This is to certify that the

thesis entitled

A Three-Pronged Approach Towards Improving the Development of Safety-Critical Software Systems

presented by

Amy C. Christensen

has been accepted towards fulfillment
of the requirements for

Master's degree in Computer Science

Betty H.C. Cheng
Major professor

Date 7/12/95

A Three-Pronged Approach Towards Improving the Development of Safety-Critical Software Systems

By

Amy C. Christensen

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science

August 1995

ABSTRACT

A Three-Pronged Approach Towards Improving the Development of Safety-Critical Software Systems

By

Amy C. Christensen

We present a proactive approach towards the integration of safety-critical software systems into the existing legal structure. First, we examine how the current legal system handles cases that involve safety-critical systems. Next, we look at incidents that have involved embedded software systems. The results show that there exist few software-specific guidelines for handling safety-critical systems. Also, there appears to be a lack of responsibility and accountability within the software development community. Because of the critical nature of the systems, immediate attention is warranted. In order to address these problems, we propose a three-pronged approach comprising complementary actions. The approach involves additions to current legislative policy and procedure, the restructuring of computer science education, and the induction of a Software Advisory Board. This dissertation explores the motivations for, the implications of, and the integrated use of these three types of actions.

I would like to dedicate this to my parents, Jerry and Claudia Christensen, whose love and support I truly treasure.

ACKNOWLEDGMENTS

I would like to thank those who helped me through the various stages of this thesis. My friends and family were always there to give me the needed support. Credit goes to both Janet and Chris who had to hear me out during some of my frustrating moments. Thanks for listening.

With the help from members of both the legal and computer science community, I was able to explore a new and exciting area. Thanks goes to Clark Turner for sharing his legal perspective during the beginning stages of my research. I appreciate also all of the input and discussion generated by members of the Formal Methods Course. In addition, Dr. Heimdahl and Dr. Weinshank provided me with extensive comments exposing me to many important issues I would have missed on my own.

Special thanks goes to Jerry Gannod who gave me both technical and emotional support throughout the entire project. As my personal information source for both formal methods and LaTeX, he was ready and willing to answer all of my questions and to frequently come to my rescue.

Most of all, I want to thank to Dr. Cheng for her time, knowledge, and motivation. I know that my topic dove into a new area that has not been subject to much research. I appreciate the opportunity she gave me to explore computer legal issues; she helped me gain an entirely new perspective on computer science, as a student and as a member of the public. I respect and admire her enthusiasm as well as her wisdom.

To everyone I offer my sincere thanks.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

Introduction

The fast-paced introduction of computer-based systems into the mainstream of society has ushered in a new area of concern: computer-related risks [1]. While the benefits of computers can be seen by the public on a daily basis, the associated dangers seem to have become obscured. A review of several past incidents exposes a problem of tantamount importance.

- The Denver International Airport was expected to be a model airport able to handle traffic even in the midst of poor weather conditions [2]. However, the grand opening was postponed for more than 1.5 years due to software errors in the automated baggage system costing the planners $1.1 million per day [2].
- In 1988, the California Department of Motor Vehicles undertook a project to update its primary database system. After spending $44 million of the taxpayer's money, the new computer program failed to work. Despite the exorbitant costs, the project was terminated [3].
- In January 1990, approximately 50 million of AT&T's long distance customers went without service as the company attempted to fix a software error in the switching program. Not only was AT&T faced with angry customers, but it also suffered a $58 million loss in the period of nine hours [4].
- In a lottery conducted by Pepsi, several "winners" were paid $19 instead of the $36,000 award promised by the contest [5]. The discrepancy was attributed to a computer error that produced 500,000+ winning tickets rather than the intended 18 tickets [5, 6]. Rather than suffer the $18 billion loss, Pepsi officials attempted to placate the winners by giving them the minimal award [6]. The error has been followed by several lawsuits, rioting, and anti-Pepsi rallies [5].

Though the above incidents proved to be costly ventures, they were not inconceivable events to the software development industry. When dealing with computer systems, people using the systems are prepared for, and maybe even expect, the systems to fail. Studies have shown that in the computer industry, two large-scale software systems are cancelled for every six that are put into operation [2]. It is not always safe to have technology use software as its backbone without taking proper precautionary steps.

Software developers will be the first to admit that current practices have no means to guarantee that all software-based systems will work correctly 100% of the time. Peter Neumann [1], an expert in the area of risk assessment, states that "guaranteed system behavior is impossible to achieve." There are always circumstances that cannot be predicted nor prevented. In addition, the human element is a major factor in determining risks because humans are imperfect and can introduce errors into several phases of the development process of such systems including the design phase, the implementation phase, the operation phase, etc. The development process has tolerated a range of discrepancies allowing for the construction of some flagrant and dangerous systems.

Do all the potential problems and obstacles mean that computer systems should be eliminated from society? Or should society be willing to accept the risks and problems that computer systems may cause? Clearly, the answer to both questions is no. Computers provide valuable services to both businesses and individuals. However, it is important to recognize the current limitations of our knowledge in building computer systems and to provide the necessary precautions to help avoid costly damages. While this task sounds simple, many have failed to satisfy such requirements.

We focus our dissertation specifically on the development of software. Our reasons stem from many factors. First, software can be very complex making it difficult to create, analyze, and examine software-based systems. The complexity continues to increase as technology grows in size and capability. Second, there are few rules and/or

guidelines for developing software. Because computer science is such a new field, it does not provide access to references containing formulas for software projects or to proven techniques for handling software. The lack of such fundamentals separates software development from many other disciplines. Third, software does not "wear out" so many systems may either run on outdated software created years ago or may contain reused software that has been forcefully melded with new code. Fourth, software is usually not a standalone application; most software systems contain some form of interaction with a user. Thus, those who create software must be concerned and aware of how the user is able to use the software in addition to having the software run correctly. Fifth, there is a common misperception that anyone can write computer programs. While there may be persons who have a limited understanding of a specific computer programming language, this knowledge does not mean that they have the expertise needed to responsibly develop reliable software. Sixth, regulations specifically addressing software-based systems are vague and sometimes nonexistent. This lack of rigor allows some systems to be used without proper review. Finally, society places great trust in software. There is little understanding that both good and bad software exist and that not all software-based systems should be trusted. These facets of software and software development emphasize the uniqueness of this field and why it needs to be explored.

Therefore, due to the huge demand placed on the computing industry and the perceived simplicity of developing software, it has been difficult to foster the guidelines needed for software development processes. Moreover, little has been done to monitor and regulate the use of software-based systems. Thus, some companies have been allowed to market faulty software-based systems containing potential dangers that have eventually exposed the public to great harm [1, 7, 8]. This dissertation specifically addresses the issues relating to the development process of safety-critical software systems. A safety-critical software system is defined to be any software-

embedded system that may contain hazards that can lead directly to some form of human loss.

> **Thesis Statement:** *The use of three complementary strategies, legislation, education, and a software advisory board, can be used used to encourage effectively and enforce responsibility and accountability in software development, thereby facilitating the improvement of software quality and the development process.*

1.1 Motivations

We feel that it is unacceptable to expose the public to harm. People use technology to improve systems. Therefore, when members of the public are harmed by the systems, sometimes fatally, we must question the safety of using such technology.

- While waiting to have an X-ray taken, one patient was crushed by a 3000 pound X-ray machine. The mistake was found to be a software error that allowed the machine to completely lower itself to the bottom of the post, overtaking the area where the patient lay [7].
- A computer-controlled set of hydraulic tongs caused injury to a worker when the automatic controller failed to turn off [8].
- Twenty-nine American soldiers were killed when a Patriot missile hit their barracks rather than the scud missile for which it was intended. The incident was attributed to a combination of software errors within the Patriot system and misuse of the system [1].

The above events may seem to be surreal and unimaginable. While the causes were not intentional, the events were perhaps preventable. Clearly, software developers and system designers do not attempt to create systems that cause harm. Still, it is often the case that the developers may not take all possible precautions to provide a "reasonable" level of care. Steps could be taken that would aid in the avoidance of such terrifying tragedies. Few will argue that we are developing new applications of high technology at a rapid rate. As more systems are integrated into society, the overall risk increases. Rather than continue to create faulty systems, we should

address the problems and begin to learn from our mistakes. As an initial step, we need to determine the source of the problem and then provide a practical solution.

Why are computer-based systems not always safe? As will be seen later, computer systems are like many other products created in the United States. People have the freedom to create systems that can accomplish great and worthy tasks, but people should be able to expect that when the systems are used, the systems will work as intended and that the user will remain free from harm. It is the responsibility of the developers of such systems to create safe systems and to keep unsafe systems from being used. Currently, it seems that this sense of duty has at times been ignored and overshadowed by the lure of monetary gain and temporary benefits. However, there appears to be a growing awareness among those who are involved with critical systems. For instance, Clark Turner, a lawyer and computer science doctoral student, is currently working on improving methods for testing systems [9]. He has also explored some of the legal aspects of various computer-related issues [10]. Furthermore, a few law schools now have courses and programs that are designed to address the issues relevant to computer systems and the law, including John Marshall Law School (Chicago, Illinois), Santa Clara Law School (Santa Clara, California), and Stanford Law School (Stanford, California). However, the examples presented thus far show that, unless more care is taken by the majority of the software industry in the creation and maintenance of the systems, some of the systems will eventually fail causing great loss, both financial and fatal. In order to minimize the loss, it is imperative that a sense of duty is maintained by both the manufacturers and developers of computer systems.

1.2 Contributions

The public places its trust upon developers and manufacturers to provide products that serve their intended purpose. The discussion of software system incidents clearly illustrates the need for change by describing injuries and loss of life caused by software. It is both frightening and frustrating to realize that many of the incidents could have been prevented had there been better mechanisms for ensuring the quality and the correct behavior of software-based products. When such systems go awry and cause harm, "someone" should be willing to address the all of the consequences, (i.e., financial, legal, moral), and remedy the situation before further damage is caused. This "someone" includes every member involved in the development process of a system.

The problem at hand is twofold. First, the software components in products sometimes lack the level of quality needed for safety-critical systems. We define quality to refer to the correctness of the system with respect to robustness. This deficiency is not because high quality software cannot be created, but simply that it is not always being created. The problem then points to the manufacturers and software developers. Why is poor quality software allowed to be used in some products? Why do some developers fail to provide quality software? Why do some developers fail to be accountable for their programs? This lack of responsibility and accountability need not be allowed. It is the ethical duty of each member in the software development process, from the manufacturer to the user, to minimize the risks in software in order to avoid the dangers [11].

Second, there is a distinct problem with the current state of regulation. Measures need to be taken to address, explicitly, software systems if they are going to be used by the public. Overall, a new sense of responsibility must be renewed within the software development community. "Accountability and the responsible practice of

computing, are social values worth sustaining and when necessary, rehabilitating [12]." Clearly, the current state of affairs necessitates rehabilitation. In order to produce safer software products, several complementary actions can be taken. The remainder of this dissertation will focus on these actions.

This dissertation will show that safety-critical software systems have at times been exempt from *normal* rules and policy applicable to consumer products that were put into place for the benefit of the public. (In the context of this dissertation, "normal" refers to what is expected of general consumer products and services.) Such exemptions do indeed favor the manufacturers of the systems but can potentially endanger the public. We will show that safety-critical software systems should not be subject to special circumstances but instead be held to normal levels of scrutiny. In the end, this change will benefit both the developer and the user.

This dissertation proposes three complementary areas that need to be addressed in order to improve the development of software, with the intent of increasing software quality: *legislation*, *education*, and an *advisory board*. The addition of new legal measures would emphasize the importance of public safety from the perspective of the courts and the public itself. The new legislation would provide further incentives for manufacturers to produce "high" quality software systems. In addition, an increase in safety can be attained by raising the quality of education of computer science graduates. Exposing future developers to their social and moral responsibilities would ultimately increase the quality of safety-critical software systems. Finally, the creation of the Software Advisory Board would help to control and oversee the new legislation, provide interaction between academia and industry, aid in the development of a uniform set of processes, and help to increase the quality of the skills of the software developer. With each approach there are actions that are proactive to the current technology and those that are reactive corresponding to responsibility and

accountability, respectively. The contributions are discussed in terms of these two areas of concern.

1.2.1 Responsibility

Some feel that employees should not be held responsible for their part in building a computer system because of reasons such as: *"the results were unintended," " too many persons were in contact with the project,"* or *"it would be difficult to determine the direct cause of the defect* [12]." Nevertheless, everyone who works on a system should be equally responsible for the respective contributions. For instance, programmers should produce code that adheres to the design specifications. Appropriate and extensive testing and other validation and verification methods should be applied to verify that the code is correct. In addition, managers should be held responsible for any independent developer's product that is created within their section of the organization. It is their job to give a "stamp of approval" to the product. Without responsibility, this approval is meaningless. When dealing with this issue, one must also recall that the intent of promoting responsibility is to ensure that, if a computer system is used by the public, it must maintain an acceptable level of "safety."

Our approach provides proactive measures to renew *responsibility* in each of the three areas. Augmenting legislation to include software provides preventative guidelines that manufacturers and developers can, and should follow. Improving the computer science education helps to equip the software developers of the future with knowledge of their technical and ethical responsibilities. Finally, the induction of the Software Advisory Board will provide industry and academia with a board of experts who have insight into software-related legal issues and the state of software development itself.

As stated previously, there are no current methods that will determine if a system will behave perfectly in every situation [1]. The responsible developer must take

the time and the effort to follow precautionary measures to help prevent dangerous situations. It is important to remember that the effects of responsibility are twofold. If a system behaves correctly, then the members of the development process can feel proud of their work and, if approved by management, be compensated for their accomplishment. However, should a problem arise, the developers should be willing to remedy the problem and appease those affected by the malfunction, in other words, they should be accountable.

1.2.2 Accountability

Accountability is a term that can be used within the software development process. In the context of computer systems, Helen Nissenbaum, the associate director of the Center for Human Values, provides a clear definition.

> Accountability means there will be someone, or several people, to answer not only for malfunctions in life-critical systems that cause or risk grave injuries and cause infrastructure and large monetary losses, but even for the malfunctions that cause individual losses of time, convenience, and contentment [12].

It is clear that the boundaries of accountability encompass various degrees and types of errors. Accountability allows for a reactive way to deal with problems. Although each member of the development team should be accountable for individual actions regardless of the malfunction, the consequences of the different malfunctions should be distributed. In an extreme comparison, compensation for a child who was permanently disfigured should be greater than that for a dented fender. The varying degrees of compensation is one aspect of accountability, but is usually referred to as *liability.* It is useful to differentiate between accountability and liability [12].

While each member of a production process should be 100% accountable for individual actions, liability can be distributed so that appropriate parties provide a fractional part of the compensation due to the victim. Therefore, liability can pro-

vide compensation for a bad situation [12], but is only part of accountability. Other steps must be taken to locate the source of the problem and to eradicate the fault(s). The combination of providing retribution for the victim and a solution to the problem is the form of accountability we wish to further promote in the computer system development community.

The three-pronged approach addresses *accountability* in each of the three areas. Liability precedents for safety-critical software systems and federal laws are proposed in order to call for legal action. In the alterations to education, emphasis is be placed on preventing, detecting, and remedying incorrect solutions to homework problems and projects. Students will also be exposed to incidents that have occurred in the past with the intent that the students will learn from the mistakes of others. Finally, we propose that the Software Advisory Board be given authority to hold accountable those who endanger the public by failing to comply to the official guidelines.

In hopes of raising the level of public safety, we have chosen an approach that promotes responsibility and accountability in both a proactive and reactive manner. This approach differs from other methods that use only reactive solutions. An example would be to only use costly legal suits to promote liability, but this approach is not feasible as we will discuss later. The combination of preventative measures with appropriate compensation can help to make developers create safer computer systems for all types of use.

1.3 Organization of Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, the scope of the dissertation will be defined in terms of safety-critical software systems. Chapter 3 examines the current procedure for dealing with critical incidents. Next we examine how the procedure handles incidents involving software systems which will expose a

need for software-specific policy and a better software development process. Finally, the three-pronged approach is explained proposing additions to legislation, alterations of computer science education, and the induction of the Software Advisory Board in Chapter 4, Chapter 5, and Chapter 6, respectively. Chapter 7 describes how three groups, industry, academia, and the legal profession can make use of the three-pronged approach. Conclusions and future investigations are discussed in Chapter 8. Also, a glossary of legal terms is given in Appendix A.

CHAPTER 2

Safety-Critical Software Systems

The need for higher quality computer systems spans almost all domains in which those systems are used. Of tantamount importance are those products that cause the greatest risk to the general public. The increasing number of incidents involving such systems calls for immediate action from within the computer science community. The primary focus of this dissertation will be on safety-critical software systems. We feel that this area is causing great risk to the public [10, 13, 14] and is not receiving the appropriate amount of attention. In order to better understand the nature of safety-critical software systems, we use this chapter to clarify what is meant by the terms "safety-critical" and "software systems."

2.1 Safety-Critical

As computers continue to be embedded in more systems, categorizing the role of the system in society becomes increasingly difficult. For certain systems, the greater the number of people interacting with the system, the greater the possible risks. However, we must ask, at what point does this risk become "safety-critical?" The answer is subject to interpretation and so the definition of safety-critical has become

ambiguous. The following will help to define the perspective taken throughout this work.

According to Webster's New World Dictionary [15], *safety* means "being safe," where *safe* is defined to be "free from damage, danger; secure; having escaped injury; unharmed." General safety can then be considered to be "the freedom from exposure to danger, or the exemption from hurt, injury or loss [16]." However, in the context of software, safety takes on a more precise definition to provide a means for measuring and rating degrees of safety.

Nancy Leveson, an expert in the field of safety analysis, defines safety as a state of minimal hazards [17]. Her definition makes an important distinction between hazards and accidents. She states that a *hazard* is "a set of conditions that can lead to an accident given certain environmental conditions [17]." This definition encompasses more than an *accident* which is "an unplanned event or series of events that leads to an unacceptable loss such as death, injury, illness, damage to or loss of equipment or property, or environmental harm [17]." The distinction shows the need to provide protection not only from undesirable events, but also the possibility of such events. The public should not have to worry about unexpected and undesired events, instead they should be confident that the systems will perform as expected.

