Hazard Analysis for Software Safety:  
A Technology Maturation Study

Jennifer Morris  
Carnegie Mellon University  
17-939A: What Makes Good Research in Software Engineering?  
April 28, 2005

Abstract

As the role of software in safety-critical systems continues to grow, so does the need for appropriate, comprehensive, and efficient methods to analyze software safety during the design process. This study reviews the historical development and maturation of software hazard analysis. Results of the study show that the major technological contributions have focused on development and internal exploration, with little effort put towards demonstration of the cost/benefit advantages of the various techniques. As a result, software hazard analysis has not yet progressed to the popularization stages of maturity.

1 Introduction

In the twenty years since its inception, software hazard analysis has made little headway into real-world development environments. Industries that have used it to some degree, such as aerospace, are often tightly-coupled to the research itself. This study reviews the historical development and maturation of software hazard analysis, using the reference model defined by Redwine & Riddle [19]. Ideally, the study of a fully mature technology would show the progress from basic research, through concept formation, development and extension, and enhancement and exploration (both internal and external), to popularization. A study of software hazard analysis, a technology that is not yet fully mature, should reveal not only the stages it has achieved, but also the factors that have inhibited its further progress.

In order to analyze the maturation of software hazard analysis I selected and reviewed major publications in the research field, including papers from conferences and journals, and textbooks. I then summarized the major research question, result, and validation of the papers using the model proposed by Shaw [20]. From these summaries I mapped the major technical contributions of the papers to stages in the Redwine/Riddle technology maturation model. Figure 1 shows the maturity model for Software Hazard Analysis.

This technology does not map perfectly to the model. Development and extension began with the basic research and extended through the concept formation. The technology remains in the internal enhancement and exploration stage, with minimal efforts to advance to external enhancement and exploration. This disjoint maturation has resulted in an extensive selection of techniques that are each somewhat deficient for their intended purpose.

2 Basic Research

Safety engineering has thrived in various domains such as aviation, rail, and chemical processing long before the introduction of computers. Leveson defines safety as “freedom from accidents or losses” and a hazard as the condition of a system that could lead to an accident or loss [8]. The goal of safety engineering is to reduce risk (the combination of the likelihood that a hazard will occur and its consequences [21]). Hazard analysis is used throughout the design process as a structured method of reasoning about hazard...
causation. Although it may not be possible to eliminate all hazards, it is still important to try to reduce overall system risk as much as possible.

Safety for software systems was first identified as a distinct research area by Leveson & Harvey in 1983 [8]. The paper asserted that the increased reliance on software in safety-critical systems required the development of new methods for safety-critical software design. It also presented the first software hazard analysis technique, Software Fault Tree Analysis (SFTA), which was based on a similar technique for hardware called Fault Tree Analysis (FTA). FTA is a backward search technique that starts from a hazard and traces through its causal event chain to identify the root causes [8]. Component failure events are connected by logical AND and OR operators to indicate when an output event requires all or at least one input event. This allows failure scenarios to be constructed and simplified using Boolean algebra. SFTA, as first presented in the Leveson & Harvey paper, traces hazardous software outputs back through the code to the logic and/or input failures. The results of the code-level SFTA are then used to determine where run-time assertions might be necessary. For very large software systems, the analysis can grow quite quickly.

Although the initial ideas of software-centric and code-level analyses were later abandoned for systems-centric and requirements analyses, the central concepts of software fault trees (causal event chains leading to hazards) and safety for software systems have remained a primary technique in software hazard analysis. The development and extension of these ideas are discussed later.

3 Concept Formation

In 1986 the publication of Leveson’s seminal survey paper of software safety [7] provided a more refined definition of the problem. This paper explained in detail what safety was and how it related to other research areas (such as security and dependability), what were the differences between software and hardware safety, and what were the current and needed techniques for design and analysis of safe software.

