An Engineering Perspective on Avoiding Inadvertent Nuclear War[1]

Prof. Nancy Leveson
Aeronautics and Astronautics Dept.
MIT

Introduction

As an engineer, I can only credibly comment on the engineering aspects of the NC^3 problem. However, the overall solutions will require the use of integrated sociotechnical approaches rather than social scientists and engineers working in isolation. There are two aspects of the problem that are the focus of this paper: (1) Preventing inadvertent detonation or launch of a nuclear weapon and (2) ability to intervene if a nuclear weapon is released either intentionally or unintentionally by ourselves or others, i.e., the missile defense problem. While there have been a few false alarms and alerts in NC^3 systems in the past 50 years, none led to a loss, mostly because of the very conservative engineering approach taken to designing these systems. We need to resist the temptation to use sexy, new, and untrustworthy engineering techniques on the NC^3 systems of the future. I am worried that the temptation will be too great to resist.

I have worked in the area of safety of complex engineered systems close to 40 years now, starting on what was, for its time, a very software-intensive defense system. Since then I have been teaching and creating new safety engineering approaches to ensuring that such systems (both defense and commercial) can be trusted. More recently, I have become involved with security and, particularly, cyber security along with safety. In this paper, I will provide my perspective on the state-of-the-art in engineering for safety and security and what is needed going forward. Most important, I suggest some new technologies that I fear will be tried and lead to catastrophe, i.e., what should not be done.

Why have we been so successful in the past?

The most successful complex systems in the past were simple and used rigorous, straightforward processes. Prevention of accidental detonation of nuclear bombs, for example, used a brilliant approach involving three positive measures (the *3 I's* of isolation, incompatibility, and inoperability) and reliance on simple mechanical systems that could provide ultra-high assurance.[2] Although there were a few incidents over a long period (e.g., Thule, Palomares, and a few others), inadvertent detonation did not occur in those cases. Introduction of software and more complex designs have been avoided in the design of the accidental detonation prevention strategies.

The more recently introduced software-intensive systems have been much less reliable. However, even there, some real-time control systems stand out. For example, the on-board Space Shuttle control software demonstrates that the nearly universal unreliability of software systems today is technically not necessary.[3] The Space Shuttle software was, for its time, highly advanced and complex but it had an extraordinarily low level of software errors in flight. Appendix B provides some information about why it was so successful, considerably more than almost all software created today.

More recently, our ability to provide highly trustworthy systems has been compromised by gratuitous complexity in their design and inadequate development and maintenance processes. For example,

[1] This paper was produced for the Workshop on NC3* SYSTEMS AND STRATEGY STABILITY: A GLOBAL OVERVIEW, organized by the Nautilus Institute for Security and Sustainability and funded by the John D. and Catherine T. MacArthur Foundation, Hoover Institute, Stanford University, Jan. 22-23, 2019.

[2] Nancy Leveson (1995), *Safeware*, Addison-Wesley Publishers, Boston, MA.

[3] Nancy Leveson (2013), "Software and the challenge of flight control," in Roger Launius, James Craig, and John Krige (eds.) *Space Shuttle Legacy: How We Did It/What We Learned , American Institute of Aeronautics and Astronautics (AIAA),* Reston VA.

arguments are commonly made for using development approaches like X-treme Programming and Agile that eschew the specification of requirements before design begins. These nonsensical arguments state as a given or assumption that something has to be built before it is possible to determine what needs to be built. While this may be true for iPhone and web apps (although I doubt it), it is patently false for safety-critical applications. The careful system and requirements analysis of the past (as demonstrated on the Space Shuttle software, for example) is being discarded for sandbox approaches. Only the most rigorous and professional development practices should be used on NC^3. This limitation implies that unassurable technologies like AI are prohibited and that the use of software is minimized if more highly assurable hardware can be used instead. We will need *"back to basics"* approaches to system and software engineering if the past high level of reliability and safety of NC^3 systems is to be continued into the future.

Providing high security is even more problematic. Again, only the most basic security techniques, such as providing an air gap to isolate critical systems, have been highly successful. The number of intrusions in today's systems is appalling and unacceptable. Clearly what we are doing is not working.

Is the DoD willing to eschew the use of sexy new but unassurable technology and strip away unnecessary complexity in the most crucial nuclear systems? Given the headlong plunge of the DoD toward more and more complexity, it seems doubtful.

The other problem is that even if we were willing to do what is technically necessary to build simple, straightforward, and trustworthy systems, the losses that have and are occurring in today's systems are as much a product of our inability to manage these systems as they are technological flaws. While we tend to focus on the technical factors involved in accidents/incidents, the major losses that are well investigated (for example, the Columbia Shuttle loss, Deepwater Horizon, and Fukushima) are found to be as much, if not more, a result of social and organizational factors (including inadequate controls and flawed decision making) than simply technical factors.[4] The same is true for security-related losses. What will be needed to provide highly trustworthy NC^3 systems in the future?

Ensuring Safety and Security in NC^3

Avoiding unnecessary complexity and even compromises in functionality are going to be necessary. At the same time, we need to improve our engineering approaches to creating more complex, trustworthy systems. Even if we could potentially create technical systems that are worthy of high trust, how would we justify that confidence? Let's look at safety first and then security.

The safety analysis techniques typically used today are all 50 to 75 years old. These techniques do not work on the software-intensive, complex systems that we are building today. They are based on the obsolete assumption that accidents result from component failures. This assumption is no longer true. Today's losses are as likely to be the result of system design flaws as component failure. Our technology today allows us to build systems that cannot be exhaustively tested and for which it is impossible to anticipate, understand, plan, and guard against all potential adverse system behavior before operational use. Engineers are increasingly facing the potential for "unknown unknowns" in the behavior of the systems they are designing.

