SAFEWARE

System Safety and Computers

Nancy G. Leveson

There will always be engineering failures. But the worst kind of failures are those that could readily be prevented if only people stayed alert and took reasonable precautions. Engineers, being human, are susceptible to the drowsiness that comes in the absence of crisis. Perhaps one characteristic of a professional is the ability and willingness to stay alert while others doze. Engineering responsibility should not require the stimulation that comes in the wake of catastrophe.

— Samuel C. Florman
The Civilized Engineer

Whoever destroys a single life is as guilty as though he has destroyed the whole world; and whoever rescues a single life earns as much merit as though he had rescued the entire world.

— Sanhedrin 37:a
Talmud

Contents

Preface

These days we adopt innovations in large numbers, and put them to extensive use, faster than we can even hope to know their consequences . . . which tragically removes our ability to control the course of events.

— Patrick Lagadec
Major Technological Risk

Although the introduction of new technology has always led to accidents, the pace of technological change is increasing along with the potential consequences of accidents. As the industrialized world has learned to cope with and protect itself from the natural forces that used to cause the majority of accidents, man-made systems have taken their place.

Concerns about the dangers of technological innovation are not new. For example, in the early 1800s, James Watt argued against the use of high-pressure steam engines because of the dangers of boiler explosions. Thomas Edison similarly warned against high-voltage electricity in the 1880s. In the past, however, the consequences of accidents caused by dangerous new technology—such as high-pressure steam or high-voltage electricity—were limited. It was possible to learn from our mistakes and to improve our technology as we found out about its dangers. This situation no longer exists. Today we are building systems—and using computers to control them—that have the potential for large-scale destruction of life and the environment: Even a single accident may be disastrous. We do not have the luxury of learning from experience, but must attempt to anticipate and prevent accidents before they occur.

Goals

This book examines what is currently known about building safe electromechanical systems and looks at the accidents of the past to see what lessons can be applied to new computer-controlled systems. One obvious lesson is that most accidents are not the result of unknown scientific principles but rather of a failure to apply well-known, standard engineering practices. A second lesson is that accidents will not be prevented by technological fixes alone, but will require control of all aspects of the development and operation of the system.

In 1968, Jerome Lederer, then the director of the NASA Manned Space Flight Safety Program for Apollo, wrote

> Systems safety covers the total spectrum of risk management. It goes *beyond the hardware* and associated procedures of systems safety *engineering*. It involves: attitudes and motivation of designers and production people, employee/management rapport, the relation of industrial associations among themselves and with government, human factors in supervision and quality control, documentation on the interfaces of industrial and public safety with design and operations, the interest and attitudes of top management, the effects of the legal system on accident investigations and exchange of information, the certification of critical workers, political considerations, resources, public sentiment and many other non-technical but vital influences on the attainment of an acceptable level of risk control. These non-technical aspects of system safety cannot be ignored [**?**, p.8].

Most of these issues are addressed in this book.

Mixing social and technical issues in one book is unusual for engineering or computer science. However, considering technology and its social implications separately can limit and distort both and can result in wasted time arguing the wrong questions, as well as tragic mistakes.

Treating technology apart from its impact on society can lead not only to negative social consequences, but also to unrealistic and unusable technology. Engineers and applied mathematicians are not working in a controlled laboratory situation: The application of scientific principles to real problems requires interaction with the messy world outside the laboratory. Furthermore, most technology is based on assumptions that have political or ethical aspects that can never be "proved." We need not only to identify these assumptions, but to understand their implications and to evaluate their reasonableness.

Conversely, considering social issues without considering what is technically feasible leads to misunderstanding and useless argument. It may, for example, produce conclusions that are overly optimistic about technological improvements or overly defeatist and negative. Social arguments based on incomplete or inaccurate understanding of the associated technology are at best misleading: To be effective, engineers and social scientists need to understand both aspects. Otherwise, they end up arguing at cross-purposes without touching the most appropriate issue: how technical, political, and social decisions should be made considering both the technological and social reality of the time.

Another underlying assumption of this book is that system safety and software safety should not be separated. Safety is a system problem; it cannot be handled in a vacuum. It is rare to find a complex system today that does not include a computer, yet few safety engineering analysis techniques include consideration of computer software, and few software engineering specification and design techniques include specific consideration of system hazards. Most standards and technical approaches to safety involve just "getting the software right" or attempting to increase software reliability to ultrahigh levels. Although in the future it may be possible to construct perfect software, the current reality is that we cannot accomplish this goal for anything but the simplest systems. Moreover, even if the software had no errors, there would be no way to know this with high confidence.

This book attempts to convince the computer science and engineering communities that a different approach is possible and should be tried. The title of the book, *Safeware*, expresses the impossibility of separating the various aspects of the system in dealing with safety issues. The approach advocated here requires a change in attitude and changes in design and development practices. These changes may result not only in safer systems, but in more reliable ones and in cost savings for equal levels of assurance.

Audience

Anyone building or managing the construction of complex, safety-critical systems should find this book helpful. The information provided will be of use to practitioners, regulators, and researchers. In keeping with the basic philosophy that system safety, software safety, and nontechnical issues cannot be separated, all three are intermingled. No assumption of knowledge about system safety is made: The book introduces basic system safety concepts and then demonstrates how they can be extended to include software.

Practitioners will find the information necessary to design a safety program, but a cookbook approach is not provided. One set of procedures that applies to all systems and all organizations does not and cannot exist. By following cookbook solutions, we are very likely to find that we feel satisfied and our jobs are greatly simplified, but risk is not appreciably reduced. The goal of this book is to educate rather than to train—to provide enough information to design safety programs that are tailored to the characteristics of the system being constructed and the organization constructing and operating it.

Those responsible for regulating computer-controlled systems should also find the book useful. A delicate balance between protecting the public and not unduly inhibiting technological progress is difficult to achieve. Any regulation of technology needs to be based on the state of the art in the relevant field and on what is practical versus what is desirable. This book attempts to distinguish between what is currently achievable and what is not. Unfortunately, the information that regulators need for optimal decision making is just not obtainable at this point. Wishing or pretending that the techniques we have work better than they do or that probabilistic risk assessments are possible for software will not make it so. The only option is to do the best possible with what is known and to be flexible enough to change quickly when the state of the art advances.

Researchers will find descriptions of a wealth of open problems on which to focus their attention. To make progress, however, we need to build on what is already known. A goal of this book is to provide the background and information necessary to create new and better solutions to our problems. I have tried to summarize what is known in order to clarify what needs to be done. It will be obvious on reading this book that there is a great deal left to be learned.

Content

Each of the four parts of the book (and most of the chapters) can stand alone. Part 1 examines the nature of risk and the causes of accidents. We must understand the problems we are attempting to solve before we can design effective solutions or select appropriate methodologies. This part provides a foundation for selecting or devising risk reduction strategies.

Part 2 provides the context for the approach to safety taken in this book. To understand a technology, we must understand the basic science that underlies it, along with the social context that provides its purpose, goals, and decision criteria [**?**]. The foundations of system safety are presented in this part as well as the distinctions between it and other approaches to safety.

Part 3 introduces some terminology in order to enhance communication among engineers and computer scientists, and it describes models of accidents and human error that underlie safety techniques. Using these techniques effectively and making appropriate choices among them requires understanding their underlying models and assumptions.

Using the background provided in the first three parts of the book, Part 4 describes the elements of a safeware program, including management, process, hazard analysis, requirements analysis, design for safety, design of the human–machine interface, and verification. Experienced safety engineers may skip to this part, although they are encouraged to skim the rest of the book. Managers may be primarily interested in Chapters **??**, **??**, **??**, and **??**. Chapters **??** and **??** are review for those already familiar with hazard analysis and hazard analysis techniques, but an introduction is included because most software engineers are unfamiliar with them.

Throughout this book, real accidents are used to illustrate the points being made. A mix of applications and types of systems are included. Many of the examples were selected because extensive information about them is available—for example, Three Mile Island, *Challenger*, Bhopal, Chernobyl, Flixborough, and the Therac-25. Oversimplification of the causes of accidents is a common problem; only a few major accidents have been investigated in enough detail to learn what really happened and thus how to prevent future ones. For readers not familiar with the details of these accidents, descriptions are included in the appendices. Some involve computers and some do not. As Kletz has noted, computers do not provide new ways of making errors, but merely new and easier opportunities for making the old errors [**?**]. A shift to computer control should not require us to relearn old lessons the hard way.

Readers will quickly realize that I have not solved the software safety problem. This book summarizes what is known, evaluates what has been proposed, and points to a path to follow. Technological development will not stop for scientists and engineers to find optimal solutions to complex problems. Systems must be built using the best approaches that we currently have. If these are unsatisfactory, then either we should reconsider whether we should be building these systems using computers, or we should admit that we may be increasing risk but that we believe the additional risk is justified with respect to the benefit the system provides.

Acknowledgements

Engineers at Hughes Aircraft Company—including Marion Moon, Bill Noble, and Pete Goddard—first introduced me to the importance of software safety problems, convinced me I should look more deeply into them, and started my education in system safety. I am indebted to them and to others at Hughes for their assistance and encouragement.

Many people provided comments on early drafts of this book. I would especiallly like to thank Capt. Oscar Overton of the U.S. Air Force, Michael Brown of the Naval Surface Warfare Center, Grady Lee, Martin Kazubowski, my colleague Prof. David Notkin, Michael DeWalt of the U.S. FAA, and the University of Washington graduate students who provided feedback, especially Kurt Partridge, Michael Van Hilst, Ross Ortega, and Gail Murphy.

Support for the research described in this book was provided by the NASA/Langley Research Center, the National Science Foundation, the FAA, the California Department of Transportation and the University of California PATH program, Sandia Labs, and private industry. I am particularly grateful to Hughes Aircraft Company and to Dr. David Eckhardt of NASA/Langley Research Center for their continued support over the years.

Part I

The Nature of Risk

A life without adventure is likely to be unsatisfying, but a life in which adventure is allowed to take whatever form it will, is likely to be short.

— Bertrand Russell

The first step in solving any problem is to understand it. We often propose solutions to problems that we do not understand and then are surprised when the solutions fail to have the anticipated effect. This section explores the nature of risk and the causes of accidents.