This paper provided a good definition of the research questions related to software safety. However, it did not provide the answers. For example, it remained unclear which stage of development was optimal for hazard analysis. The original research of SFTA focused on code, primarily because the original hardware technique FTA analyzed detailed system designs. Defects in the requirements or design, however, are much more expensive to correct if found during later stages of development.

In 1992 Lutz investigated the primary sources of safety-critical software errors [12]. She reviewed software defects found during integration and system testing of two critical spacecraft. The analysis revealed that almost all of the defects were in the requirements; very few were in the code. In addition, the safety-critical defects were more likely to be found in the interfaces between the software and the rest of the system (particularly hardware), or were related to incorrect understanding of the behavior of the software with respect to the rest of the system. This result guided further research in safety analysis and design.
towards the requirements stage of development, rather than later stages of design and code. In addition, it indicated that a safety strategy focused on software in isolation was flawed because safety of the software is reliant on the overall context of the system. This caused a second major shift in the research from software safety towards systems safety.

4 Development & Extension

The development and extension stage of software hazard analysis also began with the first Leveson & Harvey SFTA paper [10], and continued as the concepts of software safety and hazard analysis were refined. This was the seminal paper in software hazard analysis, not only because it introduced the research area, but also because it provided the first clearly defined solution. Although the solution was not perfect, many subsequent solutions were simply extensions and/or refinements of this basic technique. For example, Cha et al. continued to refine SFTA in 1988 by creating a tool for analysis of software written in ADA [1]. Although they were able to demonstrate that certain parts of the technique could be automated, such as translation from code to control-flow graphs, the actual construction and analysis of the fault trees still required an expert. The tool did, however, have a graphical user interface designed to help the analyst.

One problem with a fault tree based technique is that the event-chain model of hazard causation does not accurately capture hazards that arise from complex interactions of components and timing issues. The next major category of hazard analysis, state-based analysis, attempted to solve this problem. In 1987 Leveson & Stolzy proposed a technique based on time Petri nets [11]. A time Petri net is basically a finite state-machine with states and timed transitions. For hazard analysis, a Petri net model of the system, including hazardous states, is created and used to create a reachability graph. Paths to hazardous states are traced to identify locations to add blocking elements, such as interlocks. Although the technique has the potential to analyze hazards caused by timing issues, like SFTA it is labor-intensive. In addition, the petri-nets and reachability graphs are much more difficult to read and understand.

During the concept formation stage, when requirements defects were identified as the primary cause of software hazards, a third type of hazard analysis technique emerged. In 1993 Lutz proposed including a safety checklist during general requirements analysis [13]. This list was derived from the results of her previous study of safety-critical software defects and focused on input and output checks. It also included checks on robustness measures, such as feedback loops and mode reachability analysis. Although a checklist can remind the analyst of specific problem areas, there is no guarantee that the analyst applies the list to all components in the specification.

Shortly after the proposal of safety checklists, two new techniques improved on the safety checklist model by creating a specific list of failure modes to check for each software component. The first, Hazard and Operability analysis (HAZOP) [17], was based on the technique of the same name developed by the chemical industries. HAZOP uses guide words for flows (data flows in software) to identify possible failure modes for the different software components. A structured brainstorming of the failure modes, their causes, and possible effects can be used to identify corrective measures in the system design. The second, Software Failure Modes and Effects Analysis (SFMEA) [15, 14], is a similar technique that uses a forward search from component failure modes to the possible system effects. Both techniques present the results of the analysis in tabular format, and both have some list of failure modes that must be checked for every component. Whereas HAZOP proposed the use of general guide words for failure modes such as omission, commission, early, late, coarse incorrect, subtle incorrect, the approach proposed by Lutz & Woodhouse in 1997 proposed analyzing specific failure modes for data inputs & outputs (absent, incorrect, mistimed, and duplicate) and processes (halt, omission, incorrect logic/event, and timing/order).