For the context of this dissertation, we narrow the scope of all possible safety issues by limiting the definition of critical to only include those hazards that can potentially lead to some form of ***human loss.*** Human loss will then include direct bodily injury and death, excluding loss of property or income, though these situations may indirectly lead to human loss. The purpose is not to dismiss the severity of such other losses, but to restrict the scope of the dissertation as to provide a foundation for other research. These issues do need to be addressed within their own context.

2.2 Software Systems

Software alone cannot cause harm to anyone; it is simply the programs written for a computer [15]. However, embedded within a system, software controls machines that may have life-threatening functions. Thus, when we speak of *software systems*, we are referring to systems that are partially or completely controlled by software. The combination of software and a system constitutes a product that can be sold and marketed like any other product.

Therefore, *safety-critical software systems* are defined to be any software-embedded system that may contain hazards that can lead directly to some form of human loss. The use of safety-critical systems is not a novel concept, but, when combined with software systems, normal means of certification, regulation, and even investigation become obscured. It is this chaotic state that has allowed for incidents to occur jeopardizing the lives of many and taking the lives of some.

CHAPTER 3

Safety-Critical Incidents

In order to provide context for investigating 'incidents' that have involved safety-critical software systems, we first present 'incidents' that did not involve computer-related products. We refer to the situations as incidents rather than accidents. The reason is that accidents are perceived by some people to be events that occur by chance, where as incidents occur because something went wrong. We have characterized the following events as incidents because in all the cases, the cause was detected after investigation and in most cases, the incident could have been prevented. This discussion describes the normal course of action taken when each incident occurred. With this presentation, it should become more clear as to what changes need to be made in order to provide the public with adequate protection while using safety-critical software systems. This protection should be equal to the protection the public has when using non-computerized systems.

3.1 Incidents in Other Domains

Each critical product in this discussion falls under a specific domain that is overseen by a governing body: food and drugs, flight systems, and automobiles. These organizations work to keep products with potential dangers off the market. This protection

is maintained through specific minimum certification standards for the products, intense investigations when an incident occurs, and legal measures for those who fail to comply with the regulations.

3.1.1 Tylenol Tampering

In 1982, seven people died of cyanide poisoning after ingesting tainted Tylenol capsules [18]. As soon as paramedics determined the source of the poisoning, the Food and Drug Administration (FDA) took charge of the situation and began its attempt to protect others from being harmed. Tylenol capsules, a brand of over-the-counter drugs, were pulled from stores throughout Chicago. Though the Tylenol incident was not an accident, but a deliberate crime, the role and importance of the FDA becomes apparent, particularly given their quick response in handling the situation. The fast action and cooperation between McNeil Consumer Products Co. (the manufacturer of Tylenol) and the FDA saved lives. Their noteworthy efforts are still commended today [19].

Food and Drug Administration. The Food and Drug Administration played a large role in the Tylenol incident. The quick and responsible reaction to the first reports of the poisoning helped to minimize the number of fatalities. Before additional members of the public were able to purchase the contaminated product, the FDA removed the product from stores in the vicinity of the poisonings [18]. In addition, the FDA worked with the manufacturer to protect the general public by opting for a national recall of the product [18]. Meanwhile, investigators worked diligently to determine where the tampering had occurred, working first with the distribution chain and then investigating everyone from the manufacturer to the drugstores [18].

The FDA continued to work on the case even after the tainted Tylenol was contained and removed from the public market. The FDA went on to pass anti-tampering

laws that made tampering, false reporting of a tampering, and extortion by threatening to tamper, crimes punishable by fines and prison sentences [20]. Since 1982, several people have been convicted of the above crimes and punished [20]. The laws and punishments prescribed by the FDA have sent a strong message to the public on where the FDA stands with regard to consumer product tampering.

3.1.2 ATR Grounding

On Monday, October 31, 1994, all sixty-eight people on board an ATR-72 died as their plane crashed in Roselawn, Indiana [21]. According to aviation experts, the plane crash may have been attributed to abnormal ice formations on the wings [22]. The pilots were unaware of the formations because the flight was under autopilot rather than manual control. American Eagle Flight 4184 was the first crash of an ATR-72, according to the French-Italian company, Avions de Transport Regional (ATR) Marketing [23]. Immediately following the crash, members of the Federal Aviation Administration, the Secretary of Transportation, and the National Transportation Safety Board were available at the scene. Because the ATR-72 could hold up to 72 people, it was treated as and considered to be a large aircraft. Furthermore, it was subject to the same regulations as other aircraft such as Boeing's 747 and McDonnell Douglas' MD-11 [23].

Federal Aviation Administration. The Indiana crash spawned immediate responses from both the public and the Federal Aviation Administration (FAA). In order to minimize the possibility of future incidents, action was taken on November 9, 1994, that prohibited other ATR-72 aircraft from using autopilot while flying in icing conditions [22]. In addition, the FAA required airlines to reduce the number of planes being sent out in poor weather conditions. These measures were instituted to help protect the public while necessary testing was being performed. At this time,

studies were being conducted to analyze the development of ice formations on the aircraft's wings. However, the decision to allow planes to continue flying despite the incident was displeasing to a number of groups that work in the airline industry. The Air Line Pilots Association began advocating a ban on flights that used the ATR-72 and ATR-42 aircraft in icy conditions [24]. The association's request was overruled by individual airlines who found the requests "unfounded" [24] and by the FAA who felt the recommendation was unnecessary. To further advocate the association's concerns, some pilots worked to produce a leaflet explaining why they refused to fly the propeller planes in such conditions [24]. The intent was to warn the passengers of possible danger. Furthermore, American Eagle's flight attendant union submitted a request that the flights be grounded until the investigation into the Indiana crash was complete [24].

On December 10, 1994, after analyzing results from tunnel tests, the FAA banned both the ATR-72 and the ATR-42 from flying in weather conditions where there was the possibility of ice forming on the wings [25]. This order affected many companies that used the propeller planes built by ATR [25]. Airlines such as Continental Express and American Eagle were forced to cancel flights as they attempted to move the ATR-72s and the ATR-42s to warmer climates and bring in other aircraft to fly in the colder areas [26]. The affected airlines were not completely content with this decision because the banning order was apt to incur large costs in the figure of millions of dollars [26]. However, when weighted with the enormous "cost" of a crash, (i.e., passenger/crew fatalities, the loss of the plane, etc.), the order seemed justified. Both pilots and prospective passengers agreed. Since early December 1994, various pilot associations had been requesting such a ban. In addition, many passengers who were disrupted from their scheduled flights were not upset with the FAA. Though many were inconvenienced, most felt that a few hours of discomfort were well

worth an increase in travel safety. Others chose different modes of travel during the holidays [27].

Finally, on January 11, 1995, the FAA lifted the ban on the ATR-72 and the ATR-42. Extensive testing was done at Edwards Airforce Base that supported the claim that the cause of the crash was most likely attributed to ice forming on the wings [28]. The manufacturer's first proposal for correction focused on having the pilots look out the window for the ice formations. However, the second proposal was much more appealing to the FAA who, after much consideration, decided to lift the ban pending the following. First, pilots were required to watch for the ice formations [29]. Second, the planes were still banned from flying in freezing rain until new de-icing mechanisms were put into place and crews were retrained according to new FAA regulations [29]. Third, the aircraft was not allowed to use its flaps during certain icing conditions [30]. Fourth, the detection of freezing rain had to be improved, and planes were required to exit freezing rain areas [30]. Finally, the pilots had to keep the RPM of the propellers above a given rate [30]. The FAA expected the new guidelines to raise the level of ATR safety.

While many holiday travelers were delayed due to the restrictions on ATR flights, most feel that the "emergency airworthiness directive" initiated by the FAA helped to maintain safe travel during the holidays and worked to prevent unnecessary incidents [31].

3.1.3 Windstar Recall

In January 1994, the first wave of Ford Windstars were put into production. One year later, Ford Motor Company recalled all 1995 models that were built between January 31, 1994 and September 19, 1994 [32]. It had been found that "in all the affected vehicles a loose connection in the electrical power distribution box under the hood could cause excessive heating and ignite wire insulation and other nearby parts [32]."

In addition, approximately half of the minivans also had a possible hazard within a wire harness that, when pinched, could create an electrical fire [32]. Though none of the 13 reported fires had caused injuries, Ford chose to recall the models before injuries occurred [32].

Due to the fast action of Ford, no federal authorities were called to action.

3.2 Incidents Involving Software Systems

As seen in Section 3.1, specific measures were taken to help ensure public safety. The intent of this section is to expose the need for greater understanding of software systems in order to provide a higher level of safety for safety-critical software systems. The purpose of the discussion is not to criticize the existing governing bodies, but to give insight as to how the computer science community can aid in the governing of products involving software systems.

3.2.1 Therac-25 Incidents

In 1983, Atomic Energy of Canada, Ltd. (AECL) released the Therac-25, a dual-mode linear accelerator created to provide radiation treatment for cancer patients. Concisely stated, the following describes the functionality of the Therac-25 [10].

> Medical linear accelerators (linacs) accelerate electrons to create high-energy beams that can destroy tumors with minimal impact on the surrounding healthy tissue. Relatively shallow tissue is treated with the accelerated electrons; to reach deeper tissue, the electron beam is converted into X-ray photons.

The new dual mode Therac-25 was based on the Therac-20 and Therac-6 accelerators and included added enhancements to the software and the elimination of what was thought to be unnecessary hardware components, (e.g., protective circuits, mechanical interlocks) [10]. The software program regulated the timing mode (X-ray or

electron), strength, and duration of the dose, thus giving software more control of functions than the software used in the previous versions of the Therac [33].

Five Therac-25 machines were used in the United States and six were used in Canada [10]. Nancy Leveson and Clark Turner conducted an extensive investigation of six Therac-25 incidents [10]. By 1987, they found that six patients had been severely burned by the Therac-25 as detailed in Table 3.1.

No.	Date	Site of Incident	Injury
1	June 3, 1985	Kennestone Regional Oncology Center	Loss of breast/shoulder use
2	July 26, 1985	Ontario Cancer Foundation	Destruction of hip
3	December 1985	Yakima Valley Memorial Hospital	Tissue necrosis
4	March 21, 1986	East Texas Cancer Center	Fatality
5	April 11, 1986	East Texas Cancer Center	Fatality
6	January 17, 1987	Yakima Valley Memorial Hospital	Fatality

Table 3.1. Therac-25 Overdose Incidents

The following summarizes some of their findings.

Kennestone Regional Oncology Center. On June 3, 1985, a patient suffering from a malignant breast tumor was receiving treatment from the Therac-25 in Marietta, Georgia when she was given an overdose. It was calculated that she received a measure of 15,000 to 20,000-rads rather than the 200-rad prescription. The patient complained that she had been burned, but was informed that that was impossible [10]. Still, the patient suffered an immense amount of pain as explained in the following account [10]:

> ...She developed a reddening and swelling in the center of the treatment area. Her pain had increased to the point that her shoulder 'froze' and she experienced spasms. She was admitted to West Paces Ferry Hospital in Atlanta, but her oncologists continued to send her to Kennestone for Therac-25 treatments...About two weeks later, the physicist at Kennestone noticed that the patient had a matching reddening on her back as though a burn had gone through her body, and the swollen area had be-

> gun to slough off layers of skin. Her shoulder was immobile, and she was apparently in great pain... Eventually, the patient's breast had to be removed because of the radiation burns. She completely lost the use of her shoulder and her arm and was in constant pain.

There was no response to the incident from the center and AECL conducted no investigations after it was told of the incident. In addition, the FDA was not contacted and other users of the Therac-25 were left unnotified. The exact cause of the burn remains undetermined since the treatment prescription printout was not working at the time of the incident. Understandably, the patient filed a lawsuit that was eventually settled out of court.

Ontario Cancer Foundation. A similar incident occurred in Hamilton, Ontario, Canada on July 26, 1985. The patient was being treated for carcinoma of the cervix. This particular treatment was to be her twenty-fourth by the Therac-25 machine. On this day, 'normal' malfunctions occurred, (i.e., treatment suspensions and treatment pauses). This time the series of events resulted in the application of 13,000 to 17,000-rads, again, well exceeding the normal 200-rad dosage [10]. It has been noted that a dose of 500-rads can be fatal if applied to the entire body [10]. Understanding the patients trauma can help to clarify the serious nature of the overdose [10]:

> The patient complained of a burning sensation, described as an 'electric tingling shock' to the treatment area in her hip... The patient came back for further treatment on July 29 and complained of burning, hip pain, and excessive swelling in the region of the treatment... The patient was hospitalized for the condition on July 30... The patient died on November 3, 1985 of an extremely virulent cancer. An autopsy revealed the cause of death as the cancer, but it was noted that had she not died, a total hip replacement would have been necessary as a result of the radiation overexposure.

In this case, a few actions were taken. AECL conducted an investigation and decided that the problem was located in an "ambiguous position" message for the turntable. The turntable was controlled by a three-bit signal that rotated the table

to one of three positions: electron beam, X ray, or field light [10]. Each position allowed for different levels of treatment. An overdose could occur if the turntable was placed in the wrong position. With a three-bit signal, there are several signals that could designate more than one mode at a time, signals uninterpretable by the turntable. The code was altered to alleviate the problem and the software was revised to help provide further checks. AECL also opted for a voluntary recall in which it notified other users of the problem, yet did not mention that patient injury was involved. The FDA was informed of the problem and produced a set of modifications that had to be made in order for the Therac-25 to be officially approved by the FDA. AECL completed the first modification, but failed to comply to the rest of the FDA's directives.

After the patient returned on July 29, 1985 the Ontario Cancer Foundation Clinic ceased using the Therac-25. In addition, the clinic hired an independent consultant who found other problems with the system. A letter was sent to AECL outlining the measures that needed to be taken. AECL (apparently) disregarded the suggestions provided by the clinic and the independent investigator.

Yakima Valley Memorial Hospital. In December 1985, a patient went to Yakima Valley Memorial Hospital, located in Yakima, Washington, to receive a radiation treatment. Even after the incident, no one within AECL was willing to admit that the incident could have been caused by the Therac-25 [10].

> She [the patient] developed erythema (excessive reddening of the skin) in a parallel striped pattern at one port site (her right hip) after one of the treatments. Despite this, she continued to be treated by the Therac-25 because the cause of her reaction was not determined to be abnormal until January or February of 1986. On January 6, 1986, her treatments were completed.
>
> The staff monitored the skin reaction closely and attempted to find possible causes. The open slots in the blocking trays in the Therac-25 could have produced such a striped pattern, but by the time the skin reaction had been determined to be abnormal, the blocking trays had been

> discarded... When it was discovered that the woman slept with a heating pad, a possible explanation was offered on the basis of the parallel wires that deliver the heat in such pads. The staff x-rayed the heating pad and discovered that the wire pattern did not correspond to the erythema pattern on the patient's hip...
>
> The hospital staff eventually ascribed the skin/tissue problem to 'cause unknown'... About a year later... the staff investigated and found that the patient had a chronic skin ulcer, tissue necrosis (death) under the skin, and was in constant pain. This [condition] was surgically repaired, skin grafts were made, and the symptoms relieved. The patient is alive today, with minor disability and some scarring related to the overdose.

When contacted, AECL responded that the cause of incident could not be attributed to any Therac-25 malfunction [10]. The hospital was forced to conduct its own investigations as it was never informed of any other injuries caused by the Therac-25.

East Texas Cancer Center. Three months later in Tyler Texas, on March 21, 1986, another patient received an overdose from the Therac-25. The patient was scheduled for a treatment of 180 rads on a 10 × 17-cm section of his upperback [10]. Due to some errors in the operator's input, the machine malfunctioned and delivered an incorrect dosage. Because the video display and audio monitor were not on, the patient was unable to warn the operator of the problem. Thus, a second dose of radiation was given. It was estimated that the patient received approximately 16,500 to 25,000-rads over a 1 cm area in approximately 1 second, clearly causing much damage [10]. The following excerpt provides a more complete account of the overdose [10]:

> After the first attempt to treat him, the patient said that he felt like he had received an electric shock or that someone had poured hot coffee on his back: He felt a thump and heat and heard a buzzing sound from the equipment. Since this was his ninth treatment, he knew that this was not normal. He began to get up from the treatment table to go for help. It was at this moment that the operator hit the 'P' key to proceed with the treatment. The patient said that he felt like his arm was being shocked

> by electricity and that his hand was leaving his body. He went to the treatment room door and pounded on it. The operator was shocked and immediately opened the door for him. He [the patient] appeared shaken and upset.
>
> During the weeks following the accident, the patient continued to have pain in his neck and shoulder. He lost the function of his left arm and had periodic bouts of nausea and vomiting. He was eventually hospitalized for radiation-induced myelitis of the cervical cord causing paralysis of his left arm and both legs, left vocal cord paralysis (which left him unable to speak), neurogenic bowel and bladder, and paralysis of the left diaphragm. He also had a lesion on his left lung and recurrent herpes simplex skin infections. He died from complications of the overdose five months after the accident [original overdose].

The center's physicist temporarily stopped the center from using the Therac-25 after the incident and notified AECL. Representatives from AECL visited the center to examine the machine, but still claimed that the Therac-25 could not give an overdose [10]. The AECL representatives suggested that the cause may have been from electrical shock and allowed the center to continue to give treatments with the Therac-25.

East Texas Cancer Center. Three weeks later, another disaster occurred. On April 11, 1986, a patient received over 4,000-rads of treatment to a small 7 × 10-cm section of his face. The dosage was intended to treat his skin cancer, but instead led to the patient's death [10]. After the treatment, the patient knew that something had gone wrong as the following reveals [10]:

> The patient began to remove the tape that had held his head in position and said something was wrong. She [the operator] asked him what he felt, and he replied 'fire' on the side of his face... the patient explained that something had hit him on the side of the face, he saw a flash of light, and he heard a sizzling sound reminiscent of frying eggs. He was very agitated and asked, 'What happened to me, what happened to me?'
>
> The patient died from the overdose on May 1, 1986, three weeks after the accident. He had disorientation that progressed to coma, fever to 104 degrees Fahrenheit, and neurological damage. Autopsy showed an acute high-dose radiation injury to the right temporal lobe of the brain and the brain stem.

