fallacious reasoning appears in the safety case.

From each lesson a recommendation is developed as to how the argument should be revised so that it either conforms to the pattern or addresses the counter-evidence. Just as lessons take the form of AntiPatterns, a recommendation is a *pattern* showing how the argument may be modified to address the fallacies identified by the lesson and restore sufficiency. The recommendation is a suggested strategy for repairing the argument—it does not have to encompass all possible solutions nor must the system developer implement the particular strategy contained in the recommendation. The point is to reinforce investigators' concerns by providing guidance as to how they might be accommodated. A recommendation may address multiple lessons, and so it is not necessary for there to exist one recommendation for each lesson; however each recommendation should correspond to one or more lessons to reduce the likelihood of frivolous recommendations being issued. If investigators find that a claim cannot be supported by any means, then they may recommend that it be stricken from the argument, which in the context of Pandora has the same effect on the investigation as if the claim had been pruned.

Investigators must determine to what extent the lessons and recommendations they develop should be tailored to the specific argument being analyzed. Specific lessons and recommendations may bring about the changes desired in that argument but might miss related arguments exhibiting the same fallacy. Overly general lessons and recommendations, on the other hand, might be too confusing to be useful. As part of the proposed research, the recovery process will be refined to address the issue of generality. Kelly and McDermid have developed a process for



**Figure 4: Extensions to Pandora**

assessing the impact of a recognized challenge to an argument and then changing the argument in response to that challenge, and it is expected that their process, along with the fallacy taxonomy, will form the basis of the recovery process in Pandora [17]. As with the pruning and sufficiency evaluation processes, the feasibility and comprehensiveness of the recovery process will be evaluated through case studies once it is developed.

### 3.4. Extensions

Pandora may be applied to any failure of a system for which an explicitly defined safety case exists, and it produces lessons and recommendations for removing the flaws discovered in that system’s safety argument in the context of the failure. It would be useful, however, to be able to apply Pandora to systems for which no defined safety case exists and to immediately apply the findings of a Pandora investigation to other systems whose safety arguments might exhibit similar flaws as shown in Figure 4. Potential processes for doing so are discussed below.

**Retroactive Derivation.** Most safety-related digital systems in operation today do not have explicit safety cases; however they still have evidence and an undocumented safety argument that formed the basis of their certification or acceptance. Pandora could be applied to such a system if the undocumented argument and its underlying evidence could be derived retroactively. One possible method of doing so would be to identify the predominant standard to which the system was developed and then instantiate a safety case pattern for that standard to construct the core argument. Investigators could then tailor and fill out the template argument to accommodate information obtained from interviews with developers, system documentation, and basic evidence such as fault tree analyses, hazard and risk assessments, and the like. Cooperation from the developer would be essential because only the developer would know for certain whether the derived safety case accurately reflected the de facto argument for the system’s safety. It is expected that safety cases will have to be derived for most of the systems selected for the case study evaluation of Pandora, and so this process will be applied at least in an ad hoc fashion as part of the research using factual information from official investigations. From this experience a more systematic process will be developed.

**Feed Forward.** The lessons and recommendations that arise from a Pandora investigation are targeted at the system that was the subject of the investigation; however other systems, especially those that use the same safety case patterns as the failed system, may be affected by the results. Feeding forward the lessons and recommendations of a Pandora investigation to related safety cases would raise the value of the investigation and further reduce the probability of a failure’s recurrence. To feed a recommendation forward to other safety cases, one must first express its lesson generally enough so that essential elements of the fallacy it describes are captured but unrelated system-specific details are abstracted away. Second, one must be able to identify the set of safety cases potentially affected by the lesson due to their use of similar arguments. Since safety cases are informal, it is unlikely that it will ever be possible to completely identify the set of cases that use a certain argument; however it may be possible to approximate this set by tracking which safety cases use a particular claim, assumption, justification, piece of evidence, or pattern. When a lesson is generated concerning a tracked element, all safety cases that use the element could then be checked to see if they exhibit the AntiPattern described by the lesson. If so, they could then be recovered according to the associated recommendation prior to a failure. Until the recovery process of Pandora is finalized and evaluated, it is unknown whether feed forward will be a feasible extension to Pandora. Even if it is found to be infeasible the basic form of Pandora will prevail as a systematic approach to digital system failure analysis.