In addition, complexity is leading to important system properties (such as safety) not being related to the failure of individual system components but rather to the interactions among the components that have not failed or even malfunctioned. Accidents can occur due to unsafe interactions among individual components and subsystems that satisfy their requirements.[5]

[4] As an example, see Nancy Leveson (2007), ``Technical and managerial factors in the NASA Challenger and Columbia losses: Looking forward to the future,'' in Handelsman and Fleishman (eds.), *Controversies in Science and Technology, Vol. 2: From Chromosomes to the Cosmos*, Mary Ann Liebert, Inc., New Rochelle, NY.
[5] Nancy Leveson (2012), *Engineering a Safer World*, MIT Press, Cambridge MA.

As one example, the loss of the Mars Polar Lander was attributed to noise (spurious signals) generated when the landing legs were initially deployed during descent.[6] This noise was normal and expected and did not represent a failure in the landing leg system. The onboard software interpreted these signals as an indication that landing occurred (which the software engineers were told they would indicate) and shut the engines down prematurely, causing the spacecraft to crash into the Mars surface. The landing legs and the software performed correctly—as specified in their requirements, that is, neither "failed"—but the accident occurred because the system designers did not account for all interactions between the leg deployment and the descent-engine control software.

Another example occurred when some Navy aircraft were ferrying missiles from one point to another. One pilot executed a planned test by aiming at the aircraft in front (as he had been told to do) and firing a dummy missile. Apparently nobody knew that the "smart" software was designed to substitute a different missile if the one that was commanded to be fired was not in a good position. In this case, there was an antenna between the dummy missile and the target, so the software decided to fire a live missile located in a different (better) position instead. What aircraft component(s) failed here? We cannot even use our usual excuse which is to blame it on the human operator. The pilot did exactly as he had been instructed to do. In fact, even when there is "operator error," it is usually the result of our designing systems in which an operator error is inevitable and then blaming the results on the operator rather than the designer.

Engineering approaches used in the past (and currently) to deal with failure-related accidents have no impact on these "system interaction" accidents. There are two ways to potentially avoid them: (1) reduce the complexity in our system designs to the point where they may be unable to achieve the mission goals and those goals will have to be curtailed more than leaders are willing to do or (2) reduce complexity as much as is feasible along with developing and using much more powerful analysis and design techniques. The second approach seems the most realistic, and it is the one I personally have been taking in my research.

The more powerful analysis techniques we are developing are becoming widely and successfully used on the autonomous systems in automobiles and increasingly on commercial aircraft today, although only starting to be used in defense. These tools require making fundamental engineering paradigm changes and have met with resistance. But in the automotive and aviation domains, there is no other choice. Automobiles today contain over 100 million lines of software. Compare that to our most modern military aircraft, which probably contain between 15 and 20 million lines of code, which of course is still a significant amount. Other industries have been even more resistant to change.

Why are new tools not developed and used more? Making progress involves a paradigm change from the way engineering is done today. Thomas Kuhn analyzed the difficulty engineering and science has in accepting paradigm changes.[7] We need to overcome this resistance if we are to create highly trustworthy NC^3 systems. Current tools are based on paradigms that just do not stretch to the needs of modern systems. Such paradigm changes do *not* include introducing technologies, such as AI, for which trustworthiness is not currently (and in the foreseeable future) possible despite using new analysis and design approaches.

What types of paradigm changes are necessary? Because the nature of accidents is changing as our engineering technology changes and system complexity increases, our models and understanding of the cause of accidents need to change. This will require a change to the underlying theoretical basis for safety and security engineering and for managing critical systems.

[6] Jet Propulsion Laboratory Special Review Board (2000), Report on the loss of the Mars polar lander and deep space 2 missions, NASA JPL, March.

[7] Thomas Kuhn (1962), *The Structure of Scientific Revolutions*, University of Chicago Press.

Traditionally, accidents/losses have been conceived as resulting from chains of failure events. This model is no longer adequate and needs to be expanded to include component and subsystem interaction accidents and not just failures. The traditional approaches to safety engineering are based on reliability theory; safety is essentially equated with the reliability of the system components. This assumption about the cause of losses is no longer true. New technology and increasing complexity in our designs require a change in how accident causality is perceived and losses prevented. The change I have suggested uses System Theory, although other approaches may be possible. The bottom line is that we cannot continue to base safety and trustworthiness on assumptions that no longer hold just because we are resistant to change.

System theory was created after World War II to deal with the new technology and complexity in our engineered systems. It also arose in biology, where the interconnections and interdependencies of the human body also created a need for a paradigm change to make significant progress.[8] Before this time, complexity was handled in science and engineering by using analytic decomposition, i.e., breaking systems into their components, analyzing the components independently, and then combining the results to evaluate and understand the behavior of the system as a whole. The physical or functional components are assumed to interact in direct and known ways. For example, if the weight of an complex object is the analysis goal, the separate pieces may be weighed and the result combined to get the system weight. As another common example, the system reliability is usually evaluated by evaluating the reliability of the individual components and then the component reliabilities are combined mathematically to evaluate the system reliability. Safety is usually assumed to be simply a combination of the component reliabilities.

The success of this type of decompositional or reductionist approach relies on the assumption that the separation and individual analysis does not distort the phenomenon or property of interest. More specifically, the approach works if:

- Each component or subsystem operates independently. If time-related events are modeled, then they are independent except for the immediately preceding and following events.
- Components act the same when examined separately (singly) as when they are playing their part in the whole.
- Components and events are not subject to feedback loops and other indirect interactions.
- The interactions among the components or events can be examined pairwise and combined into a composite value.

Sophisticated levels of coupling between components violate these assumptions. If these assumptions are not true, which is the case for complex engineered systems today, including most NC^3 systems, then the simple composition of the separate analyses will not accurately reflect the value for the system as a whole.

The failure events also must be assumed to be stochastic[9] for probabilities or likelihood to be determined. Unfortunately, software and humans do not satisfy this assumption. In addition, complexity is leading to important system properties (such as safety) not being related to the behavior of individual system components but rather to the interactions among the components, as noted above. Accidents can occur due to unsafe interactions among components that have not failed and, in fact, satisfy their requirements. That is, the components can be 100% reliable but the system can be unsafe.