The center's physicist again stepped in and kept the center from using the Therac-25 until the problems had been eradicated. He and the operator performed several tests to determine the series of events that caused the overdose. It was determined that the operator had made a mistake entering the treatment data and used the UP key to make the correction [10]. This action seemed to have triggered the overdose. AECL was notified, but could not reproduce the overdose after performing several tests. East Texas Cancer Center's physicist had to explain to AECL how he produced the error and then AECL was finally able to reproduce the overdose.

AECL notified the FDA, but the FDA had already begun an investigation after the first East Texas Cancer Center incident in which it found the Therac-25 defective and unsuited for use. The FDA then required AECL to notify all users of the Therac-25 that it was defective and to provide the FDA with a corrective action plan (CAP). Until a final plan was devised, AECL recommended that the centers use a "temporary" remedy that involved removing the key cap from the UP key or placing electrical tape over the key to prevent further use [10]. This recommendation was found to be unsatisfactory by the FDA, and on May 2, 1986, the Therac-25 was declared defective [10]. The users began to form a coalition where information was passed between the various centers. After much discussion, it was found that many centers had encountered similar problems with the Therac-25. In fact, some had felt it necessary to add extra safety features to their machines. Several anomalies about the machine were revealed. In addition to the known overdoses, the users estimated that 10 to 30 percent of the cases resulted in underdosing [10]. From June 1986 to January 26, 1987, the FDA and AECL went through several series of revisions to devise a CAP in an effort to fix the problems of the Therac-25.

Yakima Valley Memorial Hospital. One week before the final CAP was submitted, another patient received an overdose. On January 17, 1987, the patient went

in for treatment for carcinoma. The intended dosage was a total of 86-rads. The following is an account of the actual treatment [10]:

> He [the operator] went into the room to speak with the patient, who reported 'feeling a burning sensation' in the chest... Later in the day, the patient developed a skin burn over the entire treatment area. Four days later, the redness took on the striped pattern matching the slots in the blocking tray. The striped pattern was similar to the burn a year earlier at this hospital that had been attributed to 'cause unknown'...
>
> The patient died in April from complications related to the overdose. He had been suffering from a terminal form of cancer prior to the radiation overdose, but survivors initiated lawsuits alleging that he died sooner than he would have and endured unnecessary pain and suffering due to the overdose. The suit was settled out of court.

The cause of the incident was attributed to race conditions in the software. * Though AECL proposed temporary solutions to the problem, the FDA felt that the software was an inadequate form of sole protection and that hardware backups needed to be installed for added protection. Finally, on February 10, 1987, the FDA told AECL that the Therac-25 was officially defective and unable to be used in the United States until further notice [10]. Canadian officials also recommended that users refrain from using the Therac-25.

Discussion. On July 32, 1987, 25 months after the first massive overdose, AECL sent a final notice to users outlining 34 modifications that were to be implemented on the Therac-25 in order to make the machine safe to use again [10]. Several questions were raised concerning the Therac-25: Why did the centers and hospitals continue to use the Therac-25 while their patients were being injured? How and why did all of the constant malfunctions of the machine go unnoticed? Why was AECL allowed to submit incomplete schedules of modifications without any recourse? Most

*Race Conditions occur when two or more processes read from or write to the same data at the same time causing the result to be dependent upon the order that the data is accessed [34].

importantly, why were patients allowed to die before proper attention was given to the Therac-25?

While we may never be able to provide answers to these questions, we can address some of the software-related problems that led to the injuries. The development of the Therac-25 was far from ideal. By observing several of the deficiencies uncovered in the investigation, it is easy to see how such malfunctions occurred. First, there was an unreliable reuse of software from the Therac-6 and the Therac-20 [10]. Some of the old programs were used in the Therac-25 under the incorrect assumption that they would perform the same in the new machine. Second, the design itself was overly complex. This complexity made it difficult to evaluate how the system performed. Third, some of the concurrent programming was invalid [10]. Race conditions and lack of mutual exclusion were some of the errors later revealed. Fourth, the importance of the user interface was neglected. Operators of the Therac-25 had become accustomed to frequent interruptions and unexplained error messages. This lead to the next problem, the documentation was very poor. Sixth, there was an unverified reliance upon the software for safety. Testing was limited and there was no validation or verification performed. Regardless, the software was expected to replace hardware as the primary safety control unit. Finally, regulation applicable to the Therac-25 was unable to handle the situation. It is apparent that several deficiencies in many of the development stages of the Therac-25 contributed to the malfunctions. Though we are not able to reverse the Therac-25 situations, we can work to find solutions to the problems uncovered in the investigation and prevent such future incidents.

3.2.2 Airbus Incidents

Airbus Industrie, the manufacturer of the Airbus aircraft, maintains the philosophy that an automated cockpit is more reliable than a cockpit controlled by a pilot [35]. Thus, the more automation there is, the better. Several past incidents, noted in Ta-

ble 3.2, have brought attention to Airbus Industrie and have caused some to question its philosophy. This European aviation company is the producer of the Airbus A300, A310, A320, A330, and A340 models, all of which are partially or completely controlled by automatic flight control systems. Of these models, the A320 was the first *fly-by-wire* system used in civil transport aircraft [36, 37]. † The A320's fly-by-wire control system works as follows. The commands from the pilots to the mechanisms of the plane are sent digitally. Data is collected from both the pilots and on-board sensors. The system will "compensate automatically" for certain common disturbances. Finally, the system has internal limits that govern different variable values [36].

The A310's control system comprises two independent software systems provided for safety reasons [38].

No.	Date	Aircraft	Site	Fatalities
1	June 26, 1988	A320	Mulhouse Habsheim	3
2	February 14, 1990	A320	Bangalore	94
3	February 11, 1991	A310	Moscow	0
4	September 1991	A300	Katmandu, Nepal	167
5	January 20, 1992	A320-100	Strasbourg, France	87
6	April 26, 1994	A300-600R	Nagoya, Japan	263
7	June 30, 1994	A330	Toulouse, France	7
8	September 19, 1994	A340	Heathrow	0
9	September 24, 1994	A310-300	Paris-Orly	0

Table 3.2. Air Industrie Incidents Involving the Airbus

Incident 1. On June 26, 1988, three passengers on board an Air France Airbus A320 were killed after their aircraft crashed into a group of trees near the Mulhouse Habsheim runway [39]. The crash occurred during a flyby maneuver performed for

† A fly-by-wire system is fully controlled by an automatic flight control system (AFCS).

an airshow. Flying at a very low altitude and attempting to make the "maximum attack angle" proved to be a fatal mistake. Minutes before the crash, the pilot disengaged the autothrottle planning to take full manual control of the aircraft as a precaution. (It was later determined to be an unnecessary action [40].) After passing the control tower, the pilot put the throttle in the maximum thrust position. This action conflicted with the program that expected a "gradual" increase of power [40]. The pilot then put the aircraft in the full nose-up position [40]. By the time the engines were fully powered, the aircraft had already collided with trees causing two engines to shutdown [40].

An extensive investigation was conducted in an attempt to determine the cause of the incident. This investigation included various tests and intense research into the controls that governed the system [36]. Investigators came to the conclusion that the incident was caused by a number of factors including: the flyover height was too low, the speed was too slow, the engine was at flight idle, and the go-around power[‡] was not activated early enough [40].

It may also be noted that the investigation uncovered several inadequacies with the aircraft. Though none of these attributed to the incident, they were fixed in other aircraft [40].

Incident 2. On February 14, 1990, 94 of the 146 members aboard an Indian Airlines Airbus A320 were killed as the aircraft crashed near the Bangalore Airport upon landing [42]. The series of events that led to the crash began with the improper selection of the 'idle open descent mode.' In addition, the engines were set to idle allowing the plane to descend at a very low speed. Finally, the automatic flight control system put the plane into go-around mode fully activating the idle engines. At this

[‡]The Takeoff/Go-Around (TOGA) mode is used during landings where the aircraft must attempt another landing and takeoff. In this mode, full power is supplied to the aircraft [41].

time, the plane was descending too rapidly and the pilots were unable to switch the other controls to ascend and try to land again.

Since this incident was the second Indian Airlines A320 crash, the owners of the aircraft, Indian Airlines, immediately grounded all Airbus A320s and refused to accept any of the other A320s that were to be delivered [42]. Airbus Industrie was not pleased by this decision. Managers asserted that the A320 and its systems were not to blame for the crash, and claimed that the aircraft's design and equipment provided no basis for the grounding [42]. While no direct link was found between the crash and the actual aircraft, the question of why the automatic pilot went into the go-around mode remained unanswered.

Incident 3. On February 11, 1991, the crew of an Interflug Airbus A310 temporarily lost control near Moscow after changing the flight modes [43]. Due to runway problems, the aircraft had to delay landing. The pilots activated the go-around mode sending the plane into automatic control. However, the control tower requested that a specific altitude be maintained, one much lower than that used by the go-around mode. When the pilot attempted to disengage the mode and return to manual control, the automated controls took over causing the plane to climb at a steep angle. Fortunately, the automatic controls were deactivated and no one was injured.

Incident 4. In September 1991, 167 people died in Katmandu, Nepal as a Pakistani A300 crashed while attempting to land [44]. Investigations determined that the autopilot was never completely disengaged, it was only temporarily deactivated pending reactivation. During the same year, a bulletin was released by Airbus Industrie warning pilots of the dangers associated with overriding the autopilot mechanism [45]. The bulletin went on to describe a sample scenario in which the autopilot system could conflict with the pilots' maneuvering and could cause a crash.

Incident 5. The following year on January 20, an Air Inter Airbus crashed killing 87 of the 96 people on board. The incident, involving a "fly-by-wire" A320-100, occurred near the Strasbourg-Entzheim Airport located in France. Upon approach, the plane was set in the "incorrect" descent mode and crashed into a mountain located only 8 nautical miles from the airport [46]. After an extensive investigation by the French authorities, the Delegation Generale pour l'Armement (DGA), and other organizations, the exact cause of the crash was left as undetermined. However, "complex human factor issues" were noted as potential causes. Pierre-Henri Gourgeon, a member of DGA, noted that Air Inter did not equip its aircraft with the Ground Proximity Warning System (GPWS) [47]. The airline's managers felt that the GPWS generated too many false alarms and could be replaced by another air traffic control system [46]. In contrast, other sources state that when used on the A320, the GPWS was highly reliable [47]. At first, it was also found that failure of the Flight Control Unit (FCU) may have played a minor role in the incident. After further analysis, a hypothetical situation was presented to DGA exploring the possibility that confused crew members may have set the FCU in the incorrect mode [48].

In response to the investigation reports, Airbus Industrie began to redevelop the FCU as a precautionary method. The problem with the flight mode was attributed to pilot error. Yet, some French pilots found that the system design was also at fault. The Airbus A320 was created to run automatically, so many manual operations were not integrated well into the design. For instance, the FCU was created with little attention to human factor issues and seemed open for error in cases where the pilot had manual control. In support of this claim, it was noted that aircraft, including the A320, required a significant amount of reprogramming in order to accommodate changing conditions during flight [49]. For example, one pilot refused to make a landing approach alteration after descending below 3000 feet because it would have required his copilot to make approximately 12 reprogramming steps without any

assistance or guidance [49]. What may have been attributed to pilot error in the Strasbourg incident, may have actually been caused by poor software design, a design approved by the manufacturer.

Incident 6. Approximately two years later, another incident was reported in Nagoya, Japan involving a China Airlines (CAL) Airbus A300 enroute from Taipei, Taiwan. On April 26, 1994, 263 of the 271 members onboard the A300-600R aircraft died as their plane crashed upon landing [50]. The Accident Investigation Committee (AIC), a part of Japan's Ministry of Transport attributed the cause of the incident to conflicts between the autopilot system and the pilots' actions. That is, the crew was supposed to choose between allowing the plane to remain in automatic pilot or to fly the plane manually. Instead, the pilots attempted a mixture of both approaches [51]. According to the flight transcript, the take off/go-around (TOGA) switch was accidentally engaged. Crew members made attempts to disengage the switch, but during the confusion, it was disengaged and then re-activated. At this point, the autopilot system took over forcing a phantom takeoff during an actual landing then sending the plane into a fatal ascent [52]. Many sources blamed only the pilots. However, others felt that there were major problems with the Airbus FCU [14].

The sequence of events that led to this incident closely parallels those mentioned in the 1991 bulletin [45]. The existence of such a bulletin showed that Airbus Industrie was aware of possible problems with the autopilot system. In addition, there had been at least six other incidents involving different models of the Airbus A300 before the Nagoya incident [44]. Airbus Industrie's reactions were questioned concerning possible lack of action. This was the seventh Airbus, though the first A300-600, to crash in the last 10 years. Airbus Industrie has made no changes in the aircraft design or in the manufacturing process [51]. The manufacturer of the aircraft (Airbus

Industrie) was not the only party questioned after the crash, the owners of the aircraft (CAL) were also subject to several investigations.

CAL had been the primary transportation for all major Chinese officials. Following the crash, the director of Taiwan's Civil Aeronautics Administration (CAA) and several top executives at CAL resigned taking responsibility for the incident [14]. Upon further investigation, it was believed that CAL had sacrificed improved safety for profits. For example, the former chairperson of CAL made a decision against the purchase of new aircraft and equipment because they were too costly [14]. Among the needed equipment were simulators for flight training. At the time, some training was done during actual commercial flights [14]. There have been charges that the CAL board of directors was primarily run by senior military officers who had little understanding of the airline industry. New management at CAL has recognized the need for a balance between maintaining safety as well as the company's standing. Improvements, such as bringing in international consultants and hiring competent board members, have put CAL back on track towards maintaining a safe airline. Had the autopilot issue been addressed earlier by any of these parties, (i.e., Airbus Industrie, CAL, CAA), the Nagoya tragedy may have been prevented.

Incident 7. On June 30, 1994, Airbus Industrie was performing tests on the autopilot system of an A330 in Toulouse, France, when during one of the test flights the aircraft crashed immediately after takeoff [53]. All four members of the flight crew and three airline representatives were killed. The flight crew was attempting to test the autopilot mechanism in the event of engine failure, a requirement for the Category 3 Certification Test. The Delegation Generale pour l'Armement (DGA), conducted investigations and determined that the loss of control was caused by contradictory requirements from the pilots and the autopilot system [41, 54]. However, the conflict between the autopilot system and the pilot's instructions could have been resolved

had the pilots acted sooner. Some have questioned the overreliance upon the autopilot system by crew members. Regardless of whether or not the pilots could have prevented the crash, the question still remains: What was wrong with the autopilot system? One account of the incident states the following [54]:

- The autopilot sent the aircraft into a mode that conflicted with the test flight plans ending the test.
- Pitch altitude was increased while the speed decreased bringing the aircraft to a speed below that of the minimum control speed.
- The crew disengaged the autopilot.

The investigative report stated that the crash could have been prevented had the autopilot system been turned off four seconds earlier [41]. If the autopilot system had been performing correctly, the pilots would not have needed to disengage the autopilot system at all. Thus, in addition to improving pilot and crew reactions, it was equally, if not more important, to have reviewed the autopilot system and made the necessary corrections. Later, it was found that the crash was partially caused by incomplete logic in the automated system [55]. There was a lack of protection for the pitch attitude in one of the autopilot modes. Airbus Industrie was able to provide a temporary fix for the logic problem and began implementing recommendations by the DGA immediately after the preliminary investigation [41].

Incident 8. On September 19, 1994, an Airbus A340, owned by Virgin Airlines, was forced into an emergency landing when critical data disappeared from the screen of the cockpit [56]. The aircraft was able to land safely in manual mode adhering to instructions from the control tower. Though no one was injured, the incident caught the attention of many in the software development community. Among several problems, it was found that some causes of the incident were partially attributed to software and hardware errors in the Flight Management Guidance System (FMGS) and the fuel management system. This was said to be "the first time that an accident report

on an A320/330/340 series aircraft specifically cites software and hardware reliability as the main problem [57]." Airbus Industrie had encountered similar problems with the FMGS on the A320 and was prepared to research the problem as to provide a solution [57].

Incident 9. On September 24, 1994, a Tarom owned A310-300 aircraft lost temporary control near the Paris-Orly Airport when the FCU entered "level change" mode without the pilot's knowledge [58]. This change caused the plane to begin a rapid ascent at a very sharp angle contradicting the pilot's manual controls. The plane then went into a steep dive continuing until the aircraft was only 800 feet above ground [38]. Though no one was killed, and the incident did not result in an crash, members of the Flight Safety Foundation Icarus Committee (a committee consisting of safety aviation experts from the United States and various European countries) were alarmed [59]. The recognition of this hazard showed that they were working to prevent the occurrence of future incidents. Committee members have found that there are approximately 360 incidents for every major commercial crash that occurs, and that if each incident had been thoroughly investigated, many of the problems would have been detected [59].

Discussion. Upon review of the numerous incidents involving the different Airbus aircraft, it can be ascertained that proper action was not always taken to ensure flight safety. Measures were taken to address the situations caused by the problems (i.e., blaming the pilots) rather than to address the problems themselves (i.e., the autopilot system). Two main inadequacies in the software development process surfaced throughout the various investigations. First, it was discovered that some of the control logic governing the autopilot system was not correct. Why the manufacturers failed to take the corrective action to fix the automated flight system is unknown.