Chapter 1

Risk in Modern Society

Living in a technological society is like riding a bucking bronco. I don't believe we can afford to get off, and I doubt that someone will magically appear who can lead it about on a leash. The question is: how do we become better bronco busters?

— William Ruckelshaus
Risk in a Free Society

Many of the most serious accidents of this century have occurred in the past 15 years. The worst industrial accident in history (in terms of deaths and injuries) took place in 1984, when the release of a highly toxic chemical, methyl isocyanate, at a Union Carbide chemical plant in Bhopal, India, killed between 2,000 and 3,000 people and seriously injured over 200,000. The worst series of accidents in the 35-year history of medical accelerators occurred between 1985 and 1987, when six people were massively overdosed by a computer-based radiation therapy machine called the Therac-25—all died or were seriously injured. In 1979 and 1986, respectively, the accidents at the Three Mile Island and Chernobyl nuclear power plants occupied the world's attention. Before the Chernobyl disaster, it was widely thought that nuclear power plant accidents involving release of significant radioactive materials to the atmosphere were highly improbable, if not virtually impossible. Just three months before Chernobyl, the *Challenger* accident gripped the world's attention. Is new technology making our world riskier?

Risk is not a new problem. Humans have always had to face risk from their environment, although the risks have changed as society and the natural environment have changed. In the past (or currently in rural areas of developing countries), the greatest concerns were natural (geological and climatological) disasters such as floods, drought, earthquakes, and tropical storms. Today, industrialization has substituted man-made hazards for those rooted in nature. In the United States, technological hazards account for 15 to 25 percent of human mortality and have significantly surpassed natural hazards in impact, cost, and general importance [?]. Damage to our ecosystem is difficult to assess, but danger signals include a high rate of species extinction and high concentrations of toxic chemicals in the environment.

Although industrialized nations have used technology to control many natural hazards, a strict distinction between natural and man-made disasters is an oversimplification. Human tampering with the environment has exacerbated and sometimes caused natural disasters, such as the destruction of watersheds leading to flooding or the release of chemicals into the atmosphere affecting climate and crops. Flood damage in the United States, for example, has increased as expenditures on flood control have increased [?]. Areas prone to flooding were not developed until we introduced flood prevention measures. When an exceptional flood overwhelms these measures (or they do not achieve their design goals), the damage is much worse than it would have been before prevention measures were introduced.

The number of recorded natural disasters in which people were killed or injured rose, on average, during the 1960s and 1970s. This increase does not reflect a change in geophysical or climatological extremes (abnormalities in the physical environment), which held relatively constant during this time, but rather reflects an increasing risk of casualties resulting from these extremes, partly due to societal and technological changes [?].

Similarly, the amount of damage caused by technological accidents may be affected by natural phenomena: The result of an accidental release of radiation or chemicals, for example, may be dependent on weather factors such as wind direction and strength.

Thus, the distinction between natural and technological hazards is often useful, but is not completely accurate. In fact, all hazards are affected by complex interactions among technological, ecological, sociopolitical, and cultural systems [?, ?, ?]. Attempts to control risk by treating it simply as a technical problem or only as a social issue are doomed to fail or to be less effective than possible.

To discover effective solutions to these problems, we must understand the factors underlying the risks we face. Part 1 of this book delineates these factors and the myriad aspects of the general problem of risk. Parts 2 through 4 examine the foundations of system safety and present an approach to controlling technological risks in the complex systems that characterize modern society, especially those systems that include computers among their components.

1.1 Changing Attitudes toward Risk

All human activity involves risk; there is no such thing as a risk-free life. Safety matches and safety razors, for example, are not *safe*, only *safer* than their alternatives; they present a reduced level of risk while preserving the benefits of the devices they replace. No aircraft could fly, no automobile move, and no ship put out to sea if all hazards had to be eliminated first [?].

Progress demands taking some risks, and despite great advances in technology, we are unable to eliminate risk altogether. However, much more so now than in the past, humans are demanding that risks be known and controlled to the extent possible and practical. Societies are recognizing the right of workers, consumers, and the general public to know what risks they face and are making worker and public safety a responsibility of the employer and manufacturer [?, ?].

The shift from purely personal to organizational or public responsibility for risk is a recent phenomenon. Until the early part of this century, workers were expected to provide their own tools, to understand the risks associated with their trade, and to accept personal responsibility for their own safety. This attitude was justified partly by the fact that workers devoted their entire careers to the manufacturing of one or two products [?]; they could thoroughly understand their jobs and had control over the way they performed their tasks.

Today, workers are more at the mercy of their employers in terms of safety, and, accordingly, responsibility has shifted from the employee to the employer. In most industrialized countries, employers are expected to provide a safe working environment and the necessary tools and equipment to maintain that environment. In addition, changes in legal liability and responsibility have led to product safety programs that are concerned with consumer as well as employee safety.

Clearly, in matters of risk, today's complex, technological society requires that the general public place its trust in "expert knowledge" [?]. Accordingly, responsibility for detection of and protection from hazards has been transferred from the public to the state, corporate management, engineers, safety experts, and other professionals. Complete abdication of personal responsibility, however, is not always wise. In some instances—such as the Bhopal accident—the public has completely trusted others to plan for and respond effectively to an emergency, with tragic results. The Bhopal Union Carbide plant was run in a way that almost guaranteed a serious accident would occur. In addition, emergency and evacuation planning, training, and equipment were inadequate. The surrounding population was not warned of the hazards, nor were they told (either before or during the chemical release) of the simple measures (such as closing their eyes and putting wet cloths over their faces) that could have saved their lives. Such incidents have aroused the public to become more involved in risk issues.

In turn, public involvement in issues that past generations took for granted—such as the hazards related to medicine, transportation, and industry—have led to government regulation and to the creation of public interest groups to control hazards that were previously tolerated [**?**, **?**]. System safety engineers, who used to focus on uncontrolled-energy accidents such as explosions or the inadvertent firing of a weapon, are now being asked to control new hazards such as air and ground pollution and to ensure that the systems we build do not produce, contain, or decompose into hazardous materials [**?**].

In writing about the Bhopal tragedy, Bogard expresses this new attitude:

> We are not safe from the risks posed by hazardous technologies, and any choice of technology carries with it possible worst case scenarios that we must take into account in any implementation decision. The public has the right to know precisely what these worst case scenarios are and participate in all decisions that directly or indirectly affect their future health and well-being. In many cases, we must accept the fact that the result of employing such criteria may be a decision to forego the implementation of a hazardous technology altogether [**?**, p.109].

Increased regional and national concern about safety has expanded in the past decade to include international issues. The greenhouse effect, acid rain, and accidents such as the release of radiation at Chernobyl do not recognize national borders. The global economy and the vast potential destructiveness of some of our technological achievements are forcing us to recognize that risk can have international implications and that control of risk requires cooperative approaches.

Because risk reduction can be expensive and often requires tradeoffs with other desirable goals, it is important to consider whether the increased public and governmental concern about technological risk is justified and whether engineers, both hardware and software, should be worrying about these problems at all. Is risk increasing in our modern society as a result of new technological achievements, or, on the other hand, are we experiencing a new and unjustified form of Luddism?[1]

[1] The Luddite disturbances occurred in Yorkshire between 1811 and 1816, when workers in

1.2 Is Increased Concern Justified?

Determining whether technological risk is increasing or not depends on the data used and its interpretation. On the one hand, Harris and colleagues argue that technological hazards, in terms of human mortality, were greater in the earlier, less fully managed stages of industrial development [**?**]. They cite data from the National Safety Council showing that occupational death and injury rates have declined steadily since the early part of this century [**?**] and conclude that technological hazard mortality is not currently rising. However, they warn that "the positive effects of technology have for some time reached their maximum effect on human mortality, while the hazards of technology continue partially unchecked, affecting particularly the chronic causes of death that currently account for 85 percent of mortality in the U.S.A." [**?**].

On the other hand, examination of the technological accident rate, rather than the occupational death and injury rate suggests that technological risk *is* increasing. Sixty percent of all the major industrial disasters from 1921 to 1989 occurred after 1975 [**?**]. Writing in 1989, Bogard argued that 12 of the 19 major industrial accidents in the twentieth century involving 100 or more deaths occurred after 1950. When we include smaller-scale incidents, transportation accidents, dam breaks, and structural collapses, the evidence supporting the hypothesis of increasing risk is even more compelling.

Complicating things further, although the total technological accident rate has been increasing, accident rates in specific *types* of systems have been decreasing. For example, the military aviation accident rate has generally improved over time. This improvement has been attributed to an emphasis on system safety programs and concerted efforts to eliminate and control hazards [**?**]. Thus, the experience of military aviation seems to support an argument that technological advances need not increase risk if efforts are made to control it. Recently, however, the decrease in military aviation accident rates attributed to the use of system safety techniques has slowed. One explanation for this slowing may simply be the naturally increasing difficulty in finding ways to make large improvements. However, other, less obvious factors may be at work here.

Clearly, we will not find the answer to our question in such ambiguous and contradictory statistical data. In fact, the major changes occurring in the post–World War II era make most long-term historical risk data inapplicable to today's world: Past experience does not allow us to predict the future

the English woolen industry tried, through violence, to stem the increasing mechanization of the mills. *Luddism* has become a generic term describing opposition to technological innovation.

when the risk factors in the present and future differ from those in the past. Examining these changes will help us understand the problems we face.

1.3 Unique Risk Factors in Industrialized Society

Risk is a combination of the *likelihood* of an accident and the *severity* of the potential consequences. Risk increases if either the likelihood or the magnitude of loss increases (as long as the other component does not decrease proportionally). Different factors may affect these two components of risk. Some factors that are particularly relevant today are the appearance of new hazards and the increasing complexity, exposure, energy, automation, centralization, scale, and pace of technological change in the systems we are attempting to build.

1.3.1 The Appearance of New Hazards

Before the Industrial Revolution, accidents were the result of natural causes or involved a few relatively well-understood, simple technological devices. In the twentieth century, scientific and technological advances have reduced or eliminated many risks that were once commonplace. Modern medicine, for example, has provided cures for previously fatal diseases and eliminated some scourges, such as smallpox, altogether.