[8] Ludwig von Bertalanffy (1969), *General Systems Theory: Foundations*, New York: Braziller; and Norbert Weiner (1965), *Cybernetics*, MIT Press
[9] Something is stochastic if it can be described by a probability distribution or pattern that may be analyzed statistically to understand average behavior or an expected range of behavior but may not be predicted precisely.

Note that so-called "systems of systems" are no different than any other system. All systems are abstractions imposed by the observers. As such, the boundary around systems can be drawn and redrawn as appropriate and necessary to achieve the immediate goals. All systems are composed of other "systems" (which are, from that viewpoint, seen as subsystems), where most of these subsystems may already exist. To wit, the concept of a "system" is recursive and adaptive. Dealing with "systems of systems" requires no new concepts or approaches than are needed to handle any system. But we need new approaches to engineer and operate all complex systems.

System theory as used in engineering was created after World War II to deal with the increased complexity of the systems starting to be built at that time.[10] It was also created for biology in order to successfully understand the complexity of biological systems.[11] In these systems, separation and analysis of separate, interacting components (subsystems) distorts the results for the system as a whole because the component behaviors are coupled in non-obvious ways. The first engineering uses of these new ideas were in the missile and early warning systems of the 1950s and 1960s.

Some unique aspects of System Theory are:

- The system is treated as a whole, not as the sum of its parts. You have probably heard the common statement: "the whole is more than the sum of its parts."
- A primary concern is *emergent properties* (see Figure 1), which are properties that are not in the summation of the individual components but "emerge" when the components interact. Emergent properties can only be treated adequately by taking into account all their technical and social aspects. Safety and security and most other important system properties, including trustworthiness, are emergent.
- Emergent properties arise from relationships among the parts of the system, that is, by how they interact and fit together.

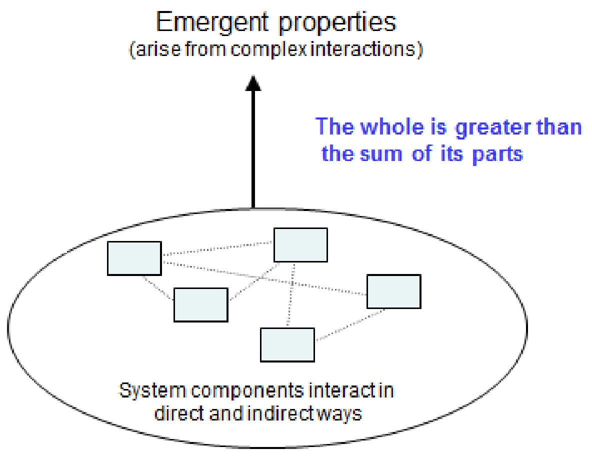
Emergent properties
(arise from complex interactions)
The whole is greater than the sum of its parts
System components interact in direct and indirect ways

Figure 1: Emergent Properties in System Theory

If emergent properties arise from individual component behavior and from the interactions among components, then it makes sense that controlling emergent properties, such as safety, security, maintainability, and operability, requires controlling both the behavior of the individual components and

[10] Basic introductions include Peter Checkland (1981), *System Thinking, System Practice*, John Wiley & Sons, and Gerald Weinberg (1075), *An Introduction to General Systems Thinking*, John Wiley & Sons. Wiener introduced the early term "cybernetics," which is now been supplanted by the term "System Theory," in Norbert Wiener (1965), *Cybernetics: or Control and Communications in the Animal and the Machine*, 2nd Edition, MIT Press, Cambridge, MA.

[11] Ludwig Von Bertalanffy (1969) , *General System Theory*, Braziller, New York.

the interactions among the components. We can add a controller to the figure to accomplish this goal. The controller provides control actions on the system and gets feedback to determine the impact of the control actions. In engineering, this is a standard feedback control loop.

The controller enforces constraints on the behavior of the system. Example safety constraints might be that aircraft or automobiles must remain a minimum distance apart, pressure in deep water wells must be kept below a safe level, aircraft must maintain sufficient lift to remain airborne unless landing, toxic substances must never be released from a plant, and accidental detonation or launch of weapons must never occur.

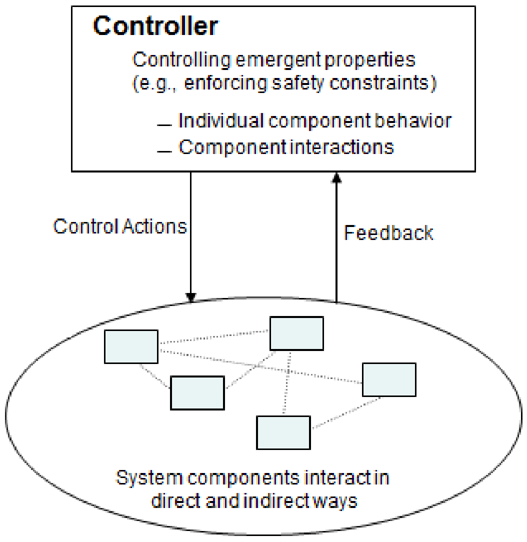
Controller
Controlling emergent properties
(e.g., enforcing safety constraints)
Individual component behavior
Component interactions
Control Actions
Feedback
System components interact in
direct and indirect ways

Figure 2: Controllers and controls can control emergent properties

Control is interpreted broadly and, therefore, includes everything that is currently done in safety engineering plus more. For example, component failures and unsafe interactions may be controlled through design, such as using redundancy, interlocks, barriers, or fail-safe design. Safety may also be controlled through process, such as development processes, manufacturing processes and procedures, maintenance processes, and general system operating processes. Finally, safety may be controlled using social controls including government regulation, culture, insurance, law and the courts, or individual self-interest. Human behavior can be partially controlled through the design of the societal or organizational incentive structure or other management processes.

