1

The Theory of System Health Management

Stephen B. Johnson
NASA Marshall Space Flight Center and University of Colorado at Colorado Springs, USA

Overview

This chapter provides an overview of system health management (SHM), and a theoretical framework for SHM that is used throughout the book. SHM includes design and manufacturing techniques as well as operational and managerial methods, and it also involves organizational, communicative, and cognitive features of humans as social beings and as individuals. The chapter will discuss why all of these elements, from the technical to the cognitive and social, are necessary to build dependable human–machine systems. The chapter defines key terms and concepts for SHM, outlines a functional framework and architecture for SHM operations, describes the processes needed to implement SHM in the system lifecycle, and provides a theoretical framework to understand the relationship between the different aspects of the discipline. It then derives from these and the social and cognitive bases some design and operational principles for SHM.

1.1 Introduction

System health management (SHM) is defined as *the capabilities of a system that preserve the system's ability to function as intended*.[1] An equivalent, but much wordier, description is "the capability of the system to contain, prevent, detect, diagnose, respond to, and recover from conditions that may interfere with nominal system operations." SHM includes the actions to design, analyze, verify, validate, and operate these system capabilities. It brings together a number of previously separate activities and techniques, all of which separately addressed specific, narrower problems associated with assuring successful system operation. These historically have included analytical methods, technologies, design and manufacturing processes, verification and validation issues, and operational methods. However, SHM is not a purely technical endeavor, because failures largely originate in the organizational, communicative, and cognitive features of humans as social beings and as individuals.

[1]The idea that SHM exists to preserve functionality was first clearly expressed in Rasmussen (2008).

System Health Management: with Aerospace Applications, First Edition. Edited by Stephen B. Johnson, Thomas J. Gormley, Seth S. Kessler, Charles D. Mott, Ann Patterson-Hine, Karl M. Reichard, and Philip A. Scandura, Jr.
© 2011 John Wiley & Sons, Ltd. Published 2011 by John Wiley & Sons, Ltd.

SHM is intimately linked to the concept of *dependability*, which refers to *the ability of a system to function as intended*, and thus SHM refers to the capabilities that provide dependability.[2] Dependability subsumes or overlaps with other "ilities" such as reliability, maintainability, safety, integrity, and other related terms. Dependability includes quantitative and qualitative features, design as well as operations, prevention as well as mitigation of failures. Psychologically, human trust in a system requires a system to consistently perform according to human intentions. Only then is it perceived as "dependable." The engineering discipline that provides dependability we shall call "dependability engineering." When applied to an application, dependability engineering then creates SHM system capabilities. This text could easily have been called *Dependability Engineering: With Aerospace Applications*. The relationship of dependability engineering to SHM is much like that of aerospace engineering to its application domain, in that there is no "aerospace subsystem," but rather a set of system capabilities designed by aerospace engineers, such as aerodynamic capabilities of lift and drag, mission plans and profiles, and then the coordination of many other subsystems to control the aircraft's dynamics, temperatures, electrical power, avionics, etc. SHM is the name of all the "dependability capabilities" which are embedded in a host of other subsystems.

Within the National Aeronautics and Space Administration (NASA), a recent alternative term to SHM is "fault management" (FM), which is defined as "the operational capability of a system to contain, prevent, detect, diagnose, respond to, and recover from conditions that may interfere with nominal mission operations." FM addresses what to do when a system becomes "unhealthy." To use a medical analogy, FM is equivalent to a patient going to the doctor once the patient is sick, whereas SHM also includes methods to prevent sickness, such as exercise and improved diet, which boost the immune system (improve design margins against failure). For the purposes of this book, FM will be considered the operational aspect of SHM. SHM includes non-operational mechanisms to preserve intended function, such as design margins and quality assurance, as well as operational mechanisms such as fault tolerance and prognostics.

Major events in the evolution of SHM are given in Table 1.1.

The recognition that the many different techniques and technologies shown in Table 1.1 are intimately related and should be integrated has been growing over time. Statistical and quality control methods evolved in World War II to handle the logistics of the massive deployment of technological systems. The extreme environmental and operational conditions of aviation and space drove the creation of systems engineering, reliability, failure modes analysis, and testing methods in the 1950s and 1960s. As aerospace system complexity increased, the opportunity for failures to occur through a variety of causal factors also increased: inadequate design, manufacturing faults, operational mistakes, and unplanned events. This led in the 1970s to the creation of new methods to monitor and respond to system failures, such as the on-board mechanisms for deep-space fault protection on the Voyager project and the Space Shuttle's redundancy management capabilities. By the 1970s and 1980s these technologies and growing system complexity led to the development of formal theory for fault-tolerant computing (Byzantine fault theory), software failure modes and fault tree analyses, diagnostic methods, including directed graphs, and eventually to methods to predict future failures (prognostics). Total quality management, which was in vogue in the late 1980s and early 1990s, was a process-based approach to improve reliability, while software engineers created more sophisticated techniques to detect and test for software design flaws. By the early 2000s, and in particular in response to the *Columbia* accident of 2003, NASA and the DoD recognized that failures often resulted from a variety

[2]The most well-known definitions of dependability are "the ability to deliver a service that can justifiably be trusted," or alternatively, "the ability to avoid service failures that are more frequent and more severe than is acceptable" (Avizienis *et al.*, 2004). We believe that our definition is functionally equivalent to the Avizienis–Laprie concepts, and prefer our simpler, more concise definition. The Avizienis–Laprie concepts use the idea of "service," which in our terms is "function," and the issues of trust and frequency are, in our definition, subsumed into the idea of "intended function." There are a variety of ways in which intended function or service can fail, whether in frequency, duration, or magnitude. If the system does not, or is not predicted/expected to, provide the intended function or service, then it will not be trusted to do so.

Table 1.1 Major events in the development of SHM

Date	Event
1950s	• Quality control • Reliability analysis, failure modes and effects analysis (FMEA) • Environmental testing • Systems engineering
1960s	• Fault tree analysis, hazards analysis • Integrated system test and "search for weaknesses" test • Hardware redundancy
1970s	• Reliability-centered maintenance • Software FMEA and software reliability analysis • Redundancy management, on-board fault protection • Early built-in test (primarily push-to-test or go/no-go testing)
1980s	• Byzantine fault theory (1982) • Software fault tree analysis, directed graphs • DoD integrated diagnostics • Boeing 757/767 maintenance control and display panel (mid-1980s) • NASA and DoD subsystem and vehicle health monitoring (late 1980s) • Aerospace Corporation Dependability Working Group (late 1980s) • ARINC-604 *Guidance for Design and Use of Built-In Test Equipment* (1988) • Principles of Diagnostics Workshop (1988) • Total quality management • Boeing 747-400 central maintenance computer
1990s	• Condition-based maintenance • *System Health Management Design Methodology* (1992) • *Dependability: Basic Concepts and Terminology* (1992) • ARINC-624 *Design Guidance for Onboard Maintenance System* (1993) • Boeing 777 onboard maintenance system (1995) • (Integrated) system health management • Directed graphs applied to International Space Station • Operational SHM control loop concept (1995) • SHM diagnostics technologies, sensor technologies, prognostics • Bi-directional safety analysis, probabilistic risk assessment
2000s	• *Columbia Accident Investigation Board Report* (2003) • Air Force Research Laboratory ISHM Conference established (2004) • Integrated System Health Engineering and Management Forum (2005) • American Institute of Aeronautics and Astronautics Infotech Conference (2005) • NASA Constellation SHM – Fault Management (FM) Methodology • NASA Science Mission Directorate FM Workshop (2008) • Prognostics and Health Management Conference (2008) • Control System and Function Preservation Framework (2008) • *International Journal of Prognostics and Health Management* (2009) • Prognostics and Health Management Society established (2009) • Constellation FM team established (2009) • *NASA Fault Management Handbook* writing begins (2010) • NASA FM Community of Practice (2010) • *SHM: With Aerospace Applications* published (2011)

of cultural problems within the organizations responsible for operating complex systems, and hence that failure was not a purely technical problem.

The term "system health management" evolved from the phrase "vehicle health monitoring (VHM)," which within the NASA research community in the early 1990s referred to proper selection and use of sensors and software to monitor the health of space vehicles. Engineers soon found the VHM concept deficient in two ways. First, merely monitoring was insufficient, as the point of monitoring was to take action. The word "management" soon substituted for "monitoring" to refer to this more active practice. Second, given that vehicles are merely one aspect of the complex human–machine systems, the term "system" soon replaced "vehicle," such that by the mid-1990s, "system health management" became the most common phrase used to deal with the subject. By the mid-1990s, SHM became "integrated SHM" (ISHM) within some parts of NASA, which highlighted the relatively unexplored system implementation issues, instead of classical subsystem concerns.

In the 1980s, the DoD had created a set of processes dealing with operational maintenance issues under the title "Integrated Diagnostics." The DoD's term referred to the operational issues in trying to detect failures, determine the location of the underlying faults, and repairing or replacing the failed components. Given that failure symptoms frequently manifested themselves in components that were not the source of the original fault, it required "integrated" diagnostics looking at symptoms across the entire vehicle to determine the failure source. By the mid-1990s the DoD was promoting a more general concept of condition-based maintenance (as opposed to schedule-based maintenance), leading to the development of a standard by the early 2000s. By the 2000s "enterprise health management" was becoming a leading term for the field.

Another recent term for SHM is prognostics and health management (PHM), though from all appearances, the subject matter is identical to SHM, since SHM encompasses prognostics as a specific technique for maintaining system health. The PHM term graces a new "PHM society" established in 2009 and with its own conferences and an online journal, the *International Journal of Prognostics and Health Management*, which was formed in 2009.