At the same time, science and technology have also created new hazards. Misuse and overuse of antibiotics has given rise to resistant microbes. Children no longer work in coal mines or as chimney sweeps, but they are now exposed to man-made chemicals and pesticides in their food or to increased environmental pollution [**?**]. The harnessing of the atom has increased the potential for death and injury from radiation exposure.

Some of the new hazards are more pervasive and harder to find and eliminate than were the ones eliminated or reduced in the past [**?**]. In addition, we have no previous experience to guide us in handling these new hazards. Much of what has been learned from past accidents is passed down through codes and standards of practice. But appropriate codes and standards have not yet been developed for many new engineering specializations and technologies. Sometimes lessons learned over centuries are lost when older technologies are replaced by newer ones, for example when digital computers are substituted for mechanical devices.

Many of the approaches that worked on the simpler technologies of the past—such as replication of components to protect against individual component failure—are ineffective in controlling today's complex risks. Although redundancy provides protection against accidents caused by individual component failure, it is not as effective against hazards that arise from the interactions among components in the increasingly complex and interactive systems being engineered today. In fact, redundancy may increase complexity to the point where the redundancy itself contributes to accidents.

1.3.2 Increasing Complexity.

Many of the new hazards are related to increased complexity (both product and process) in the systems we are building. Not only are new hazards created by the complexity, but the complexity makes identifying them more difficult.

Perrow distinguishes between accidents caused by component failures and those, which he calls *system accidents*, that are caused by interactive complexity in the presence of tight coupling [**?**]. High-technology systems are often made up of networks of closely related subsystems. Conditions leading to hazards emerge in the interfaces between subsystems, and disturbances progress from one component to another. As an example of this increasingly common type of complexity, modern petrochemical plants often combine several separate chemical processes into one continuous production, without the intermediate storage that would decouple the subsystems [**?**].

In fact, analyses of major industrial accidents invariably reveal highly complex sequences of events leading up to accidents, rather than single component failures. Whereas in the past, component failure was cited as the major factor in accidents, today more accidents result from dangerous design characteristics and interactions among components [**?**].

The operation of some systems is so complex that it defies the understanding of all but a few experts. Increased complexity and coupling make it difficult for the designer to consider all the hazards, or even the most important ones, or for the operators to handle all normal and abnormal situations and disturbances safely [**?**].

Not only does functional complexity make the designer's task more difficult, but the complexity and scope of the projects require numerous people and teams to work together. The anonymity of team projects dilutes individual responsibility [**?**]. Moreover, many new specializations do not have standards of individual responsibility and ethics that are as well developed as those of older professions.

Kletz points out the paradox that people are willing to spend money on complexity but not on simplicity [**?**]. Consider the following accident that occurred in a British chemical plant. In this plant, a pump and various pipelines had several different uses, which included transferring methanol from a road tanker to storage, charging it to the plant, and moving recovered methanol back from the plant. A computer set the various valves, monitored their positions, and switched the transfer pump on and off.

On this particular occasion, a tank truck was being emptied:

> The pump had been started from the control panel but had been stopped by means of a local button. The next job was to transfer some methanol from storage to the plant. The computer set the valves, but as the pump had been stopped manually it had to be started manually. When the transfer was complete the computer told the pump to stop, but because it had been started manually it did not stop and a spillage occurred [**?**, p.225].

In this case, a simpler design—independent pipelines for different functions, which were actually installed after the spill—makes errors much less likely and may not be more expensive over the lifetime of the equipment.

Computers often allow more interactive, tightly coupled, and error-prone designs to be built, and thus may encourage the introduction of unnecessary and dangerous complexity. Kletz has noted, "Programmable electronic systems have not introduced new forms of error, but by increasing the complexity of the processes that can be controlled, they have increased the scope for the introduction of conventional errors" [**?**]. If we accept Perrow's argument that interactive complexity and coupling are a cause of serious accidents, then the introduction of computers to control dangerous systems may increase risk unless great care is taken to minimize complexity and coupling.

1.3.3 Increasing Exposure

Not only is our *technology* becoming more complex, but our *society* has become more complex, interdependent, and vulnerable [**?**]. The consequences of an accident depend not only on the hazard itself but also on the exposure of the hazard—that is, the length of time and the environment within which the hazard exists.

More people may be exposed to a given hazard today than were previously. Passenger capacity in aircraft, for example, is increasing to satisfy economic concerns. Siting of dangerous facilities near large populations is increasing as more people move to cities and larger plants need larger workforces within commuting distance. Interdependence and complexity can cause ripple effects beyond the immediate exposure area of the hazard, magnifying the potential consequences of accidents.

1.3.4 Increasing Amounts of Energy

Another factor related to increased risk is the discovery and use of high-energy sources—such as exotic fuels, high-pressure systems, and atomic fission—which have increased the magnitude of the potential losses. Other new systems use more conventional energy sources, but they involve technology that requires larger amounts of energy than was required in the past.

The larger amounts of energy increase both the surrounding area potentially affected by an accident and the amount of damage possible. New hazards that can cause genetic damage and environmental contamination introduce the potential for affecting not only the current generation but our descendants as well.

1.3.5 Increasing Automation of Manual Operations

Although it might seem that automation would decrease the risk of operator error, the truth is that automation does not remove people from systems—it merely moves them to maintenance and repair functions and to higher-level supervisory control and decision making [?]. The effects of human decisions and actions can then be extremely serious. At the same time, the increased system complexity makes the decision-making process more difficult.

Automation often removes the operator from the immediate control of the energies of the system [?]. Consequently, an individual moving within the physical system may find it difficult to anticipate the possible energy flows—for example, the physical behavior of an industrial robot. This difficulty is enhanced in systems that use exotic chemical and physical processes that are not well understood.

In addition, operators in automated systems are often relegated to central control rooms, where they must rely on indirect information about the system state: This information can be misleading. In 1977, New York City had a massive and costly power blackout [?]. When the operator followed prescribed procedures to handle the initial symptoms, the electrical system was brought to a complete halt. The operator did not know that there had been two relay failures—one that would lead to a high flow of current over a line that normally carried little or no current (and thus would have alerted the operator to the real problem) and a second relay failure that blocked the flow over the line, making it appear normal. The operator was unaware of the particular set of circumstances that made the zero-current reading abnormal and treated it as normal. Lack of direct information increases the probability of such faulty hypotheses. Operators then become the scapegoat when accidents are blamed on human error, even though the "error" was induced by features of the automated system.

These problems will only get worse with the current trend toward decentralization of control. Microprocessors embedded in the plant or system are taking over most control functions, with only high-level information being fed back to the central control room or control point. This design limits even further the operator's options and hinders broad comprehension of the system state [?].

A control loop, by its very nature, masks the occurrence and subsequent development of a malfunction precisely because it copes with the immediate effects of the problem, at least for a time [?]. But this masking does not continue indefinitely: When the malfunction is finally discovered, it may by then be more difficult to control, or the symptoms may be hidden or distorted. By the time a human gets involved, the symptoms may be referred forward or backward by the major loops in the overall process. For example,

> In 1985, a China Airlines 747 suffered a slow loss of power from its outer right engine. This would have caused the plane to yaw to the right, but the autopilot compensated, until it finally reached the limit of its compensatory abilities and could no longer keep the plane stable. At that point, the crew did not have enough time to determine the cause of the problem and to take action: the plane rolled and went into a vertical dive of 31,500 feet before it could be recovered. The aircraft was severely damaged and recovery was much in doubt [?, p.138].

The energy saving systems incorporated in many process plants, particularly during the 1970s, to conserve energy and to improve thermal economy complicate this problem further [?]. For example, heat generated by a process might be recovered through a complex exchange system. As is often the case, multiple goals—in this case safety and economy—lead to conflicts. The energy-saving systems introduce component interactions that make the functioning of the system less transparent to its designers and operators [?]. From the designers' standpoint, systematic analysis and prediction of events that could lead to accidents become increasingly difficult. From the operators' standpoint, the extra complexity makes diagnosis of problems more difficult and again leads to masking and referral of symptoms: The place in the plant where signs of trouble first emerge may not be where the problems occurred [?].

1.3.6 Increasing Centralization and Scale

Increasing automation has been accompanied by centralization of industrial production in very large plants and the potential for great loss and damage to people, equipment, and the environment. For many decades, plant size has been increasing, resulting in a change of scale in production: Devices and whole processes are being extrapolated into untested areas. In nuclear power, for example, Bupp and Derian observed that by 1968, manufacturers were taking orders for plants six times larger than the largest one then in operation. "And this was in an industry which had previously operated on the belief that extrapolations of two to one over operating experience were at the outer boundary of acceptable risk" [**?**, p.73]. The Browns Ferry Nuclear Power Plant, which was the site of a serious accident in 1975, was 10 times the size of any plant already in operation at the time its construction began in 1966. In fact, it was to become one of the world's largest electrical generating facilities [**?**].

Ocean shipping is another industry experiencing enormous changes in scale that are alien to its previous caution and conservatism. In order to maximize profits, supertankers are being built without the sound design and redundant systems (such as double hulls) they once had. Mostert writes about these superships:

> The gigantic scale of vessels creates an abstract environment in which crews are far removed from direct experience of the sea's unforgiving qualities and potentially hostile environment. Heavy automation undermines much of the old-fashioned vigilance and induces engineers to lose their occupational instincts—qualities that in earlier days of shipping were an invaluable safety factor [**?**].

1.3.7 Increasing Pace of Technological Change

A final risk factor is the increasing pace of technological change in this century: The average time required to translate a basic technical discovery into a commercial product has decreased from 30 years during the early part of this century (1880–1919) to 5 or fewer years today. In addition, the number of new products or processes is increasing exponentially [**?**]. The twentieth century has seen a major acceleration in the growth of new industries stemming from scientific and technological innovation, such as gene splicing. Dangerous substances are being handled on an unprecedented scale, and economic pressures often militate against extensive testing [**?**].

