Introduction to System Health Engineering and Management in Aerospace

Stephen B. Johnson
NASA Marshall Space Flight Center
Advanced Sensors and Health Management Systems Branch, EV23

ABSTRACT

This paper provides a technical overview of Integrated System Health Engineering and Management (ISHEM). We define ISHEM as "the processes, techniques, and technologies used to design, analyze, build, verify, and operate a system to prevent faults and/or minimize their effects." This includes design and manufacturing techniques as well operational and managerial methods. ISHEM is not a "purely technical issue" as it also involves and must account for organizational, communicative, and cognitive features of humans as social beings and as individuals. Thus the paper will discuss in more detail why all of these elements, from the technical to the cognitive and social, are necessary to build dependable human-machine systems. The paper outlines a functional framework and architecture for ISHEM operations, describes the processes needed to implement ISHEM in the system life-cycle, 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 a set of design and operational principles for ISHEM.

Introduction and Definition

Integrated System Health Engineering and Management (ISHEM) is defined as *the processes, techniques, and technologies used to design, analyze, build, verify, and operate a system to prevent faults and/or mitigate their effects*. It is both something old and something new. It is old in that it consists of a variety of methods, techniques, and ideas that have been used in theory and practice for decades, all related to the analysis of failure and the maintenance of the health of complex human-machine systems. It is new in that the recognition of the relationships between these various methods, techniques and ideas is much more recent and is rapidly evolving in the early 21st century.

The recognition that these different techniques and technologies must be brought together has been growing over time. This can be seen in a variety of ways:

- the creation of reliability theory, environmental and system testing and quality methods in the 1950s and 1960s
- the total quality management fad of the 1980s and early 1990s
- the development of redundancy management and fault tolerance methods from the 1960s to the present
- the formulation of Byzantine computer theory in the 1970s and 1980s
- the development of new standards such as integrated diagnostics and maintainability in the 1990s
- the emergence of vehicle and system health management as technology areas in both air and space applications in the 1990s and early 2000s
- the recognition of "culture problems" in NASA and the Department of Defense as crucial factors leading to system failure in the 2000s.

We argue that these disparate but related ideas are best considered from a broader perspective, which we call Integrated System Health Engineering and Management (ISHEM). The term ISHEM evolved in the late 1980s and early 1990s from the phrase "Vehicle Health Monitoring (VHM)," which within the NASA research community referred to proper selection and use of sensors and software to monitor the health of space vehicles. Within year or two of its original use, space engineers found the phrase Vehicle Health Monitoring deficient in two ways. First, merely monitoring was insufficient; the real issue was rather what actions to take based on the parameters so monitored. The word "management" soon substituted for "monitoring" to refer to this more active idea. Second, given that vehicles are merely one aspect of the complex human-machine systems that aerospace engineers design and operate, 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.

The Department of Defense during this same period had created a set of processes dealing with similar topics, but under the title "Integrated Diagnostics." The DoD's term referred to the operational maintenance issues (usually in an aircraft environment) that the DoD faced in trying to detect faults, determine their location, and replace them. Given that fault symptoms frequently manifested themselves in components that were not the source of the original fault, it required "integrated" diagnostics looking at many aspects of the vehicle in question to determine the actual source of the fault, and hence what component should be replaced. This word soon found its way into the NASA terminology, becoming "Integrated System Health Management" (ISHM). Motivation to use "integrated" in the NASA terminology almost certainly related to the issue of separating "system-level" issues from the various subsystems and disciplines that dealt with failures within their own areas. Highlighting the system aspects helped to define ISHM as a new system issue, instead of an old subsystem concern.

Finally, in 2005, the program committee for organizing the Forum on Integrated System Health Engineering and Management in the Fall of 2005 decided to add the word "Engineering" to the title. The motivation to add yet another word to the term was to distinguish between the technical and social aspects of the problem of preventing and mitigating failures. The major difference between the discussions of the 1990s and in the early 21st century is the growing recognition of the criticality of social and cognitive issues in dealing with failures. The word "engineering" in ISHEM now refers to the classical technical aspects of the problem. This now distinguishes technical aspects from the organizational and social issues, which the word "management" clearly implies by common usage. It is important for old-time VHM personnel to realize that in the new definition, the implication of "activity" versus "passivity" in the term management is still correct, but it now also has the added nuance of the social and cognitive aspects of system health.

A synonym for ISHEM is "Dependable System Design and Operations." Both phrases (ISHEM and Dependable System Design and Operations) signify that the new discipline deals with ensuring the "health" of a technological system, or alternatively, preventing its degradation and failure. This includes design and manufacturing techniques as well operational and managerial methods. ISHEM is not a "purely technical issue" as it also involves and must account for organizational, communicative, and cognitive features of humans as social beings and as individuals.

For simplicity, the subject matter of ISHEM, or of Dependable System Design and Operations, is "dependability." This word connotes more than other "ilities" such as reliability (quantitative estimation of successful operation or failure), maintainability (how to maintain the performance of a system in operations), "diagnosibility" (ability to determine the source of a fault), "testability" (the ability to properly test a system or its components), "quality" (a multiply-defined term if ever there was one) and other similar 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 perform according to human expectations. A system that meets human expectations is dependable, and is ISHEM's goal, achieved by focus on its opposite, failure.