Within NASA's Science Mission Directorate, the recognition that on-board design to address failures had become a major cost and schedule issue within science spacecraft programs was highlighted in the Fault Management Workshop of 2008. This workshop led to a set of findings about the typical technical, organizational, cost, and schedule problems associated with these on-board functions, to the institution of a Fault Management Group in the Constellation Program, and to the creation of a *Fault Management Handbook*, which as of October 2010 is in development. There is still some debate as to the scope of FM versus SHM, whether these are synonyms for each other, or whether FM is the operational subset of SHM. In this book we shall interpret FM as the latter.

As described in the Foreword, this text emerged from the NASA Marshall Space Flight Center and Ames Research Center-sponsored Forum on Integrated System Health Engineering and Management, held in Napa, California, in November 2005. The book editors decided that the term "system health management" most concisely captured the major goal of the discipline, to manage the health of the system, through organizational and technical means. As systems by their nature are and should be integrated, the editors decided not to use the term "integrated" in the title. The goal of this book is to provide SHM practitioners, who are typically expert in one set of techniques and one application area, an educational resource to learn the basics of the related disciplines, issues, and applications of SHM in aerospace. Though the "system applications" in Part Six are aerospace focused, the rest of the sections are general in nature and apply to many different application areas outside of aerospace. Even the aerospace applications provide very different emphases that have similarities to those of other applications outside of aerospace, such as chemical process industries, electrical power generation and distribution, and computer network applications.

Organizing SHM as a discipline provides a conceptual framework to organize knowledge about dependable system design and operations. It also heightens awareness of the various techniques to create and operate such systems. The resulting specialization of knowledge will allow for the creation of theories and models of system health and failure, of processes to monitor health and mitigate failure, all with greater depth and understanding than exist in the fall of 2010. We feel this step is necessary, since the disciplines and processes that currently exist, such as reliability theory, systems engineering, management theory and others, practiced separately, have not and cannot separately meet the challenge of our increasingly sophisticated and complex systems. As the depth of SHM knowledge increases, the resulting ideas must be fed back into other disciplines and processes in academic, industrial, and government contexts.

1.2 Functions, Off-Nominal States, and Causation

SHM's primary goal is to preserve the system's ability to function as intended. To understand the ramifications of this goal, we must introduce and define a number of concepts, including: system, intended functions, states and behaviors, classes of off-nominal states, and causation of off-nominal states.

According to the International Council on Systems Engineering, a system is "a construct or collection of different elements that together produce results not obtainable by the elements alone" (INCOSE, 2010). For engineered systems, these "results" are the purposes (goals, objectives) for which the system was created. The system's designers and operators "intend" for the system to perform these goals, and it is these intentions that ultimately define whether the system is performing properly. Intent is defined by anyone that "uses" or "interacts" with the system, whether as designer, manufacturer, operator, or user.

In mathematical terms, a system performs a function $\mathbf{y} = f(\mathbf{x})$, where $\mathbf{x}$ is the input state vector, $\mathbf{y}$ is the output state vector, and f is the system process that transforms the input state into the output state. The system can be divided into subsystems and components, each of which can be described in the same mathematical, functional way. Functions are allocated to mechanisms (which can be hardware, software, or humans), such that the function f can be implemented in several possible ways. Functions are implemented by mechanisms, and their operation is characterized by how they affect system states. The temporal evolution of a state is called a "behavior." Behaviors can also be characterized as states, as they simply form a new state vector of individual states from a set of time samples. During system operations, it is not possible to definitively know the system's "true state." Instead, operators only have information from which an "estimated state" is determined. In general, when we refer to "states" in operations, we mean the "estimated state." In analysis and testing, the "true state" is often assumed or known, such as when a known fault is injected into a system test.

A system is considered "nominal" if the output state vector matches the intentions of the designer and/or operator. A system is "off-nominal" if it does not. Off-nominal states come in three types: failures, anomalies, and degradations. *Failure* is the *unacceptable* performance of the intended function. *Degradation* is the *decreased* performance of the intended function. *Anomaly* is the *unexpected* performance of the intended function.[3]

Contrary to common wisdom, off-nominal states are not definitively specified in a system's requirements or specifications. No single specification or model defines all of the details of what as system is intended to do, and how people expect it to behave. This is ultimately due to the fact that each person that interacts with the system has his or her own individual model of the system, which could be formal (a computer simulation or mathematical model) or informal (in a person's mind, based on the person's prior interactions with the system). This model determines how a person translates the (designers' or operators') intentions for the system into individual expectations of current and future system behaviors. It is ultimately an *individual*, and very frequently a *social* (or community),

[3]These are our definitions, and they do not precisely match a variety of other standards and documents that have their own definitions. These definitions, and others in this chapter, have been developed and refined over the course of more than two decades. The Preface briefly describes much of this heritage, except for the significant work of Laprie and Avizienis since the mid-1990s on the nature of faults and failures in computing.

decision as to what constitutes an off-nominal state, and there can be significant arguments between groups and individuals about how to classify a given state. Over time, as a particular off-nominal state is investigated, the states can be reclassified from nominal, degraded, anomalous, or failed into one of the other categories. This typically happens because people change their minds about the proper classification once they understand the cause(s) of the observed states and behaviors. They then adjust their individual (formal and/or informal) models, leading to a different classification. This reflects normal learning about the system over time during operations.

It is possible to have states that are anomalous and failed, anomalous and not failed, and failed but not anomalous. For example, the loss of *Pioneer 10* due to component wearout of its radioisotope thermal generator was a failure, but there was no anomaly, because the failure was predicted years in advance and happened just as predicted and expected. Conversely, anomalies that are not failures are common, such as a power signature that differs from previous signatures of the same function. An example is a transient fluctuation when a switch is closed that differs from previous times when that same switch was closed. In most cases, failures are also anomalous. This same logic holds for the relationship of anomalies to degradations. Degraded and failed states are both on the same axis of “functional acceptability,” where degraded is not ideal but still acceptable, but failed is not acceptable. Anomalous behavior is a separate, orthogonal classification axis.

The typical response to an anomaly is to investigate the behavior so as to determine its cause, because it could indicate component degradation or low-level component failure that manifests as a different behavioral signature than when that component was operating nominally. Investigation of the anomaly leads to three possible outcomes: (1) the anomaly remains not understood, and remains an anomaly; (2) the anomaly is judged as acceptable behavior and is reclassified as nominal; (3) the anomaly is judged unacceptable and is deemed a failure. In the second and third cases, the former anomaly now is understood and “expected,” and system models used to classify system states are adjusted so that future occurrences of the behavior are automatically classified as nominal or as failures. Two prominent examples of the second case are from the Space Shuttle Program. Both the Shuttle foam strike phenomenon and partial charring of O-rings were initially classified as anomalies and failures according to the Shuttle’s initial specifications, but were reclassified over time to be normal behavior, in the sense that they were reclassified from safety to maintenance issues (Vaughan, 1996).[4] While both of these cases led to tragedies, the fact remains that, for complex systems, there are many behaviors that prior to operations are considered anomalies and/or failures, but are reclassified as normal once flight experience has been acquired and the anomalies and failures are investigated. The issue is not whether classification occurs – it does, all the time – but rather if it is done correctly.

The basis for reclassification is most often the determination of the cause of the off-nominal state. That is, knowledge of the cause, and projection of the future effects from that cause, determine whether an off-nominal state is acceptable or not, and hence what actions should be taken. Causation of off-nominal behavior is therefore a critical SHM topic. Internal causes of failures are called faults, whereas external causes of failure are simply called external causes. A *fault* is defined *as a physical or logical cause internal to the system, which explains a failure*. Simply put, it is an *explanation* for observed failure effects. If an anomaly exists, and an investigation judges the anomalous behavior as failed and the cause is internal to the system, then the cause is a fault. Conversely, if the behavior is judged as nominal, or as failed but the cause is external, then no fault exists. The term “fault” in common English has two meanings of significance for SHM: *causation* and *responsibility*. Investigation of failure, which is a significant aspect of SHM, addresses both of these concerns. Therefore, it is important for dependability engineering as a discipline to have a definition of “fault” that encompasses both meanings. The definition of a fault as a cause internal to the system enables both interpretations. For example, if a Mars lander lands on a rock that is too large, it will tip over and fail. In this situation, we would not normally say that it is “Mars’s fault” that the lander tipped over, particularly if the risks

[4]The O-ring case became the basis of Vaughan’s concept of “normalization of deviance.”

were known, low, and acceptable. We would say that it was just bad luck in this case. However, if the operators knew that the region in which the landing was to occur had many such rocks, and that the operators took unnecessary risks by landing there, then there is a fault, which is that the operators made a flawed decision to land at this location.

Fault and failure are interdependent, recursive concepts of “cause” and “effect.” Seen from one perspective, a fault explains a given failure, but from another, that same fault is seen as the failure that needs an explanation. For example, in the *Columbia* tragedy of 2003, the hole in the leading edge of the wing is the failure that needs explaining, and its cause is a chunk of insulation foam hitting the wing and causing a structural breach. However, from the perspective of the designers of the external tank, the foam falling off the external tank is the failure to be explained, and its cause was air bubbles in the foam insulation. In turn, the air bubbles in the foam can be seen as the failure, and flaws in the foam application process seen as the fault that explains it. This process can continue for quite a long time in a failure investigation, but ultimately the investigation stops and no further causes are sought. The first causes in these long chains of explanation are the *root causes* – failure is often the result of several root causes that interact. The term “root cause” is also relative, because as far as one group are concerned, the explanation that satisfies them so that they require no deeper explanation is their root cause. However, another group may not be satisfied with this. For them, the original group’s root cause is not a cause at all, but a failure to be explained. When they stop their investigation, their first causes are the root causes. The recursive nature of these terms helps to explain the major difficulties that many groups have had in defining them, but also explains their utility.[5] Figure 1.1 illustrates the relationship of a number of these concepts.