### 3.5. Evaluation

The evaluation of Pandora will assess the quality of the lessons and recommendations it produces by comparing them to those of official investigations and, time permitting, other approaches to digital system failure analysis. Four distinct, phased evaluations will be conducted: a preliminary feasibility study of Pandora, an evaluation of the fallacy taxonomy separate from Pandora, an evaluation of Pandora's effectiveness in producing lessons and recommendations, and, time permitting, an evaluation of Pandora's efficiency. The preliminary feasibility study will be conducted by applying Pandora to a well-understood failure to assess the merit of the overall process as it has been described in this proposal and to discover weakness that need to be addressed in the subsequent development steps.

Prior to the evaluation of Pandora itself, the taxonomy of safety-argument fallacies will be evaluated through a controlled experiment to determine its usefulness in detecting and classifying fallacies in system safety arguments. A set of small, hypothetical safety cases will be prepared from real-world cases but into some of which fallacies have been injected. The set of safety cases will then be presented to a control and an experimental group of test subjects, most likely graduate students, who will be asked to examine the safety cases and identify any fallacies that are present. At least one of the safety cases will, to the best of the experimenters' knowledge, be non-fallacious, and at least one of the safety cases will be a pure real-world safety argument. Both groups will be trained to read and interpret the safety cases, which will include training in understanding GSN; however the experimental group will also be provided with copies of the fallacy taxonomy and the accompanying descriptions and AntiPatterns for each fallacy. The results of each test subject's performance will be aggregated, and the aggregate results from the control and experimental groups will be compared to determine to what extent, if any, the fallacy taxonomy affected the experimental group's accuracy in detecting the injected fallacies. Moreover, members of the experimental group will be asked to classify each fallacy they detect according to the taxonomy so that their ability to accurately classify true positives may also be measured. Because a study of this magnitude will incur a significant resource expense, including the acquisition of test subjects and the development of training materials and the experimental safety cases, prototype studies will be conducted to refine the taxonomy and experimental materials beforehand.

Once its fault detection and recovery processes have been developed, the complete Pandora process will be evaluated through a series of cases studies, which is a common method of evaluating novel approaches to failure analysis [15, 22, 23]. Each case study will concern a real-world system failure for which an official investigation report exists to be used as a benchmark. Ideally, safety cases would be available for the systems involved in the case studies; however very few safety cases are publicly available and those that have been located concern systems for which there are no known failure reports [8]. Instead, safety cases will be derived retroactively for the systems selected for study based upon available documentation. To minimize the possibility of contaminating the derived safety cases with information gleaned from the failure, the party who derives the safety cases will not be exposed to the results of prior investigations and will be separate from the party performing the Pandora analysis. Counter-evidence for the case studies will be obtained from official investigation reports and other sources as it is demanded by Pandora. The lessons and recommendations Pandora produces will be compared to those of the official investigations and possibly other approaches. Pandora will be judged to be an effective approach to failure analysis if it produces unique lessons and recommendations, and it will be judged to be superior if it is also able to reproduce all of the pertinent lessons and recommendations from the benchmark investigations.

Finally, the efficiency with which Pandora is able to arrive at its lessons and recommendations with a minimum of wasted effort will be evaluated if time is available. The manner in which efficiency will be evaluated has not yet been decided, and a metric will have to be developed for comparing Pandora to other approaches. One possible metric is the ratio of the number of man-hours spend obtaining evidence from which lessons were derived to the total number of man-hours spent obtaining evidence. This metric would give the percentage of time spent obtaining "useful" evidence; although it is possible that the same lessons and recommendations could have been derived from alternate and more-easily-obtained evidence. The primary objective of development Pandora is to produce unique lessons and recommendations that are arrived at systematically, and so efficiency is not essential provided that Pandora is not so exhaustive as to make it impracticable.