The increased pace of change lessens opportunity to learn from experience. Small-scale and relatively nonhazardous systems can evolve gradually by trial and error. But learning by trial and error is not possible for many modern products and processes because the pace of change is too fast and the penalties of failure are too great. Design and operating procedures must be right the first time when there is potential for a major fire, explosion, or release of toxic materials. Nuclear energy is just one example. Christopher Hinton, who was in charge of the first British atomic energy installations, said in 1957,

> All other engineering technologies have advanced not on the basis of their successes but on the basis of their failures. The bridges that collapsed... have added more to our knowledge of bridge design than the ones which held; the boilers that exploded more than the ones that had no accidents.... Atomic energy, however, must forego this advantage of progressing on the basis of knowledge gained by failure (quoted in [**?**, p.53]).

As a result, empirical design rules and equipment standards are being replaced by reliance on hazard identification and control or by attempts to build ultrareliable systems that never fail. The feasibility of accomplishing either of these goals and the amount of protection they afford are unknown in comparison to the older methods involving learning by experience and using well-tested standards and guidelines.

Given the difficulty that industry is having in coping with all the changes, it is not surprising that government agencies charged with the licensing and safety monitoring of these industries are also having problems keeping pace. New technology, such as digital computers, requires new standards, regulatory procedures, and expertise that take time to develop and perfect. Industry and society are unwilling to slow progress while these agencies catch up.

1.4 How Safe Is Safe Enough?

We are back where we started—with no real proof that a problem exists but with much supporting evidence. Examining the risk factors described, a strong argument can be made that concern is justified.

Because of the recent changes and unique conditions for which historical experience does not apply, the nonoccurrence of particular types of accidents in the past is no guarantee that they will not occur in the future. In addition, if we must learn from accidents and if failure teaches us more than success (as Petroski [**?**] and others have argued), then what can we do about systems in which a single accident can have such tragic consequences that the process of learning from accidents is unacceptable to society?

The system safety approach to reducing risk is to anticipate accidents and their causes in before-the-fact hazard analyses (rather than relying on after-the-fact accident investigations) and to eliminate or control hazards as much as possible throughout the life of a system. The goal is to understand and manage risk in order to eliminate accidents or to reduce their consequences. Frola and Miller claim that system safety investment has reduced losses where it has been applied rigorously in military and aerospace programs [**?**].

Unfortunately, hazards will never be eliminated completely from all systems. In addition to the technical difficulty of anticipating and reducing risks, there is the basic problem of conflicting goals. Desirable qualities tend to conflict with each other, and design tradeoffs between safety, performance, and other goals are required. A large industrial robot arm, for example, carrying a heavy load at high speed cannot be stopped quickly in emergencies without damaging the arm. The longer it takes the arm to stop in response to a deadman switch or other safety device, the less the wear on the arm but also the more likely that the arm will hit something before it stops [**?**]. Likewise, human–machine interfaces that are designed to be easy to use often are less safe. For example, computer input errors can be reduced if operators are required to repeat operations, but that requires extra time and seemingly wasted effort.

Designing a system to protect against a variety of hazards is clearly possible, but designing a system to protect against all hazards, no matter how perverse or remote, might require making so many compromises in functionality and other goals that the system is not worth building at all [**?**]. Finding the right balance is difficult.

Wolf suggests that we may be "in the presence of a contradiction of technological culture that can neither prevent potentially disastrous accidents nor accept their consequences" [**?**]. But even if the number of accidents and their associated losses increase, we are unlikely to abandon the new and risky systems that represent technological progress. The benefits of technology usually come with disadvantages, and society is unwilling to live without many of those benefits. If this assumption is correct, then the process for determining exactly which systems to build and what new technology to introduce into them becomes critical.

Several ways of making this decision have been suggested. At one extreme is an anti-technology stance that blames accidents on new technology alone and concludes that advanced technology should not be used in dangerous systems. Such a simple, negative stance is not the solution to complex engineering and ethical problems. At the opposite extreme is the pro-technology position that all new technology is good: If it can be built, then it should be. Those who hold this position often put the blame for accidents on humans and assume that risks will be reduced by replacing human operators by computers. Again, this position is oversimplistic.

The prevailing position in our society is the utilitarian view that the only reasonable way to make technology and risk decisions is to use risk–benefit analysis. This belief is so widespread that we often accept risk–benefit analysis as the *only* way to make technology and risk decisions, without realizing that there are alternatives.

1.4.1 Risk–Benefit Analysis and the Alternatives

To utilitarians, catastrophic accidents (such as Bhopal) are one of the risks of high technology, which, in the long run, are outweighed by the technology's overall benefits. Decisions can be made by comparing these risks and benefits.

To apply this approach, we must be able (1) to measure risk and (2) to choose an appropriate level for decision making. Unfortunately, it is impossible to measure risk accurately, especially before a system is built: Systems must be designed and built while knowledge of their risk is incomplete or nonexistent. Even if past experience could be used, it might not be a valid predictor of future risk unless the system and its environment remain static, which is unlikely. Small changes may substantially alter the risk involved [**?**].

Risk assessment tries to solve this dilemma. The goal is to quantify risk, including both likelihood and severity of a loss, before historical data is available. The accuracy of such risk assessments is controversial. William Ruckelshaus, former head of the U.S. Environmental Protection Agency, argues that the current use of risk assessment data is a kind of pretense: "To avoid paralysis resulting from waiting for definitive data, we assume we have greater knowledge than scientists actually possess and make decisions based on those assumptions" [**?**, p.110].

In fact, risk assessment may never be able to provide definitive answers to these types of risk questions. Estimates for extremely unlikely events, such as a serious nuclear reactor accident, can never have the same scientific validity as estimates of events for which there are abundant statistics. Because the required probabilities are so small (such as 10^{-7} per reactor per year), there is no practical possibility of determining this failure rate directly—that is, by building 1,000 reactors, operating them for 10,000 years, and tabulating their operating histories [**?**].

Instead, probabilities of serious accidents are calculated by constructing models of the interaction of events that can lead to the accident. In practice, the only events included are those that can be measured. At the same time, the causal factors involved in most major accidents (see the appendices for some examples) are almost all unmeasurable. The difficulty of performing risk assessments is discussed in later chapters of this book. In brief, the technique is controversial, and the results are far from universally accepted as meaningful.

Even if risk could be measured, there is still the problem of choosing the level of risk to use in decision making. The most common criterion is that of *acceptable risk*: A threshold level is selected below which risk will be tolerated. But who determines what level of risk is acceptable in comparison to the benefits?

Often, those getting the benefits are not those assuming the risks. The people who are negatively affected are rarely asked their opinion, especially when they are not represented by an influential lobby or trade association [**?**]. The attitude of some decision makers is reflected in a statement by the director of Electricité de France who explained French secrecy about nuclear power this way: "You don't tell the frogs when you are draining the swamp" [**?**, p.156]. Hence, along with technical problems, utilitarianism and risk–benefit analysis present many philosophical and ethical dilemmas.

The moral implications of risk–benefit analysis are epitomized by the Ford Pinto gas tank case. Reportedly, Ford knew of the danger of explosion upon impact. But after doing a risk analysis weighing the cost of fixing the gas tank against the incidence of rear-end collisions and the damages usually assessed in wrongful death law suits at that time, the company decided that it would be cheaper to settle lawsuits after explosions than to fix the gas tank design. Here the benefits went to Ford while the (nonmonetary) risks were unknowingly assumed by the drivers and passengers of the Pinto. Admiral Bobby Inman argued in the wake of the *Challenger* accident that "There is a difference between risks taken with the unknown and risks taken to save on costs" [**?**, p.58]. Perrow suggests that the issue ultimately is not risk but power—the power to impose risks on the many for the benefit of the few [**?**].

Another moral difficulty with risk–benefit analysis involves selecting a common unit of measurement to compare losses and benefits. Usually, dollars are chosen. This choice raises the question of whether human suffering should simply be regarded as a cost and assigned a dollar value. Moreover, there is the problem of how to do this assignment. The most common approach is to use the amount of money the person would have earned from the point of death to his or her statistical life expectancy. Thus, a young, healthy, high-earning person would be worth more than a young, low-earning person or an older person close to retirement. The moral difficulties with this approach to assigning dollar values to human life are obvious, especially if you consider what dollar value you personally would place on the life of your child or spouse.

Alternatives to the use of acceptable risk have been proposed. *Optimal risk* involves a tradeoff that minimizes the sum of all undesirable consequences [?]. Optimal risk is achieved when the incremental or marginal cost of risk reduction equals the marginal reduction in societal costs—that is, where the sum of the cost of risk abatement and the expected losses from the risk is at a minimum. Estimating expected losses, however, still requires the ability to make probabilistic likelihood estimates of accidents and losses.

In *Normal Accidents* [?], Perrow offers a model for making decisions that attempts both to limit the potential for accidents resulting in large numbers of deaths and to minimize the effects of abandoning high-risk technology. This model uses catastrophic potential (severity) and cost of alternatives, but not probabilistic likelihood. Perrow views accidents as normal and therefore inevitable in complex systems. According to this perspective, we should assume that accidents will occur and make decisions accordingly, rather than assume that accidents will not occur on the basis of low probability estimates.

Perrow divides high-risk systems into three categories. The first category includes those systems with either low catastrophic potential or high-cost alternatives, such as chemical plants, aircraft, air traffic control, dams, mining, and automobiles. These systems should be tolerated and could be further improved with modest effort. The second category includes those technologies with moderate catastrophic potential and moderate-cost alternatives. These are the systems that we are unlikely to be able to do without (such as marine transport) or where the expected benefits are substantial (such as recombinant DNA). Perrow suggests that these systems should be strictly regulated. The final category includes systems with high catastrophic potential and relatively low-cost alternatives. He places nuclear weapons and nuclear power in this group and argues that the systems in this final category should be abandoned and replaced.

One problem with Perrow's approach is that it does not provide any way to make decisions about specific systems; it deals only with large classes of systems. An alternative is to require that accident rates not be increased when new technology is introduced. If, for example, accidents have occurred in a specific type of system at a certain historical rate, then the new technology should only be required to achieve an equivalent rate to be considered acceptable. This approach is based on the belief that if the public currently accepts a technology with a particular accident rate, then they will continue to accept this level of risk.