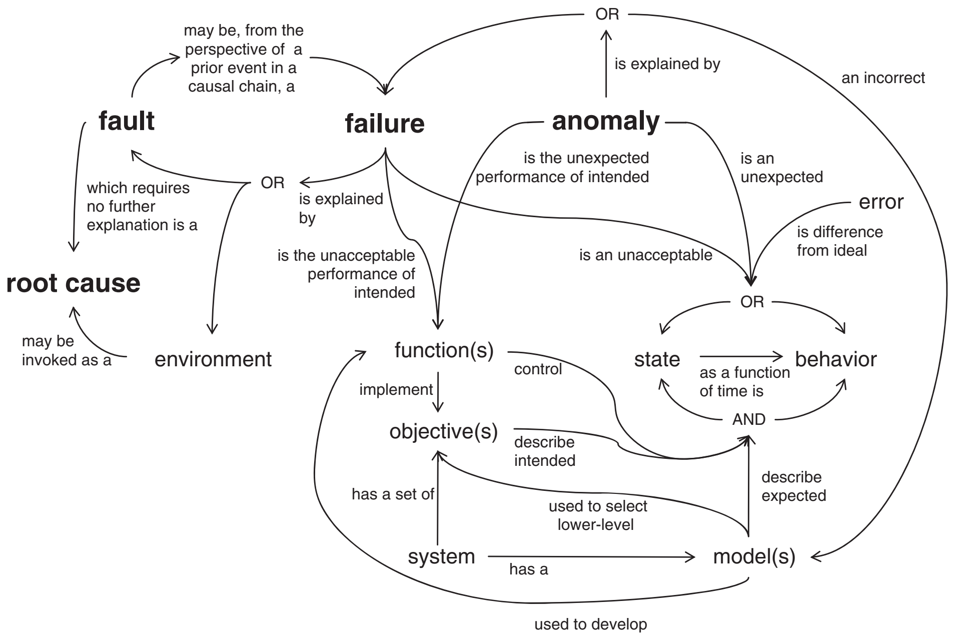

Figure 1.1 Concept diagram for major SHM terms

[5]These definitions of fault and failure as cause and effect are related to the notion in the philosophy of science that “explanation” is fundamental to the purpose of science. An example through a colorful story is on pages 132–4 of Van Fraasen (1980).

Human causation of the majority of faults is a key axiom of SHM theory. *Human faults*, whether individual or social via miscommunication or lack of communication, *are the root causes for most failures*, other than the relatively small percentage of failures caused by expected system wearout or environmental causes. While comprehensive statistical studies have not been compiled, discussions with those who have compiled aerospace failure databases suggest that the vast majority (most likely 80% or more) of failures are ultimately due to one of two fundamental causes: *individual performance failures* and *social communicative failures*. This should come as little surprise. Humans create and operate systems for their own purposes and with their own individual and social processes, and it is human failings in these areas that lead to internal faults in design, manufacturing, or operations. Human causation of the majority of faults is the basis of the Columbia Accident Investigation Board's finding that NASA's "culture" is deeply implicated in disasters in human spaceflight. In essence, if you search deep enough for root causes, you find most of them are human causes, and thus addressing these individual cognitive and social communicative issues is relevant to reducing the probability of system failure, by reducing the number of faults introduced into the system by humans.[6]

The results of human faults differ, depending on when they occur in the system lifecycle. Human mistakes in the design phase generally lead to "design faults" or "common mode failures," since they are common to all copies of the system. Mistakes in manufacturing generally lead to faults in single copies of the system. These are typically called "random part failure," though the label of "random" is usually a cover for our inability to find the human fault that is almost always the root cause. In manufacturing, faults can also lead to failures in all copies of the system, but when this is true, the fault is in the design of equipment that manufactures multiple copies, in which case the fault is ultimately a design flaw. Mistakes in operations are generally considered human operational faults and are often blamed on the operators. However, most failures are ultimately due to humans and thus share this fundamental similarity.

The implication of human causation is that SHM must address all failure causes, whether "design faults," "manufacturing faults," or "operator faults," and that the basic rates of occurrence of these faults are roughly the same due to common human causation.

1.3 Complexity and Knowledge Limitations

Humans regularly build systems that produce behaviors that the human designers and operators did not expect or understand prior to their occurrence. This basic fact defines the concept of complexity. We define something as *complex* when it is *beyond the complete understanding of any one individual*. In passing, we note that many systems such as the Space Shuttle elude the complete understanding of entire organizations devoted to their care and operation. Because of their complexity, aerospace systems *must* have several or many people working on them, each of whom specializes in a small portion. The system is subdivided into small chunks, each of which must be "simple" enough for one person to comprehend. A fundamental limitation on any system design is the proper division of the system into cognitively comprehensible pieces. This is a key facet of systems engineering, though it is not generally expressed in this manner (Johnson, 2003).

The inability of humans to understand their creations has a specific implication for SHM, which is that SHM engineers must assume that the system has behaviors that nobody understands or expects to occur. Since they are not expected, they are assumed improbable, but logic requires that these be more probable than the usual assumption of "so improbable as to ignore," and history indicates that it is nearly certain that some of these will manifest themselves during system operation. Some of these behaviors can cause system failure. The SHM design must include capabilities to address this issue.

[6]Over the last few decades, research on the nature of technology in the social science community has made the connection between humans and the technologies they develop quite clear, most obviously in the theory of the "social construction of technology." A foundational text is Bijker *et al.* (1987).

1.4 SHM Mitigation Strategies

The purpose of SHM is to preserve system functionality, or, stated differently, to control state variables within an acceptable range, in the presence of current or predicted future failures. If the system always functioned perfectly and ideally, SHM would not be necessary. However, over time, due to regular wear and tear, components degrade, or due to other internal or external causes they fail such that the system's nominal control systems are unable to sustain function. For an active control system, this is often because the sensors, processors, or actuators that the control system assumes are operating normally have failed, making the control algorithm ineffective since it usually assumes these components are working properly. For passive control of a state variable, such as is typical for structures, the structures themselves degrade and fail due to dynamic or thermal cycling. A common example is an aircraft's wings, which have a finite lifetime under standard dynamic loads. In either case, the system's active or passive design margins are exceeded and failure ensues. SHM provides passive capabilities to prevent faults, active capabilities that take over when the regular control system fails, and active capabilities to predict future failure and take action to prevent or delay it.

To design SHM into the system, the first step must be to identify the system's functions. This is typically accomplished by performing a systems engineering functional decomposition, which defines system functions from the top down. Typically this is represented as a tree structure (a function or "success" tree) and/or in a functional flow block diagram. The former is a semi-time-independent functional decomposition, whereas the latter is an event sequence diagram, addressing the time and/or mission sequencing of system functions.

Each system function has the possibility of failure, and the goal to preserve functionality in the face of impending or actual failure implies that each function defined from the system decomposition must be assessed from the standpoint of how that function can be preserved or protected. The SHM designer must determine the function preservation strategy. At the highest level, there are only two choices for preserving function: to prevent failure, or to tolerate failure, as shown in Figure 1.2.

Failure prevention can be accomplished by design-time fault avoidance, or by operational failure avoidance. *Design-time fault avoidance* means that failure causes are prevented, usually through design margins and quality assurance measures. *Operational failure avoidance* means operationally predicting when a failure will occur, and taking operational measures to prevent its occurrence. These operational measures can include retirement of the system prior to failure if repair is not possible (thus avoiding a hazardous or undesirable consequence should the system fail, such as retiring an aircraft before its wings fail), alteration of the system's operation to delay the inevitable failure (reducing loads on the stressed components), or repair or replacement of the stressed component before failure (such as schedule-based or condition-based maintenance).

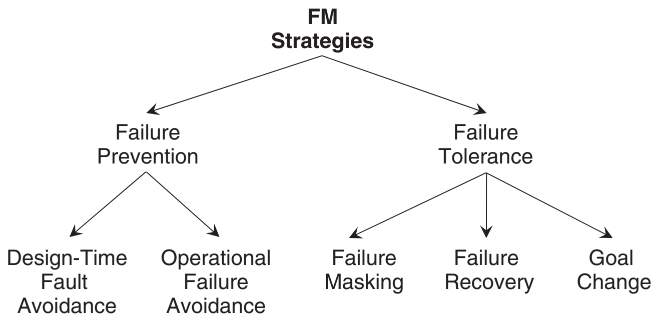

Figure 1.2 SHM function preservation strategies

Failure (or fault) tolerance strategies include failure masking, failure recovery, or goal change. *Failure masking* is the strategy of allowing a failure in the system, but preventing that failure from compromising the critical function of interest. This is usually performed by detection and containment of the failure before it propagates downstream to the function that cannot be compromised. The *failure recovery* strategy is possible if the function can be temporarily compromised. In this strategy, a failure occurs and the function is compromised, but it is detected and a response taken that reestablishes control such that the function is once again controlled acceptably, before any system goals are permanently compromised. System goals are not changed in the failure recovery strategy. The *goal change* strategy is applied when the failure effects are such that the current system functions cannot be maintained. In this strategy, the system switches to less demanding goals than the original set, such as a "safing" mode to preserve power and reestablish communication with Earth for a robotic spacecraft, or an abort mode to save the crew for a crewed spacecraft.