HAZOP and SFMEA, like their predecessors, suffer from the same scalability limits and reliance on analyst expertise. In particular, every software input, output, and process must be checked for every possible failure mode. In some systems these components and failure modes can be quite numerous. Also, the expe-
rience used for validation in the paper relied on at least seven different iterations of the SFMEA. Repetition is a characteristic of all hazard analysis techniques because each time a hazard is identified and corrected, a new analysis must be completed.

Although this stage of development began with a technique that applied to code, subsequent techniques quickly shifted focus to analysis of requirements. This change reflects the results of the concept formation research that occurred midway through development and extension. The shift from a software-in-isolation view of safety towards a systems view of safety came more gradually as techniques began to look more closely at the software interfaces of the system. By the time internal enhancement and exploration arrived, requirements were the primary targets of hazard analysis, and most techniques analyzed the effect of software behaviors on system safety.

5 Enhancement & Exploration (Internal)

The internal enhancement and exploration stage of software hazard analysis began with the use of combined techniques. Goddard demonstrated a combined approach to hazard analysis that relied on Petri-nets and SFMEA [4]. In this method, an SFMEA is performed on the Petri net to analyze possible failure modes of the Petri net, such as failure of a transition to fire, and their affects on the overall system. The Lutz & Woodhouse paper of 1987 not only presented an improved technique for SFMEA, but it also demonstrated that combining techniques (SFMEA & SFTA in this example) could produce a more thorough analysis. The researchers used SFMEA to identify which faulty inputs, outputs, and/or processes could lead to hazards, and SFTA to determine whether those faults could occur. In both cases, combining techniques with complimentary benefits provided a more comprehensive hazard analysis.

At this time, researchers also began to explore other uses of software hazard analysis. Gorski and Wardzinski used SFTA to derive real-time requirements [5]. Their method produces a formal model of the system-level fault-tree in a duration calculus. Real-time requirements are derived from the causes of hazards related to timing issues. In 1998 Hansen & Ravn used a similar technique to derive safety requirements from SFTA [6]. They also produced a formalized fault tree of the system in a duration calculus, and then traced the system hazards back to their sources. All underlying causes of the system hazards were then used to derive safety requirements for the software in the system. Maxion and Olszewski built upon the ideas from hazard analysis techniques to create a structured diagram of exception types to improve software dependability during design [16]. Each of these techniques illustrates the extension of the general software hazard analysis to other problem domains.

Another enhancement of the original idea of SFTA was the extension to include support for product-line analysis [2]. This technique added information to the fault tree that identified which components were common across product lines, and which varied. Analysts could use this information to reduce the fault tree for a particular version of the product line by eliminating analysis of sections that were not common to all versions, or were not chosen for the particular version. Hazard analysis techniques are all quite resource-intensive, so development of techniques for partial re-use of fault trees is useful for advancement to further stages of external exploration and popularization.

In 2004, Leveson extended the idea of state-machine based hazard analysis with the Systems Theoretic Accident Model and Process (STAMP) [9]. She asserted that hazards cannot be analyzed with event-chain models of causation, that safety analysis must only be done at the system level, and that analysis techniques should look at failures in control processes to enforce constraints, not at component failures. A STAMP hazard analysis models the entire system with a state-machine model to look for process control failures that allow safety constraints are violated [3].

The primary trend of this development stage has been the use of enhanced or modified existing techniques. This has resulted in techniques that are better suited for their purpose (STAMP, product-line SFTA, combined techniques), or that are useful for another domain (SFTA for real-time & safety requirements).
However, no research has yet solved the problem of scalability of software hazard analysis. All techniques require an expert to perform much of the analysis by hand. Also, little research has been devoted to comparing techniques. Although software hazard analysis remains in this stage today, the following section describes a small amount of work that has been done in external enhancement and exploration.