Aside from the technical problem of how to determine that the accident rate in the new system will be equivalent, difficult moral problems again arise if the new technology has the potential to reduce the accident rate but this reduction requires tradeoffs and increased costs. From an ethical standpoint, equivalent safety in this case may not be adequate. Consider air bags and other improvements in automobile safety: By the "acceptable risk is what has been accepted by the public previously" argument, such safety improvements are unnecessary, since people apparently accept the current risk by their willingness to drive.

The use of computers, in particular, may offer a potential increase in safety, but, at the same time, allow a decrease in safety margins; their use, therefore, provides the possibility of economic or productivity benefits along with the same historic level of safety. Should equivalent risk levels be accepted when reduced risk is possible? Even worse, do computers really reduce risk as much as is assumed when these types of tradeoff decisions are made?

It appears that there are no entirely satisfactory methods for making these decisions. Part of the explanation for this lack of mathematical and engineering solutions is that the decisions involve deep philosophical and moral questions—not simply technical choices.

1.4.2 Trans-Scientific Questions

A basic problem with utilitarian approaches is that they attempt to use scientific methods and arguments to answer what are fundamentally *not* scientific questions. Alvin Weinberg, former head of Oak Ridge National Laboratory, writes,

> Many of the issues that lie between science and politics involve questions that can be stated in scientific terms but that are in principle beyond the proficiency of science to answer.... I proposed the term "trans-scientific" for such questions... Though they are, epistemologically speaking, questions of fact and can be stated in the language of science, they are unanswerable by science; they transcend science... In the current attempts to weigh the benefits of technology against its risks, the protagonists often ask for the impossible: scientific answers to questions that are trans-scientific [**?**].

Even though scientists cannot provide definitive answers to such risk questions, they still have an important role to play: to provide what scientific information they can about the question at hand and to make clear where science ends and trans-science begins. Weinberg contends that the debate on risks versus benefits would be more fruitful if we recognized those limits.

Making decisions such as how safe is safe enough involves addressing moral, ethical, philosophical and political questions that cannot be answered fully by algebraic equations or probabilistic evaluations. "Scientific truth is established in the traditional methods of peer review: only what has value in the intellectual marketplace survives. By contrast, where trans-science is involved, wisdom (rather than truth) must be arrived at by some other mechanism" [?].

Although scientists and engineers can legitimately disagree about the extent and reliability of their expertise, they often appear reluctant to concede limits on their ability to answer what are essentially trans-scientific questions. Answering such questions involves moral and aesthetic judgments: they deal not with what is *true* but rather with what is *valuable.* As such, where there is no consensus on these values, the decisions must be made by political processes.

Unfortunately, conflicts of value present special difficulties for the predominantly scientific and technocratic modes of rationality of Western society [?]. Cotgrove argues that beliefs about risk are embedded in complex belief and value systems that constitute distinct cultures: The way individuals see the world and evaluate risk is part of this culture. Until recently, the master value of our industrial society has been wealth creation—the overall goal for society was taken for granted to be maximizing economic growth and the production of goods and services. Today, this dominant value system is starting to be rejected by some members of our society. Table 1.1 outlines some of the features of these competing social paradigms [?].

For groups that hold such different paradigms, communication is nearly impossible: They essentially inhabit different worlds. What is rational and reasonable from one perspective is irrational from another. Each side is unable to comprehend alternative viewpoints, which requires, in Kuhn's terms [?], a paradigm shift. If the goal is maximizing output, for example, then not only are nuclear risks justified, but it would be unreasonable not to take them. From a different perspective about how the world works and what kind of society is desirable, to take even the possibly small—but in practice incalculable—risks for future generations stimulates moral indignation. Perhaps the explanation behind scientists' often-expressed frustration with the "irrational" way many people evaluate risk is just that different groups evaluate it from very different cultural viewpoints, all of which are rational within their different contexts.

Because of the futility of attempting to change deep-rooted cultural beliefs, trans-scientific questions will not be covered in this book. However, it is important to point out which issues are truly scientific and which are trans-

	Dominant Technocratic Paradigm	**Alternative Environmental Paradigm**
Core values	Material (economic growth)	Nonmaterial (self-actualization)
	Natural environment valued as a resource	Natural environment intrinsically valued
	Domination over nature	Harmony with nature
Economy	Market forces	Public interest
	Risk and reward	Safety
	Individual, self-help	Collective, social provision
Form of governing	Authoritarian (experts influential)	Participatory (citizen, worker involvement)
	Hierarchical	Nonhierarchical
Society	Centralized	Decentralized
	Large-scale	Small-scale
	Ordered	Flexible
Nature	Ample reserves	Earth's resources limited
	Nature hostile or neutral	Nature benign
	Environment controllable	Nature delicately balanced
Knowledge	Confidence in science and technology	Limits to science and technology
	Rationality of means	Rationality of ends
	Separation of fact from value, thought from feeling	Integration of fact and value, thought and feeling

Table 1.1: Alternative social paradigms (Adapted from Stephen Cotgrove. Risk, value conflict, and political legitimacy. In Richard F. Griffiths, editor, *Dealing with Risk: The Planning, Management, and Acceptability of Technical Risk*, Manchester University Press, Manchester, England, 1981, page 129.)

scientific so that communication lines can be kept open. We must also realize that decisions about safety will cause legitimate disagreements that cannot be resolved by simple utilitarian arguments. Some opposition to our technological inventions goes beyond questions of measurable risks and economic benefits and instead focuses on social, political, and psychological risk and intangible, unmeasurable risk considerations [**?**]. According to Weinberg, it is the scientist's duty to inject some order into this often chaotic debate by distinguishing scientific from trans-scientific issues. An attempt is made throughout this book to identify these trans-scientific problems.

Chapter 2

Computers and Risk

We seem not to trust one another as much as would be desirable. In lieu of trusting each other, are we putting too much trust in our technology?. . . Perhaps we are not educating our children sufficiently well to understand the reasonable uses and limits of technology

— T.B. Sheridan
Trustworthiness of Command and Control Systems

And they looked upon the software, and saw that it was good. But they just had to add this one other feature. . .

— G.F. McCormick
When Reach Exceeds Grasp

Just as James Watt's invention of the first practical steam engine fueled the Industrial Revolution, the invention of the first practical computer 50 years ago has drastically altered our society. The uniqueness and power of the digital computer over other machines stems from the fact that, for the first time, we have a general-purpose machine (Figure 2.1). We no longer need to build a mechanical or analog autopilot from scratch, for example, but simply to write down the "design" of an autopilot in the form of instructions or steps to accomplish the desired goals. These steps are then loaded into the computer, which, while executing the instructions, in effect *becomes* the special-purpose machine (the autopilot). If changes are needed, the instructions can be changed instead of building a different physical machine from scratch.

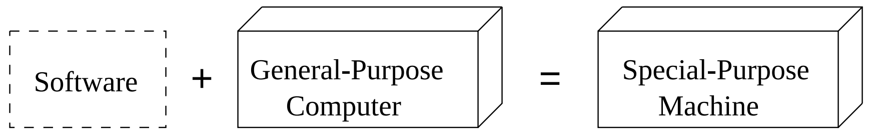


Figure 2.1: Software plus a general purpose computer creates a new special purpose machine.

Machines that previously were physically impossible or impractical to build become feasible, and the design of a machine can be changed quickly without going through an entire retooling and manufacturing process. In essence, the manufacturing phase is eliminated from the lifecycle for these machines: The physical parts of the machine can be reused, leaving only the design and verification phases.[1] The design phase also has changed: Emphasis is placed only on the steps to be achieved without having to worry about how those steps will be realized physically.

The advantages of computers have led to an explosive increase in their use, including their introduction into potentially dangerous systems. This chapter explores the role of computers in accidents, some of the myths related to their use, and why we seem to have so much difficulty with the engineering of software.

2.1 The Role of Computers in Accidents

Few systems today are built without computers to provide control functions, to support design, and sometimes to do both. Computers now control most safety-critical devices, and they often replace traditional hardware safety interlocks and protection systems—sometimes with devastating results. Even if the hardware protection devices are kept, software is often used to control them. Frola and Miller describe the problem of computer-related hazards in military aircraft:

> A relatively new breed of hazards and associated problems has appeared. They appear primarily in flight control systems, armament control systems, navigation systems, and cockpit displays. They add new dimensions to the human-error problem. Some of

[1]Although duplication of software might be considered to be manufacturing, it is usually a relatively trivial process.

> the hazards are passive until just the right combination of circumstances arrives.. . . Some result from the crew's multitude of choices in aircraft system management, often during prioritization of tasks. Conversely, computer-based systems are supposed to relieve pilot workload, but perhaps too much so in some instances, with resultant complacency and/or lack of situation awareness [**?**, p.7-13].

Computers can be used in safety-critical loops in several ways. Figure 2.2 shows some of these uses, including:

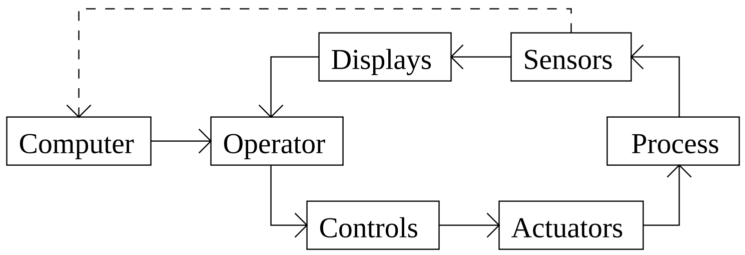
1. Providing information or advice to a human controller upon request (2.2a)
2. Interpreting data and displaying it to the controller who makes the control decisions (2.2b)
3. Issuing commands directly, but with a human monitor of the computer's actions providing varying levels of input (2.2c)
4. Eliminating the human from the control loop completely (2.2d).

Even if the human is eliminated from direct control, the computer still needs to be supervised: The computer closes the control loop, but humans may be assigned the role of setting initial parameters, making intermittent adjustments, and receiving information from the computer.