Institutionally, these strategies are typically implemented by several groups. When fault avoidance is selected for a function, the implementation of design margins occurs through design groups, manufacturing, and quality assurance organizations. Failure prediction and operational failure avoidance are typically implemented by operations groups, while the three fault tolerance strategies are implemented by SHM and element designers. Analysis of the effectiveness of these strategies is also split, this time between the SHM and element engineers, operations engineers (availability), and safety and mission assurance (reliability etc.) organizations. The SHM engineer has the primary role at the beginning of a project to determine the function preservation strategies, and then for implementation and analysis of these strategies to the extent that these are not already covered by other organizations. The assessment of the total effectiveness of the strategies in mitigating mission risks is typically split between SHM and reliability analysts.

1.5 Operational Fault Management Functions

When the design-time fault avoidance strategy is selected, its implementation leads to appropriate design margins, which are then analyzed for their effectiveness in ensuring appropriate component reliability for the various system components (components in this sense can mean hardware, software, or humans that implement functions). However, SHM engineers are typically not involved with the implementation, though they may be involved with analysis and testing to ensure proper implementation. For all of the other strategies, SHM engineers are involved. It is, in fact, the growing sophistication and pervasiveness of active prediction, operational failure avoidance, and fault tolerance designs that is the primary spur to the development of SHM as a discipline. This section will describe the functions of operational implementation, which is the "fault management" (FM) subset of SHM.

Under nominal conditions, the system has passive design margins and active control systems that provide system functionality, by maintaining state variables associated with each function within acceptable bounds. FM is the set of operational capabilities that perform their primary functions when the nominal system design is unable to keep state variables within acceptable bounds. To do this, FM, just like the nominal control system, operates as an active control loop, with the system providing information to functions that detect off-nominal conditions, functions to determine the cause(s) of these conditions, decision functions to determine what to do about these conditions, and actions (responses) that implement these decisions to preserve system function. The detection functions are continually active to monitor state variables and determine when they have reached an off-nominal condition. The diagnostic functions to isolate and identify causes of off-nominal conditions, the decision functions, and response functions to take action to remediate off-nominal conditions do not generally execute unless and until an off-nominal condition exists. Together, these "FM loops" define new system control regimes, which enable the system to function under a variety of off-nominal conditions. For failed conditions, the FM loops provide capability precisely when the nominal control regime can no longer control state variables. For degraded conditions that will eventually lead to failure, FM loops preempt

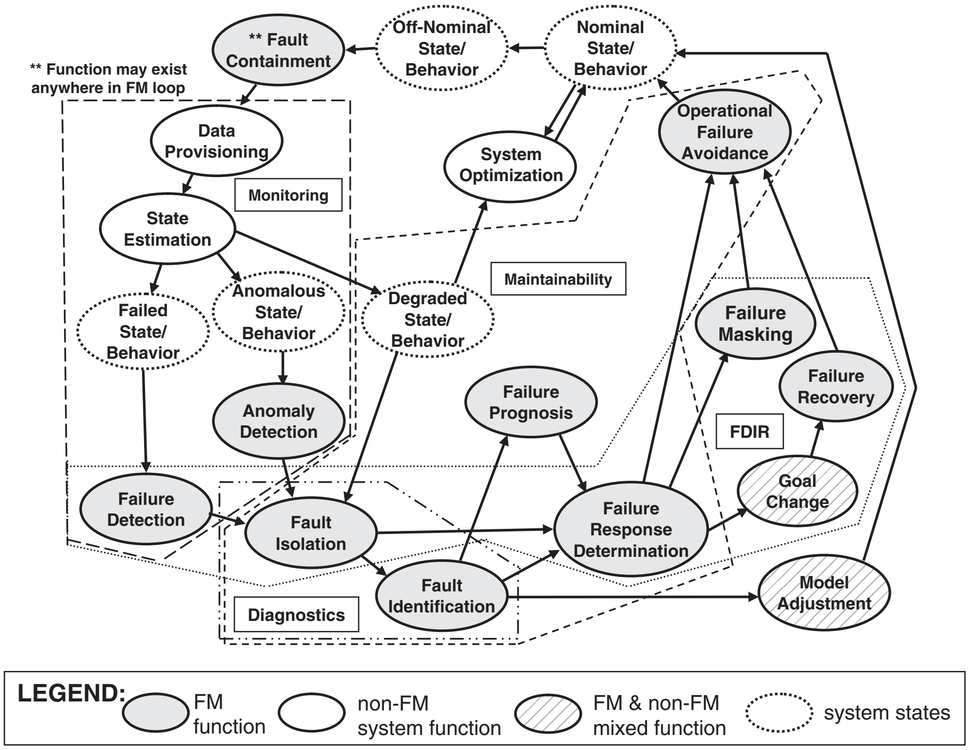

Figure 1.3 Operational FM control loops

failures so that the regular control system never loses control. As described above, design-time fault avoidance is not an active control loop, as it is a passive function. It is therefore not part of FM.

Figure 1.3 illustrates the relationship of FM functions, which operate in characteristic control loops. The diagram also shows the boundaries of some of the common labels under which SHM activities have historically occurred. The FM functions are defined below:

- *Anomaly detection:* Deciding that an anomaly exists.
- *Failure detection:* Deciding that a failure exists.
- *Failure masking:* An action to maintain intended function in the presence of failure.
- *Operational failure avoidance:* An action to prevent a failure from occurring.
- *Failure prognosis:* Predicting the time at which a component will fail.
- *Failure recovery:* An action taken to restore functions necessary to achieve existing or redefined system goals after a failure.
- *Failure response determination:* Selecting actions to mitigate a current or future failure.
- *Fault containment:* Preventing a fault from causing further faults.
- *Fault identification:* Determining the possible causes of off-nominal behavior.
- *Fault isolation:* Determining the possible locations of the cause of off-nominal behavior, to a defined level of granularity.
- *Goal change:* An action that alters the system's current set of objectives.
- *Model adjustment:* Modifying the model of the system upon which expectations of future states and behaviors are based.

Each FM loop consists of a suite of these functions, and together the entire FM loop must operate faster than the failure effects that the FM loop is intended to mitigate. The FM loop latencies are the sum total of the latencies required to sense, detect, isolate, decide, and respond to a predicted or current failure. These latencies must be less than the *time-to-criticality* (TTC), which is the amount of time it takes for failure effects to propagate from the failure mode along failure effect propagation paths (FEPPs) to the first *critical failure effect* (CFE). Both the FM loop latencies and the TTC are based on the physics by which the nominal or failure effects propagate, which can change along the FEPPs and FM loop paths. For example, failure effects propagating through electrons in a wire generally propagate on the order of a few milliseconds for wire lengths typical of aerospace systems, whereas failure effects propagating via fluid flows are typically on the order of several hundred milliseconds via atoms and molecules, and thermal effects in seconds or minutes. There can be multiple FEPPs and, less frequently, multiple FM loop paths for a single fault.

The CFE is *not* always the effect at the moment the mission actually fails. Rather, it is some intermediate effect, which, if it occurs, has irrevocably compromised system objectives, even if the ultimate mission failure or degradation will occur sometime further in the future. Consider a loss of propellant failure scenario in the cruise phase of a planetary probe. The effects of the propellant loss may not be ultimately manifested for months or years when the vehicle must perform orbit operations to gather science data. The relevant time to measure for each FM loop is to the CFE, which in this case is "the amount of time, given the rate of loss of propellant based on the current (and projected) leak size, when there will be not enough propellant to meet mission objectives." When several CFEs occur for a given fault, then the CFE of relevance is the one to which the failure effects propagate soonest.

Both the failure effect propagation times and FM loop latencies are complicated by the fact that many of the individual times are statistical in nature, when assessed during the design phase before any specific failure has occurred. For example, in a typical launch vehicle liquid rocket engine, small variations in input conditions can create dramatically different effects due to the nonlinearity of fluid flows. Post-failure analysis tracks a particular failure event sequence with a specific, deterministic time, but, seen prior to the fact, that particular sequence of events and times is only one of many possibilities. The FM designer must ultimately understand the effectiveness of the FM design, which in part is based on the temporal race conditions of the FM loops versus the TTCs for a given failure scenario, and then summed for all FM loops in all relevant failure scenarios. In complex cases, the analysis is often statistical in nature, based on Monte Carlo runs of physics-based models of the particulars of the system and its failure modes. Sometimes full Monte Carlo runs are unnecessary, such as when a worst-case analysis can be performed to bound the effectiveness estimate. The FM design must provide some quantitative improvement in system reliability and/or availability, and in some cases might only have to meet a given requirement, which could be shown through the worst-case analysis.

1.5.1 Detection Functions and Model Adjustment

Mitigation of off-nominal states requires that the system be able to detect that off-nominal states exist. Detection of failures and degradations requires a calculation of the *deviation between an estimated state and the ideal* ***intended*** *state* for a given variable, which we define as a *control error* in the control theory sense.[7] The ideal state is associated with the function that is implementing a system objective. Detection of anomalies is different, as it is based on the *deviation between an estimated state and the ideal* ***expected*** *state*, which is a *knowledge error* in the control theory sense. Failure and degradation detection signify that the system's behavior is no longer ideal, whereas anomaly detection signifies

[7]The term "error" has many meanings in common usage, including what we call failure, fault, mistake, and several others. We shall use the term only in the control system sense, as a control or knowledge error, unless stated otherwise.

that the knowledge of the system's behavior is inaccurate. The concept of error is most appropriate for continuous variables. For discrete variables (true or false, 1 or 0), the "error" is a simple inequality or mismatch between the desired value and the estimated value.