Pe
dic
sio
sys
ens
des
hav
dres
mee
were
may
is of
syste
famil
altere
needs
deem

3.2.3

In the
occurr
tion ra
The ob
critical
and in t
involvi
incident

Perhaps the system was "too complex" or the remedy too expensive. Maybe officials did not have proper knowledge of computer systems needed to make corrective decisions. Regardless, the Airbus aircraft were at least partially controlled by software systems. And as with other types of systems, it is the duty of the manufacturer to ensure the public's safety by locating and fixing any known problems. Second, the design allowed for conflicting modes. Though the problem with the design may not have been caused by a programming error, it is an issue that could have been addressed in the design stage. In addition, the pilot-computer interface did not always meet the pilots' needs, thus bringing human factor issues to light [35]. The pilots were unable to differentiate between the various modes. Also, parts of the system may have been too complex for the pilots. Even if the system performs correctly, it is of equal importance that users of the system are able to understand and use the system properly. To help remedy the human factor issues, the pilots need to become familiar with the new automated cockpits and the automated cockpits need to be altered to better accommodate the pilots. Little action was taken to address these needs. The obvious question is how many incidents have to occur before changes are deemed necessary and implemented?

3.2.3 Anti-lock Brake Systems

In the previous incidents involving the Therac-25 and the Airbus, several fatalities occurred, bringing much attention to the incidents. However, we hope to have attention raised before such incidents actually occur, thus reducing the number of fatalities. The objective should be to prevent incidents by focusing on hazards within safety-critical software systems. This approach would assist in the prevention of incidents and in the end prevent loss of and injury to human life. The following set of incidents involving the braking systems have no reported fatalities. Still, the high number of incidents warrants investigation in the hope that no fatality will occur.

The automotive industry has recently introduced new braking systems to the public. The Anti-lock Brake System (ABS) was designed to help prevent incidents in situations where the wheels lock up due to excessive braking. By providing an automatic pumping action, wheels remain unlocked, thus giving the driver more time to take control of the vehicle [60]. The potential widespread use of ABS in new passenger vehicles was enabled by the significant reduction in brake costs. The primary technology used in ABS is as follows [61]:

> The most important component of the ECU [electronic control unit], and the entire ABS system, is the MPU [microprocessor unit]. A single-chip electronic microprocessor or microcontroller device, the MPU serves as the central processing unit (CPU) for all ABS control and monitoring functions. [§]

Detailed information regarding details of the ABS system can be found in [61, 62, 63]. Additional testing data can also be found in [64].

One main concern is that there are several different anti-lock brake systems [65]. Each manufacturer may have their own ABS model [66]. In addition to all-wheel systems and rear-wheel only systems, there are also different ABS systems for passenger vehicles, truck tractors, trailers, and light trucks, to name a few. Thus, it has been stated by one carmaker official that there needs to be some type of rating system so that members of the public will know if the ABS on their automobile is safe [60]. Apparently, all ABS systems are not behaving as predicted. The National Highway Traffic Safety Administration (NHTSA) has received:

- 146 complaints, including four accident reports, and zero injury reports for failing ABS on 1988-93 models of Buick Regal, Pontiac Grand Prix, Chevrolet Lumina, and Oldsmobile Cutlass Supreme [67]
- 156 complaints, including 57 accident reports, and 22 injuries for ABS on light trucks made by General Motors [68]

[§]The MPU receives data from wheel sensors, performs calculations to determine which wheel, if any, requires braking action, and then sends the appropriate signal to the modulator of braking pressure.

- 142 complaints, including five accident reports, and two injuries for ABS on 1991-93 Chrysler Corporation minivans [69].

In each of these cases, NHTSA has conducted preliminary evaluations and begun investigating the reports.

The primary hypothesis about the brakes is that the vehicle owners do not understand how to react when the automated system is activated. In order for ABS to work correctly, the driver must apply intense direct pressure to the brakes [70]. The problem is that many drivers do not expect the automatic pumping so they take their foot off the brake causing the automated braking to disengage [13]. Another problem is the lack of sufficient pressure to the brake [70]. In other instances, drivers attempt to manually pump the brake causing the system to again fail [13]. The inability to correctly use the anti-lock brakes has caused inaccuracies in the expected decline in collisions. These problems, perhaps, explain why the large investment of $1.7 billion dollars in ABS technology has not aided in the reduction of collisions [66].

There have been proposals for fixing ABS systems to make them more usable. One approach, taken by Lucas Industries Inc., adds a sensor to their existing ABS system. This addition hopes to better reflect the reactions of the driver during critical situations. By calculating the speed of the braking, the automatic brake pulsing will be applied quickly if the pedal is depressed fast, and the automatic brake will be applied more slowly if the pedal is depressed in a more 'progressive' movement [70]. Another suggestion has been to improve the education of drivers of automobiles that have ABS. In addition to distributing videos and literature that describe how the system will react in locking situations, some companies use simulators at the dealerships to train and educate new car owners [71]. Though these fixes seem to provide safer automobiles and safer drivers, the conflict between the system and the driver is not the only problem with ABS. Thus, even with the remedies discussed above, there are still problems that cause the systems to perform unexpectedly.

National Highway Traffic Safety Administration. The reduction of death and injury on highways has become a primary concern of the administration [72]. Thus, if the National Highway Traffic Safety Administration (NHTSA) decides to mandate the use of ABS on all passenger vehicles and light trucks, NHTSA must also make sure that the addition of ABS reduces the number of injuries and fatalities. NHTSA currently attempts to control such incidents by thoroughly examining and testing the various anti-lock brake systems. However, much of the testing is done by individual manufacturers [66]. The NHTSA must also remain updated on possible additions to ABS. Two such additions have already been proposed: the addition of sensors to detect braking pressure [70] and also the incorporation of traction control with the existing ABS [60]. Once new features are added, testing needs to be performed on the new system as well as on the individual features. The responsibility of overseeing the regulation of ABS is not a small task, but an important one for automobile drivers across the nation. About twenty years ago, heavy trucks were required to have ABS. However, many of the systems were defective and prone to failure causing the mandate to be revoked [73]. It is clear that ABS systems must be carefully reviewed before legislature is passed to mandate their use.

NHTSA comprises 660 professionals whose responsibility is to help lower the number of injuries and fatalities across national highways [72]. However, there has been some controversy associated with NHTSA. Some feel that despite its efforts there is still an estimated 20 million defective vehicles on the road [74]. The procedure of NHTSA is as follows. Most complaints come from the public via free hotline numbers [74]. If NHTSA obtains consent from the informant to notify the manufacturer, a report is forwarded to the manufacturer. When a problem arises, a voluntary recall is suggested. If the manufacturer declines to recall the vehicle then a mandatory recall is declared. The law states that manufacturers are required to provide free corrections of safety problems if the car is less than nine years old [74]. In most

cases, the manufacturers cooperate, but in some cases, conflicts arise. The method seems to be working, but some find the timeliness of the system to be slow. They state that this method allows hazardous cars to be driven many miles before they are repaired [74]. Also, many automobiles never receive the correction and continue to be driven in their unsafe state.

Since Dr. Ricardo Martinez recently assumed the role of Administrator of NHTSA, there has been a renewed vow to prevent injury [72]. The ABS incidents have thus far involved no fatalities. The effectiveness of the new administration will potentially be measured by the handling of the ABS brakes and its ability to prevent injury and the loss of life.

Discussion. Because the use of ABS is new, a full understanding of the situation is not available. However, studies have already revealed some of the software-related problems with ABS. First, the users seem to be unable to correctly use ABS. The addition of new technology is useless if people are unable to use the new technology. Second, testing is the primary means of certification. The computer science field has already determined that testing alone is not necessarily the most reliable nor cost effective way to remove defects from a system [75]. This leads to the final problem, the manufacturers of ABS are then unable to verify the level of safety. There are multiple types of ABS: each manufacturer can have its own type of ABS, there is all-wheel and rear-only ABS, and various vehicles have to have different ABS. What types of validation and verification are performed on each type of ABS? How do consumers know if the ABS in their vehicle is safe?

3.3 Problems with Software Development

After reviewing the examples involving software systems, several problems relevant to the software development process have surfaced. First, unacceptable software is being created. Though we may not be able to prove that software is 100% correct, there are some techniques that can be used to help aid in the programming stages. It is apparent that these techniques are not always being applied. Second, many deficiencies in the development stage were encountered. Explicit design specifications, validation, and verification techniques were not always used. Third, there was a lack of understanding between the users of the system and the designers of the system. Thus, the users were unable to use the systems properly. The final problem encompasses all of the above. In each of the incidents, new technology was added without proper analysis. Just because new technology can be added to a system, that alone does not make the system better. These problems are of utmost importance especially when found within safety-critical software systems. These problems can largely be remedied by current technology, thus providing safer systems. The difficulty is providing those parties involved in the development and maintenance with the incentives and resources needed to use preventative and proactive means to develop high-quality systems. The three-fold solution calls for changes to legislation and education in addition to the induction of an advisory board.

CHAPTER 4

Legislation

The current legal system attempts to keep products on the market safe. However, the existing legal structure does not provide the needed "technology-specific" guidelines [76]. In order to maintain the expected level of safety and allow the use of embedded software within products, two main issues need to be addressed. First, standard liability claims need to include products containing software systems so that when a defective product causes damage, a product liability claim can be filed to compensate the victim and punish the party at fault. Second, it is also important that regulation boards governing the approval of products change their policies to reflect the impact of the increased use of embedded software.

4.1 Liability

Black's Law Dictionary defines product liability as: "The legal liability of manufacturers and sellers to compensate buyers, users, and even bystanders, for damages or injuries suffered because of defects in goods purchased [77]." *Because critical systems are usually considered to be 'goods,' product liability applies to the systems discussed in the previous chapters. One lawyer, who has researched the legal aspects of software

*A glossary of legal terms is found in Appendix A.

in business, finds that safety-critical software systems may be prime candidates for product liability claims [78]:

> Software programs with the greatest vulnerability to products liability claims would include those used to operate devices that could harm people if they malfunctioned, such as airplane guidance systems, dosage control circuits for X-ray machines and triggers for an automobile's braking system.

In order to better understand how safety-critical systems should be treated in liability claims, guidelines of strict liability are briefly overviewed.

4.1.1 Strict Liability

The law provides a means of compensation for victims who have been injured by a defective 'good' under the premise of strict liability. The defect can be attributed to both the design and manufacturing stages of the product [79]. There are four main facets for strict liability that separate it from other claims, three of which are directly relevant to safety-critical software systems [80].

1. Disclaimers common to many other claims are not permissible.
2. Economic loss is not sufficient, injury to a person or a person's property must be shown.
3. Negligence is not an issue; the manufacturer is held at fault.

By holding the manufacturer "liable without fault," manufacturers can then be allowed to place potentially dangerous products on the market by providing the public with the assurance that safety will be maintained. If, for any reason, that safety is compromised, a remedy must be provided. This approach encourages the manufacturer to be responsible for its product by making the product as reliable as possible as to avoid strict liability claims. The law was established for situations where it is impossible for the manufacturer to guarantee that each product is 100% safe. The

manufacturer is expected to do the best job possible and is held accountable when an incident occurs.

> Concept of strict liability in tort is founded on the premise that when the manufacturer presents his goods to the public for sale, he represents [claims] they are suitable for their intended use, and to invoke such doctrine is essential to prove the product was defective when placed in the stream of commerce [77].

While the definition of strict liability seems to be explicit, this is not always true. Each application of strict liability is case specific and the ruling is sometimes dependent upon the court. One primary concern is that the judicial system will not handle cases involving safety-critical software systems in the same manner as cases not involving embedded software. This differentiation means that there could be cases where software systems are held exempt from strict liability claims simply because they contain software. This difference in treatment should clearly not be allowed. When software is embedded within a system, it becomes a part of the entire system. Even if the software component of a system is created by a contracted party, then the manufacturer should still be held liable for placing the product on the market. There have been cases where the strict liability ruling held for a software-controlled system, but when dealing with software-related products, the claims seem to become more complex. The following cases help to illustrate the point.

Lewis v. Timco v. Joy. In Lewis v. Timco, Inc., the defendant held that a defective set of computer controlled tongs caused the defendant serious injury [81]. † Mr. Alfred Lewis was caught in a "snub line" when the tongs failed to disengage after the throttle was released. The tongs did not disengage because of a design defect. However, Mr. Lewis was also found guilty of negligence in that he did not adjust the "snub line" as necessary [81]. Though the case has gone through the court of

†The tongs are hydraulically controlled and are used to disjoin tubing joints. Explicit controls are set to allow the tongs to work properly. [81].

appeals, each trial has found that the notion of strict liability was applicable [8, 81]. Still, there has been some controversy in the application of *comparative negligence.* Though several parties were involved in the chain of actions that led to the use of the tongs, the first verdict absolved some of the parties of any liability and reduced the amount of the claim, placing part of the blame on Mr. Lewis' own negligence. However, in the appeal, the court chose to follow the premise that, "where a machine design presents open and obvious dangers, the law holds the human operator to a reasonable standard only [82]." Thus, Mr. Lewis' negligence was absolved. The court also overturned the verdict involving the "sharing of responsibility" among some of the members. The court felt that [81]:

> If strict products liability seeks to impose enterprise responsibility upon the entity exposing others to the risks of its defective products... then the facts of the instant case dramatize the inappropriateness of applying comparative negligence principles. We therefore decline to apply the doctrine of comparative negligence in this strict products liability action in admiralty.

In the case notes, the court did acknowledge that some courts may have upheld the use of comparative negligence as did the first court. In addition, the court felt that perhaps in other cases, comparative negligence may apply. Comparative negligence varies from one state to another such that in some states, it is never applied while in others it is applied when appropriate [8]. This case illustrates the case-specific and court-specific applicability of strict liability and comparative negligence. The use of comparative negligence in strict liability cases could have profound effects on software developers. If the courts are going to disperse accountability to various parties involved in the case, then there is the possibility that the software developers may also be held accountable.

Aetna v. Jeppesen. In a more controversial case, Aetna Casualty and Surety Co. v. Jeppesen & Co., the courts found Jeppesen strictly liable for manufacturing

defective instrumental landing charts. In November 1964, all passengers aboard a Bonanza aircraft were killed as the plane crashed on approach to land in Las Vegas, Nevada [83]. Aetna attempted to recover the costs paid in the wrongful death suits for the deceased passengers. Jeppesen felt that they were not to blame and that the airline company was partially at fault. The court found that the crash was caused by defective instrument approach charts created by Jeppesen. The discrepancy involving the charts was that the two graphical views depicted in the charts were drawn to different scales. Aetna felt that, "it [Jeppesen] had failed in its design goal of graphically representing this information in a readily understandable way [84]." Even though there was numerical indicators on the charts that cited the differing scales, the crew took it for granted that they would be drawn to the same scale.

In the first trial, the court found Jeppesen to be primarily at fault and held that the crew members were not found to be negligent. Thus, the verdict was that Jeppesen was 80% at fault and Bonanza was 20% at fault [83]. Bonanza should have detected the defect and passed the information onto the pilots.

In the appeals trial, the court was willing to apply comparative negligence as explained: "a defendant remains strictly liable for injuries caused by a defective product, but plaintiff's recovery is reduced to the extent that its lack of reasonable care contributed to the injury [83]." The court felt that the crew was negligent in misinterpreting the charts and did not find the previous verdict acceptable. Instead, the court felt that the negligent party should be more liable than the strict liability party [83]. The previous rulings on the case were annulled and the damages were to be reassigned [83].

The case sparked controversy outside the realm of Aetna, Jeppesen, and Bonanza. Many felt that because the charts were computer-generated [84], the charts were simply output from a program and not an actual item to be sold. Thus, any application of strict liability should have been dismissed. The courts determined that since the

charts were technical tools, strict liability was applicable. This verdict upset many software developing companies [78]. They felt that the technical charts behaved more like a book. Had the data been treated as information in a book, Jeppesen would not have been held at fault. Though the courts and the companies disagreed on the verdict, one positive result from the ruling occurred, the software industry recognized the reality of product liability claims for software-related products. It is clear that anyone who plays a role in the development of a product may be held liable if the product causes harm [78]. In addition, the role of the user may come to play a larger role in the courts placing greater importance on correct use of systems.

Discussion. Current cases are beginning to set precedents for future computer-related strict liability claims. When dealing with safety-critical software systems, the applicability of strict liability does not appear to be overly controversial. The greatest controversy comes from those who believe that computers are too complex in nature, thus making the determination of the defect difficult and liability inapplicable. Such controversy is understandable, but it must address the appropriate problem. If it is found that strict liability should not be applicable, then the solution should be to find or create a new set of liability rules rather than eliminate liability for those who manufacture and sell computer-related products. Some feel, however, that currently the best option available is that of strict liability [80]:

> If society has made the irrevocable decision to use computers, as it surely has, and computers are influencing our lives more and more each day, society must come to grips with the issue of who will pay for the damage than an errant computer can cause. Strict liability seems to be a rational and just theory to apply in many instances.

Interestingly, this perspective is shared also by members of the computer science field. Helen Nissenbaum, whose research also involves issues dealing with computer ethics, gives the following observation [12]:

> Software seems, therefore, to be *precisely* the type of artifact for which strict liability is necessary—assuring compensation for victims, and sending an emphatic message to producers of software to take *extra*ordinary care to produce safe and reliable systems.

The effects of liability claims upon the software development community are great. If it is known that safety-critical software systems are going to be held to the laws of strict liability, then it will also be recognized that any party involved in the production of such a product may be held proportionately liable for any defects. This view will gain widespread recognition by software developers, software companies, and the court system, with appropriate changes to current laws.

4.1.2 Limits of Liability

There is a need for additional legislation beyond that which is discussed in product liability claim courts. The question may arise as to why there is a need to address issues besides liability. The following reasons should help to alleviate such wonder: court/case specificity, liability changes, ill-defined computer law, and lack of responsibility.

Court/Case Specificity. First, the applicability of liability fluctuates and cannot provide needed stability. For instance, liability is determined in a court of law. Such litigation is costly with respect to both time and money. For computer-related cases, many courts (including judges, lawyers, and jurors) are unfamiliar with the domain and thus are not able to always provide a well-informed verdict. This insufficiency is due in part to the failure of the laws to provide adequate guidelines for addressing computer-related cases. Also, with rising court costs, it is not always feasible for the victim to sue the manufacturer who is responsible for the defective product.