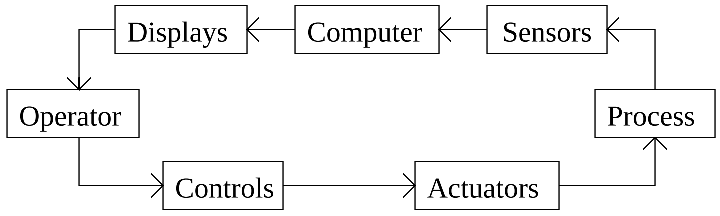
The safety implications of computers exercising direct control over potentially dangerous processes are obvious (Figure 2.3a). Less obvious are the dangers when (as depicted in Figure 2.3b)

1. Software-generated data is used to make safety-critical decisions (such as air traffic control and medical blood analyzers)
2. Software is used in design analysis (such as CAD/CAM)
3. Safety-critical data (such as blood bank data) is stored in computer databases.

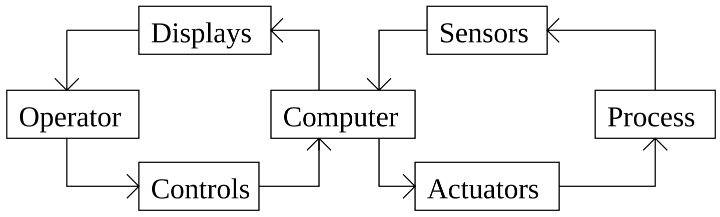
The FDA has received reports of software errors in medical instruments that led to mixing up patient names and data, as well as reports of incorrect outputs from laboratory and diagnostic instruments (such as patient monitors, electrocardiogram analyzers, and imaging devices) [**?**]. In 1979, the discovery of an error in the software used in the design of nuclear reactors and their supporting cooling systems resulted in the Nuclear Regulatory Commission's temporary shutdown of five nuclear power plants that did not satisfy earthquake standards. There is a serious danger of overreliance on the accuracy of computer outputs and databases.



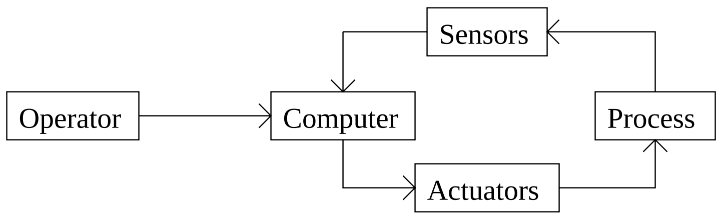
(a) Computer provides information and advice to controller perhaps by reading sensors directly.



(b) Computer reads and interprets sensor data for operator.

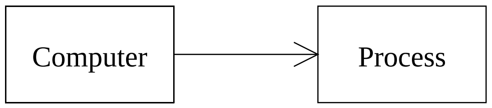


(c) Computer interprets and displays data for operator and issues commands; operator makes varying levels of decisions.

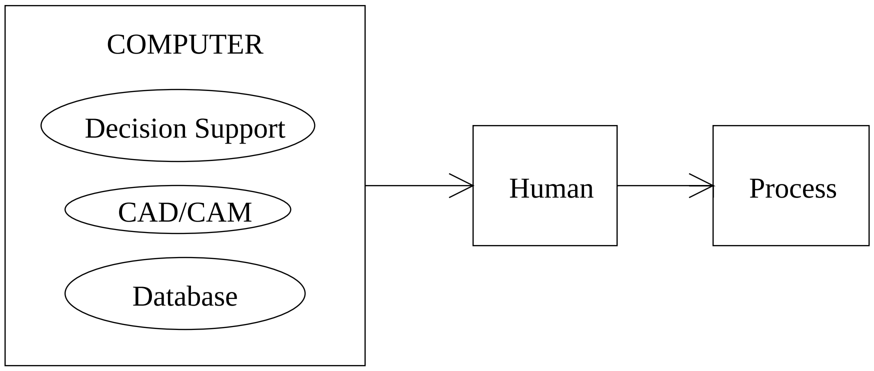


(d) Computer assumes complete control of process with operator providing advice or high-level direction.

Figure 2.2: Alternative uses of computers in control loops.



(a) Direct Control



(b) Indirect Control

Figure 2.3: Direct and indirect control.

In some cases, companies and government agencies have argued that software that generates data but does not make decisions, such as air traffic control software, is not safety critical or is less so than direct-control software because the human controller makes the ultimate decision, not the computer:

> If diagnostic devices produce incorrect results, the errors may be readily noticed or may be inconsistent with other clinical signs. Thus, the risk to the patient is less than in the case of software-driven devices that directly affect patients [**?**, p.6].

Although risk *may* be reduced by the use of a human intermediary, this reduction is by no means assured. Often, information from a computer or technological device is more readily believed than conflicting information that is directly observed. Even more frequently, systems are built that require the operator to rely on computer-generated information that the human has no independent way to check. In almost all cases, risk from incorrect computer operation is not eliminated by having a human in the loop, and it may not even be reduced.

Arguments about who makes the ultimate decision and therefore is responsible may be appropriate in a courtroom when affixing blame and determining financial liability, but not when the goal is to build safer systems. Engineers need to consider all the contributors to accidents if their efforts to reduce risk are to be effective. Often the argument that software providing information or advice to humans is not safety critical is used to avoid the difficult task of ensuring the safety of the software. If system safety truly is to be increased, then all the components whose operation can *directly* or *indirectly* affect safety must be considered, and the related hazards must be eliminated or reduced.

In addition to software's direct and indirect contributions to accidents, computers also add difficulty and cost to accident investigations. New procedures are required because, unlike failures of mechanical devices, "malfunctioning electrons will not be found in the wreckage" [**?**]. In the case of the Therac-25 medical accelerator, overdoses were first denied and not investigated or were attributed to transient hardware failures [**?**]. Even if the possibility of software error *is* investigated, subtle errors that cause accidents in well-tested and sometimes long-used systems are not easy to find (or to prove that they may or may not exist). One software error cost millions of dollars to investigate after it caused the loss of an F-14 military aircraft [**?**].

The wide-spread use of computers in safety-critical systems is creating new problems for software and system engineers. Methods to ensure the safety of computer-controlled systems have lagged behind the development of these systems. Proven system safety engineering techniques do not include software, and, because of the unique characteristics of this new technology, are not easily adapted to software. In addition, because of the relatively recent introduction of computers to control potentially dangerous systems and the relatively safe nature of the computer itself (in terms of explosion, fire, or other direct hazards), few software engineering techniques have been developed to cope with safety problems. For the most part, standard software engineering techniques and processes are being used to develop safety-critical software without any consideration of the special factors and unique requirements for enhancing safety.

Communication problems are making efforts to rectify these deficiencies more difficult. System and software engineers have few common models or tools, and even their vocabulary is different. Computer science, by developing separately from engineering, has created its own technical vocabulary—sometimes the two groups use different names for the same things or they give the same name to different things. Kletz has described the problems operators, engineers, and programmers have in working together:

> Operating and design staff have been known to complain that programmers resent being watched or checked and that they produce programs that are not resistant to mistakes, cannot tolerate plant errors, and are difficult to understand. No doubt programmers make similar remarks about operators and designers [**?**, p.1].

2.2 Software Myths

If there are problems, why are computers being used so widely? The basic reason is that computers provide a level of power, speed, and control not otherwise possible; they are also relatively light and small. Many other supposed advantages of using computers are myths, however, usually stemming from looking only at the short term. Understanding these myths is important if we are to make competent decisions about using computers to control safety-critical processes.

Myth 1. *The cost of computers is lower than that of analog or electromechanical devices.*

Reality: This myth, like most myths, has some superficial truth: Microcomputer hardware is cheap relative to other electromechanical devices. However, the cost of writing and certifying highly reliable and safe software to make that microprocessor useful, together with the cost of maintaining that software without compromising reliability and safety, can be enormous. The onboard Space Shuttle software, for example, while relatively simple and small (about 400,000 words) compared to more recent control systems, costs NASA approximately $100,000,000 a year to maintain [**?**].

Designing an electromechanical system is usually much easier and cheaper, especially when standard designs can be used. Of course, software *can* be built cheaply, but then lifetime costs—including the cost of accidents and required changes when errors are found—increase and may become exorbitant.

Myth 2. *Software is easy to change.*

Reality: Again, this myth is superficially true: Changes to software *are* easy to make. Unfortunately, making changes without introducing errors is extremely difficult. And, just as for hardware, the software must be completely reverified and recertified every time a change is made, at what may be an enormous cost. In addition, software quickly becomes more "brittle" as changes are made—the difficulty of making a change without introducing errors may increase over the lifetime of the software.

Myth 3. *Computers provide greater reliability than the devices they replace.*

Reality: Although true in theory—software does not "fail" in the sense this term usually implies in engineering—there is little evidence to show that erroneous behavior by software is not a significant problem in practice.

When systems were made only from electromechanical and human components, engineers always had to worry about mechanical failure and operator or maintenance error. Techniques were developed to reduce greatly (but not eliminate) random wearout failures and human errors and to mitigate their consequences.

Now that computers are being introduced into these systems, a new, completely abstract factor—software—has been added. Since software is pure design, there is no need to worry about the random wearout failures found in physical devices, but system behavior now can be significantly affected by software design errors.

Little hard data is available on the reliability of operational software, especially data that compares software reliability to the reliability of equivalent systems that do not use computers. What is available mixes too many types of software (such as game, business, and control software) to be useful.

A study by the British Royal Signals and Radar Establishment used commercially available tools to examine the number of errors in software written for some highly safety-critical systems [?]. Up to 10 percent of the program modules or individual functions were shown to deviate from the original specification in one or more modes of operation. Discrepancies were found even in software that had undergone extensive checking using sophisticated test platforms. Many of the detected anomalies were too minor to have any perceptible effects—for example, a discrepancy of 1 part in 32,000 in a computation using 16-bit arithmetic. However, about 1 in 20 of the functions found to be faulty (that is, about 1 in 200 of all new modules) contained errors with direct and observable effects on the performance of the system being controlled. For example, potential overflows in integer arithmetic were detected that caused a change in the direction of deflection of an actuator—such as "turn left" when the correct action was "turn right."

On the surface, it seems that the solution to the software reliability problem is simply to get the software right. Although human error is a factor (since software is designed by humans), ample time *appears* to be available to use sophisticated techniques to eliminate any design errors before the software is used. However, accomplishing this goal has turned out to be harder than expected. Very few sophisticated software systems have been fielded that have not contained a significant number of errors.