### 4. Contributions

The key contribution of this research is the development of *Pandora*, a systematic approach to the analysis of digital system failures founded on the system safety argument. Although Pandora is not the only process for investigating failures of digital systems, it is unique in that it systematically drives the analysis following the same reasoning from which the safety of the system was originally argued. Pandora is expected to provide the following benefits:

## 4.1. Expected Contributions

- *Systematization of the collection of evidence.* Digital systems typically do not generate overt evidence when they fail, and even if they have been instrumented to do so there is no guarantee that the information they provide will help explain the failure. Consequently, investigators typically must rely upon a system's development and maintenance history, which may span several years or even decades, to assemble most of the evidence for their investigation. Pandora provides precise objectives to guide the elicitation of evidence that are based upon the claims made in *that system's* safety argument, allowing investigators to target their review and ensuring that the evidence obtained is indeed relevant to the investigation. For example, Pandora might ask investigators to seek out evidence refuting the claim that the design diversity scheme employed in a system sufficiently mitigated the hazard of common faults in redundant components. Eliciting evidence refuting this specific claim is much easier than attempting to answer the generic question of whether a system's redundancy architecture contributed to a failure.

- *Systematization of the development of lessons and recommendations.* The ultimate goal of system failure analysis is to produce lessons and recommendations that document how a failure occurred and how to prevent it from occurring again, respectively. It is not always clear how the recommendations obtained from an investigation address the lessons or, in some cases, how the lessons themselves pertain to the particular failure that was observed. Pandora directs investigators to revisit each system safety claim that is relevant to a failure. If evidence refuting the claim is obtained or if the argument supporting it is inherently flawed, the unsatisfied claim is documented as a lesson, and investigators then develop recommendations as to how the argument should be revised so that it satisfies the claim. Thus, each recommendation issued from a Pandora-based investigation follows from one or more lessons, and each lesson is derived from a systematic review of the system safety argument.

- *A comprehensive yet efficient approach to analyzing digital system failures.* The systematic elicitation of evidence and derivation of lessons and recommendations make Pandora a comprehensive method of analyzing failures of digital systems; however such an approach is of marginal value if it is too exhaustive to be practicable. Pandora achieves rigor while preserving efficiency in three ways. First, it requires investigators to consider only those details of a system's design and development and maintenance histories that impact the safety of the system by focusing the investigation on the system's safety argument. Second, Pandora removes from consideration safety claims that do not pertain to the failure observed, allowing investigators to concentrate their efforts on sections of the argument that could actually affect the likelihood of recurrence. Finally, Pandora greatly reduces the possibility of extraneous lessons and recommendations being produced by an investigation by ensuring that each recommendation corresponds to a lesson and each lesson to a flaw in the safety argument that could have contributed to the failure.

## 4.2. Achieved Contributions

In addition to the expected contributions of Pandora itself, the research motivating the development of Pandora has already contributed the following:

- *Discovery of the enhanced safety-case lifecycle.* The enhanced safety-case lifecycle describes the feedback loop between system development and failure analysis in which systems are progressively improved through lessons learned from observed system behavior [12]. Kelly and McDermid introduced the core safety-case lifecycle by developing a process for revising the safety cases for a system when a challenge such as a mishap arises to that system's safety claims [4, 17]. The enhanced lifecycle builds upon this work by advocating the use of the safety case to guide failure analysis after a mishap to determine what challenges to the safety argument exist. Pandora implements this aspect of the lifecycle as a process for performing such an analysis and repairing the safety argument accordingly.

- *Development of a taxonomy of fallacies in system safety arguments.* Pandora is concerned with the discovery and repair of fallacious inferences in system safety arguments, which are attempts to support claims with irrelevant, unacceptable, or insufficient premises. To assist investigators in detecting and repairing these inferences when applying Pandora to an investigation, a taxonomy of common fallacies in system safety arguments was developed based upon a case study of fallacies observed in industrial safety cases [11]. The taxonomy provides a description of each fallacy, a description of how it is likely to appear in safety arguments, and, when possible, a suggested strategy

for removing the fallacy from the argument. In addition to its role in Pandora, developers of system safety arguments could also use the taxonomy to avoid committing the fallacies before a system is put into operation.