Liability Changes. Second, the rules encompassing liability are constantly changing. The House Republicans are now attempting to move the governing of Tort Law from the state to the federal level [85]. The proposed changes include:

- Having the court cost be paid by the loser of the lawsuit [85].
- Making it more difficult to sue product sellers [85].
- Placing limits on product liability claims [86].
- Creating maximum amounts for punitive damage claims [86].

In summary, the bill would make it more difficult to bring claims against many manufacturers. One particular part of the bill will release drug and medical companies from any liability as long as they fulfill the requirements provided by the Food and Drug Administration (FDA) [85]. This legislation assumes that the FDA is able to provide adequate certification criteria. An illustrative example of the applicability of this bill is the fact that had this bill been in place, women harmed by silicone breast implants and Dalkon Shield IUDs would not have been able to sue the makers of the defective products [85]. It is true that such changes will lower the number of cases brought to court, but the changes will not necessarily raise the level of the quality of products. Instead, some of those at fault may be absolved of liability. The goals of some of the parties backing the bill seem to be contrary to those who support another bill that affects the car industry.

Another set of cases that will affect liability claims is being sent to the high courts. Families of tractor-trailer rigs want to hold the manufacturers liable for unsafe vehicles *including* those that meet the safety standards enforced by the Department of Transportation (DoT) [87]. Automakers are usually absolved of any responsibility once it is shown that the vehicle in question complied with the safety standards. The DoT and families of crash victims are attempting to eliminate this defense under the premise that the DoT's standards only provide *minimum* guidelines. The government's position is as follows [87]: "The government contends that federal safety standards are

minimum standards and that suits should be permitted against manufacturers who don't go beyond the minimum standard." An example of the applicability of this bill would be that automobile manufacturers who comply to minimum standards but do not provide airbags would be held liable in court cases under the premise that they sold unsafe vehicles [87].

The stances on the FDA cases and the DoT cases seem to be in opposition. This contradiction simply supports our claim that the laws of liability are unstable. If either of these bills become law, then the entire structure of liability claims will be redefined. Even if neither of the propositions are passed, the existence of such claims reinforces the premise that we cannot rely upon liability alone to ensure public safety.

Ill-defined Computer Law. Third, Computer Law is not a precisely defined law as of this date. Many of the fundamental principles of existing laws can be applied to computer-related cases, but computer law as a whole is ill-defined. The fact that many computer-related issues do not fit exactly into the existing structure further complicates the problem at hand. The court cases discussed previously attempted to apply current product liability law to computer-related cases. In addition, there have been other computer-related suits. However, there still are no exact guidelines for dealing with computer-related liability claims.

> Legal precedents relating to situations in which computer systems were involved are in a state of flux. Computer technology is changing rapidly; on the other hand, the law changes slowly. Thus, despite the approaching maturity of computers, the new legal questions which they present have as yet been only superficially addressed... There are certain unique characteristics of computers upon which traditional contract law has not yet been brought to bear. A comprehension of these characteristics is essential to the development of contract law rules for computer technology that can provide the connectedness and dependability which society expects the law to provide [79].

Clearly, it is up to both the legal community and computer scientists to provide a distinct set of laws that can govern society that include software guidelines. As of

this date, no set of laws exists. Until they are adopted, we must depend upon other methods to promote safety.

Lack of Responsibility. Finally, liability does not promote a sense of responsibility. While it holds the defendant liable for certain damages, liability is divided so that each defendant is assigned a percentage of responsibility. The existence of a trial allows the defendant to attempt to absolve himself/herself of any responsibility and reduce the degree that he/she is accountable to the plaintiff. This mode of action runs contrary to the desired environment where the manufacturer is responsible and willing to be accountable for its actions. We feel that such qualities are essential for our domain. Maintaining a certain level of responsibility and accountability can help to prevent incidents such as those presented in this dissertation, highlighting the overall preventative approach towards maintaining safer systems.

4.2 Software Systems Policy

Product liability deals with defective products that were already placed on the market. This section addresses proactive measures that can be taken to prevent defective products from being placed on the market. Several regulation agencies currently develop strict guidelines for certifying products. However, as seen previously, the purpose of these guidelines can be disputed. Some feel that they should be treated as minimum standards while others feel that they provide adequate testing criteria. The problem with safety-critical software systems is that many of the guidelines do not include clauses for addressing software issues. Though we know that software errors within a system have the ability to cause fatalities, many times the software is released for use without undergoing a rigorous review process, thus leaving open the possibility for the public to use defective and potentially harmful products.

Should a product fall through loopholes simply because it contains software? Clearly the answer is no. Therefore, the regulations that guide various certification standards need to be updated to include concise software requirements. Revisions to current policy need to work to prevent hazards not just incidents. Possibilities for software-related policy additions include the need for: testing criteria, documentation standards, design specifications, etc. Examples of such include those defined for general software engineering courses, process models for the Space Shuttle, and in those in support of the Capability Maturity Model [88, 89, 90]. Had such additions been in place, the Therac-25 accidents may have been prevented.

Some may argue that software is too complex and cannot be adequately reviewed. This complexity does not mean that software should not be subject to regulation guidelines, but that in certain cases it may be appropriate to refrain from using certain software programs until they can be shown to meet safety standards. Should we allow the use of software in systems though the software may have a high degree of complexity and is difficult to test? If the possible hazards are non-threatening, (e.g., computer game software), the answer to this question may be yes. However, in cases where the application is safety-critical, it clearly is not acceptable to use software systems that have not been thoroughly tested and examined. While this requirement may mean that manufacturers may sometimes have to revert to more reliable products that may not be state-of-the-art, the overall increase in safety provides justification for their use. Some call for "simpler design" as a means of promoting safer systems [12]. A well known quip seems to capture the intent, "It is better to be safe than sorry." This added precaution parallels the procedures that govern the FDA. Before a drug is placed on the market, the FDA conducts several tests and examinations. Thus, even though a drug may seem to be producing favorable results, the product will not be available to the public until those results can be tested and confirmed. Why should software be treated less stringently?

Thus, the solution does not call for a completely new set of laws, but for an augmentation of the existing certification laws that govern critical systems to include software criteria. In addition, we should add to current recall regulations so that they include software safety checks and make software inspection part of the investigation process. For any critical system that does not have domain-specific certification/ recall laws associated with it, certain software specific guidelines should be created. Finally, we must make sure that software systems are not found to be exempt from standard liability laws. Software systems should be treated no differently than any other product on the market. The fact that software is embedded within a critical system should not overshadow the safety-critical nature of the system itself.

Example. John Rushby, an expert in the area of Formal Methods for critical systems, has presented a method that could be used to help define the level of formalism required to meet specific certification guidelines. He provides a way to categorize critical systems into four Levels of Rigor defined as follows [91]:

Level 0: No use of formal methods.

Level 1: Use of concepts and notation from discrete mathematics.

Level 2: Use of formalized specification languages with some mechanized support tools.

Level 3: Use of fully formal specification languages with comprehensive support environments, including mechanized theorem proving or proof checking.

One suggestion would be to require that software within safety-critical systems be subject to Level 3 Rigor. Requiring such formalism would give the developers a way to examine the requirements, specifications, and designs [91]. Properties such as mutual exclusion, correctness, liveness, absence of deadlock, lack of starvation, and others could be proven to exist in the specification. Though expecting software developers to adhere to Level 3 Rigor for critical systems may by costly and the level

of complexity may have to be reduced, this requirement would provide the needed basis for providing safer software systems.

CHAPTER 5

Education

It is emphasized that the purpose of expanding the current legislation is not an attempt to increase the number of litigations per year. Instead, the intent is to define a rigid set of guidelines to be followed and a clear set of consequences for those who fail to follow the guidelines. This change would make the responsibilities of the manufacturers more concise. Manufacturers will be able to understand what is expected of them and then pass on what will be expected of the contributors to their products, (i.e., software developers). Once these revisions are made, industry must be made aware of the changes. This dissertation proposes that in addition to notifying industry it is equally important to make the guidelines known in academia. We must recall that the students of today are the system developers of the future. If we expect change to occur, both the field and computer science education must change [7].

5.1 Need for Change

The need for change within computer science education, including greater and repeated emphasis on responsibility and accountability, is motivated by the increasing demands placed on the computing industry discussed in previous chapters. The de-

mand, in addition to the desire, for high quality software is rising. However, upon review of the current state of computer science education, many of the fundamental ethical issues are not being taught. In order for the educational system to meet the needs of the growing industry, educators must reevaluate their current programs and make the necessary changes.

5.1.1 Software Demands are Increasing

As we move into the Information Age, it is clear that the demands placed upon software are rapidly increasing. The examples presented in Chapter 3 illustrate the increased use of software within critical systems ranging from airplanes and automobiles to medical machines. Not only is more software being used, but software itself has become increasingly complex. This increase in complexity has raised several issues. First, the development and the maintenance of complex software has become increasingly more difficult as the complexity rises. In addition, it is clear that fixing faulty software is not a trivial task [92]. Also, as the processing power of computers increases, the demand for software that makes use of this power is increasing. This creates a bottleneck because the processing power of the computer is stunted by a lack of software that is equally as powerful [93]. This problem motivates the need for more research in the area of software design and the need for better software developers.

5.1.2 Current Education System is not Adequate

The current state of Computer Science education has received criticism from various members of the computer science community. The following comments represent a sampling of the nature of the complaints from members of computer science education and industry.

- Many of the courses were not distinctly formed, but created as a conglomeration of several courses attempting to cover too much material [94].

- There is little or no emphasis on making corrections and giving feedback for incorrect programs and/or design [94].
- Large programs are built with little or no use of system analysis [94].
- Theory and practice have diverged [94].
- Industry finds that most academic research is 'basic and theoretical—usually not a good return on investment [95].'
- Students are taught to create new programs failing to utilize previous results [93].
- Poor understanding of algorithmic concepts [96].
- Lack of necessary mathematical foundations [96].
- Software engineers do not possess adequate ethical and technical training [96].

In addition to educating the students poorly, there is also sentiment that computer science programs do not offer a complete understanding of Computer Science. Several professionals have noted that Computer Science requires not only technical skills, but also societal understanding [7, 11].

One author likened Computer Science to Architecture by showing that it is important for a designer to understand the structural limitations of the science and also the purpose for each project [7]. The scenario involved an architect that was required to build a door [7]. In addition to knowing the technical knowledge needed to keep a door, sturdy, the architect also needed to know who would be using the door, what type of door it would be, and what type of access should be given to users of the door (e.g., if locks were needed). If the architect did not take into consideration these social aspects, then a door with no locks could be used as a front door to a bank and a sliding glass door could be used as an entry to a bathroom.

Clearly, computer science deals with many social aspects of today, but very little emphasis is placed on societal understanding in the educational process. This lack of understanding means that we are potentially providing students with a curriculum that does not include all aspects of the field. The purpose of discussing these issues

is to uncover a strategy to help improve the quality of computer science graduates. While all computer-related courses should be reviewed, the most direct approach is to begin at the undergraduate university level.

5.2 Role of University in Change

Several factors help account for the current state of Computer Science (CS) education. First, Computer Science itself is a new field. The first "computers" were developed in the late 1940s and early 1950s [80]. Compared to other disciplines, such as mathematics, that have existed for centuries, Computer Science cannot be expected to be perfect or well-understood. Second, Computer Science departments were formed very early relative to the emergence of Computer Science as a discipline [94], perhaps before any substantial definition of Computer Science had been established. The rush to establish Computer Science departments as the field was still in its infancy may explain the diversity of CS programs nationwide. However, this diversity is not only because CS is a young field, but also because CS technology is rapidly changing [79]. Quantitative growth is not bad, but it must be accompanied by qualitative growth. CS departments need to mature as new ideas emerge. Mary Shaw, a professor of Computer Science and a scientist at the Software Engineering Institute, outlines the primary shifts of programming any-which-way to programming-in-the-small to programming-in-the-large in CS research during the last three decades [93]. As more and more is discovered about CS, it is important that academic departments learn about the changes so that they can be taught to students. This propagation of new technology has not always been the case.

David Parnas, a well-known researcher of CS, states that, "A university's primary responsibilities are to its students and society at large [94]." This responsibility implies that it is the duty of the university to remain aware of changes within

the research domain and the current state of industry. Such responsibility can be maintained through a two-fold approach that restructures the current courses and maintains a bridge between academia and industry.

5.3 Course Restructuring

Before we begin, we should note that each CS department across the nation is different. However, each department should provide a group of core classes that present the fundamentals of the field. We examine the current computer science curriculum to expose two necessary changes: the insertion of a new course into the beginning of the course schedule and the integration of social values and formal methods into existing courses. To illustrate how the changes can be implemented, we look at a sample CS undergraduate program and propose a new set of course descriptions. Though the sample program is not certified by the Computing Sciences Accreditation Board (CSAB), explanation of how the proposed changes would affect the CSAB required courses is given in Section 5.5.

5.3.1 Adding a New Course

We believe that problem solving is the basis of Computer Science where the purpose is to use computers to solve problems. This characterization implies that one must have a thorough understanding of both the problem, its domain, and the computing resources needed to provide the best solution. Students need to be able to understand the problem, create a clear representation of the problem, devise a precise solution, and then implement the solution. While much emphasis is given to the implementation stage, very little attention is paid to the early stages. The *ad hoc* method of implementing a solution before appropriate analysis is performed can have adverse effects on the solution. These effects may not be apparent in classroom problems, but

will be encountered when the student enters industry and attempts to develop large and complex systems without fully understanding the problem.

The first computer science course usually focuses on teaching the students a programming language. This approach is applied at both the secondary education level (high school) and at the undergraduate education level. We feel that the first computer science course should be a problem solving course emphasizing discussion of various issues such as:

- different interpretations of problems
- problem decomposition
- several solutions to problems
- analysis of problem domains
- the ambiguities encountered when solving problems
- the role of mathematics: logic, probability, discrete
- appreciation of the differences between languages, operating environments, and machines available for use
- the ramifications of incorrect solutions
- how errors should be fixed

This type of course would provide a better understanding of problems that can be used and referenced in all of the other courses throughout the program. Many computer science students complain about having to take math courses stating that they are not necessary, but indeed they are. The above proposed course would provide the motivation for taking discrete mathematics, computer architecture, etc. exposing them to different perspectives of Computer Science. Whether the student becomes a programmer or a technical manager, this understanding is vital to producing quality computer science graduates. Thus, we propose that we insert a new course into the very beginning of the CS program.

In addition to the above, teaching this course would foster the maturity of development techniques discussed by Mary Shaw. Recall, she stated that the type of research done today is different than that done thirty years ago. The most current research focuses on programming-in-the-large and deals with issues such as those listed in Table 5.1 [93].

Attribute	Programming-in-the-Large
Characteristic problems	Interfaces, management system structures
Data issues	Long-lived databases, symbolic as well as numeric
Control issues	Program assemblies execute continually
Specification issues	Systems with complex specifications
State space	Large, structured state space
Management focus	Team efforts, system lifetime maintenance
Tools, methods	Environments, integrated tools, documents

Table 5.1. Research Focus: 1975-1985

The problem solving course would not provide the students with specific techniques for solving all types of problems. Instead, it is intended to give them general problem solving skills and a better understanding of issues that they must deal with when solving problems. Then, when other courses are taken dealing with more specific techniques, a general understanding would already be in place.

5.3.2 Changes to Existing Courses

In order to help Computer Science students understand the fundamental responsibility that they have to society, the notion of responsibility and accountability must be emphasized throughout the entire program. Such concepts are not separate from any area, nor are they excluded from any part of professionalism. One method is the integrative approach which attempts to incorporate specific aspects of Computer

Science into all courses [7]. Two primary components must be integrated: social needs and formal methods.

Social Needs. Emphasis can be placed on the fact that the systems built in industry will be used by the public and that some systems have the potential to cause harm. This connection is rarely made. In the classroom setting, if a program gives an incorrect value, then the grade is adjusted accordingly. The ramifications of an erroneous program are minimized. However, when a program written for a critical system produces an incorrect value, the product may cause harm and, in turn, lead to extreme consequences. The effects that the students, as future system developers, will have on society will be great, thus giving them a large responsibility with respect to public safety. Understanding the social aspect of Computer Science is vital to promoting responsibility. "Computer technologies are a medium for an intensely social activity; and that system design—though technical as an enterprise—involves social activity and shapes the social infrastructure of the larger society [7]." This responsibility is important in all specializations of Computer Science, and it can be easily seen that the work of software developers can directly affect the public. Programs have been written that control major communication systems, space craft tests, combat control systems, etc. [11]. Some define software development itself to be social: "The software process, from creation to purchase, to use to effect, is a social process involving human participants who want and need different goods in the process [11]." Once it is understood by the students that safety-critical software systems can have a great impact on society, it should follow that the software developers need to take all available precautions to make the systems safe. Thus, for software that is used in critical applications, the need for higher quality software, precise design specifications, and formal methods is made clear.

Formal Methods. Developing software has become more than simply writing programs that produce a certain output. At one time, the computer was thought to be a simple black box that mysteriously and almost magically produced a given output [80]. This myth has been replaced by a science that studies the methods and the actions that control the "black box." Issues such as the form of the output, the needs of the user, and the correctness of the program all become important. There are many situations where previous methods of programming are no longer acceptable. Consider the following scenarios:

- The user needs a program to perform some function, but the actual needs are defined ambiguously.
- The system is very complex and cannot be tested as a whole (i.e., space shuttle).
- The system is old and needs to be updated.

All of these scenarios are common problems in the industry and all could benefit from the use of formal methods. A *formal method* is characterized by a formal specification language and a set of rules governing the manipulation of expressions in that language [97]. With respect to the application of formal methods for software systems, John Rushby, a longtime researcher in the formal methods community offers this definition [91]:

> ...the use of mathematical techniques in the design and analysis of computer hardware and software ... [that] allow properties of a computer system to be predicted from a mathematical model of the system by a process akin to calculation.