These are not just "teething" problems that go away after the software is used for a while: Software-related errors usually occur over the entire system lifetime, sometimes after tens or hundreds of thousands of hours of use. The Therac-25 worked correctly thousands of times before the first known overdose, and the accidents were spread out over two and a half years [?]. The Space Shuttle software has been in use since 1980, and NASA has invested an enormous amount of effort and resources in verifying and maintaining this software. Despite this effort, since the Shuttle started operation in 1980, 16 severity level 1 software errors[2] have been discovered in released software. Eight of those remained in code that was used in flights, but none have been encountered during flight. An additional 12 errors of lower severity have been triggered during flight—none threatened the crew, three threatened the achievement of the mission, and nine were worked around. These problems occurred despite NASA having one of the most thorough and sophisticated software develop-

[2]Shuttle flight software errors are categorized by the severity of their potential consequences, without regard to the likelihood of their occurrence: severity 1 errors are defined as errors that could produce a loss of the Shuttle or its crew; severity 2 errors could affect the Shuttle's ability to complete its mission objectives; severity 3 errors affect procedures for which alternatives, or workarounds, exist; severity 4 and 5 errors are minor coding or documentation errors.

ment and verification processes in existence.

Although reliability is easily augmented by the use of redundancy for random, hardware wearout failures, techniques that provide the equivalent reliability enhancement for software design errors have not been found. For various reasons (described in the next section), including the fact that eliminating failures caused by design errors is inherently much more difficult than predicting and eliminating wearout failures, highly effective techniques are unlikely to be found. Vendors often tout tools or approaches that will lead to "zero-defect" software, but these claims are more sales than science.

Even if the techniques did exist to produce perfect software, there is usually not enough time to accomplish this goal: The ideal conditions—including unlimited funds and time—seldom if ever exist [**?**]. Instead, there are competing needs to reduce software costs (which already exceed hardware cost in many large systems). Time pressures also become severe, since the time required to develop software often controls the pace of the overall project.

But even if this myth were true—that is, computers are more reliable than other devices—this fact would not necessarily mean that they were *safer* than the devices they replace.

Myth 4. *Increasing software reliability will increase safety.*

Reality: Software reliability can be increased by removing software errors that are unrelated to system safety, thus increasing reliability while not increasing safety at all.

In addition, software reliability is defined as compliance with the requirements specification, while most safety-critical software errors can be traced to errors in the requirements—that is, to misunderstandings about what the software should do. Many software-related accidents have occurred while the software was doing exactly what it was intended to do—the software satisfied its specification and did not "fail." Software can be correct and 100-percent reliable and still be responsible for serious accidents. (Examples are given in later chapters.) Safety is a system property, not a software property.

Safety and reliability, while partially overlapping, are not the same thing: Increased computer or software reliability does not necessarily result in increased system safety.

Myth 5. *Testing software or "proving" (using formal verification techniques) software correct can remove all the errors.*

Reality: The limitations of software testing are well known. Basically, the large number of states of most realistic software makes exhaustive testing impossible; only a relatively small part of the state space can be covered. Although research has resulted in improved testing techniques, no great breakthroughs are on the horizon, and mathematical arguments have been advanced for their impossibility [**?**].

The use of mathematical techniques to verify the consistency between the software instructions and the specifications is another way to gain assurance. Although not currently practical, mathematical verification of software is likely to be so in the future. Unfortunately, such verification will not solve all our problems. The process requires that the "correct" behavior of the software first be specified in a formal, mathematical language. This task is not easy and may turn out to be as difficult and error-prone as writing the code.

In addition, although the basic computations and algorithms are easily specified and verified, the most important errors may not lie in these aspects of the code. For example, many software-related accidents have involved overload. A recent case occurred in England when a computer that was dispatching emergency ambulance services stopped working because it was unable to handle the number of calls it received [**?**]. Such sophisticated timing problems are much more difficult, and perhaps impossible, to verify formally because they involve more than just the application software itself.

Most important, as stated earlier, practical experience and empirical studies [**?**] have shown that most safety-related software errors can be traced to the requirements and not to coding errors (which tend to have less serious consequences in practice). Writing adequate software requirements is a difficult and unsolved problem. This book presents some techniques that may help, but the problem is far from solved and may remain unsolved for quite some time.

Myth 6. *Reusing software increases safety.*

Reality: Although reuse of proven software components can increase reliability, reuse has little or no effect on safety. In fact, reuse may actually *decrease* safety because of the complacency it engenders and because the specific hazards of the new system were not considered when the software was originally designed and constructed. Examples of safety problems arising from the reuse of software include the following:

- The Therac-20, parts of which were reused for the Therac-25, contained the same error responsible for at least two deaths in the Therac-25. The error had no serious consequences when encountered in the Therac-20; it resulted only in an occasional blown fuse and not in a massive overdose, and so was never detected and fixed (see Appendix A).

- Software used successfully for air traffic control for many years in the United States was reused in Great Britain with less success. The American developers had not worried about handling zero degrees longitude (since that was not relevant in the United States); as a result, the software basically folded England along the Greenwich Meridian, plopping Manchester down on top of Warwick [**?**].
- Aviation software written for use in the northern hemisphere often creates problems when used in the southern hemisphere [**?**]. In addition, software written for American F-16s has caused accidents when reused in Israeli aircraft flown over the Dead Sea, where the altitude is less than sea level.

Safety is not a property of the software itself, but rather a combination of the software design and the environment in which the software is used: It is application-, environment- and system-specific. Therefore, software that is safe in one system and environment may be unsafe in another. Reuse is *not* a solution to the safety problem.

Myth 7. *Computers reduce risk over mechanical systems.*

Reality: Computers have the potential to decrease risk, but not all uses of computers achieve this potential. Computers can automate tedious and potentially hazardous jobs such as spray painting and electric arc-welding, thus reducing the risk to workers in these particular jobs. However, other arguments that computers can reduce risk are debatable:

1. **Argument**: Computers allow finer control in that they can check parameters more often, perform complicated computations in real time, and take action quickly.

   Counter-argument: Computers *do* provide finer control computations in real time, and they *can* take action quickly. But finer control allows the process to be operated closer to its optimum, and the safety margins can be cut. The resulting systems have economic benefits, because they will, theoretically, shut down less often, and productivity may be increased by allowing more optimal control. However, any potential safety benefits of the finer control may be negated by the decrease in safety margins—perhaps without the concomitant attainment of the high software reliability on which the arguments for smaller safety margins were based. There is no way to know before extensive use outside the test environment whether high software reliability has been achieved.

2. **Argument**: Automated systems allow operators to work farther away from hazardous areas.

   Counter-Argument: Because of lack of familiarity with the hazards, more accidents may occur when operators *do* have to enter hazardous areas. Assumptions that plants controlled by robots will not require operators to intervene physically are usually wrong and can lead to accidents. For example, a computer-controlled robot killed a worker in a plant that the designers had assumed would require a minimum of intervention—they did not include walkways for humans or standard safety devices such as audible warnings that the robot was in motion [**?**]. After the plant was operational, the operators found that they needed to enter the hazardous areas 15 to 20 times a day to bail out the robots and maintain adequate productivity: The original assumption that all robots (and thus the plant) would be shut down before humans entered hazardous areas became impractical. The designers had overestimated the ability of the plant to work adequately without human intervention and had not foreseen changes that would be required to planned operating procedures to meet productivity goals.

3. **Argument**: By eliminating operators, human errors are eliminated.

   Counter-argument: Operator errors are replaced by human design and maintenance errors: Thus, humans are not removed from the system, they are merely shifted to different jobs. It should not be too much of a surprise that human designers have been found to make the same types of errors as operators (see Chapter **??**).

   Moreover, as noted in Chapter 1, when humans are removed from direct contact with the system, they lose information that is necessary for correct decision making. Physically removing operators from the processes they supervise may simply lead to new types of errors and hazards.

4. **Argument**: Computers have the potential to provide better information to operators and thus to improve decision making.

   Counter-argument: While theoretically true, in reality this potential is very difficult to achieve. The subject is complex, and a detailed discussion is deferred until later. Briefly, computers make it easy to provide too much information to operators and to provide it in a form that is less usable for some purposes than traditional instrumentation.

5. **Argument**: Software does not fail.

> **Counter-argument**: This common belief is true only for a very narrow definition of "failure." Later chapters propose precise definitions of relevant engineering terms and their application to software. The important point here is that computers can exhibit incorrect and hazardous behavior, whether we call this behavior failure or not.
>
> One of the results of substituting computers for mechanical devices is a reduced ability to predict failure modes. Most mechanical systems have a limited number of failure modes, and often they can be designed to fail in a safe way—for example, a valve can be designed to fail closed or a relay can be designed to fail with its contacts open. In comparison to software, the limited number of physical failure modes also simplifies (1) the analysis of a system for potentially unsafe behavior, (2) the process of assuring that the design is adequately safe, and (3) the elimination or control of hazards to make the system safer. The unpredictability of software behavior and the potentially large number of incorrect behaviors often preclude the same type of failure-mode analysis and fail-safe design.

In summary, computers have the potential to increase safety, and, surely, this potential will be realized in the future. But we cannot assume that we know enough now to accomplish this goal. In addition, any increased potential may not be realized if those building the systems use it to justify taking more risks.

Computers will not go away: Their use and importance in complex systems is only going to increase. Software engineers, often with little training or experience in safety engineering, are building software for safety-critical systems. At the same time, safety engineers are finding themselves faced with ensuring that computer-dominated control systems are safe. To achieve and ensure safety in these systems, software must be included in the system-safety activities, and the software must be specifically developed to be safe using the results of system hazard analysis. A goal of this book is to provide information and ideas about how to do this.

2.3 Why Software Engineering is Difficult

Why do we have so much trouble engineering software when, for the most part, the software is performing the same functions as the electromechanical devices it is replacing? Shouldn't the same engineering approaches apply since the same type of design errors can be made in both? Shouldn't they be equally hard or easy to construct?

Parnas [?] and Shore [?] have written excellent descriptions of the unique engineering problems in constructing complex software. Much of the following discussion comes from these two sources.

2.3.1 Analog versus Discrete State Systems.

In control systems, the computer is usually simulating the behavior of an analog controller. Although the software may be implementing the same functions previously performed by the analog device, the translation of the function from analog to digital form may introduce inaccuracies and complications. Continuous functions can be difficult to translate to discrete functions, and the discrete functions may be much more complex to specify.