Detection functions generally classify the estimated state as "nominal," or one of the three "off-nominal" categories of failure, anomaly, degraded. We explicitly identify two FM detection functions: failure detection and anomaly detection. Though under consideration, we do not currently define a "degradation detection" function, but degradations must also be identified based on the criteria of decreased performance compared to the ideal. This comparison would be either against the system's original ideal performance value for a function, or against a current or recent performance value, depending on the purpose of classifying a state as degraded. For all three detection types, separation of a nominal from an off-nominal state requires some threshold to separate one from the other, or alternatively some mechanism that allows for a quantitative measurement of comparative nominal or off-nominal characteristics. Most typically a precise threshold value is used.

Anomaly and failure detections are significantly different functions, which frequently have different implementations. Failure detections can be based on knowledge of specific, known failure states and behaviors based on FMEA-identified failure modes, or they can be designed based on knowledge of what level of function compromise is unacceptable for achievement of the function's objective, regardless of the underlying failure mechanisms. The response to failure detection is ultimately to take some mitigating action. Anomaly detections are of very different design, identifying deviations of current behavior from prior or predicted behavior as the criterion of "unexpectedness." These are often based on artificial intelligence, training-based algorithms, but also encompass the engineering judgment of operators. The response to anomaly detection is to investigate the cause of the anomaly. If successful, the investigation classifies the estimated state as nominal, degraded, or failed. If not, the anomaly remains an anomaly. The investigation, if successful, acts as a failure or degradation detection function.

In many cases, the other result of an anomaly detection is a model adjustment, which modifies the observer's model of system behavior to reflect the new information learned from the investigation. After model adjustment, a future recurrence of the state originally detected as an anomaly would quickly be classified as degraded, failed, or nominal.

The FM detection and model adjustment functions are all aspects of state estimation, whose purpose is, as the term suggests, to make the best estimate of the true state of the system. As state estimation functions, their effectiveness of these functions is measured by false positive and false negative metrics for individual detection mechanisms, and by coverage metrics for the entire suite of failure detections. Put another way, the coverage metric determines what percentage of the total number of system faults (failure modes) the failure detection mechanisms can detect, adjusted by the probability of the various failure modes. The effectiveness metric reduces this fraction using the false positive/false negative (FP/FN) metrics.

Off-nominal detection algorithms often include filtering algorithms to separate transient events from permanent ones. These same algorithms can often provide information about the cause of the off-nominal detection. A typical simple example is a "three-strike algorithm," which is based on the idea that it is physically impossible for the state variable being measured to physically change rapidly, so that if drastic value changes occur, it is very likely an artifact of the physical mechanisms that measure the state variable, or the digital system that is transporting the measurement data, or an environmental effect on these mechanisms, and thus that this measurement is not providing any information about the state variable in question. Under the physical assumption that the rapid change is a single event upset (SEU), it is highly improbable that it will occur twice in a row. Requiring three consecutive measurement effectively eliminates the possibility of a false positive resulting from a relatively probable SEU event. If the big jump persists, then there is probably a permanent problem in the digital processing system, as opposed to the state variable that is being measured by the relevant observation/sensor.

Model adjustment, if done incorrectly, can lead to disastrous consequences. In the well-known cases of system failure such as the *Challenger* and *Columbia* accidents, anomalies were classified incorrectly as normal behaviors, instead of as unacceptable failures or even to remain anomalous. These model adjustments were called “normalization of deviance” by Diane Vaughan in her book *The Challenger Launch Decision*, but were not recognized as a normal engineering activity, nor by the “model adjustment” name. Far from being “deviant” or “incorrect” or “abnormal,” model adjustment occurs all of the time; the question is whether it is done properly. There are many system models. In fact, it is likely that there are as many models of the system as there are people who interact with the system, and then there are the many formal models of the system as well. Thus, “model adjustment” is an unrecognized and usually haphazard process, which can lead to misunderstandings and, as the *Challenger* and *Columbia* accidents teach us, to disaster. Far more attention must be paid to this function in the future than has been done in the past.

1.5.2 Fault Diagnosis

Fault diagnosis is the term that encompasses the two FM functions of fault isolation and fault identification. It can be considered as a composite function that aims to determine the location and mechanism of the underlying failure cause(s). Both fault isolation and identification are measured via ambiguity groups, which are groupings of components that cannot be distinguished from each other based on the detection signature provided by the failure detection and/or anomaly detection functions. If a specific set of failure detections or anomaly detections occurs, there are several possible components in which the underlying failure cause may exist, and it is not possible to determine which component among the set contains the causal mechanism.

Fault isolation attempts to answer the question of *where* the causal mechanism of an off-nominal state exists. The FM usage of the phrase “fault isolation” as a diagnostic function should not be confused with the use of the same phrase to describe a mechanism to prevent failure effects or causal mechanisms from spreading from one location to another. This usage is common in electrical applications and electrical engineering. In FM terminology, this is called “fault containment” and/or “failure containment,” and examples include mechanisms such as optical isolators or breaker circuits. The term “fault isolation” is historically used in fault management and its predecessors, but is somewhat misleading. The fault isolation function determines the location not just of faults (causes of failure inside the system boundary), but also of environmental failure causes. So it would be somewhat better termed “failure cause isolation,” though for historical reasons we hold to the commonly used term “fault isolation.”

Fault identification (sometimes called fault characterization) attempts to answer the question of *what* the causal mechanism of an off-nominal state is (or alternatively, *why* the failure occurred, which is usually explained by identifying the causal mechanism). Its implementation sometimes operates similarly to the fault isolation function in that automated diagnosis tools use similar techniques of forward and backward tracing of failure effects to determine the location of faults as it does to determine the possible failure modes that cause the off-nominal state. However, fault identification is frequently implemented quite differently than fault isolation, with humans performing tailored analyses to determine off-nominal behavior causes. As with fault isolation, fault identification seeks for causes that can be inside the system boundary (faults), or outside the boundary in the environment. It is frequently true that fault identification is not necessary for an effective failure response to be implemented. Often it is only necessary to determine the location of a fault, so as to remove it from the control loop.

The effectiveness of fault diagnosis functions is measured by ambiguity groups that list the sets of possible causes and locations, along with associated false positive and false negative rates assessed against these ambiguity groups.

1.5.3 Failure Prognosis

Prognosis is simply defined as prediction of future states or behaviors, and failure prognosis predicts when failure will occur. Failure prognosis typically uses a formal model to evaluate the future consequences of current system behavior. Current system state, behavioral, and environmental data is fed into the formal model, which is usually (though not always) physics based. Knowing the expected future goals and operations of the system, this model is then used to predict the point in time, or range of times, in which the function performed by these components may be compromised. This information is passed to the failure response determination function, which decides whether to take an operational failure avoidance action, safe the system (goal change), retire the system (such as retire an aircraft before its wing fails), or wait until failure occurs and take a failure response.

Prognosis is a particularly important FM function for systems that have long operational lives and failure effects that have long times to criticality (days, weeks, years), in which the deterioration of components can be tracked and fed into the relevant physics-based models. Deep-space probes with multi-year missions monitor key components for deterioration, such as their power sources (degradation of solar panels, batteries, or radioisotope thermal generators). Fleets of aircraft (and, historically, the Space Shuttles) also have strong prognostic programs, often focused on deterioration of the highest-stressed structural components.

Failure prognosis as an operational FM function should not be confused with design-time analysis and prediction of behavior, the results of which are then built into the automated system. FM control loops must detect failure early enough to initiate and execute a successful failure response prior to the failure propagating to a severe consequence. The failure detection and diagnosis functions in effect must predict the future consequences of this failure so as to determine what response(s) to execute, and when. Thus, they have embedded in them a sense of prognostics, as the logic of the entire FM loop (failure detection, fault isolation, failure response) is based on a built-in predictive capability. If failure A is detected, then in the future, critical system function X will be compromised, and this means that failure response R must be executed now. Despite its predictive content, this example is not normally thought of as failure prognosis, mainly because the prediction is done at design time and not during operations.

1.5.4 Failure Response Determination

Failure response determination is the FM decision function to determine appropriate mitigation actions to current or predicted failure. Failure response determination contains several key sub-functions, which include functional assessment, identifying failure response options, determining the likely outcomes of the response options, prioritizing the response options, selecting which response(s) to initiate, and notifying the system to implement the response(s). Functional assessment determines the compromises to system functionality that are occurring now and will occur in the future given the current failures, how they propagate, and how they affect the system's ability to meet mission goals. Failure response determination can be implemented through automated mechanisms or human operators (ground or flight crew). The location of the failure response determination mechanism is intimately linked to the issue of "locus of control" for a system (i.e., who or what has decision authority to take actions, for nominal or off-nominal purposes).

1.5.5 Failure Response

Failure response is a composite function that covers four FM functions: goal change, failure recovery, failure masking, and operational failure avoidance. It generically describes actions taken to mitigate the effects of failure.

1.5.5.1 Goal Change

Goal change is the action that alters the system's current set of objectives. This can be executed by the system for a variety of reasons and is thus not exclusively an FM function, but FM is one of the primary initiators of this function. In the FM context, a goal change is initiated to attempt to regain the system's ability to control the system state (achieve some function) in reaction to a failure. Usually the goal changes to a "degraded goal" or a subset of the system's original goals. For example, with spacecraft safing, the current science objectives may be abandoned while the spacecraft maintains the goals of maintaining system power and communicating with Earth. In the case of a human-rated launch vehicle, an ascent abort abandons the goal of achieving orbit, but protects the goal of keeping the crew safe. For an aircraft a typical example would be rerouting the flight to an alternate airport or destination.