## 5. Research Agenda

To achieve the contributions listed in section 4, this thesis will develop the Pandora process and evaluate its feasibility, comprehensiveness, and efficiency as a method of analyzing digital system failures. The motivation for Pandora has already been provided by the development of the enhanced safety-case lifecycle, which shows that systemic defects in a system may be traced to flaws in the system's safety argument and that lessons and recommendations for avoiding those defects may be expressed as changes to the argument. The overall Pandora process has also been developed along with a taxonomy of logical fallacies in system safety arguments to assist investigators in applying the process. The remaining work concerns the development of the fault detection and recovery processes of Pandora, evaluation, and further enhancements as described below. The work is expected to be completed in the spring of 2006.

### 5.1. Prerequisite Milestones

*M1. Assess the feasibility of the overall Pandora process by applying it to a simple case study (Spring 2005).*

Pandora as described in section 3 will be applied to a previously-studied failure such as the Minimum Safe Altitude Warning System (MSAW) failure in order to determine the weaknesses in its current form [12]. The deficiencies identified in Pandora by this feasibility study will guide the completion of subsequent milestones.

*M2. Develop safety-case AntiPatterns for the fallacies described in the taxonomy (Summer 2005).* Developing AntiPatterns for the fallacies in the taxonomy will support future milestones and add utility to the taxonomy itself.

### 5.2. Development Milestones

*M3. Develop a process for pruning irrelevant premises from system safety arguments (Summer 2005).* The pruning process will improve the efficiency of Pandora by excluding irrelevant and unacceptable premises, which contribute nothing to the safety argument, from further consideration. Section 3.2 discusses the pruning process in detail.

*M4. Develop a process for evaluating the sufficiency of system safety arguments (Fall 2005).* Systemic faults that contribute to safety-related failures can be traced to weakly supported claims in a system's safety argument. The sufficiency-evaluation process will comprise the core fault-detection capability of Pandora, and it will be the mechanism through which lessons are derived from the failure. Section 3.2 discusses this process in further detail.

*M5. Augment the sufficiency-evaluation process to support recovery of the safety argument (Fall 2005).* Once a fault has been discovered in a safety argument, the argument must be revised to remove the fault, which may in turn require changes to the system itself. The addition of support for safety-argument recovery will enable Pandora to produce recommendations for preventing a failure from recurring. Section 3.3 discusses recovery in more depth.

### 5.3. Evaluation Milestones

Section 3.5 discusses the evaluation of Pandora and the safety-argument taxonomy in detail.

*M6. Evaluate the effectiveness of the safety-argument fallacy taxonomy (Summer 2005).* The fallacy taxonomy provides much of the support for Pandora's fault detection capability, and it may be used outside of Pandora to improve the quality of safety arguments before they are put into operation. Evaluation of the taxonomy through a controlled study will assess its effectiveness in supporting these efforts.

*M7. Evaluate the comprehensiveness of Pandora's lessons and recommendations (Spring 2006).* Pandora will be applied to case studies of real-world failures to assess its ability to detect systemic faults, recover safety arguments, and produce lessons and recommendations. Pandora's effectiveness will be evaluated by comparing its lessons and recommendations to those of official investigations and, time permitting, other failure analysis approaches.

*M8. Evaluate the efficiency of Pandora as a whole (time permitting).* An efficiency evaluation would assess the cost of applying Pandora and give a measure of the fraction of time spent investigating fruitful leads.

## **5.4. Further Enhancements**

Extensions to Pandora to support retroactive derivation of the safety case and to feed-forward lessons and recommendations to related systems are discussed in section 3.4. These milestones will be completed if time permits.

*M9. Develop a process to feed forward the lessons and recommendations of a Pandora investigation to related safety arguments.* Feed-forward will support the proactive elimination of faults in similar, vulnerable systems.

*M10. Develop a process for retroactively deriving the safety case for a failed system.* Support for retroactive safety-case derivation will enable Pandora to be applied to failures of systems for which no explicit safety case exists.

## **6. Related Work**

In addition to Pandora, other techniques exist for repairing safety arguments and analyzing system failures. The most notable of these are safety-case maintenance, Why-Because Analysis, STAMP, and PARCEL. The following sections discuss each of these techniques and compare them to Pandora.