To model complex sociotechnical systems requires a modeling and analysis technique that includes both social and technical aspects of the problem and allows a combined analysis of both. Figure 3 shows an example of a hierarchical safety control structure for a typical regulated industry in the U.S. International controllers may be included. Notice that the operating process (the focus of most hazard analysis) in the lower right of the figure makes up only a small part of the safety control structure. There are two basic hierarchical control structures shown in Figure 3—one for system development (on the left) and one for system operation (on the right)—with interactions between them. Each level of the structure contains controllers with responsibility for control of the interactions between and behavior of the level below. Higher level controllers may provide overall safety policy, standards, and procedures (downward arrows), and get feedback (upward arrows) about their effect in various types of reports, including incident and accident reports. The feedback provides the ability to learn and to improve the effectiveness of the safety controls.

Manufacturers must communicate to their customers the assumptions about the operational environment in which the original safety analysis was based, e.g., maintenance quality and procedures, as well as information about safe operating procedures. The operational environment, in turn, provides feedback to the manufacturer and potentially others, such as governmental authorities, about the performance of the system during operations. Each component in the hierarchical safety control structure has responsibilities for enforcing safety constraints appropriate for that component, and together these responsibilities should result in enforcement of the overall system safety constraints.

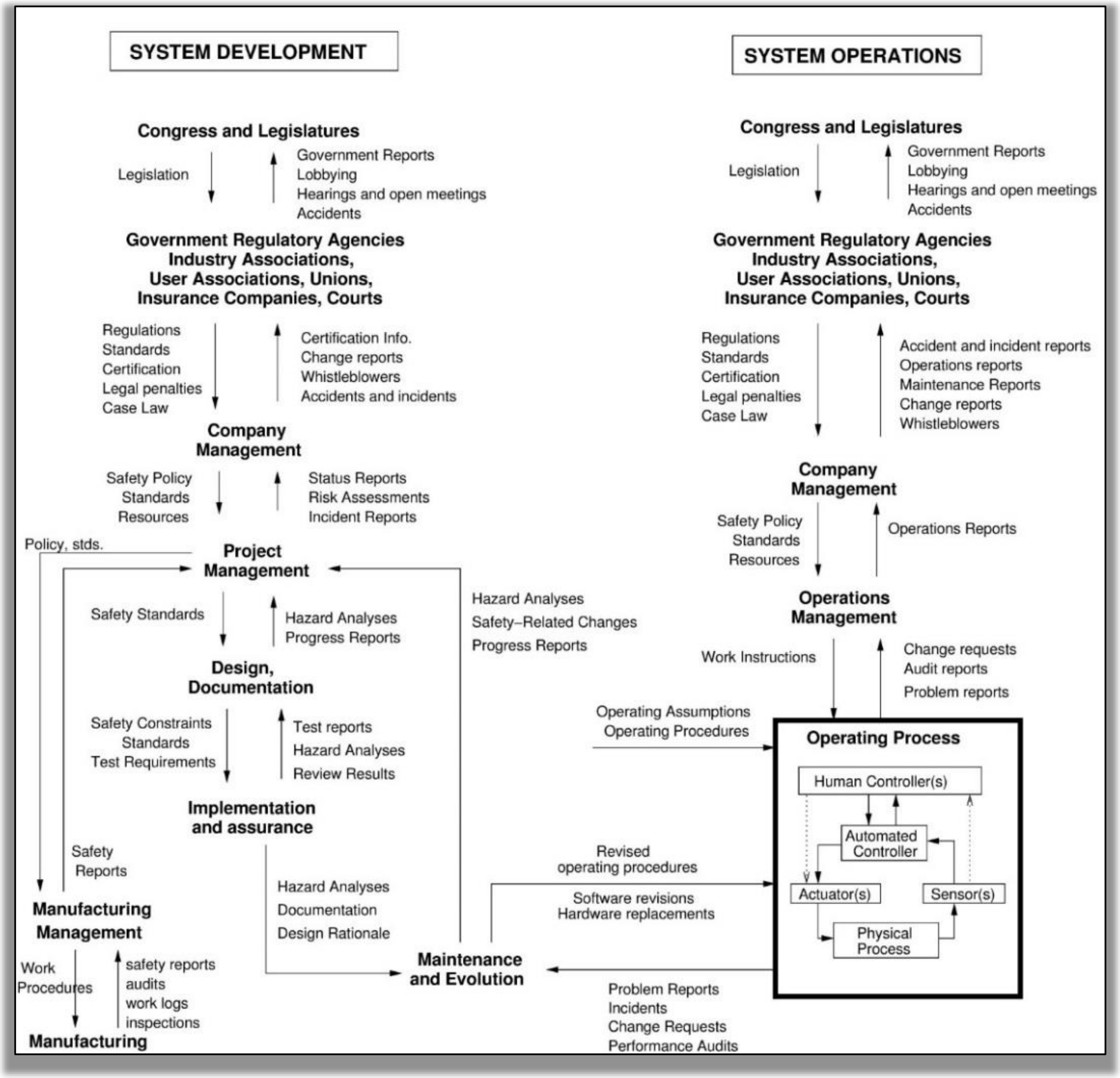

Figure 3: A generic safety control structure

Note that the use of the term “control” does not imply a rigid command and control structure. Behavior is controlled not only by engineered systems and direct management intervention, but also indirectly by policies, procedures, shared value systems, and other aspects of the organizational culture. All behavior is influenced and at least partially “controlled” by the social and organizational context in which the behavior occurs. Engineering this context can be an effective way to create and change a safety culture, i.e., the subset of organizational or social culture that reflects the general attitude about and approaches to safety by the participants in the organization or society [13]. Formal modeling and analysis of safety must include these social and organizational factors and cannot be effective if it focuses only on the technical aspects of the system. As we have learned from major accidents, managerial and organizational factors and often governmental controls (or lack of them) are as important as technical factors in accident causation and prevention. For space reasons, Figure 3 emphasizes the high-level components of a safety control structure and not their detailed design. Each can be quite complex. For example, the operating process (lower-right-hand box) would include all the physical parts of an NC^3 system.