1.5.5.2 Failure Recovery

Failure recovery is the FM function of restoring system functions necessary to achieve existing or redefined system goals after a failure. Failure recovery occurs in two contexts: (1) when the system can temporarily sustain a compromise to function and the failure recovery action is activated without any goal change; and (2) after a goal change (typically safing) to return the system from the safe state back to normal operations. In some cases, normal operation may be identical to the operations occurring prior to the failure, with no change of objectives or functions. However, normal operation may require a new goal (one different from the original goal) for the system, which by comparison to the system's original goal before the failure could be less demanding. An example is failure recovery after a safing action, in which safing (a goal change) changed the system's objectives to something achievable, but often by abandoning some original goal, such as performing all of the science objectives in favor of performing only some of the science. In this case, the failure permanently compromises some of the mission. After the ground or flight crew (or the system autonomously) evaluate the situation, they determine which of the original system objectives can be attained, and command the system into a new configuration with new mission goals and plans.

Failure recovery has been a label typically applied to in-flight operational systems, but not always to maintenance or manufacturing/supportability actions. This is incorrect, as maintenance actions to repair or replace components after failures are failure recovery actions. An example is the failure of a launch vehicle component prior to launch, leading to a launch scrub and recycle. The failure recovery in this case may include repair and/or replacement of the failed component, reloading propellant tanks, and recycling the launch sequence to a point where it can be restarted.

1.5.5.3 Failure Masking

Failure masking differs from failure recovery in that failure masking is implemented when a system function cannot be compromised even temporarily. In failure masking a low-level failure propagates effects, which are halted before compromising the critical function. A classical example is a voting mechanism in a set of fault-tolerant computers. A triplex or quad set of computers perform identical operations, and the voting mechanism ensures that if any one of the computers fails, it is outvoted by the others, so that the output of the vote is always the correct set of information. The location at which the failure effects stop propagating is often called a "failure containment zone (or region) boundary."

1.5.5.4 Operational Failure Avoidance

Operational failure avoidance is an action to prevent a predicted future failure from occurring. Thus it is not a response to a current existing failure, but to a future predicted failure. It differs from failure masking in that failure masking prevents a failure from spreading beyond a certain location, whereas operational failure avoidance prevents the failure from happening to begin with. Whereas fault avoidance is a design-time passive implementation of design margins and quality assurance mechanisms to prevent faults (and hence failures), operational failure avoidance is an operational action to delay the onset of predicted failure or stop it altogether. An example is a component that has degraded in such a way that the regular mission profile, which normally would have been acceptable, creates high temperatures that can accelerate the degradation of the component so that it will fail in the near future. The system can be operated in a way that avoids these temperature ranges, so the mission operations team changes the mission profile so that this component is now kept in the shade, whereas it would normally have been in attitudes in which it was exposed to the Sun. Reliability-centered and condition-based maintenance are other typical examples of operational failure avoidance.

1.5.6 Fault and Failure Containment

Failure masking, fault tolerance, and fault and failure containment are closely linked concepts. To prevent loss of life, loss of the system (or vehicle), or loss of mission, both faults and failures must be contained. The concept of failure containment is easy to understand: failure effects, as they spread along failure effect propagation paths, must be stopped or contained to prevent system or mission failure. The location at which a particular set of failure effects are stopped is called a failure containment zone boundary. The set of failure containment zone boundaries creates a failure containment region, in which certain classes of failure effects are contained.

Fault containment is a related, but more complex, concept. Its name implies the difference from failure containment – fault containment is defined as preventing a fault (a cause of failure) from causing further faults (further causes of failure). An example best describes the nuances of the concept. Assume that an electrical short circuit occurs in Component A, and the system is designed in such a way that this leads to an overvoltage that propagates to a neighboring Component B, in which the overvoltage creates physical damage and another short circuit. Then assume that further overvoltages are contained so that further components do not experience this condition. Next, a fault diagnosis is performed, leading to the isolation of Component A as the component in which the fault originated. Technicians then replace Component A with a new Component A′. When the system is tested, it still does not function, because Component B has a permanent fault. Only when Component B is also replaced will the system function properly. This is an example of fault propagation, as opposed to merely failure propagation, and in this case "fault containment" did not exist between Components A and B, for that type of fault and resulting failure effects. One can therefore have "fault containment zones" that are different from "failure containment zones."

If the failure recovery function operates properly and successfully, then failure effects are generally contained, and, for this reason, failure containment is not considered a separate, independent FM function. It is encompassed in the overall process and functions of failure detection, fault isolation, and failure recovery. However, fault containment is a separate issue that must be addressed separately from the containment of failure effects. The prevention of the spread of permanent physical (or logical damage, with software) must be addressed with means different from those of containing failure effects.

Fault and failure containment boundaries operate only against certain types of faults and failures, but not others. Thus any boundary, when specified or identified, must be associated with that set of faults and failures that it is effective against. It is meaningless unless those classes are specified or identified as the items that the boundary mechanism addresses.

1.6 Mechanisms

1.6.1 Fault Tolerance

We define *fault tolerance* and *failure tolerance* as direct synonyms: *the ability to perform a function in the presence of any of a specified number of coincident, independent failure causes of specified types.*

Fault tolerance is unusual in that it is the ability to tolerate failures, but also is tied to the idea of a certain number of faults that cause the failures to be tolerated.

Fault tolerance is valid only against certain types or classes of faults and failures. For example, the triplex voting computer system handles regular "random part failures," but not design faults in the voting mechanism, or "Byzantine faults." Specification of fault tolerance, without identifying what is being tolerated, is not only incomplete, but potentially dangerous, as it can mislead designers and managers into believing it is effective against all faults and failures, which is incorrect. Fault tolerance at a low level (or closer to the failure cause location) enables failure masking at a higher level (further "downstream" from the causal location).

1.6.2 Redundancy

Redundancy is a fundamental aspect of FM designs, as all FM mechanisms rely on some form of redundancy. The importance of the concept is revealed when modeling FM design mechanisms in a fault tree or success tree. In a success tree model, for a function to succeed, all of its sub-functions must succeed, which are therefore represented as AND gates. A failure of any function means the AND function fails, and the system function above the gate fails. In a fault tree, the logic is reversed, with a failure of any of the functions underneath a higher-level function causing failure of the high-level function. These are modeled with OR gates. FM design mechanisms add new functions to the system, but they improve system reliability because they are effectively the "opposite gate" for these trees. In a success tree full of AND gates, the FM mechanism appears as an OR gate. In a fault tree, the tree full of OR gates shows the FM mechanism as an AND gate. FM design mechanisms are thus revealed in abstract modeling representations as system redundancies, and therefore the FM designer's concept of redundancy must be expanded to match.

In each case, the FM mechanism operates only against certain classes of faults and failures. The classic example is that in hardware identical redundancy, the FM mechanism mitigates against random part failure in any of the redundant strings, but cannot mitigate against a design flaw common among all of the redundant strings. Finally, the FM mechanism is not generally 100% effective even for those faults and failures it does mitigate, due to issues of false positive and false negative detections, fault isolation imperfections, and failure response limitations.

Redundancy is fundamental to FM design, verification and validation, and operations. When analyzing the system using logical techniques of fault and success trees, the nature of the redundancy must be identified, the limitations of that redundancy (what it does and does not apply to, and its effectiveness) assessed and in some cases calculated, and these limitations addressed in the FM design or in the justifications for why some risks are acceptable (or not). Whether formally analyzing the FM design in this way or not, the principle of redundancy *always* applies to FM, and the principle can be used to understand, assess, and justify the design and the risks of not having FM when those risks are acceptable.

1.6.2.1 Hardware Identical Redundancy

The most obvious kind of redundancy is hardware identical redundancy. A typical example is a triplex or quadruple redundant computing system, which mitigates random part failures or single event upsets in a single string of the computing system. Three or four exact copies of the relevant portion of the system are replicated, under the assumption that the bulk of the failures of that system are "random." By definition, it cannot mitigate against common mode faults, that is, faults whose effects impact all of the redundant strings simultaneously, such as design faults. Hardware identical redundancy can be, and is frequently, utilized both for failure detection and for failure responses. Thus, the voting mechanism in a triply redundant computing system is both a mechanism for detecting failures in one of the computers and a fault isolation mechanism in that it determines the location of the originating fault as somewhere in the faulty string and not others. Finally, it is a failure response mechanism in that it can physically deactivate the failed string or it can merely vote out the bad results, thus removing it from the active control loop on a cycle-by-cycle basis.

1.6.2.2 Functional (Dissimilar/Analytic) Redundancy

Functional redundancy is the use of dissimilar hardware, software, or operations to perform identical functions. Typically, different parts of the system are designed to, or by their physical nature, have known relationships. Thus, in electrical systems, there are known physical relationships between voltage, current, and resistance, while gas flows have specific relationships between pressure, volume, and temperature. It is possible to use these relationships and the known design of the system to develop physical and/or logical relationships between these variables, which provide means for non-identical measurement to provide the same information content as if there were identical measurement. Another example is a set of triply redundant computers, using different processors and different software designed by separate organizations, as occurs for commercial aircraft flight control. Dissimilar redundancy is typically utilized for failure detection, but it can also be potentially utilized for failure response, where, for example, a thruster might be used to replace the function of a failed reaction wheel on a spacecraft.