6 Enhancement & Exploration (External)

The external enhancement and exploration of software hazard analysis is limited to two textbooks published in the mid 1990’s. The first was “Safeware: System Safety and Computers” published by Leveson in 1995 [8]. The second, “Safety-Critical Computer Systems,” by Neal Storey came in 1997. These two texts were intended not only for use in safety-critical systems classes, but also for system designers in industry, as reference manuals for system safety. In addition to hazard analysis, the books covered a wide variety of topics related to safety-critical systems, such as design, testing, and verification & validation.

7 Research Strategy

Most of the papers published in this technology field follow a common research strategy. In the Shaw model, the common research question is a method for analysis (How can I evaluate hazards in software systems?), the common result is a new/improved technique for hazard analysis, and validation is almost always by example or experience. In the seminal SFTA paper, Leveson and Harvey validated the technique by applying it to the software of a small spacecraft, the NASA/ESA FIREWHEEL [8]. Their analysis revealed one possible source of hazards, an infinite while loop in the software. This pattern was repeated by subsequent techniques (Petri Nets, HAZOP, SFMEA, etc.) [11, 1, 13, 17, 15, 14].

This research strategy might be sufficient for basic proof of concept, but it falls short as a convincing argument for one technique or another. First, the analysts using the hazard analysis technique are almost always the researchers who created them. Analysts in a real-world environment would not be so well-trained in the correct usage of the techniques. Second, the validation experiences often only involve one system, which is often a toy example or research system that is not intended for production. Third, the hazard analyses performed in the validation are not complete. In the SFTA on the FIREWHEEL spacecraft, only one hazard was explored. A real system would contain several potential hazards that must be analyzed. Finally, the results of the analysis focus on what hazards/causes were found. Little or no attempt is made to compare the analysis with other techniques, or to provide some cost/benefit analysis. It is important to know whether or not a technique can find hazards and their causes, but it is equally important to know how much work has to be done to find them.

One paper attempted to compare different hazard analysis methods. Modugno et al. applied three different techniques, in succession, to a high-speed transport guidance system at NASA Ames [18]. The results of the analysis revealed that each technique found different hazards in the system, and that no technique found them all. In fact, simply creating the models for the state-machine based technique revealed several sources for hazards in the system. The study was not as rigorous as it should have been; the analysts were all graduate students or academic researchers and only one system was studied. However, it is at least an attempt to answer a different question, which is “How do different hazard analysis techniques compare?”

The study by Lutz on hazardous software defects also approached a different research question, generalization [12]. This research focused on answering, “What are the most common causes of safety-related software errors in safety-critical, embedded systems?” Although the question was different, the primary method used to obtain and validate the results was experience (studying the defect data from two safety-critical spacecraft). This experience is limited to two very similar safety-critical systems, in the same application domain, possibly by the same development team. Once again, validation by experience of one analysis on two such similar systems falls short for the intended research question and result, namely a generalization/characterization of safety-critical software defects.
8 Discussion and Conclusions

The results of this study indicate that software hazard analysis has stalled at the internal enhancement & exploration stage of the Redwine/Riddle technology maturation model. Much work has been done to create new and improved software hazard analysis techniques, yet these methods have not been widely adopted in industry. SFTA and SFMEA are used to some extent in the aerospace industry, but this domain is highly coupled with the researchers and institutes that produced the techniques. So why is there little external exploration and no popularization?

Each technique is resource-intensive, and there is no technique that is fully automated. Most analyses are completed manually by domain experts. The techniques that are somewhat automated, such as Petri-nets and other state-machine models, still require a domain expert to generate the model to be analyzed. As a result, the techniques are limited by the expertise of the analyst. Given that the techniques are so expensive, development organizations with tight budgets must be convinced of the value they receive from the added expense of the analysis. There have been few attempts to identify the cost/benefit ratio of any particular technique, let alone compare it with another. The research in software hazard analysis thus far has failed to provide convincing evidence of this value.

Future progress in this research area must include some standardized methods for comparing techniques. In addition, significant market penetration requires more automation. In order to effectively transfer the technology to industry, researchers must provide convincing evidence of its benefits, and lower its costs.