### **6.1. Safety-Case Maintenance**

Kelly and McDermid have developed a systematic process for updating a safety case in response to a challenge against a claim, piece of evidence, or contextual detail in its safety argument [17]. The challenges they envision may stem from new regulatory requirements, system design changes, revised assumptions, operational experience, or observations of operator behavior. Once a challenge has been identified, their process covers assessing the impact of the change on the safety argument and recovering the argument to a “correct, complete, and consistent state.” In the context of Pandora, lessons may be viewed as challenges to the safety argument, and once a lesson has been documented Kelly and McDermid’s process could be used to develop recommendations that address it. Indeed, it is likely that the recovery process of Pandora will be based in part upon their maintenance process. Unlike Pandora, however, they assume that the change to the argument has been recognized prior to the application of their process, and so their process cannot be used to detect faulty arguments or derive lessons from failures.

### **6.2. Why-Because Analysis**

Why-Because Analysis (WBA), developed by Ladkin and Loer, is a method of developing rigorous, highly formal event-chain models of accidents and incidents based on Lewis’s counter-factual reasoning [22]. WBA begins with a semi-formal graphical reconstruction of an accident’s event sequence, which is then translated into temporal, counter-factual logic and verified for logical consistency and sufficiency. The result of this analysis is a highly rigorous causal explanation as to why an event occurred. WBA does not address the issues of how the event sequence is initially determined or what evidence is needed to establish the event sequence, nor does it suggest where recommendations should be issued to prevent the causal chain from recurring (although one might infer some of this information from the graphical depiction of the event sequence). Thus, WBA is useful for those who have a hypothesis as to why an accident occurred and wish to verify its soundness. Pandora is designed to facilitate the development of an accident hypothesis based upon fallacies discovered in the system safety argument as well as recommendations for repairing the system that addresses the deficiencies expressed in the hypothesis. Nevertheless, those desiring the rigor afforded by WBA could develop an event chain of an accident based upon the lessons derived from a Pandora investigation and then use that event chain to produce a WBA graph.

### **6.3. STAMP**

Leveson’s Systems-Theoretic Accident Model and Processes (STAMP) is an accident model grounded in systems theory that views accidents as the result of undesired interactions between system components [23]. STAMP models systems and accidents as control and informational feedback loops in order to depart from event-chain models and seek out management, organizational, governmental, and other social-technical factors that might have influenced the event sequence preceding an accident. STAMP classifies accidents as arising from either inadequate enforcement of constraints on system behavior, inadequate execution of control actions, or inadequate or missing feedback.

Despite its name, STAMP is primarily a model, not a process. Expressing the relationships between management and organizational structures and the system itself can certainly provide insight into how those structures might have influenced the system's development or operation and possibly contributed to an accident, but STAMP does not define a rigorous process for detecting dysfunctional interactions or determining where in the social-technical structure remedies should be applied. By focusing on the system safety case, Pandora is able to examine the safety requirements, evidence of safety, and contextual details of a system's development while providing a systematic process for detecting faults and removing them. That said, Pandora and STAMP are compatible, and both techniques may be used in an investigation. This compatibility exists because safety constraints from higher-level control systems in a socio-technical structure function as safety obligations that lower-level systems must conform to by implementing their own lower-level constraints. Safety constraints appear in a system's safety argument either as goals or contextual information that frames the argument, and so they may be analyzed by Pandora. Likewise, the lessons from a Pandora investigation may be expressed in STAMP since they correspond to inadequate or unenforced safety constraints.

#### 6.4. PARCEL

Johnson and Bowell's PARCEL (Programmable Electronic Systems Analysis for Root Causes and Experience-based Learning) is an accident analysis technique specifically developed for investigating failures of programmable systems that "traces the causes of adverse events back to the lifecycle phases a common requirements of the IEC 61508 standard" [15]. PARCEL is actually pair of causal analysis techniques. The first is a lightweight flow-charting technique that can be applied relatively quickly to suggest typical potential causal factors; this approach might be applied to low-cost accidents or incidents. The second approach involves a more complex events and causal factors (ECF) analysis that could be reserved for incidents with a higher risk of recurrence. Both of the techniques map possible causal factors in the accident or incident to lifecycle phases in system development as specified in the IEC 61508 standard. Casual factors are classified according to a taxonomy developed from the same standard.