1.6.2.3 Information Redundancy

Information redundancy utilizes extra "information" to detect and potentially to respond to certain types of failures. The most common example is error detection and correction (EDAC) codes.[8] In EDAC, extra bits are added to a message, such that if a cosmic ray or some other phenomenon causes one or more bits to flip (a single event upset), then the receiving device can use the extra, redundant information to reconstruct the original message, in effect "unflipping" the bit(s) that had been changed. While this particular example addresses a failure caused by the environment, the principle can apply to failures with internal system causes (faults) as well. In this example, information redundancy is used for detection, isolation, and response.

1.6.2.4 Temporal Redundancy

Temporal redundancy refers to the practice of repeating a function should it fail upon a single execution. A typical example is the use of several measurement over time of the same state variable, because any single measurement could be corrupted by a single event upset. Another common example in computer processing is the checkpoint–rollback capability, when a series of computations have produced suspect results. In the checkpoint–rollback, the computer state is reverted (rolled back) to the computer state at a previous point in time that had been stored for potential future use (the checkpoint), and then restarted from the checkpoint to recompute the original set of calculations.

1.6.2.5 Knowledge Redundancy

The redundancy examples shown so far demonstrate that prior to actual system operations, detecting failures in general requires some source of knowledge independent of the part of the system under observation (in which the failure is to be detected) in order to determine that a problem exists. In

[8]Error in this case really means a low-level failure – unacceptable performance of intended function – but we shall follow the historical, though inaccurate, use of this phrase.

other words, one cannot depend on that portion of the system that is being monitored to detect its own failures, because that portion of the system might have a failure that prevents its own detection mechanism from operating. The examples shown so far use identical mechanisms, non-identical mechanisms related by physical laws, and extra information to detect and respond to failures. In the most general sense, these are examples of the application of separate sources of "knowledge" to identify failures in any given system or portion of a system.

The principle applies beyond these cases. For example, when a human cross-checks another human's calculations or command sequences, this is an example of the application of a separate source of knowledge from the originator of the products being checked, under the assumption that it is difficult for the originator to see his or her own mistakes. When this knowledge is encapsulated in an automated command sequence checker, or an artificial intelligence algorithm of some kind, this merely automates and encapsulates the separate knowledge source into a machine-based mechanism.

Even simulation, which is not normally considered from this perspective, can be thought of as an application of knowledge redundancy. When a system is tested using a simulation, what is happening is the application of a separate knowledge source that attempts to duplicate the environment in which the system will operate, and the way in which the system is to be operated within it. The "ultimate" and most accurate knowledge source that by definition has no flaws or simplifications is the system's operational environment. The point of cross-checking, analyzing, verifying, and validating the system is to find faults before the system is operated in this final, ultimate environment.

1.7 Summary of Principles

This section summarizes the core principles of SHM (or dependability engineering):

- SHM exists to preserve system functionality.
- SHM utilizes basic concepts of systems theory, including the system boundary, hierarchical decomposition, and recursion.
- Classification of states is based on models that define individual and group expectations of system behavior. Reclassification of a state is based on modification of the model(s) that define expectations of behavior.
- There are three categories of off-nominal states: anomalies, degradations, and failures. Anomalies refer to knowledge errors, and failures and degradations refer to control errors.
- Faults are causes of (explanations of) failure internal to the system. Fault and failure are recursive concepts.
- Humans build complex systems that they cannot fully understand. SHM must address failures due to unpredicted causes.
- The root causes of the vast majority of failures are human communicative and cognitive faults.
- SHM is deployed based on assessment of system function, and the risk to the system should that function fail.
- SHM strategies to preserve system function are: design-time fault avoidance, operational failure avoidance, failure masking, failure recovery, and goal change. The first is a passive design and quality assurance function, and the latter four are operational FM strategies.
- Operational FM is an extension of control theory, in that it provides control regimes to enable system function in the presence of failures that are, or are predicted to become, beyond the ability of the normal (passive or active) control system to successfully maintain function.
- Control theory ideas such as characteristic times (FM loop latency versus failure effect propagation times to the critical failure effect), knowledge and control errors, and state estimation versus control functions extend directly to FM.
- All operational FM implementations use some form of redundancy.

1.8 SHM Implementation

While many of the chapters of this book describe various aspects and implementations of SHM, this section will provide a brief overview of the major issues and strategies.

SHM is inherently an aspect of systems engineering. It is not a "subsystem," but rather a capability implemented across the entire system. Health management engineers (HMEs) assess risks across all system functions and design the capabilities to mitigate those risks. SHM is thus best organized as an aspect of systems engineering, both on projects and within matrix organizations, as an independent functional organization. A HME position or group (depending on project size) created and funded at the system level significantly aids dependability design. This engineer or team works alongside the chief engineer and the system engineer. The HME develops health management (HM) plans, performs system HM design and coordinates the subsystem HM designs and testing, and performs and coordinates HM-related analyses, including coordination with safety and mission assurance groups. The HME is then responsible for actively seeking trouble spots in the design, in particular interactive problems that cross subsystem boundaries. This engineer also orchestrates explicit HM design reviews that put teeth into the efforts to design dependability into the system. These reviews parallel the standard design reviews for the system and subsystems, but focus explicitly on preventing and mitigating failure across the entire system (Scandura, 2005).[9]

The first task of the HME is to assess risks to system functions (which in turn aim to achieve objectives), and to allocate SHM functions and design mechanisms to mitigate those risks. Functions are defined from the top down, usually through a systems engineering functional decomposition, typically represented as a success tree and/or an event sequence. Once defined in this way, the HME, along with reliability, safety, and subsystem analysts, defines the consequences should these functions fail, and provides preliminary estimates of required and likely reliability and availability (R&A) based on an implementation concept assuming a single string design without any operational FM capabilities. Where the likely single string R&A falls short of what is required, then SHM capabilities must be provided, either to beef up reliability for the function through design margins (fault avoidance), or to add redundancy managed by operational FM mechanisms. These FM mechanisms are then allocated to appropriate engineering groups to implement. Some FM mechanisms must be put in place to account for currently unpredicted failures, primarily by deploying mechanisms to protect functions even when the R&A assessments based on current knowledge predict a low probability of failure.

Analysis of the system's R&A, and of the effectiveness of the FM mechanisms as part of the system assessment, requires top-down, bottom-up, and middle-out assessments. Top-down analyses to determine threats to function include probabilistic risk analyses, reliability analyses, and hazard analyses, represented in fault trees and event sequences. Bottom-up analyses are typified by FMEAs, which determine the failure modes of each system component and a rough determination of the failure effects of each failure mode. Middle-out analyses use information from both top-down functional assessments and bottom-up FMEA data to feed a directed graph representation of the system design. This capability allows connection of the bottom-up failure modes to the top-down functions, and is the core model for model-based diagnostics and state estimation. Assessments of state estimation functions such as anomaly and failure detection, fault diagnosis, and failure prognosis typically are assessed via FP/FN and coverage metrics, with diagnosis functions having the added complexity of ambiguity groups. Control functions such as failure response determination and failure responses must be measured for the probability of successful mitigation, accounting in particular for the race condition of TTC versus FM loop latencies for each mitigation. These assessments are structured through failure scenarios, which are the unique system behaviors under failure conditions. Failure scenarios are constructed primarily from the bottom up based on the particular ways that the system

[9]This particular name has never been used in practice. However, its function has existed in a variety of projects, such as JPL's "Fault Protection Engineer," and also sometimes in the "Chief Engineer" or "Systems Engineer" positions.

can fail and how it reacts to failure (including the FM mitigations), but which must connect to the top-down information as to which functions are important and the consequences of the failures.

Verification and validation also use bottom-up and top-down information. Because the system does not normally fail, faults must be injected into the system to test SHM capabilities, and these faults ultimately reside in the FMEA. System verification to assess whether the system is meeting requirements uses a mapping from the full set of failure modes to a smaller set of tests designed to verify those requirements. System validation by contrast often uses top-down information to assess "use cases," which for SHM are tied to the failure scenarios of unique system failure behaviors. For SHM capabilities to mitigate against "unknown unknowns," engineers often "artificially" compromise functionality by altering thresholds or introducing multiple failure modes to stimulate the functional failure. These tests tie to the analyses described above, because it is impossible to test all possible failure behaviors in all conditions, and there must be an analytical mapping from the tests to the analyses for completeness. The testing is often overseen by quality assurance personnel. Manufacturing is another primary location of quality assurance activities to minimize the number of faults introduced into the system by humans building the system.

Finally, operations is a major locale for SHM implementation, particularly for prognosis and operational failure avoidance functions, including repair and maintenance. If a diagnostic or "truth" model has been built during design, then it is usually cost effective to port this model to be the basis of a model-based, operational diagnostic system. This system can also be used for operational personnel training and education.

1.9 Some Implications

There are many implications of the theory of SHM described above. We shall highlight a few of them.

1.9.1 Detecting Unpredicted Off-nominal States

As described above, SHM must address unpredicted states. It turns out that detecting unpredicted states is generally possible, but responding appropriately to them remains problematic. While it is impossible to know all possible causes of failure, it is possible to know which failures matter, because for a failure to matter, it must compromise a function. Since in theory we can determine all functions needed for a system to achieve its goal, it is possible to construct mechanisms that detect deviations from our intent for these functions. Also, we can develop algorithms to detect unexpected events regardless of causes (anomaly detections) based on past system behavior. Thus in theory, failure of critical functions can be detected with nearly 100% certainty.

This good news is partially compromised by the existence of latent faults. Latent faults are faults embedded in the system, but do not show any symptoms until some later event creates the conditions in which the failure effects manifest themselves. The classic example is when a switch has failed such that it will stick in the current position that it currently inhabits. Only when someone tries to flip the switch will its inability to change state become apparent.