Figure 4 shows the basic form of the interactions between the levels of the control structure, where the controller imposes control actions on the controlled process. The standard requirements for effective management—assignment of responsibility, authority, and accountability—are part of the control structure design and specification.

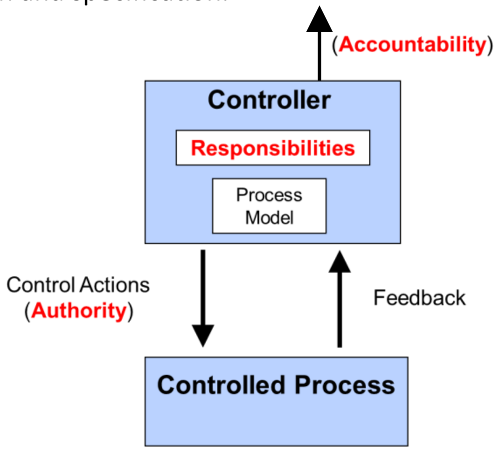

Figure 4. The Basic Building Block for a Safety Control Structure

The importance of feedback becomes apparent here. In engineering, every controller must contain a model of the controlled process in order to provide effective control. For human controllers, this model is usually called a *mental model.* This process model or mental model includes assumptions about how the controlled process operates and the current state of the controlled process. It is used to determine what control actions are necessary to keep the system operating effectively.

Accidents in complex systems often result from inconsistencies between the model of the process used by the controller and the actual process state, which results in the controller providing unsafe control actions. For example, the autopilot software thinks the aircraft is climbing when it really is descending and applies the wrong control law; a military pilot thinks a friendly aircraft is hostile and shoots a missile at it; the software thinks the spacecraft has landed and turns off the descent engines prematurely; or the early warning system thinks the country has been targeted and launches an interceptor at a friendly target. Note that it does not matter, such as in the last example, whether the incorrect process model was a result of an unintentional or intentional cause so that security is handled in the same way. The Stuxnet worm in the Iranian reactor program is an example. The worm made the controller's process model think that the contrifuges were spinning slower than they were and the controller reacted by sending "increase speed" commands to the centrifuges, wearing them out prematurely.

Part of the challenge in designing an effective safety control structure is providing the feedback and inputs necessary to keep the controller's model of the controlled process consistent with the actual state of the controlled process. An important component in understanding accidents and losses involves determining how and why the controls may be ineffective in enforcing the safety constraints on system behavior; often this is because the process model used by the controller was incorrect or inadequate in some way.

To apply system theory to safety, a new accident causality model is required that extends what is currently used. STAMP (System-Theoretic Accident Model and Processes) expands the traditional model of causality beyond a chain of directly-related failure events or component failures to include more complex processes and unsafe interactions among system components.[12] In STAMP, safety and security are treated as a dynamic control problem rather than a failure prevention problem. No causes are omitted from the STAMP model, but more are included and the emphasis changes from preventing failures to enforcing constraints on system behavior. Some advantages of using STAMP are that:

- It applies to very complex systems because it works top-down rather than bottom up.
- It includes software, humans, organizations, safety culture, etc. as causal factors in accidents and other types of losses without having to treat them differently or separately.
- It allows creating tools, such as STPA, accident analysis (CAST), identification and management of leading indicators of increasing risk, organizational risk analysis, etc. These tools have been shown to be much more powerful and effective in both analytical and empirical evaluations.

The goal of this paper is not to sell this particular model but to argue that we can add new technology and complexity to our NC^3 systems, but a paradigm change is going to be needed. Making incremental small improvements in current approaches or simply creating new names for the old approaches that failed (a favorite of researchers and others) and pretending the new name will somehow magically make it work better is not going to allow improvements in NC^3 system trustworthiness.

An example of the use of the new system-theoretic approach on the safety analysis of the new U.S. Ballistic Missile Defense System is provided in Appendix A.

As alluded to above, security is also an emergent property. The primary approach to cybersecurity today is intrusion prevention and detection, implemented by performing various types of threat analysis. The goal is to identify the threats and prevent them by identifying when they might result in intrusions and trying to prevent such intrusions. This approach is clearly not working given the enormous number of intrusions that are occurring. In addition, the focus in cybersecurity has been on information security. But the major problem in NC^3 is mission assurance, not information security (although it admittedly is a part of the problem). In fact, security can and should be treated in the same way as safety. Consider the following definitions:

Definition: <u>Safety</u> is freedom from losses.

Definition: An a<u>ccident/mishap</u> is any undesired or unplanned event that results in a loss, as defined by the system stakeholders.

Losses may include loss of human life or injury, equipment or property damage, environmental pollution, mission loss (non-fulfillment of mission), negative business impact (e.g., damage to reputation, product launch delay, legal entanglements), nuclear war, etc. There is nothing in the definition that distinguishes between inadvertent and intentional causes. In fact, the definition does not limit the causes specified. The current focus in security on intrusions and information loss, therefore, can be supplemented in the mission assurance world with a focus on preventing losses. Systems need to be designed so that even if an enemy penetrates a system boundary (which they will), they cannot do anything harmful. The same approach to preventing a loss can be used, for example, if an operator

[12] Nancy G. Leveson, *Engineering a Safer World*, MIT Press (2012), Cambridge MA.

inadvertently issues a command to launch a missile or if the operator was tricked into doing it by some type of intrusion. The first is considered a safety problem and the second a security problem, but the same analysis and design approaches can be used to prevent them.

This change in the way that cyber security is treated is again a radical paradigm change from what is done today. In Air Force evaluations, it has been found to be much more effective.[13] But such changes require a willingness to make them.