References


[Method of Analysis, Technique, Example]
This paper presents a tool to perform Software Fault Tree Analysis (SFTA) on Ada programs. This was the first example of a hazard analysis technique for higher-level software languages, and the first tool in this research field.


[Method of Analysis, Technique, Example]
This paper presents a technique for using Software Fault Tree Analysis (SFTA) for product-line software. Information about commonality and variability are added to the software fault tree. These trees are then reduced for individual versions by removing the variable components which are not used.


[Method of Analysis, Technique, Example]
This paper presents the technique to apply STAMP for hazard analysis. The system is modeled as a state-machine, and the state machine is then analyzed to find violations of the safety constraints.

This paper presents a combined hazard analysis technique (Petri nets + SFMEA). First the Petri net is constructed, then the model is analyzed with a SFMEA to identify failures in transitions that could lead to hazards.


This paper presents a method for deriving real-time requirements from a Software Fault Tree Analysis (SFTA). The fault tree is constructed, then formalized in duration calculus to capture timing issues. The roots of this formal model are then used as the real-time requirements.


This paper presents a technique for using Software Fault Tree Analysis (SFTA) of a system to derive the software safety requirements.


This was the seminal survey paper that defined software safety and enumerated its challenges.


This was the first textbook on safety for software systems. In addition to software hazard analysis, the book covers other issues in safety, such as design, testing, and verification and validation.


This paper presented a new model for accident causation the Systems Theoretic Accident Model and Process (STAMP). In the model, accidents are caused by failures of control processes to enforce safety constraints. This paper also asserts that system safety has to be analyzed as a whole (there can be no separate software and hardware analyses).


This paper was the first paper to identify the need for research on software safety. It also presented the first software hazard analysis technique Software Fault Tree Analysis (SFTA), which was adapted from a hardware hazard analysis (FTA).

This paper presents a hazard analysis technique that uses time Petri nets models. The state-machine model of the system is captured in the Petri net, and then the reachability graph of hazardous states are analyzed. Blocking factors, such as interlocks, are then added to the model to prevent reaching of hazardous states.


This paper attempted to identify the primary sources of safety-related software defects. The results showed that most safety-critical defects were found in the requirements and involved the interface of the software with the rest of the system.


This paper presents a safety checklist for use during requirements analysis. The checklist is intended to remind the analyst to check for defects in the requirements that are safety-related.


This paper presented the first good example of Software Failure Modes and Effects Analysis (SFMEA). Data faults (absent, incorrect, wrong timing, duplicate) and event faults (halt, omission, incorrect logic/event, timing/order) are traced forward to hazards. This paper also provided a combined approach (SFMEA + SFTA).


This paper presents a method for combining Software Failure Modes and Effects Analysis (SFMEA) and Software Fault Tree Analysis (SFTA). The technique starts with an FTA of the system, then uses SFMEA and SFTA on the software.


This paper presents a tool (a fishbone diagram of exception types) that can be used during development to help improve exception handling. It provides a thorough explanation of the experimental method, including experimental design, participant selection, the procedure, and the evaluation criteria. It also provides a statistical analysis of the results.

This paper presents a software hazard analysis technique based on the chemical industries’ Hazard and Operability studies (HAZOP). This method uses structured brainstorming of potential problems in flows (data). Each flow is checked against a set of guide words, such as too high, too low, early, late, etc.


The researchers wanted to know whether or not multiple hazard analysis techniques could be combined, and how they compared. The results showed that the techniques found different hazards, and no technique found all of them. It also showed that the expertise of the analyst is a primary factor in the success of the technique.


This paper presents a model for software technology maturation. This model includes 6 stages: basic research, concept formation, development & extension, internal enhancement & exploration, external enhancement & exploration, and popularization.


This paper characterizes the major research strategies (question, results, & validation) used in software architecture research.


This was the second textbook on safety for software systems, released shortly after the Leveson text. This book covers similar issues of software system safety analysis and design.