PARCEL identifies causal factors in a digital system failure by searching for deviations between the system's development lifecycle and the lifecycle prescribed by the standard to which it was developed. Although PARCEL is based upon the IEC 61508 standard, Johnson envisions adapting it to other popular standards such as DO-178B and DS 00-55. It is not clear, however, whether PARCEL would be able to discover a flaw in the standard itself. Most digital system standards, including those mentioned, are prescriptive, and Weaver has argued that prescriptive standards do not assure safety because they over-emphasize the importance of following an unproven development process [32]. Thus, even if a system were developed in compliance with standards it is possible for the system to exhibit a safety-related failure due to a systemic fault. Whether PARCEL would be able to detect such a fault is questionable because the fault would not correspond to any deviation from the standard. Pandora does not make assumptions about the manner in which a system was developed, and so it does not suffer from this limitation; however it does assume that the system has a pre-failure safety case (or that one may be derived retroactively), and the extent of the lessons Pandora is able to derive from the safety case could be limited by the comprehensiveness of the safety case itself. Nevertheless, if the safety case is flawed, Pandora will be able to derive some lessons from it and provide at least an incremental improvement to the safety of the system.

Although PARCEL is intended to be used as the driving analysis technique in an investigation, Johnson notes that other techniques may be used to perform the causal analysis instead of ECF such as STAMP or WBA. To make the process compatible with Pandora, the requirements imposed by IEC 61508 could be expressed as a safety-case template from which the safety argument for the system under investigation is derived. Pandora could then analyze the argument, and the lessons and recommendations arising from Pandora could then be incorporated into PARCEL. Thus, investigators could choose which causal analysis techniques they wished to apply according to the nature of the system and the needs of the investigation: STAMP to analyze interactions between subsystems, WBA to produce a rigorous proof of causal arguments, or Pandora to detect fallacious reasoning in safety arguments.

#### References

- [1] Adelard. "The Assurance and Safety Case Environment – ACSE." 31 Jan. 2005. <<http://www.adelard.co.uk/software/asce/index.htm>>
- [2] Adelard. *The Adelard Safety Case Development Manual*. 31 Jan. 2005. <<http://www.adelard.co.uk/resources/ascad/index.htm>>
- [3] Avižienis, Algirdas, et al. "Fundamental Concepts of Computer Dependability." *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004): 11-33.