Using formal specification languages facilitates the early evaluation of a software design and verification of its implementation through the use of formal reasoning techniques [91, 97, 98, 99]. A formal specification can be rigorously manipulated to allow the designer to assess the consistency, completeness, and robustness of a design before it is implemented. Each step in the development process can be supported by

mathematical proof, thus minimizing the number of errors due to misinterpretation and ambiguity. This type of approach is supported by many methodologies [98, 100, 101, 102].

Though some may find formal methods useful, we feel that they are necessary for providing the public with safe systems. Because of their potential dangers, safety-critical software systems should require a high level of formalism. Some may argue that formal methods are too difficult to teach in the classroom. However, if formal methods can be introduced in the beginning problem solving course, then formal techniques can be used in each of the following computer science courses with the use of formal methods increasing throughout the student's education. This approach will equip the student with a thorough understanding of formal methods one that can be applied in industry.

5.4 Technology Transfer

The other area of computer science education that is lacking is that which deals with technology transfer. In order for the educational system to be effective, there must be a link between academia and industry. Several suggestions have been proposed that expose the students to real-life problems, yet at a minimized level. We propose that every computer science student enroll in a practicum course, participate in a co-op program, or be involved in joint research.

5.4.1 Practicum

In an attempt to link both the social and technical aspects of Computer Science, some have proposed the use of practicum courses within the curriculum [7]. The practicum course usually involves organizing the class into small groups and having them work on real projects in various domains. The projects could be fulfilling needs

of the university or perhaps engaging in small contracts with industry. No matter who the contact is, the students are exposed to issues such as dealing with the customer, understanding the customer's needs, project planning, including working with deadlines, complying with monetary constraints, and researching the product's environment. The primary disadvantage to having practicums is that the project can be time consuming for both the students and the professor. One must also realize that the practicum class alone will not address all of the social issues [7].

5.4.2 Co-op Program

To aid in the transition from student to employee, educators, students, and employees consider co-op and internship programs to be valuable experiences [94]. The students are able to benefit from the experience of working on a real project and from dealing with employers and the employer's needs. Most students can receive credits for co-op programs while also earning a salary. While this may seem trivial to some, receiving monetary compensation for good work is motivating. This program must be run with direct contact between sponsoring faculty and project supervisors in order to provide a quality experience [94].

5.4.3 Joint Research

Although it may not be feasible for undergraduate programs, some have proposed more industry-sponsored academic research [95]. The research conducted would be funded by private industry and there would be two liaisons: one from academia and one from industry [95]. Both industry and academia would benefit from the research. First, industry would be able to utilize the resources of academia, have first-hand knowledge of state-of-the-art research, and be able to obtain useful results from the research. In addition, students would be exposed to current problems found by

industry, help define the leading edge, and gain experience working with professionals. The research would be a synthesis of coursework applied to industrial applications.

5.5 Sample Course Requirements

This section reviews the existing undergraduate computer science education requirements for Michigan State University (MSU) [103]. As stated previously, though MSU is not accredited by CSAB, it does contain courses that correspond to the primary computer course requirements. CSAB accreditation guidelines require, among many other criteria, that the curriculum contain a set of core computer courses covering the following areas: theoretical foundations of computer science, algorithms, data structures, software design, the concepts of programming languages and computer elements, and architecture [104]. Thus, any references to MSU courses can also be applied to CSAB curriculum.

We propose modifications to the program and to specific courses that support the education of students based on ideas mentioned in the previous sections. Emphasis is given to promote the use of formal methods as well as the integration of social values.

Core Courses
1) Problem Solving
2) Algorithms & Computing
3) Discrete Structures in Computer Science
4) Computer Organization & Assembly Language Programming
5) Data Structures & Programming Concepts
6) Automata & Formal Language Theory
7) Operating Systems
8) Computer Architecture
9) Compiler Construction
10) Design/Synthesis/Capstone Course

Table 5.2. Required Computer Science Courses

5.5.1 Core Courses

We have taken the computer science major requirements and added the problem solving course in order to create the revised course requirements. We have included a Project Course that will allow the student to gain industry experience *. The results are in Table 5.2. The following augments the course descriptions to include social responsibility and formal methods.

Problem Solving: This course examines the process of problem solving: understanding the problem, expressing the problem, deriving a solution, expressing a solution, and verification of the solution. This course includes the presentation of various techniques for addressing each of the steps and the ability to choose the appropriate technique. Discussion of factors that affect problem solving with respect to computer science are also included: computer system hierarchies, languages, probability, statistics, etc.

Algorithms & Computing: †Introduction to different approaches to implementing algorithms, the fundamental logic needed to describe such algorithms, and a specific programming language to implement the algorithms. Beginning programming concepts, (i.e., text input/output, selection, repetition, arrays, functions and procedures, scope of variables, records, sequential files [105]), and programming techniques, (i.e., searching, sorting, randomness, and recursion), should be taught. In addition to teaching the technical aspects of basic programming principles, it is important to incorporate the practical nature of using such techniques to solve problems into the lecture. For example, in one lesson about counting loops, a professor asked the class to write a program according to the specification: "If the heart stops beating

*This courses replaces the choice between the two courses: Design of the Intelligent Systems and Software Tools for Concurrent Systems.

†Corresponds to the CSAB course focusing on algorithms.

for 10 seconds, the pacemaker's program should send a signal to give the heart one electrical shock [7]." The common solution was to implement a loop going from one to ten. The problem is that the machine cycles were much shorter than one second. As such, most of the pacemakers were programmed to shock the patient about every 0.000001 seconds, electrocuting the patient [7]. This example presents the type of problems that occur when discussing programming concepts, machine specifics, and social impacts.

Discrete Structures in Computer Science: [‡] Discrete Structures is the first pure theory course required for computer science majors. It addresses several of the fundamentals of formal methods including mathematical logic, set theory, induction, etc. This course gives the tools needed to be able to use formal methods. To help the students understand the importance of the course, explanations of how the principles relate to real-world problems are necessary. It is quite beneficial if students can appreciate the need for theory early within their coursework.

Data Structures and Programming Concepts: [§]Focusing on various data types and structures and the construction and analysis of algorithms, formal methods could be incorporated by having the assignments be specified using a formal specification language. The subject matter should refer to practical situations that help illustrate the use of the material and guidelines for determining what method to use. Feedback should be given as to whether or not the students' assignments were correct as well as to provide measures to correct the assignments.

Automata and Formal Language Theory: This is the second formal theory course that the students encounter. In addition to learning about languages and gram-

[‡]Corresponds to the CSAB course focusing on theoretical foundations of computer science.
[§]Corresponds to the CSAB course focusing on data structures.

mars, emphasis should be placed on why various types of languages and paradigms are used. When introducing Turing computability and undecidability, practical problems containing similar properties should also be discussed. For instance, the student may be able to visualize the Traveling Salesman Problem and then be able to understand its complexity.

Computer Organization & Assembly Language Programming/Operating Systems/Computer Architecture: ¶Each of these courses deals with machine specific properties. In addition to the regular material covered in these courses, it is important that the students understand the relationship between the machine and the application levels. This could include discussion of how machine-dependent factors affect the entire system. We have already discussed previously an example that illustrates how clock cycles affect programming concepts. Other discussions could focus on the semantic gap that exists between the machine's core concepts and high-level language concepts [106]. Attention could be given to the discrepancies that such a gap can create.

Design/Synthesis/Capstone Course: A real-life project course should be taken that takes an existing problem and goes through the entire cycle of producing a working solution. Students should be able to get credit for this course by participating in a practicum course, a co-op program, or joint research. This gives the student flexibility to find a project relevant to the student's area of interest. Regardless of the project course, the students should be able to use formal methods and problems solving skills to utilize the knowledge gained in the previous courses. It would be beneficial to have this course be one of the final courses taken.

¶Corresponds to the CSAB course focusing on architecture.

Optional Courses
1) Software Engineering
2) Computer Networks
3) Artificial Intelligence & Symbolic Programming
4) Translation of Programming Languages
5) Organization of Programming Languages
6) Computer Graphics
7) Vector & Parallel Programming
8) Database Systems

Table 5.3. Optional Computer Science Courses

5.5.2 Optional Courses

The following courses are offered to the students as optional. The students are expected to choose three to four of the courses listed in Table 5.3. It is important to notice that all of the courses deal with systems that can be embedded within safety-critical software systems. Thus, it is important that each address the responsibilities associated with creating such systems.

Software Engineering: ‖This course should expand further upon the issues that have been addressed in the previous chapters. The problem solving process should be discussed proving guidelines for solving software-related problems. It should provide the needed foundation for the development of software in any domain. Because software development expands into virtually all domains, review of example systems can show what is currently run by software, and exposure to software failure can show the effects of poor development or poor understanding of the requirements. The course should include the discussion of complexity, specification and verification techniques, testing methods, and tools for software development.

‖Corresponds to the CSAB course focusing on software design.

Computer Networks: The process of problem solving is clearly applicable to this area. Though "real-life" examples may not be as accessible, much attention can be paid to the social aspects of networks. The importance of this topic is evident by the discussions involving the Internet. There are several articles that address the following: privacy issues, ownership issues, interstate/international issues, etc. [1, 107, 108, 109, 110, 111].

Artificial Intelligence & Symbolic Programming: Techniques for using the models that have been used in the Artificial Intelligence Community are presented. This course typically discusses searching and storing information. These programs can become increasingly complex. In addition, they can be used in systems that can potentially cause danger. Thus, programs written in the area of Artificial Intelligence need to be subject to a high level of rigor if they are to be used in the industry. As for the social aspects of Artificial Intelligence, the area studies various aspects of human life. Thus, studies are made to understand how people perform specific tasks as well as why they perform those tasks.

Translation of Programming Languages: **The theory behind the translation of programming languages is taught and followed by the application of the theory through programming projects. This course provides a good example of how theoretical mathematical concepts and grammars are actually applied within the domain. In addition, it allots for understanding of machine specifics. Emphasis should be placed not only on programs that perform the translations, but also on the fundamental theory.

**Corresponds in part to the CSAB course focusing on concepts of programming languages and computer elements.

Organization of Programming Languages: ††The emphasis should be placed on the various types of languages and paradigms upon which the languages are based. This shows what types of languages are available and what type of capabilities each language group has. Small programming problems can be given to illustrate the amenability of language for one type of application over another. Likewise, discussion of recent incidents that have occurred involving programming language concepts will also be beneficial. For example, when discussing concurrent programming, issues such as the need for mutual exclusion and the elimination of race conditions are of primary importance. The Therac-25 incidents show some consequences of poor programming that fails to acknowledge the necessity of such precautions.

Computer Graphics: The need for precision within computer graphics programs is great. Thus, use of some form of formalism can be warranted. As for the social aspects, understanding of the human factors issues can be discussed. We not only create accurate systems but ones that can be interpreted correctly by the audience. Human-Computer Interfaces are becoming more and more important as computer systems are further integrated into society. The interface issues should be dealt with during the design specification stage. In applications such as aircraft cockpits, the data as well as its graphical representation to the pilots is of great importance. If correct data can not be interpreted, then the data is useless. As seen in the Airbus incidents, interface issues can be major factors [55].

Vector & Parallel Programming: It is clear that we have not found methods for perfecting sequential programs. The addition of parallel processing adds an additional level of complexity to programming. Thus, formal techniques should be used to specify and verify the correctness of such complex programs.

---

††Corresponds in part to the CSAB course focusing on concepts of programming languages and computer elements.

Database Systems: In addition to the structuring of data within the database, issues such as: *data integrity* and *data protection* should be handled. The students need to understand the potentially wide-sweeping effects and impact of incorrect data. The need for non-ambiguous data is especially critical for systems that have a direct impact upon human life. The Therac-25 incident involving the turntable showed the harm that could be caused by ambiguous data. Though the switch signal was generated by the system, it could have been data stored in a database. There are several incidents that involve people being declared "dead" in a database and therefore losing their identity. Another effect of incorrect data involves several possible situations that could occur if certain parties gained access to confidential information. In addition, the need for privacy and data security are becoming very important issues. Though these may not directly cause human loss, the effect of confidential information being accessed could potentially have serious adverse consequences.

CHAPTER 6

Software Advisory Board

A potentially significant factor that contributes to the lack of a sense of responsibility and accountability for software developers is the lack of a central agency that deals primarily with software. The unique qualities of software, as discussed in Chapter 1, warrant the need for a board that deals specifically with software-related issues. While many of the domain-specific regulatory boards, such as the FDA, FAA, and DoT, have members on their committees that handle software issues, there is no uniformity between domains. As the use of software-controlled products increases, the various domains will have to adopt explicit guidelines for handling software systems. If each domain is allowed to develop its own standards, then it is unlikely that uniform standards will be applied. Uniform standards could be used to provide minimum guidelines designed to promote and preserve public safety. Any attempt in the future to create uniform guidelines between the domains will be difficult, if not impossible.

This problem of failing to uniformly deal with software systems is also found within the domains themselves. For example, since the Therac-25 incidents, the FDA has added policies to specifically deal with software systems [10]. The changes, however, were separate from the new methods that the American Association of Physicists in Medicine enacted to improve computer-assisted radiation treatment [10]. While each of the groups' recognition of the need to change their policy is commendable, there

remains the need for a body that can facilitate communication between each of the groups within and among each domain. Such a body would not only create a thread of cohesiveness, but it would also aid each of the domains in the creation of new policy.

We propose the establishment of a Software Advisory Board (SAB) to oversee software-related projects across all domains. The needs for the SAB are numerous. Rather than list all of the current problems, we examine the proposed functions of the SAB and address some of the issues that are currently being neglected. In the previous chapters, propositions for change within the legislature and the educational system have been examined. The SAB plays an instrumental role in bringing those propositions to fruition. As software becomes a major part of American culture, it is logical that there should be a governing body that can help regulate its use.

6.1 Regulatory

In Chapter 4, we discussed some of the problems with the legal system when dealing with cases that involve software products. The discussion was divided into two main parts: liability and policy. It is these two areas that the regulatory role of the SAB will address.

6.1.1 Liability

First, the SAB can help provide guidance for liability cases in each of the domains. Having a representative body that fully understands the uses as well as the limits of software will aid in concluding comparable decisions for similar cases. The SAB will develop a set of standards that accomplish a standard of care as explained in the following: "The standard of care provides a tool to distinguish between malfunctions (bugs) due to inadequate practices and those that occur despite a programmer or

designer's best effort [12]." Without a SAB, there will be no source that would provide guidance for handling similar software liability cases. Instead, each case will be based upon domain specific analyses performed by each court. This approach can lead to poor legal practice causing each court to give independent rulings on software systems. In addition, it is very costly. The SAB could act as a regulatory board that keeps a set of standards for software systems within the known limits of current technology, making software liability cases more concrete.

6.1.2 Policy Creation

Work would be done to ensure that each manufacturer of software systems understands what level of quality is expected. This function brings us to the remaining regulatory role of the SAB, policy. The SAB could provide expert knowledge of what should be expected of software systems. Amendments to current policy regarding software would help to clarify the guidelines the manufacturer should adhere to and also outline limits of software. While it is known that large software systems cannot be proven to be 100% perfect, there are common procedures that can be used to help prevent hazards. These procedures include, but are not limited to issues discussed in the education chapter, (i.e., design documentation, formal specification of requirements, testing procedures, minimum guidelines). A member of the SAB would be expected to know the limits as well as the advances of current evaluation techniques. In the situation where the system design exceeded the limits, the design should replace the software with a more reliable mechanism, such as hardware.

6.1.3 Criminal Justice

In the case of the Tylenol Tamperings, the Food and Drug Administration had the authority to enact laws that prohibited tampering or alleged tampering with consumer

products [20]. The SAB should also have analogous authority. If there are incidents where the manufacturers explicitly disregard the policy produced by the SAB and place the public in danger, then the SAB should be able to take action. By giving the SAB input into the creation of federal laws intended to protect the public, safety can be improved.

6.1.4 Extraneous Systems

Another regulatory role of the SAB would be to directly oversee the production of any safety-critical software system that is not specifically governed by any domain. In such cases, the SAB would be able to establish certification guidelines and have the authority to recall the product when the SAB felt that the product threatened the public's safety.

6.2 Approval

The SAB would be able to provide industry and academia with a set of SAB approved validation/verification methods or agencies that provide vital services.

6.2.1 Independent Testing Agencies

The SAB could publish lists of "SAB Approved" Independent Testing Agencies. A common practice is to have someone within the development process conduct testing and decide whether or not the product is marketable. To many, this approach seems counterproductive in that there is an obvious bias.

> Today, software is treated like other commercial products, both mass-produced and contracted. In cases in which correct software performance is considered critical or at the request of the buyer, the software is tested (usually by providers) for safety and correctness. Unlike special products such as children's toys, new medicines, nuclear plants, and chicken, software is rarely tested by independent agencies [11].

Though there may be a variety of reasons contributing to why software is not being independently tested, if a list of credible testing agencies were publicized, perhaps more manufacturers could utilize the resource and begin independent testing. Having a product approved by an independent testing agency would help and certainly not hinder the product's retail value. In a recent report regarding the Space Shuttle's software system, a committee, independent of NASA, assessed the shuttle's software development process [89]. The committee recognized the use of some verification and validation from within NASA, but recommended that additional independent reviews be conducted [89].

6.2.2 Rating Methods

A list of acceptable Reliability Rating Methods could also be given. One main problem with the current state of affairs is that software seems to escape normal certification procedures. We have already stated that the SAB could work to update regulation policy. We also propose that the SAB offer additional means of rating the systems in order to provide a way to obtain added safety. Software, regardless of its complexity, needs to be controlled.

> It is unreasonable to exempt software from quantitative reliability requirements, whenever dependability and risk are otherwise treated in a quantitative fashion. In these cases, there should be an obligation to convincingly and rigorously demonstrate that such requirements are satisfied [92].

As with the Independent Testing Agencies, providing an approved list of reliability measurements produces the possibility that more will try to integrate them into normal practice.

Example. One method that is increasingly being used as a metric for evaluating the quality of the development process and organization is the Capability Maturity Model (CMM) [90]. The intent of CMM is to provide organizations with a way to

rate their current process and assess what should happen in the future [90]. CMM is comprised of the following five levels [90]:

Level 1 - Initial: Process is ad hoc or chaotic.