Conclusions and a Way Forward

We have avoided nuclear catastrophe by using very conservative techniques and avoiding software in critical functions. That will not be possible going forward. Another approach, probably involving a paradigm change, is going to be necessary.

In this paper I have argued that we need to:

- Avoid unnecessary complexity. Strip systems to their basics and use rigorous and careful development processes
- Emphasize less technology, not more: throwing technology at the problem will make it worse. In particular, do not use new technology like AI for which it is impossible to provide adequate trust. In almost all cases, it is not needed but used because it is "sexy" and fun and people, not understanding it, assume it will provide miracles.
- Improve NC^3 systems not by pretending that our current technology is perfect or even adequate, but by acknowledging the limitations. Instead, develop more powerful, socio-technical and system engineering and risk management approaches that involve paradigm changes from the approaches that are no longer working. These are only now coming into existence and will need technical advances and refinement.

Appendix A: An Example Application of STPA to a Ballistic Missile Intercept System

STPA is a new, more powerful hazard analysis method based on STAMP. In 2005, soon after it was invented and in one of its first applications to a real system, STPA was used to ensure that inadvertent launch would not occur in the U.S. Ballistic Missile Intercept System.[14] While safe engineering efforts (including STPA) are most effective when begun early in the system life cycle, they may be done later for various reasons.

In this case, the Missile Defense Agency (MDA) used STPA to characterize the residual safety risk of the Ballistic Missile Defense System (BMDS) right before deployment and field test. As described by Pereira, Lee, and Howard,[13] BMDS was developed as a layered defense to defeat all ranges of threats in all phases of flight (boost, mid-course, and terminal). It is comprised of a variety of components including sea-based sensors on the Aegis platform, upgraded early warning radars (UEWR), the Cobra Dane Upgrade (CDU), Ground-based Midcourse Defense (GMD) Fire Control and Communications (GFC/C), a Command and Control Battle Management and Communications (C2BMC) Element, and Ground-based interceptors (GBI). Future block upgrades were originally planned to introduce additional Elements into the BMDS, including Airborne Laser (ABL) and Terminal High Altitude Area Defense (THAAD).

While each of the elements of this integrated and very complex system had active safety programs, the complexity and risk introduced by their integration as a single system went beyond what is possible

[13] William Young, dissertation

[14] Steven Pereira, Grady Lee, and Jeffrey Howard 2006). "A System-Theoretic Hazard Analysis Methodology for a Non-advocate Safety Assessment of the Ballistic Missile Defense System," *Proceedings of the 2006 AIAA Missile Sciences Conference*, Monterey, CA, November 14-16.

to analyze using traditional hazard analysis techniques. Some of these elements involved upgrades to already fielded systems. Many of them had historically been developed independently.

The STPA analysis is performed on a control structure, as described in the main part of this paper. Clearly, the actual control structure cannot be shown, but a fictional one is used (FMIS or Fictional Missile Intercept System) as an example. Similar to programs within the real BDMS, FMIS uses a hit-to-kill interceptor that destroys incoming ballistic missiles through force of impact. The FMIS control structure is shown in Figure A.1.