- [4] Bishop, Peter, and Robin Bloomfield. "A Methodology for Safety Case Development." *Proc. of Safety-critical Systems Symposium*, Feb. 1998, Birmingham, U.K.
- [5] Butler, Ricky W. and George B. Finelli. "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software." *IEEE Transactions on Software Engineering* 19.1 (1993): 3-12.
- [6] CET Advantage, Ltd. "GSNCasemaker: Goal Structuring Notation Tool." 2004. 31 Jan. 2005. <<http://www.cetadvantage.com/website/GSNCasemaker.aspx>>
- [7] Damer, T. Edward. *Attacking Faulty Reasoning: A Practical Guide to Constructing Fallacy-Free Arguments*. 5<sup>th</sup> ed. Belmont: Wadsworth. 2005.
- [8] Dependability Research Group. "Safety Case Repository." 2004. 28 Jan. 2005 <<http://dependability.cs.virginia.edu/research/safetycases/safetycasesexamples.php>>
- [9] Fenton, Norman. "How to Improve Safety-Critical Standards." *Safer Systems*. Proc. of 5<sup>th</sup> Annual Safety-Critical Systems Symposium, 1997. Ed. T. Redmill and T. Anderson.
- [10] Govier, Trudy. *A Practical Study of Argument*. 2<sup>nd</sup> ed. Belmont: Wadsworth. 2005.
- [11] Greenwell, William S., C. Michael Holloway, and John C. Knight. "A Taxonomy of Fallacies in System Safety Arguments." 2004. 17 Jan. 2005 <<http://www.cs.virginia.edu/~wsg6p/docs/papers/dsn.05.pdf>>.
- [12] Greenwell, William S., Elisabeth A. Strunk, and John C. Knight. "Failure Analysis and the Safety Case Lifecycle." *Human Error, Safety, and Systems Development*. Proc. of 7<sup>th</sup> Working Conference on Human Error, Safety, and Systems Development, 22-27 Aug. 2004, Toulouse, France. Ed. Chris W. Johnson and Philippe Palanque. Boston: Kluwer, 2004.
- [13] Johnson, C.W. *Failure in Safety-Critical Systems: A Handbook of Incident and Accident Reporting*. Glasgow: U of Glasgow Press, 2003.
- [14] Johnson, C.W. "Forensic Software Engineering: Are Software Failures Symptomatic of Systemic Problems?" *Safety Science* 40.9 (2002); 835-847.
- [15] Johnson, C.W. and M. Bowell. "PARCEL: Using Software Development Standards to Guide the Investigation of Computer-Related Incidents and Accidents." 14 Jul. 2004. 28 Jan. 2005. <<http://www.dcs.gla.ac.uk/~johnson/papers/parcel.PDF>>
- [16] Kelly, Tim. "Software Safety – by Prescription or Argument?" *Safety Systems*. Nov. 1997. <<http://www-users.cs.york.ac.uk/~tpk/tpkscsarticle.pdf>>
- [17] Kelly, Tim, and John McDermid. "A Systematic Approach to Safety Case Maintenance." *Reliability Engineering and System Safety* 71 (2001): 271-284.
- [18] Kelly, Tim and John McDermid. "Safety Case Construction and Reuse Using Patterns." Proc. of 16<sup>th</sup> International Conference on Computer Safety, Reliability, and Security (SAFECOMP'97), 7-10 Sept. 1997, U of York. New York: Springer-Verlag, 1997.
- [19] Kelly, Tim and John McDermid. "Safety Case Patterns – Reusing Successful Arguments." Proc. of IEE Colloquium on Understanding Patterns and Their Application to System Engineering, London, Apr. 1998.
- [20] Kelly, Tim and Rob Weaver. "The Goal Structuring Notation – A Safety Argument Notation." Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases, 1 July 2004, Florence, Italy.
- [21] Kelly, Timothy P. "Arguing Safety - A Systematic Approach to Managing Safety Cases." Diss. U of York, 1998.
- [22] Ladkin, Peter and Karsten Loer. *Why-Because Analysis: Formal Reasoning About Incidents*. RVS-Bk-98-01, 1998. 28 Jan. 2005. <<http://www.rvs.uni-bielefeld.de/publications/books/WBAbook/>>
- [23] Leveson, Nancy. "A New Accident Model for Engineering Safer Systems." *Safety Science* 42.4 (2004): 237-270.
- [24] Leveson, Nancy G. *Safeware: System Safety and Computers*. Boston: Addison-Wesley, 1995.
- [25] Littlewood, Bev and Lorenzo Strigini. "Validation of Ultrahigh Dependability for Software-based Systems." *Communications of the ACM* 36.11 (1993): 69-80.
- [26] Lowrance, William W. *Of Acceptable Risk: Science and the Determination of Safety*. Los Altos: William Kaufman, 1976.
- [27] McDermid, John A. "Software Safety: Where's the Evidence?" *Conferences in Research and Practice in Information Technology* 3 (2001): 1-6. Proc. of 6<sup>th</sup> Australian Workshop on Safety-Critical Systems and Software, 6 July 2001, U of Queensland. Darlinghurst: Australian Computer Society, 2001.
- [28] Perrow, Charles. *Normal Accidents: Living with High-Risk Technologies*. Princeton: University Press, 1999.
- [29] Rushby, J. "Using Model Checking to Help Discover Mode Confusions and Other Surprises." Proc. of 3<sup>rd</sup> Workshop on Human Error, Safety, and Systems Development, U of Liège, Belgium, Sept. 1999.
- [30] Storey, Neil. *Safety-Critical Computer Systems*. Harlow: Prentice Hall, 1996.
- [31] United States. Natl. Transportation Safety Board. *Controlled Flight Into Terrain, Korean Air Flight 801, Boeing 747-300, HL7468, Nimitz Hill, Guam, August 6, 1997*. Aircraft accident report NTSB/AAR-00/01. 13 Jan. 2001. 27 Jan. 2005. <<http://www.nts.gov/publicatn/2000/AAR0001.pdf>>
- [32] Weaver, Robert A. "The Safety of Software - Constructing and Assuring Arguments." Diss. U of York, 2004.