Level 2 - Repeatable: Some basic management processes; techniques that work well on one project are used/repeated on future projects.

Level 3 - Defined: There is a standard software process; most projects closely follow this process.

Level 4 - Managed: The process and product is closely evaluated; quantitative measurements are used.

Level 5 - Optimizing: Feedback coupled with new ideas is given regarding process; proper changes are then made.

Further discussion of CMM can be found in the cited reference [90].

Another method that could be examined by the SAB is the verification and validation techniques used by both NASA and the independent assessment committee. The committee's method was based upon an approach that reviews reports and changes to the shuttle software by using three main areas of analysis that are summarized as follows [89]:

Limited Analysis: Reviews the problems, impacts, requirements risks, and disposition.

Focused Analysis: Checks the code, testing, verification, documentation, and safety issues.

Comprehensive Analysis: Studies implementation of other systems and performs complete tests and verification.

Though a complete understanding of the assessment process is not available in this dissertation, the report does provide an additional example of a rating method that is currently being used. More detailed information about the methodology behind the analysis can be found in [89].

6.2.3 Education Certification

A final set of guidelines that could be approved is the certification criteria for Computer Science education. This would include providing universities with a list of requirements that are needed to become a certified academic institution. The SAB could review existing accreditation criteria such as that provided by CSAB or could simply have the CSAB become a part of the SAB. The educational certification standards could provide a basis for material found on a future "Computer Science Professional" exam.

6.3 Interactive

There are several benefits that could be realized if the SAB were used as an intermediate link between various parties involved with software systems. The SAB would help the Software Development Community, the Regulatory Agencies, and academia interact to provide safer software systems. The role of the SAB as it relates to each of these communities is discussed in turn.

6.3.1 Software Development Community

The SAB could be used to interact with members of the Software Development Community. The advantages to such a communication link are many, but two primary advantages are considered.

First, the SAB could utilize the experiences of the community to help develop the regulatory policy presented in Section 6.1 and also the approval lists that were discussed in Section 6.2. To better understand the community's needs, direct contact must be maintained.

A second provision that the SAB would provide would be a link between the software developers and a body of authority. Though this relationship may seem

trivial, it is in fact an important asset. Expected to meet specific deadlines, many products are sent out before adequate testing has been performed. While the product may meet the minimum guidelines, there still could be other dangers associated with the product. The decision to send out a product containing software before it is ready is not the decision of the software developer, but that of the product manger. This conflict puts the developer into an awkward and undesirable situation having to send out a product that has not been proven to be safe. We examine a situation that occurred ending in a tragedy. However, the true tragedy, as will be shown, was that the incident could have been prevented had the complaints of certain members of the development team been addressed.

Challenger Incident: In 1986, millions of citizens watched the Space Shuttle Challenger explode only 73 seconds after liftoff [112]. All crew members, including the "Teacher in Space," Christa McAuliffe, were killed [112]. Tragedy overcame the Nation as it attempted to understand the shocking incident. However, the incident was not shocking to all. Several design engineers had made attempts to postpone the takeoff predicting such a disaster [112].

The cause of the incident was determined to be a leak in the O-Rings though knowledge of the problem with the O-Ring had been available since 1979 [112]. Furthermore, a particular engineer, Roger Boisjoly, had been conducting tests that proved that the rings were subject to failure during low temperatures [112]. He made a notable effort to make NASA aware of the existing problems, though to no avail. Finally, the evening before takeoff, he was part of a teleconference between the company he worked for, Morton Thiokol, and NASA [112]. During the conference, he proceeded to explain why he felt an explosion could occur. Despite his substantiated plea, the Morton Thiokol managers gave into the desires of NASA and gave the go ahead for the launch. Roger Boisjoly had done everything possible to prevent the disaster and

was forced to sit and watch the explosion, the same explosion he predicted. Some engineers choose to attribute the cause of the Challenger Disaster to poor management, "The Challenger did not crash because of a poorly designed O-Ring, it crashed because of a poorly designed decision-making system—a system where engineers have little input and no vote [112]."

What happened after the incident causes even more disbelief. Several engineers testified of the events that had occurred. After their testimony, they were criticized by their company's management and demoted from their positions [112]. Other employees at the company placed blame upon the engineers for criticizing and giving bad publicity about the company. Roger Boisjoly eventually left the company due to "the hostile work environment" [112]. He now lectures on the importance of engineering ethics nationwide [112].

The fact that the engineers were ignored and shunned is shocking. However, it is easy to see a similar situation occurring where the software developer does not feel that a given product has received adequate testing or analysis. Confrontations such as these occur daily, though not always with critical systems. There is obviously a glaring defect in the corporate ethic system. In the National Society of Professional Engineers Code of Ethics [112], it states,

> The members of the profession recognize that their work has a direct and vital impact on the quality of life for all people. Accordingly, the services provided by engineers require honesty, impartiality, fairness, and equity, and must be dedicated to the protection of the public health, safety and welfare... Engineers shall hold paramount the safety, health, and welfare of the public in the performance of the professional duties.

While there exists controversy as to whether or not software engineering should be considered a profession, few can argue with the fact that software developers have a large impact upon society. In this manner, software designers are similar to engineers and so their concern with public safety should be the same. Thus, adherence to a similar set of ethics is not unreasonable.

In reference to the Challenger disaster, it has been stated that, engineers are not required to take courses that study professional ethics and there is no place to report incidents dealing with ethical dilemmas [112]. The same holds for software developers. Chapter 5 has already addressed the need for ethics within the computer science curriculum. We feel that the SAB would be able to act as a grievance board for those who feel that unsafe software-related products were being sent to the market.

6.3.2 Other Regulatory Agencies

Another purpose of the SAB would be to provide interaction between regulatory agencies that deal with safety-critical software systems. The relevant agencies include, but are not limited to: the Food and Drug Administration, the American Association of Physicists in Medicine, the Federal Aviation Administration, and the Department of Transportation. These are the agencies that regulate the examples presented in this dissertation.

6.3.3 Academia

The tie between the SAB and academia has already been explored in previous chapters. Any useful research that is produced in the university setting can be submitted and reviewed by the SAB. If the research is found to be applicable to the software development community, such as new reliability rating models, the SAB can publicize the results and add the method to the list of approved methods. Once new results are found, the SAB would have the power to integrate the results into the system, keeping software regulation current.

CHAPTER 7

Cooperative Balance

We have presented a three-pronged approach to promote higher quality software development. In order for the approach to be successful, there must be a cooperative balance between three primary organizations: industry, the legal system, and academia. If one fails to take the prescribed measures, then the others would likewise fail. The necessity of each organization is complemented with a direct impact on various members of the organizations.

7.1 Industry

The call for reform addresses the need for safer computer-based systems. In addition to providing a higher level of safety, the approach also directly affects members of the safety-critical software development team: management and developers.

7.1.1 Management

Management is a critical factor in the adoption of more systematic, repeatable, and formal methods within industry. It is managers that allocate the resources needed to apply more rigorous methods. The lack of technical knowledge as well as the lack of appreciation for software safety may be reasons why some managers do not sup-

port formal methods. Currently, software safety is often neglected in large industrial projects, projects where they are needed the most [113].

Our approach facilitates the bridging of the gap between management and developers. By including managers as an active part of the development process, we place new standards on them calling for managers that understand and appreciate the formal design process. The impact of such an assumption carries over to many management-related areas. First, if managers are able to fully understand the tasks of the workers, there would be a higher level of respect, trust, and interaction between management and workers. Secondly, the added awareness would help managers recognize 'good' employee skills and the value of educational background (e.g., advanced degrees) leading to better merit evaluations. Also, managers would be able to recognize the need for certain skills and thus be able to provide opportunities for improvement through enrollment in workshops and educational programs. Next, knowledgeable managers would be attentive to the needs of the worker and allow for more accurate budgeting of funds, time, and resources. In addition, because managers would be directly involved in their team's progress, management could play a concrete role in being held accountable for their projects. This role includes willingness to fix errors and to receive commendation for a job well done. Finally, managers could interact with the SAB enabling them to keep in contact with other managers within their domain and to remain up-to-date with new technology.

7.1.2 Workers

The crux of our approach has addressed the lack of responsibility and accountability of the members in the actual development process. Some of the primary members in this process are the workers, the developers of the systems. However, we have found that this discrepancy coincides with several factors such as insufficient education. Our approach addresses this issue and provides the following for workers.

First, workers would be able to have more trust in their projects. The concept of management would become a more positive aspect of the development process. If management demonstrates through actions a genuine/sincere concern for the quality of computer-based systems, there would be more interaction between management and the workers to help alleviate problems early on. Also, the deadlines management places on the workers would be much more reasonable and achievable taking into account time and effort needed to use rigorous approaches to development. Furthermore, adequate resources should be allocated for independent verification/validation efforts. The use of formal methods would allow for more explicitly defined expectations of the workers. Second, workers would stay aware of current techniques through distributions produced by the SAB and also through workshops and short courses sponsored by management. Third, the appropriate awards for good work and education would be acknowledged through the hiring and promotional processes. Finally, if a worker experiences a problem, (e.g., compromising safety), within the company and has made several attempts to have it remedied, the SAB would act as a resource to support the workers ethical responsibilities to the public.

7.2 The Legal System

It has been shown that the legal system has been asked to handle computer-related issues. Our approach provides the legal community with sound guidelines that should be followed when dealing with safety-critical software systems. We do not attempt to define the law but instead to provide the information needed by the legal community to interpret and understand the law with respect to computer-related issues. Providing judges and lawyers with finite properties that can be expected of software systems would help them to interpret regulations and laws and provide a beginning foundation for computer-related legal issues.

7.2.1 Judges

Our approach exposes a need for judges who have a certain level of understanding of computer-based systems. First, the proposed additions to regulation policy would provide the judges with a concrete set of guidelines to examine when faced with a faulty software system. Second, having a more well-defined computer science graduate would raise the level of expectations of the courts and provide a better point of reference in dealing with what is expected of the "reasonable computer scientist." Next, some of the impact of our approach would stem from the induction of the SAB. The SAB would act as a national authority that can provide information to the judges regarding the current state of technology and formal methods. The uses of this SAB role would range from making known recent verdicts within the various domains, providing input in the redefinition of permissible courtroom evidence, discussion on the use of professional witnesses and juries, to the discussion of new criminal acts for those who disregard public safety. Finally, the SAB would be able to tie together all domains that deal with safety-critical software systems by having each domain comply to similar guidelines. This centralization would help to minimize the work of the judges and also provide a place for software-related precedents. If this centralization were not the case, then each domain would be able to create its own set of regulations and actions making each court case domain dependent, a time consuming and expensive approach.

7.2.2 Lawyers

Our approach calls for competent lawyers to handle computer-related cases while at the same time creating new opportunities for lawyers nationwide. First, regardless whether a lawyer represents a manufacturer or a member of the public, our approach helps to define distinct guidelines for software-related court cases. The responsibility

of the manufacturer and the developer have been discussed to provide higher expectations in both roles. The changes to education help to define the level and quality of competence expected from computer science graduates. Second, the attempt to provide a uniform handling of software-related systems would help to create precedents set by standard legal cases that can be studied and reviewed by the lawyers. Thus, a lawyer working on a computer-related case would not have to start from scratch, but would have cases to reference. The next impact directly follows implying that there would be a need for lawyers with an understanding of computer-based systems and development processes that also have the drive to forge into the unknown. Lawyers working in this field would be faced with exciting and challenging issues. They would have to take fundamental legal properties, (i.e., punitive damages, contract law, and comparative negligence), and interpret them with respect to computer-related cases. Though the work may be difficult, it certainly presents a worthy challenge.

7.3 Academia

The final area that is directly affected by our approach is academia. Because there are many complaints as to the quality of computer science graduates, as seen in Section 5.1.2, change should be welcome.

7.3.1 Professors

The impact of our approach expands into the university faculty as follows. First, the changes to education as well as the SAB provide direct opportunities for linking professors with industry. The results are twofold: professors would be able to stay in touch with industry trends and professors would have more opportunities to present their ideas to be used in industry, thereby making their instructional techniques and research more accessible for use. The changes in education would also require

professors to have current and updated lecture material and assignments in order to give the faculty and the students access to state of the art technology and to the needs of industry. Second, the revised education structure would provide a new level of expertise in future students. By introducing fundamental concepts early in the curriculum, professors would be able to focus on the subject matter rather than spending large amounts of time reviewing basic techniques. This approach applies to both undergraduate and graduate level courses. It has been stated that often graduate programs spend much of the time teaching students what should have been taught in the undergraduate program [96]. Hopefully, the new manner that the courses would be taught (i.e., including more real-life examples and ethical issues) would bring new motivation to computer science students. Next, the existence of the SAB would provide professors with a direct frame of reference to where their program should be with respect to other universities. Accreditation standards, sample course syllabi, and SAB approved methods for verification/rating would all be available from the SAB. Finally, because the foundation for each of the complementary aspects of our approach is rooted in academia, this would give professors the upperhand on understanding the technology, formal methods, and the responsibilities associated with software development. The raised level of respect could potentially provide them with more opportunities to consult in industry, to be sponsored by industry, or to serve on the SAB.

7.3.2 Students

We believe that the students would be the group to benefit the most from our approach. First, the changes to education attempt to provide understanding of the entire view of computer science. If the students can understand in the first course how important mathematical skills and ethical responsibilities are, then they would be able to apply these basic skills in other courses while gaining an appreciation for

the other courses, giving them a sense of motivation. Second, the interaction between academia and industry would keep professors familiar with current techniques rather than allowing them to become detached from society. Third, students would be better prepared for various areas of industry. Equipped with technological skills and ethical responsibility, they would be an asset to the workplace. The experience gained from the project course would create a solid foundation for entering industry. Finally, the SAB would provide the uniformity needed to allow computer science graduates to be able to enter into various domains. Without the SAB, students would have to study domain-specific standards and expectations for testing, validation, and verification. In other words, each domain would have different requirements and guidelines. In general, the computer science student would be faced with a myriad of opportunity upon graduation.

7.4 Discussion

We have reviewed the impact that our approach has on each organization involved within the approach. The cooperative balance between each of the organizations fosters many advantages including good communication links, a higher level of ethical responsibility and accountability, higher-quality development teams, and most importantly, safer systems. We must remember that though each of the groups is affected in its own specific manner, each member of the development process, the legal system, and education system is not only a member of the respective organization, but is also a member of the public. As a primary goal to provide a safer environment for the general public, the increase in public safety is by far the most important impact.

CHAPTER 8

Conclusions and Future Investigations

The promise of public safety is being compromised as potentially dangerous software-controlled products become a part of daily life. We have examined a few tragic incidents that expose vulnerabilities in the software development process, namely a lack of responsibility and accountability. Though the occurrence of such incidents is unfortunate, it does provide motivation for change. The purpose of this dissertation is to provide an approach for promoting both responsibility and accountability through proactive and reactive measures. Thus, as a remedy to the problems we have discovered throughout this dissertation, we present a three-fold strategy that attempts to renovate the legal system to include software systems, adds to the education curriculum with the intent of teaching more rigorous development techniques and social responsibilities, and proposes the creation of a software advisory board whose primary concern is software safety.

Updates to legislation include the application of *standard liability* to software systems and the recognized need for *software-specific policy.* The suggested renovation to the undergraduate education program for computer scientists places more emphasis on problem solving, ethical responsibilities, and the need for formal methods, espe-

cially when dealing with safety-critical systems. Finally, the notion of the Software Advisory Board was introduced exhibiting its ability to regulate the use of software, give approval to various techniques in certain areas, (i.e., system testing, process rating, and educational certification), and assist by interacting between members of the development process. When combined and used constructively, these three actions provide a foundation for responsibility and accountability in the software development domain.

We have presented the need for a cooperative balance between three primary areas: industry, the legal system, and academia. Everyone involved in the software development process could be greatly and positively impacted by the three-pronged approach. This impact would include everyone from the professors and students in academia to the managers and project workers in industry to the judges and lawyers involved in the legal system. The possible outcome of applying this approach provides many benefits to each of the groups. However, the approach also unveils many other issues that have not been addressed.

We hope that serious thought is given to the principles discussed throughout the dissertation. The legal perspective has produced several other computer-related topics for review. Investigations should be conducted regarding the application of legal matters in other computer-related areas, such as evaluating the interpretation of Contract Law, the adoption of criminal statues, the possibility of computer programmer malpractice, and the study of privacy/security issues all with respect to computer systems.

Insights into the educational system have also brought attention to the need for better education. In addition, many currently debated topics are revealed. What role should Formal Methods play in the education of undergraduates? Should Computer Science be an engineering discipline? Should Software Engineering sever itself from general Computer Science and become a separate discipline? Questions such as these

then lead to a prime debate among industry, Should there be a profession of computer scientists? If so, which specializations within Computer Science should be included or excluded? Should we certify software engineers? Though issues such as these seem to plague the Computer Science community, it is somewhat reassuring to know that ideas are being discussed and considered.

Finally, the induction of the Software Advisory Board brings to light the role of the government in software development. Should the SAB be federally controlled? If so, how would funds be allocated for such a purpose? Who will regulate the SAB? Will the SAB make specific formal specification languages, testing strategies, and rating methods mandatory for all software systems? How will the cost of implementing such features affect industry? How will the SAB enforce specific guidelines?

It is clear that there are many issues left to be addressed. We presented what we felt to be the most pressing issue, the development of safety-critical software systems. Though discussion of development is important, the importance of public safety cannot be ignored. We realize that this thesis does not give a method for ensuring public safety 100% of the time, but it does provide a model for increasing the level of safety and the level of awareness. Furthermore, it exposes the need to integrate legal awareness into Computer Science and the need to integrate computer issues into the legal system. As we enter a world of new technology, it is critical to ensure that the development process of such technology also advances and evolves to handle the additional complexity and the added responsibility to the public. Looking to new advances in the future is acceptable as long as we do so without forgetting the fundamental values of the past, values such as responsibility and accountability.