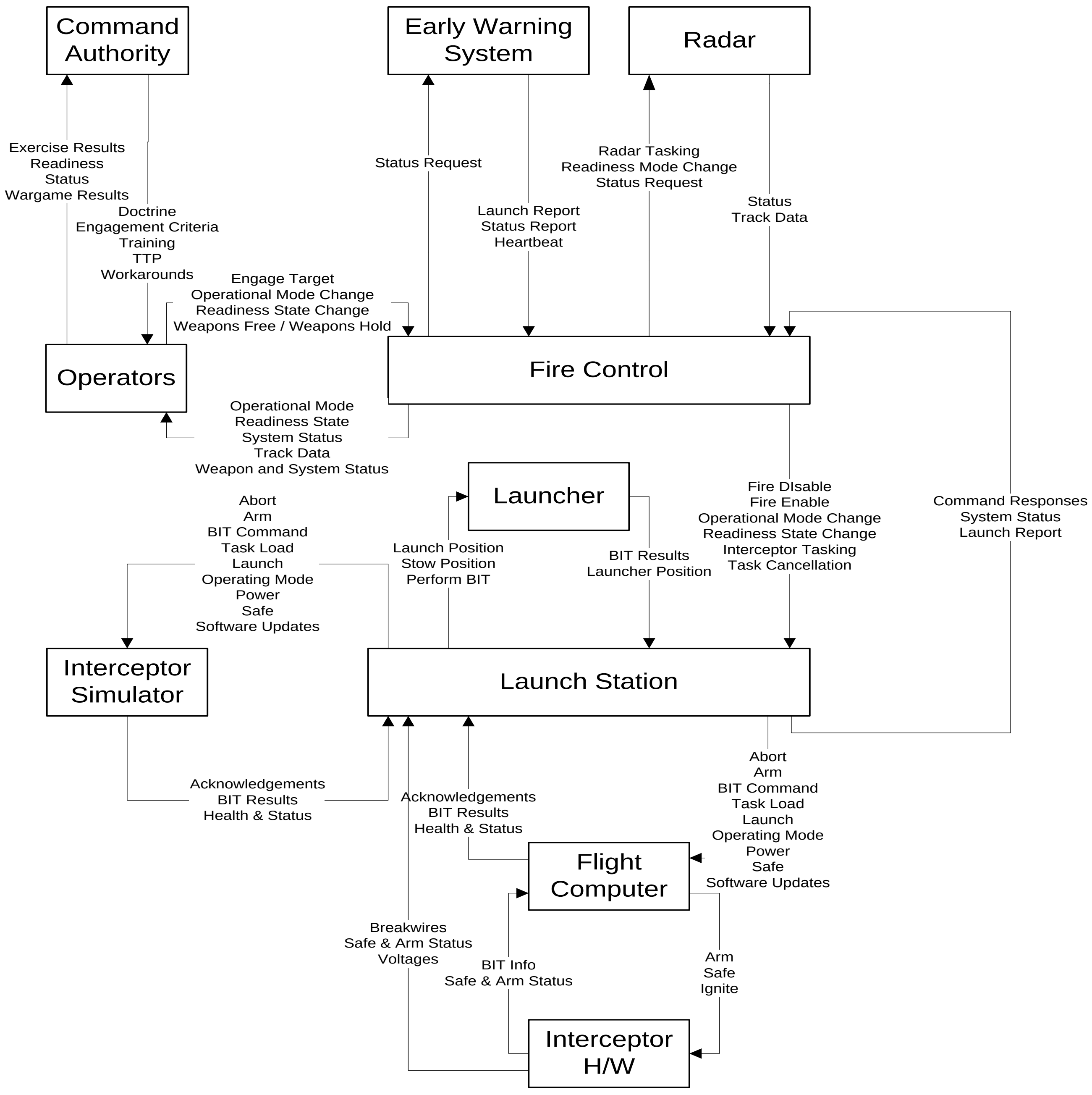

Figure A.1: The control structure of the fictional missile intercept system (FMIS)

In the STPA analysis, the system is viewed as a collection of interacting loops of control. For example, command authorities exert control over system operators by issuing guidance, providing training, and establishing tactics, techniques, and procedures (TTPs). Command authorities receive feedback in the form of reports and performance during training exercises. Operators exert control over software by inputting commands, and they receive feedback from displays and aural alerts. Software exerts control over other software and hardware by sending messages and commands and receives feedback in the form of measured values and status information. Safety is an emergent property of the system, arising from the interactions among software, hardware, and humans. Safety is maintained by placing constraints on the behavior of the system's components. Mishaps occur 1) when the constraints are insufficient to maintain safety and 2) when the controls present in the system are inadequate to enforce the safety constraints.

STPA, in this use, provided information about where constraints may be insufficient to maintain safety (prevent identified losses), when controls may be inadequate to enforce safety constraints, and how mitigations can be developed to reduce safety risk with minimal impact on the system's design and operations.

Details about the analysis and how it is performed are beyond the scope of this paper. But an example of the types of problems found is instructive. Again, the hazard is inadvertent launch. Each of the control actions (see Figure A.1) was first analyzed to identify under what context it could lead to the hazard and then the scenarios are identified that could lead to that hazardous context and control action. For example, consider the control action "*Fire Enable*" from the Fire Control system to the Launch Station. Clearly that control action could potentially be related to an inadvertent launch if given in a specific context, e.g., when a real threat does not exist. The goal of the analysis is to identify why "*Fire Enable*" might be given without the existence of a legitimate threat.

The analysis starts with a control structure that is used to carry out an organized and rigorous identification of unsafe control actions and the context in which these control actions become unsafe. In this example, we will not describe this structured process, but instead start with the Unsafe Control Action: *Fire Enable issued when there is no threat*. If the *Fire Enable* command is provided to a launch station when there is no real threat, the launch station will transition to a state where it accepts interceptor tasking and can progress through a launch sequence. In combination with other incorrect or mistimed control actions, this command could lead to an inadvertent launch.

After identifying unsafe control actions and the context in which they become unsafe, the next step is to identify the scenarios that could lead to the unsafe control action generation so that the scenarios can be eliminated from the design. The following are examples of the types of scenarios described by the people that did the analysis:[15]

> "The Fire Control computer is intended to send the *Fire Enable* command to the Launch Station upon receiving a *Weapons Free* command from an FMIS operator and while the fire control system has at least one active track. According to the requirements and design specifications, the handling of the *Weapons Free* command is straightforward. Although the specification requires an "active" track, it is more difficult to determine what makes a track active. Interviews with the development staff clarified that activity criteria are specified by the FMIS operators according to their operational procedures. The software supports declaring tracks inactive after a certain period with no radar input, after the total predicted impact time for the track, and/or after a confirmed intercept. It appears one case was not well considered: if an operator deselects all of these options, no tracks will be marked as inactive. Under these conditions, the inadvertent entry of a *Weapons Free* command would send the *Fire Enable* command to the

[15] Pereira, Lee, Howard, *op cit*.

> launch station immediately, even if there were no threats to engage currently tracked by the system."
>
> "The FMIS system undergoes periodic system operability testing using an interceptor simulator that mimics the interceptor flight computer. Hazard analysis of the system [using STPA] identified the possibility that commands intended for test activities could be sent to the operational system. As a result, the system status information provided by the launch station includes whether the launch station is connected only to missile simulators or to any live interceptors. If the Fire Control computer detects a change in this state, it will warn the operator and offer to reset into a matching state. However, there is a small window of time before the launch station notifies the Fire Control component of the change during which the Fire Control software might send a *Fire Enable* command intended for test to the live launch station."
>
> The fire control system may identify a potential threat and send the *Fire Enable* command. If that threat is later determined to be friendly, then the Fire Control Computer will send a *Fire Disable* command. The commands are sent on different communication channels. The *Fire Enable* command could potentially be delayed enough that the two commands are received in the opposite order than they were sent and the Launch Station would think that the *Fire Enable* command was active.

Note that in the first two examples, neither of the causal factors identified by the assessment involved component failures. In both cases, all the components involved were operating exactly as intended; however, the complexity of their interactions led to unanticipated system behavior (often called "unknown unknowns" by engineers) and unsafe component requirements. Component failure can, of course, be a cause of inadequate control over a system's behavior, and STPA does include those possibilities. However, the traditional hazard analysis techniques used today are based on an assumption that accidents are caused by component failures and thus consider only failure events and not the effects of complex system interactions. The third example could be caused by either a component failure, degraded performance of a component, or once again a design error that does not involve any failure.