In addition, the mathematics of continuous functions is well understood; mathematical analysis often can be used to predict the behavior of physical systems. The same type of analysis does not apply to discrete (software) systems. Software engineering has tried to use mathematical logic to replace continuous functions, but the large number of states and lack of regularity of most software result in extremely complex logical expressions. Moreover, factors such as time, finite-precision arithmetic, and concurrency are difficult to handle. There is progress, but it is very slow, and we are far from being able to handle even small software. Mathematical specifications or proofs of software properties may be the same size as the program, more difficult to construct, and often harder to understand than the program. They are therefore as prone to error as the code itself [?].

Physical continuity in analog systems also makes them easier to test than software. Physical systems usually work over fixed ranges, and they bend before they break. A small change in circumstances results in a small change in behavior: A few tests can be performed at discrete points in the data space, and continuity can be used to fill in the gaps. This approach does not work for software, which can fail in bizarre ways anywhere in the state space of inputs; the failure behavior need not be related in any way to normal behavior [?].

2.3.2 The "Curse of Flexibility."

A computer's behavior can be easily changed by changing its software. In principle, this feature is good—major changes can be made quickly and at seemingly low cost. In reality, the apparent low cost is deceptive, as discussed earlier, and the ease of change encourages major and frequent change, which often increases complexity rapidly and introduces errors.

Flexibility also encourages the redefinition of tasks late in the development process in order to overcome deficiencies found in other parts of the system. During development of the C-17, for example—a project that has run into great difficulties largely because of software problems—the software was changed to cope with structural design errors in the aircraft wings that were discovered during wind tunnel tests. This case is typical. As Shore says, “Software is the resting place of afterthoughts” [**?**].

With physical machinery, major design modifications are much more difficult to make than minor ones. The properties of the physical materials in which the design is embedded provide natural constraints on modification. The design of a computer application, on the other hand, is stored in electronic bits and presents no physical barriers to manipulation. Thus, while natural constraints enforce discipline on the design, construction, and modification of a physical machine, these constraints do not exist for software.

Shore explains this difference by comparing software with aircraft construction, where feasible designs are governed by mechanical limitations of the design materials and by the laws of aerodynamics. In this way, nature imposes discipline on the design process, which helps to control complexity. In contrast, software has no corresponding physical limitations or natural laws, which makes it too easy to build enormously complex designs. The structure of the typical software system can make a Rube Goldberg design look elegant in comparison (see Figure 2.4). In reality, software is just as brittle as hardware, but the fact that software is logically brittle rather than physically brittle makes it more difficult to see how easily it can be broken and how little flexibility actually exists.

The myth of software flexibility also encourages premature construction, before we fully understand what we need to do. The software medium is so forgiving that it encourages us to begin working with it too soon. Although we often intend to go back and start again after the details are worked out, this iteration process rarely happens in practice, and design decisions made in prototypes and early design efforts usually remain unchanged. Few engineers would start to build an airplane before the designers had finished the detailed plans.

Another trap of software flexibility is the ease with which partial success is attained, often at the expense of unmanaged complexity. The untrained can achieve results that appear to be successful, but are really only partially successful: The software works correctly most of the time, but not all the time. Attempting to get a poorly designed, but partially successful, program to work all of the time is usually futile; once a program’s complexity has become unmanageable, each change is as likely to hurt as to help. Each new feature may interfere with several old features, and each attempt to fix an error may create several more. Thus, although it is extremely difficult to build a large program that works correctly under all required conditions, it is easy to build one that works 90 percent of the time. Shore notes that it is difficult to build reliable aircraft too, but it is not particularly easy to build planes that fly 90 percent of the time.

Few people would dare to design an airplane without training or after having built only model airplanes, but there seem to be few such qualms about attempting to build complex software without appropriate knowledge and experience. Shore explains,

> Like airplane complexity, software complexity can be controlled by an appropriate design discipline. But to reap this benefit, people have to impose that discipline; nature won't do it. As the name implies, computer software exploits a "soft" medium, with intrinsic flexibility that is both its strength and its weakness. Offering so much freedom and so few constraints, computer software has all the advantages of free verse over sonnets; and all the disadvantages [**?**].

Another type of discipline is also necessary—limiting the functionality of the software. This discipline may be the most difficult of all to impose. Theoretically, a large number of tasks can be accomplished with software, and distinguishing between what *can* be done and what *should* be done is very difficult. Software projects often run into trouble because they try to do too much and end up accomplishing nothing. When we are limited to physical materials, the difficulty or even impossibility of building anything we might think about building limits what we attempt. The flexibility of software, however, encourages us to build much more complex systems than we have the ability to engineer correctly. A common lament on projects that are in trouble is "If we had just stopped with doing x and not tried to do more. . . ." McCormick notes that

> A project's specification rapidly becomes a wish list. Additions to the list encounter little or no resistance. We can always justify just one more feature, one more mode, one more gee-whiz capability. And don't worry, it'll be easy—after all, its just software. We can do anything.
>
> In one stroke we are free of nature's constraints. This freedom is software's main attraction, but unbounded freedom lies at the heart of all software difficulty [**?**].

2.3.3 Complexity and Invisible Interfaces.

One way to deal with complexity is to break the complex object into pieces or modules. For very large programs, separating the program into modules can reduce individual component complexity. However, the large number of interfaces created introduce uncontrollable complexity into the design: The more small components there are, the more complex the interface becomes. Errors occur because the human mind is unable to fully comprehend the many conditions that can arise through the interactions of these components [**?**].

An interface between two programs is comprised of all the assumptions that the programs make about each other. Shore notes that such dependencies can be subtle and almost impossible to detect by studying the programs involved. For example, one program might work properly only if another program can be relied on to finish its job in a specific amount of time. When changes are made, the entire structure collapses.

Finding good software structures has proven to be surprisingly difficult [**?**]. In the design of physical systems, like nuclear power plants or cars, the physical separation of the system functions provides a useful guide for effective decomposition into modules. Equally effective decompositions for software are hard to find. In addition, the relatively high cost of the connections between physical modules helps to keep interfaces simple. As Shore points out,

> Physical machines such as cars and airplanes are built by dividing the design problems into parts and building a separate unit for each part. The spatial separation of the resulting parts has several advantages: It limits their interactions, it makes their interactions relatively easy to trace, and it makes new interactions difficult to introduce. If I want to modify a car so that the loudness of its horn depends on the car's speed, it can be done, at least in principle. And if I want the car's air conditioner to adjust automatically according to the amount of weight present in the back seat, that too can be done—again in principle. But in practice such changes are hard to make, so they require careful design and detailed planning. The interfaces in hardware systems, from airplanes to computer circuits, tend to be simpler than those in software systems because physical constraints discourage complicated interfaces. The costs are immediate and obvious [**?**].

In contrast, software has no physical connections, and logical connections are cheap and easy to introduce. Without physical constraints, complex interfaces are as easy to construct as simple ones, perhaps easier. Moreover, the interfaces between software components are often “invisible” or not obvious: It is easy to make anything depend on anything else. Again, discipline and training are required to control these problems, but when the software reaches a certain size (which is often found in control systems today), the complexity can overwhelm even the few tools we have to control it. McCormick suggests, “The underlying premise is suspect, namely that we really can build any system, no matter how complicated. The right tool, the right process will let us do anything, or so the salesmen assure us” [**?**]. Those waiting for tools to solve our problems are likely to be disappointed:

> Regrettably, humans can cope with very little complexity. Better tools and methods can help us with many of the rote aspects of system development; the tools and methods we use are valuable, even indispensable. But consultants and tool vendors often perpetuate a delusion, the delusion that we can cope with endless complexity, if only we would use a better tool or a different method.
>
> Tools can only be an aid to judgment. Tools cannot substitute for the physical constraints encountered naturally in other disciplines. Without a harsh and uncaring nature forcing us to make hard choices, we tend to rationalize the complexity we see growing before us on each new project.... Despite the best intentions of highly skilled people, each new increment of complexity seems entirely plausible on its own. We are willingly seduced.
>
> I submit that the grand failures of big, software-intensive systems have been due primarily to this willing seduction. Postmortem analysis of such projects routinely reveals a specification that grew in complexity until project cancellation. Natural constraints simply do not apply to software, and nobody knew when to say no. After all, it was only software [**?**].

2.3.4 Lack of Historical Usage Information.

A final difficulty with software not found in hardware systems is that no historical usage information is available to allow measurement, evaluation, and improvement on standard designs. Software is almost always specially constructed, whereas physical systems benefit from the experience gained by the use of standard designs over long periods of time and in varied environments. Consider the difficulty that would ensue if every part of an airplane or car were completely changed for each new model or version and the entire design process started anew. That basically describes the situation for software. To complicate matters further, the features that are most likely to change from one complex system design to another are exactly those that are most likely to be controlled by or embedded within software.

2.4 The Reality We Face

When systems were composed only of electromechanical and human components, engineers knew that random, wearout failures and human errors could be reduced and mitigated but never completely eliminated. They accepted the fact that they had to devise ways to build systems that were robust and safe despite random failures. Design errors, on the other hand, could be handled fairly well through testing and reuse of proven designs.

Because software has only design errors, the primary approach used to deal with reliability and safety problems has been simply to get the software correct. Theoretically, the possibility does exist for finding a set of techniques or methodology that will allow us to build perfect software. Much energy has been invested in looking for this methodology and less in finding ways to build software and systems that are robust and safe in the presence of software errors.

In reality, the time to create perfect software is never there, and perhaps it never can be. We may be seeking an impossible goal: software that is free of requirements and implementation flaws and that will always do what is required under all circumstances, no matter what changes occur to it or to the environment in which it operates.

Those who believe that the methodology exists that will allow us to construct such perfect software will find this book quite unsatisfactory. For those who have reached the conclusion that this goal is impossible to achieve—or at least not reachable now or in the immediate future—and that other solutions, perhaps adapted from those developed to cope with similar problems in hardware, are necessary, this book will provide some clues as to what might be done. But to devise and effectively use these solutions, we first need to understand why accidents occur.