The second problematic point is that the symptoms and the consequences of the fault may be such that by the time an off-nominal behavior becomes visible, the system has already failed, or the time between detection of this behavior to the critical failure effect is so fast that nothing can be done about it. So even though it is nearly certain that any fault we care about will create off-nominal behaviors that we can detect, this is no particular cause for celebration if the system is doomed by the time we see it.

1.9.2 Impossibility of Complete Knowledge Independence

As stated above, "knowledge redundancy," having a different source of knowledge about the system than the one built into the system, is one of the primary means to check for mistakes or off-nominal behaviors. Along with knowledge redundancy, another typical implementation of cross-checking is to have an "independent" review, with the idea that complete independence is best. However, complete independence of knowledge is futile for the purpose of cross-checking or review. Someone that has sufficiently different background to have "complete" independence of knowledge will by definition know little or nothing about the thing they are asked to verify or cross-check. The problem with someone from the same organization as the one building and operating a device is that they have many or all of the same assumptions, background, and training. Someone with complete independence will have few or none of the assumptions, background, and training of the organization they are trying to verify. Without any common background or knowledge, they will be useless in verifying the operation or device. Some commonalities must be eliminated, but others must remain to allow for any kind of verification. This conundrum cannot be evaded. Since it is futile to attain complete knowledge independence, we must have different people having different backgrounds, each of which has some commonality with the item and organization in question, but collectively having many differences. Systems engineering was developed to deal with the issues of technical complexity and failure, using social means with semi-independent knowledge sources (Johnson, 1997).

Another way around the conundrum is to correctly replicate the operational environment, and then operate the system in that environment. This avoids needing overlaps in knowledge of the system design, in favor of detailed knowledge of the operating environment, which is often better known than the system itself.

1.9.3 The Need for, and Danger of, Bureaucracy

Bureaucracy is needed to consistently repeat dependability processes, but humans tend to lose cognitive focus during repetitive actions and to suppress the reasoning behind bureaucratic rules, creating conditions for human faults. Put another way, humans are at their best in situations that are neither wholly chaotic nor wholly repetitive. The nature of large complex aerospace systems is such that they require millions of minor actions and communications, a fault in any of which can lead to system failure. Humans cannot maintain strong focus in situations of long-term repetitive action, whether it is assembly-line wrench-turning or the launch of 50 consecutive Space Shuttle flights. One solution to this problem is to automate repetitive functions using machines. Unfortunately, this is not always possible. Humans must have some mind-stimulating activities to maintain proper awareness. One solution is proper education and training to keep operators alert to possible dangers. A variety of methods are used already, and even more are necessary. Training through use of inserted faults (which are infrequent and vary in effects) in simulations is an excellent and typical method for operations. Another necessary method is to train designers, manufacturers, and operators in the fundamental theories and principles regarding the origins and nature of faults and failures, and how to deal with them.

1.9.4 "Clean" Interfaces

A number of typical practices and guidelines are geared to reduce complexity, although the reasons for their effectiveness are typically left unexplained. One example is the use of clean interfaces, which is defined as the practice of simplifying the connections between components. However, the reason that simplification of interfaces is an effective practice is not usually explained. The reasons are ultimately related to human cognitive and social abilities. First, the fewer the number of physical and logical (software) connections and interactions, the more likely it is for humans to understand the entirety of connections and interactions and their implications for other parts of the system. Second, a physical interface is usually also a social interface between two or more organizations and people. Simple interfaces also mean simpler communication between people and organizations and their individual cultures and idiosyncrasies. Miscommunication becomes less likely, reducing the chances of failures due to miscommunication.

1.9.5 Requirements, Models, and Islands of Rigor

A common belief among engineers is that requirements or specifications at least in theory can or should provide a complete definition of what the system should do. This is an often implicit premise in the systems engineering process, and is made explicit in contractual arrangements, in which requirements govern what a contractor will provide to a customer. Unfortunately, such precision is not attainable, certainly not in common-English language of requirements, which are notoriously difficult to write in an unambiguous way. Many experts have therefore called for more rigor and formality. We endorse the need for more rigor and formality, but perhaps for different reasons, and in different ways than often envisioned.

Given that the vast majority of system failures are due to social communicative and individual cognitive faults (which are generally discovered through communication with others), finding ways to improve communications is crucial, particularly for systems engineering, which is the set of methods to integrate diverse subsystems into a functioning whole. One might ask, given these root causes of failure, why do systems operate with much higher reliability and availability than the basic failure rate of humans (which is in the range of 1 to 10% for well-trained humans on complex tasks)? It is most likely because formal models, usually mathematical or logical, and most often computer based, dramatically reduce the number of faults in the system. This is because mathematical and logical models cannot tolerate ambiguity, and require specific, complete inputs to operate properly. One does not typically load English-language statements into a computer, as English is far too ambiguous. The reductions in communicative ambiguities engendered by formal models are key to the reduction of social communicative fault rates, and hence to the number of faults that are "built in" to the system. In general, what is needed is to move from informal models and results communicated in natural language to formal models in systems engineering, and for the off-nominal aspects of systems engineering: dependability engineering or SHM.

What is needed to improve system dependability and hence for SHM is the development of formal modeling methods to support design, analysis, verification and validation, manufacturing, and operations. These methods need theoretical depth equivalent to control systems engineering or structural engineering, built into tools that can hide the theoretical complexity yet enforce proper deployment of the concepts. This formality will dramatically reduce the number of faults in the system by uncovering improper system behaviors under failure conditions well before operations, and by reducing the number of faults introduced during system operation. Reducing the number of faults will reduce the number of failures, improving the safety and reliability of these systems.

1.10 Conclusion

SHM is a comprehensive umbrella for a variety of disparate methods that have developed over decades to analyze, prevent, and mitigate failures. We have outlined here the basic concepts, terms, theory, and principles that form the foundation of SHM practices and technologies, so as to aid in the implementation of SHM in new and existing systems, and so that researchers will focus their efforts in the right directions in providing tools, techniques, and technologies that will make the systems we create more dependable.

Bibliography

Albert, J., Alyea, D., Cooper, L. *et al*. (1995) Vehicle health management (VHM) architecture process development. Proceedings of SAE Aerospace Atlantic Conference, Dayton, Ohio, May.

Avizienis, A., Laprie, J.-C., and Randell, B. (2000) Fundamental concepts of dependability. Third Information Survivability Workshop (ISW-2000), Boston, Massachusetts, October 24–26.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004) Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, **1** (1), 11–33.

Bijker, W.E., Hughes, T.P., and Pinch, T. (eds.) (1987) *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology*, MIT Press, Cambridge, MA.

Campbell, G., Johnson, S., Obleski, M., and Puening, R. (1992) *Final Report – SHM Design Methodology, Rocket Engine Condition Monitoring System (RECMS).* Prepared for Pratt & Whitney by Martin Marietta Space Launch Systems Company. Purchase Order F435025, MCR-92-5014, October 30.

Columbia Accident Investigation Board (2003) *Report*, Volume 1, National Aeronautics and Space Administration, Washington, DC, August.

INCOSE (International Council on Systems Engineering) (2010) A Consensus of the INCOSE Fellows: http://www.incose.org/practice/fellowsconsensus.aspx (accessed October 24, 2010).

Johnson, S.B. (1997) Three approaches to big technology: operations research, systems engineering, and project management. *Technology and Culture*, **38** (4), 891–919.

Johnson, S.B. (2002) *The Secret of Apollo: Systems Management in American and European Space Programs*, The Johns Hopkins University Press, Baltimore, MD.

Johnson, S.B. (2003) Systems integration and the social solution of technical problems in complex systems, in *The Business of Systems Integration* (eds. A. Prencipe, A. Davies, and M. Hobday), Oxford University Press, Oxford, pp. 35–55.

Johnson, S.B. (2010) From the secret of Apollo to the lessons of failure: the uses and abuses of systems engineering and project management at NASA, in *NASA's First 50 Years: A Historical Perspective* (ed. S.J. Dick), NASA SP-2010-4704, NASA, Washington, DC, Chapter 12.

Johnson, S.B. and Day, J.C. (2010) Conceptual framework for a fault management design methodology. AIAA Infotech Conference, Atlanta, Georgia, April, AIAA paper 227006.

Kurtoglu, T., Johnson, S.B., Barszcz, E. *et al*. (2008) Integrating system health management into early design of aerospace systems using functional fault analysis. International Conference on Prognostics and Health Management, Denver, Colorado, October.

Laprie, J.C., Avizienis, A., and Randell, B. (2004) Dependability and its threats: a taxonomy, building the information society. Proceedings of the 18th IFIP World Computer Congress, Toulouse, France, August.

Leveson, N.G. (2009) Engineering a safer world: system safety for the 21st century (or systems thinking applied to safety). Unpublished manuscript: http://sunnyday.mit.edu/book2.pdf (accessed January 27, 2011).

Perrow, C. (1984) *Normal Accidents*, Basic Books, New York.

Rasmussen, R.D. (2008) GN&C fault protection fundamentals. 31st Annual American Astronautical Society Guidance, Navigation, and Control Conference, AAS 08-031, Breckenridge, Colorado, February 1–6.

Scandura, P. (2005) Integrated vehicle health management as a systems engineering discipline. IEE 24th Avionics Systems Conference, Washington, DC, October 30.

Van Fraasen, B.C. (1980) *The Scientific Image*, Oxford University Press, Oxford.

Vaughan, D. (1996) *The Challenger Launch Decision: Risky Technology, Culture, and Deviance at NASA*, Chicago University Press, Chicago.