Appendix B: Why the Space Shuttle Software was so Good

A mythology has arisen about the Space Shuttle software with claims being made about it being "perfect software" and "bug-free" or having "zero-defects," all of which are untrue. But the overblown claims should not take away from the remarkable achievement by those at NASA and its major contractors (Rockwell, IBM, Rocketdyne, Lockheed Martin, and Draper Labs) and smaller companies such as Intermetrics (later Ares), who put their hearts into a feat that required overcoming what were tremendous technical challenges at that time. They did it using discipline, professionalism, and top-flight management and engineering skills and practices. And the results surpass what has been possible to achieve more recently on similar challenges.

There can always be differing explanations for success (or failure) and varying emphasis placed on the relative importance of the factors involved. Personal biases and experiences are difficult to remove from such an evaluation. But most observers agree that the process and the culture were important factors in the success of the Shuttle software as well as the strong oversight, involvement, and control by NASA.

1. Oversight and Learning from the Past: NASA learned important lessons from previous spacecraft projects about the difficulty and care that need to go into the development of the software. These lessons include that software documentation is critical, verification must be thorough and cannot be rushed to save time, requirements must be clearly defined and carefully managed

before coding begins and as changes are needed, software needs the same type of disciplined and rigorous processes used in other engineering disciplines, and quality must be built in from the beginning. By maintaining direct control of the Shuttle software rather than ceding control to the hardware contractor and, in fact, constructing their own software development "factory," NASA ensured that the highest standards and processes available at the time were used and that every change to human-rated flight software during the long life of the Shuttle was implemented with the same professional attention to detail.

2. Development Process: The development process was a major factor in the software success. Especially important was careful planning before any code was written, including detailed requirements specification, continuous learning and process improvement, a disciplined top-down structured development approach, extensive record keeping and documentation, extensive and realistic testing and code reviews, detailed standards, and so on.
3. The Software Development Culture: Culture matters. The challenging work, cooperative environment, and enjoyable working conditions encouraged people to stay with the project. As those experts passed on their knowledge, they established a culture of quality and cooperation that persisted throughout the program and the decades of Shuttle operations and software maintenance activities.

With the increasing complexity of NC^3 and the enormous amount of software that is or will be included in such systems, a lesson that can be learned is that we will need better system engineering, including system safety engineering than is usually practiced today. NASA maintained control over the system engineering and safety engineering processes in the Shuttle and employed the best technology in these areas at the time. The two Shuttle losses are reminders that safety involves more than simply technical prowess, however, and that management can play an important role in accidents and must be part of the system safety considerations. In addition, our system and safety engineering techniques need to be upgraded to include the central role that software plays in our complex defense systems. Unfortunately, the traditional hazard analysis techniques used in the Shuttle do not work very well for the more software-intensive systems being engineered today and will need to be considerably changed and improved.[16]

Beyond these lessons learned**,** some general conclusions and analogies can be drawn from the Shuttle experience to provide guidance for the future. One is that high quality software is possible but requires a desire to do so and an investment of time and resources. Software quality may be given lip service in many industries, where often speed and cost are the major factors considered, quality simply needs to be "good enough," and frequent corrective updates are the norm.

Some have suggested that the unique factors that separated the Shuttle from other software development projects are that there was one dedicated customer, a limited problem domain, and a situation where cost was important but less so than quality.[17] But even large government projects with a single government customer and large budgets have seen spectacular failures in the recent past such as new IRS software,[18] several attempted upgrades to the Air Traffic Control system,[19] a new FBI system,[20] and even an airport luggage system.[21] That latter baggage system cost $186,000,000 for construction

[16] Nancy Leveson (2012), *Engineering a Safer World*, MIT Press.
[17] Charles Fishman (1996), "They Write the Right Stuff," *Fast Company*, December.
[18] Anne Broache (2007), "IRS Trudges on with Aging Computers," CNET News, April 12, http://news.cnet.com/2100-1028_3-6175657.html
[19] Mark Lewyn (1993), "Flying in Place: The FAA's Air Control Fiasco," *Business Week*, April 26, pp. 87, 90
[20] Dan Eggan and Griff Witte (2006), "The FBI's Upgrade That Wasn't," *The Washington Post*, August 18, http://www.washingtonpost.com/wp-dyn/content/article/2006/08/17/AR2006081701485.html
[21] Kirk Johnson (2005), "Denver Airport Saw the Future. It didn't work," *New York Times*, August 27.

alone and never worked correctly. The other cited projects involved, for the most part, at least an order of magnitude higher costs than the baggage system and met with not much more success. In all of these cases, enormous amounts of money were spent with little to show for them. They had the advantage of newer software engineering techniques, so what was the significant difference?

One difference is that NASA maintained firm control over and deep involvement in the development of the Shuttle software. They used their experience and lessons learned from the past to improve their practices. Other government projects have ceded control to contractors, often with conflicts between quality and other goals. With the current push to privatize the development of space vehicles, it will be interesting to see whether the lesser oversight and control lead to more problems in the future with manned spaceflight.

In addition, software engineering has moved in the opposite direction from the process used for the Shuttle software development, with requirements and careful pre-planning relegated to a less important position than starting to code (e.g., Agile and X-treme programming). Strangely, in many cases, a requirements specification is seen as something that is generated after the software design is complete or at least after coding has started. Many of these new software engineering approaches are being used by the firms designing defense systems today.

Why has it been so difficult for software engineering to adopt the disciplined practices of the other engineering fields? There are still many software development projects that depend on cowboy programmers and "heroism" and less than professional engineering environments. How will the DoD and international entities ensure that the private companies building more software-intensive NC^3 systems instill a successful culture and professional environment in their software development groups? Ironically, many of the factors that led to success in the Shuttle software were related to limitations of computer hardware in that era, including limitations in memory that prevented today's common "requirements creep" and uncontrolled growth in functionality as well as requiring disciplined control over the system requirements. Without the physical limitations that impose discipline on the development process, how can we impose discipline on ourselves and our projects?

The overarching question is how will we ensure that the hard learned lessons in the past are conveyed to those designing future systems and that we are not, in the words of Santayana, condemned to repeat the same mistakes.