APPENDICES

APPENDIX A

Glossary of Legal Terms

The following provides a list of definitions for some of the fundamental legal concepts discussed in this dissertation. This glossary is meant to assist those who have had minimal exposure to such legal terms. Though there are many interpretations, we provide the definitions that correspond to the term with respect to the context of this dissertation. The definitions have been paraphrased from Black's Law Dictionary [77].

Comparative Negligence: Negligence that is split according to the percentage that the party was at fault; recovery is given to the plaintiff based upon the amount that the defendant was negligent and the amount that the plaintiff was negligent.

Contract Law: Legal obligations incurred by deviation from a contract, an agreement between two parties. There are many types of contracts each with specific guidelines.

Disclaimer: Rejection of rights normally retained by certain persons.

Liability: Having an obligation to perform some duty (i.e., pay a debt, make justice, provide some service).

Negligence: Failure to provide care equal or greater than that of "the reasonable man" where the reasonable man is held to the standards of the average person.

Punitive Damages: Damages awarded to plaintiffs that exceed compensation values; they are meant to punish the defendant for a wrongdoing and are usually applied in cases involving violence, intended harm, and the like.

Strict Liability: Liability with determining fault imposed upon the seller of defective and potentially dangerous products.

Tort Law: Violations of the law committed to a person or a person's property that was independent of any contract.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] P. G. Neumann, *Computer-Related Risks.* Addison-Wesley Publishing Company, 1995.

[2] W. W. Gibbs, "Software's Chronic Crisis," *Scientific American*, vol. 271, pp. 86–95, September 1994.

[3] "Risks to the public in computers and related systems," *Software Engineering Notes*, vol. 19, pp. 4–12, July 1994.

[4] I. Peterson, "Finding fault: The formidable task of eradicating software bugs," *Science News*, vol. 139, pp. 104–106, February 1991.

[5] M. J. McCarthy, "PepsiCo Is Facing Mounting Problems In the Phillipines," *The Wall Street Journal*, December 1994.

[6] J. Dalbey, "Pepsi promotion misfires - computer error," *RISKS-FORUM Digest*, vol. 16, December 1994.

[7] B. Friedman and P. H. K. Jr., "Educating computer scientists: Linking the social and the technical," *Communications of the ACM*, vol. 37, pp. 65–70, January 1994.

[8] S. E. Lipner and S. Kalman, *Computer Law: Cases and Materials.* Merrill Publishing Company, 1989.

[9] C. S. Turner, "Private Communication," January 1995.

[10] N. G. Leveson and C. S. Turner, "An Investigation of the Therac-25 Accidents," *IEEE Computer*, vol. 26, pp. 18–41, July 1994.

[11] W. R. Collins, K. W. Miller, B. J. Spielman, and P. Wherry, "How Good is Good Enough?," *Communications of the ACM*, vol. 37, pp. 81–91, January 1994.

[12] H. Nissenbaum, "Computing and accountability," *Communications of the ACM*, vol. 37, pp. 73–80, January 1994.

[13] K. Crain, "ABS education needed," *Automotive News*, p. 12, February 20, 1995.

[14] J. Baum, "Safety First," *Far Eastern Economic Review*, vol. 157, pp. 74–75, June 16, 1994.

[15] V. Neufeldt, Ed., *Webster's New World Dictionary.* High Technology Law Journal, 1990.

[16] J. Bowen and V. Stavridou, "Safety-Critical Systems, Formal Methods and Standards," tech. rep., Oxford University Computing Laboratory, 1992.

[17] N. G. Leveson, "Software Safety in Embedded Computer Systems," *Communications of the ACM*, vol. 34, pp. 34–46, February 1991.

[18] K. L. Ropp, "Tylenol Tampering," *FDA Consumer*, vol. 26, p. 16, October 1992.

[19] D. Kirkpatrick, "Intel's Tainted Tylenol," *Fortune*, vol. 130, pp. 23–24, December 1994.

[20] A. Hecht and M. Shaffer, "Tamperings, False Reports Bring Arrests, Jail," *FDA Consumer*, vol. 20, pp. 28–29, September 1986.

[21] Anonymous, "Agency Says Craft Are Safe," *The New York Times*, vol. 144, p. 9, December 3, 1994.

[22] I. Molotsky, "F.A.A. Restricts Some Icy-Weather Flying," *The New York Times*, vol. 144, p. A24, November 10, 1994.

[23] M. L. Wald, "Flight Recorders Found At Indiana Crash Scene," *The New York Times*, vol. 144, p. A20, November 2, 1994.

[24] A. Bryant, "Pilots at Odds With Airline Over Wing Ice," *The New York Times*, vol. 144, p. A17, December 1, 1994.

[25] A. Bryant, "2 Types of Planes Grounded by F.A.A. in Icy Conditions," *The New York Times*, vol. 144, p. 1, December 10, 1994.

[26] A. Bryant, "Flights Canceled After Commuter Planes Are Banned," *The New York Times*, vol. 144, p. 30, December 11, 1994.

[27] E. McDowell, "Holiday Travelers Cope With Threat of Chaos," *The New York Times*, vol. 144, p. 20, December 25, 1994.

[28] E. H. Phillips, "NTSB Studies Jetstream Crash, ATR Icing Data," *Aviation Week & Space Technology*, vol. 142, pp. 28–30, January 2, 1995.

[29] A. Bryant, "Agency Lifts Restriction on Use of Plane in Cold Weather," *The New York Times*, vol. 144, p. A14, January 12, 1995.

[30] E. H. Phillips, "FAA Lifts Icing Ban on ATR Aircraft," *Aviation Week & Space Technology*, vol. 142, pp. 28–29, January 16, 1995.

[31] J. T. McKenna, "ATR Problems Force American Eagle to Shuffle Aircraft, Crews," *Aviation Week & Space Technology*, vol. 142, pp. 31–32, January 2, 1995.

[32] Anonymous, "Ford Is Recalling 133,476 Windstar Vans," *The New York Times*, vol. 144, p. 9, January 14, 1995.

[33] R. C. Thompson, "Faulty Therapy Machines Cause Radiation Overdoses," *FDA Consumer*, vol. 21, pp. 37–38, December/January 1987/1988.

[34] A. S. Tanenbaum, *Operating Systems: Design and Implementation.* Prentice-Hall, Inc., 1987.

[35] D. Hughes and M. A. Dornheim, "Accidents Direct Focus on Cockpit Automation," *Aviation Week & Space Technology*, vol. 142, pp. 52–54, January 30, 1995.

[36] Anonymous, "Cabin Crew Evacuated A320 Passengers Until Smoke, Flames Became Too Intense," *Aviation Week & Space Technology*, vol. 132, pp. 98–99, June 25, 1990.

[37] J. Rushby and F. V. Henke, "Formal Verification of Algorithms for Critical Systems," in *Proceeding of the ACM SIGSOFT '91 Conference on Software for Critical Systems*, pp. 1–15, ACM, December 1991.

[38] C. Covault, "A310 Pitches Up, Dives on Orly Approach," *Aviation Week & Space Technology*, vol. 141, p. 37, October 3, 1994.

[39] Anonymous, "French Report Details 1988 Crash of A320 Following Air Show Flyby," *Aviation Week & Space Technology*, vol. 132, pp. 107+, June 4, 1990.

[40] Anonymous, "Commission Proposes More Preparation, Special Training for Airshow Flyovers," *Aviation Week & Space Technology*, vol. 133, pp. 90+, July 30, 1990.

[41] P. Sparaco, "A330 Crash to Spur Changes at Airbus," *Aviation Week & Space Technology*, vol. 141, pp. 20–22, August 8, 1994.

[42] J. M. Lenorovitz, "Indiana A320 Crash Probe Data Show Crew Improperly Configured Aircraft," *Aviation Week & Space Technology*, vol. 132, pp. 84–85, June 25, 1990.

[43] M. A. Dornheim, "Dramatic Incidents Highlight Mode Problems in Cockpits," *Aviation Week & Space Technology*, vol. 142, pp. 57–59, January 30, 1995.

[44] A. Pollack, "261 Die When a Flight From Taiwan Crashes in Japan," *The New York Times*, vol. 143, p. A3, April 27, 1994.

[45] M. Mecham, "Bulletin Warned Pilots Against AP Overrides," *Aviation Week & Space Technology*, vol. 140, p. 32, May 9, 1994.

[46] P. Sparaco, "Human Factors Cited in French A320 Crash," *Aviation Week & Space Technology*, vol. 140, pp. 30–31, January 4, 1994.

[47] J. M. Lenorovitz, "French Investigators Seek Cause of Rapid Descent," *Aviation Week & Space Technology*, vol. 136, pp. 32–33, January 27, 1992.

[48] J. M. Lenorovitz, "Confusion Over Flight Mode May Have Role in A320 Crash," *Aviation Week & Space Technology*, vol. 136, pp. 29–30, February 3, 1992.

[49] J. M. Lenorovitz, "Crash Likely to Focus New Attention on Need For Pilot Interface on Advanced Transports," *Aviation Week & Space Technology*, vol. 136, p. 29, February 3, 1992.

[50] Anonymous, "Pilot's Go-Around Decision Puzzles China Air Investigators," *Aviation Week & Space Technology*, vol. 140, p. 26, May 2, 1994.

[51] M. Mecham, "Autopilot Go-Around Key to CAL Crash," *Aviation Week & Space Technology*, vol. 140, pp. 31–32, May 9, 1994.

[52] Anonymous, "New CAL 140 Transcript," *Aviation Week & Space Technology*, vol. 140, p. 32, May 23, 1994.

[53] Anonymous, "A330 Crashes on Test Flight, Killing Seven," *Aviation Week & Space Technology*, vol. 141, p. 20, July 4, 1994.

[54] P. Sparaco, "Autopilot a Factor in A330 Accident," *Aviation Week & Space Technology*, vol. 141, pp. 26–27, July 11, 1994.

[55] M. A. Dornheim, "Modern Cockpit Complexity Challenges Pilot Interfaces," *Aviation Week & Space Technology*, vol. 142, pp. 60–63, January 30, 1995.

[56] L. Hatton, "A340 shenanigans," *RISKS-FORUM Digest*, vol. 16, December 1994.

[57] P. Ladkin, "Re: A340 incident at Heathrow," *RISKS-FORUM Digest*, vol. 16, March 22 1995.

[58] P. Sparaco, "A310 Inquiry Targets FCU," *Aviation Week & Space Technology*, vol. 141, p. 32, October 10, 1994.

[59] P. Sparaco, "French Issue Airbus Alert," *Aviation Week & Space Technology*, vol. 141, p. 21, October 24, 1994.

[60] Anonymous, "ABS and Other Features," *Maclean's*, vol. 106, p. AS10, November 22, 1993.

[61] P. Greer and D. Boehmer, "Improved electonics aid ABS," *Automotive Engineering*, vol. 98, pp. 30–35, September 1990.

[62] T. P. Mathues, "Extending the scope of ABS," *Automotive Engineering*, vol. 102, pp. 15–17, July 1994.

[63] B. Nadel, "Anti-lock Brakes for $400," *Popular Science*, vol. 237, pp. 78–81, October 1990.

[64] R. Eddie, "Anti-lock brakes on snow and ice," *Automotive Engineering*, vol. 102, pp. 47–49, April 1994.

[65] D. Reed, "Anti-lock Brake Systems," *Automotive Engineering*, vol. 102, p. 59, April 1994.

[66] G. Soodoo, "Private Communication," May 1995. Office of Crash Avoidance, National Highway Traffic Safety Administration.

[67] M. Gates, "NHTSA looks at GM brakes," *Automotive News*, p. 24, October 3, 1994.

[68] M. Gates, "Feds look at brakes on GM trucks," *Automotive News*, p. 23, August 1, 1994.

[69] M. Gates, "NHTSA investigates ABS in Chrysler's Minivans," *Automotive News*, p. 6, May 9, 1994.

[70] J. Keebler, "Lucas develops fix to help ABS users," *Automotive News*, pp. 3+, February 20, 1995.

[71] A. J. Zack, "ABS simulator educates drivers," *Automotive News*, p. 14, April 3, 1995.

[72] D. Reed, "Injury Costs More Than Crime," *Automotive Engineering*, vol. 103, p. 75, January 1995.

[73] D. Jewett, "Truckmakers say ABS is tough sell," *Automotive News*, March 27, 1995.

[74] Anonymous, "Is the Car-Safety Agency up to Speed?," *Consumer Reports*, vol. 59, p. 734, November 1994.

[75] M. E. Fagan, "Advances in Software Inspections," *IEEE Transactions on Software Engineering*, vol. SE-12, pp. 744–751, July 1986.

[76] P. G. Neumann, "Technology, laws, and society," *Communications of the ACM*, vol. 37, p. 138, March 1994.

[77] H. Black, Ed., *Black's Law Dictionary.* West Publishing Company, 1990.

[78] V. Slind-Flor, "Ruling's Dicta Causes Uproar," *The National Law Journal*, pp. 3+, July 29, 1991.

[79] J. V. Vergari and V. V. Shue, *Fundamentals of Computer — High Technology Law.* The American Law Institute, 1991.

[80] M. C. Gemignani, *Computer Law.* The Lawyers Co-operative Publishing Company, 1985.

[81] "Lewis v. Timco, Inc. v. Joy Manufacturing," *Federal Reporter of the United States Court of Appeals, Fifth Circuit*, vol. 697, no. 2d, pp. 1252–1256, 1983.

[82] B. Lathrop, "Design-Induced Errors in Computer Systems," *Computer/Law Journal*, vol. X, pp. 87–126, Winter 1990.

[83] "Aetna Casualty and Surety Company v. Jeppesen & Company," *Federal Reporter of the United States Court of Appeals, Ninth Circuit*, vol. 642, no. 2d, pp. 339–344, 1980.

[84] P. Samuelson, "Liability for defective information," *Communications of the ACM*, vol. 36, pp. 21–26, June 1993.

[85] Anonymous, "GOP prepares to overhaul negligence law," *The Grand Rapids Press*, p. A1, February 19, 1995.

[86] A. R. Dowd, "Regulatory Relief," *Fortune*, vol. 131, p. 24, January 16, 1995.

[87] M. Gates, "High court takes on carmaker safety obligations," *Automotive News*, p. 16, January 30, 1995.

[88] R. S. Pressman, *Software Engineering: A Practitioner's Approach.* McGraw Hill, third ed., 1992.

[89] Aeronautics and N. R. C. Space Engineering Board, *An Assessment of the Space Shuttle Flight Software Development Process.* Washington, D.C.: National Academic Press, 1994-1995.

[90] W. S. Humphrey, *A Discipline for Software Engineering.* Addison-Wesley Publishing Company, 1995.

[91] J. Rushby, "Formal Methods and the Certification of Critical Systems," Technical Report SRI-CSL-93-07, SRI International, Computer Science Laboratory, Computer Science Laboratory,333 Ravenswood Avenue, Menlo Park, CA 94024-3493, November 1993.

[92] B. Littlewood and L. Strigini, "Validation of Ultrahigh Dependability for Software-Based Systems," *Communications of the ACM*, vol. 36, pp. 69–80, November 1993.

[93] M. Shaw, "Prospects for an Engineering Discipline of Software," *IEEE Software*, pp. 15–24, November 1990.

[94] D. L. Parnas, "Education for Computing Professionals," *IEEE Computer*, pp. 17–22, January 1990.

[95] C. K. Chang and G. B. Trubow, "Joint Software Research Between Industry and Academia," *IEEE Software*, pp. 71–77, November 1990.

[96] D. Gries, "Teaching Calculation and Discrimination: A More Effective Curriculum," *Communications of the ACM*, vol. 34, pp. 45–55, March 1991.

[97] J. M. Wing, "A Specifier's Introduction to Formal Methods," *IEEE Computer*, vol. 23, pp. 8–24, September 1990.

[98] B. H. C. Cheng, "Synthesis of Procedural Abstractions from Formal Specifications," in *Proc. of COMPSAC'91*, pp. 149–154, September 1991.

[99] B. H. C. Cheng, "Applying formal methods in automated software development," *Journal of Computer and Software Engineering*, vol. 2, no. 2, pp. 137–164, 1994.

[100] C. B. Jones, *Systematic Software Development Using VDM.* Prentice Hall International Series in Computer Science, Prentice Hall International (UK) Ltd., second ed., 1990.

[101] E. W. Dijkstra, *A Discipline of Programming.* Englewood Cliffs, New Jersey: Prentice Hall, 1976.

[102] D. Gries, *The Science of Programming.* Springer-Verlag, 1981.

[103] Michigan State University, *Undergraduate Program– Department of Computer Science*, first ed., September 1992.

[104] Computing Sciences Accreditation Board, Suite 209, Two Landmark Square-Stamford, CT 06901, *Program Evaluator* Evaluation Handbook, 1994-1995.

[105] Pepperdine University, 24255 Pacific Coast Highway, Malibu, CA 90263-4392, *Pepperdine University Seaver College Catalog, 1994-95*, March 1994.

[106] G. J. Myers, *Advances in Computer Architecture.* John Wiley and Sons, 1982.

[107] P. G. Neumann, "Risks on the Information Superhighway," *Communications of the ACM*, vol. 37, p. 114, June 1994.

[108] P. G. Neumann, "Expectations of Security and Privacy," *Communications of the ACM*, vol. 37, p. 138, September 1994.

[109] L. M. Zanger and L. G. Oei, "Electronic-Record Storage Checklist," *IEEE Software*, vol. 111, pp. 102–103, July 1994.

[110] M. Betts, "Computerized records: An open book?," *ComputerWorld*, vol. 27, pp. 1+, August 9, 1993.

[111] R. Bergman, "Laws sought to guard health data from falling into the wrong hands," *Hospital and Health Networks*, vol. 68, p. 62, January 20, 1994.

[112] K. A. Pace, "The legal professional as a standard for improving engineering ethics: Should engineers behave like lawyers?," *High Technology Law Journal*, vol. 9, no. 1, pp. 93–130, 1994.

[113] S. S. Cha, "Management Aspect of Software Safety," *IEEE*, pp. 35–40, 1993.