We argue that ISHEM should be treated and organized as a coherent discipline. Organizing ISHEM as a discipline provides an institutional means to organize knowledge about dependable system design and operations, and it heightens awareness of the various techniques to create and operate such systems. The resulting specialization of knowledge will allow for 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 heretofore. We feel this step is necessary, since the disciplines and processes that currently exist, such as reliability theory, systems engineering, management theory and others, have been found wanting as the sophistication and complexity of systems continues to increase. As the depth of ISHEM knowledge increases, the resulting ideas must be fed back into other disciplines and processes both in academic and institutional contexts. When ISHEM is taught as an academic discipline in its own right, and when ISHEM is integrated into engineering and management theories and processes, we will begin to see significant improvement in the dependability of human-machine systems.

The new discipline includes classical engineering issues such as advanced sensors, redundancy management, artificial intelligence for diagnostics, probabilistic reliability theory and formal validation methods. It also includes "quasi-technical" techniques and disciplines such as quality assurance, systems architecture and engineering, knowledge capture, testability and maintainability, and human factors. Finally, it includes social and cognitive issues of institutional design and processes, education and training for operations, and economics of systems integration. All of these disciplines and methods are important factors in designing and operating dependable, "healthy" systems of humans and machines.

Complexity, Human Abilities and the Nature of Faults

A driving force in the recognition of ISHEM as a discipline is the growth of complexity in the modern world. This complexity, in turn, leads to unexpected behaviors and consequences, many of which

result from or result in system failure, loss of human life, destruction of property, pollution of the natural world, huge expenses to repair or repay damages, and so on. One definition of "complex" in *Webster's Dictionary* is "hard to separate, analyze, or solve." (Webster's 1991) We extrapolate from this definition, defining 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.

Complex technologies can be beyond the grasp of any single person when one of the following four conditions applies. First, the technology could be *heterogeneous*, meaning that several disparate kinds of devices are involved, such as power, propulsion, attitude control, computing, etc. Second, technologies can be *deep*, meaning that each type requires many years of study to master. This is true of almost all aerospace technologies. Third, even if the technologies are of a single type and are relatively simple individually, there may be so many of them that the *scale* of the resulting system is too large for any one person to understand. Fourth, the *interactivity* of the system within its internal components, or with its environment can also be complex, in particular as they become more autonomous. Most of these factors exist in aerospace systems, and some systems display all of these issues. The same issues often hold true for other systems with fewer and simpler technologies but more people performing diverse functions. (Johnson 2002b, Chapter 1)

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 must be subdivided into small chunks, each of which must be "simple" enough for one person to comprehend. "Simple" in this case is the opposite of complex: "that which a single person can completely understand." Of course, "completely" is a relative term, depending on the potential uses of that portion of the system that the single individual masters, that person's cognitive abilities, and the nature of that system element. Thus a fundamental limitation on any system design is the proper division of the system into cognitively comprehensible pieces. Since understanding of each portion of a system is the first step to understanding how the system will behave when each portion is connected to other portions, social division of the system into smaller elements, each of which is comprehensible with individual human cognitive capabilities, is fundamental to dependable design and operation.

Since each person can master only a small part of the whole, by definition each *must* communicate with others in order to develop, build, and operate the system. The implication is clear. For any complex system to function properly, those designing, manufacturing, and operating the system must communicate well with each other. Engineers have realized this to some extent; this is the reason for the existence of systems engineering as a discipline. However, systems engineering is not conceptualized in terms of communication or social skills, focusing instead on culturally congenial technical and process issues. This is a fundamental flaw in the conception of systems engineering, and for engineering in general. (Johnson 2003)

Analysis of failures confirms this line of reasoning. While comprehensive statistical or other studies on this matter do not yet exist, this author's experience and analysis of various aerospace failures suggest that the vast majority (perhaps 90% 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 same areas that lead to internal flaws in design, manufacturing, or operations.

Typically, people recognize faults as due to failures of hardware, software, or operational components or processes. While the "immediate cause" is seen in the technology, in fact the ultimate root cause of these component and process failures result from either individual performance or social communicative failures. The *Columbia Accident Investigation Board Report* of 2003 makes this clear with the Columbia tragedy, but it is typical for many other failures as well. This is easier to understand when we realize that technologies are merely the final products of human knowledge applied to creating useful artifacts. How could it be otherwise? *An artifact merely embodies and incarnates knowledge from its creators, and hence if it has a fault, this is ultimately due to a flaw in the knowledge of its creators or in a mismatch*

between the knowledge of its creators and that of its users. We see in this statement that faults result from both individual and social causes. The most obvious example of this fact is the requirements capture process, which is one of the most common places where design faults come into existence. Another is where the operators use the system in a way not envisioned by the designers. Thus the use of the ARPANET for email instead of data communications and simulation, and the Internet for e-commerce are both uses of network technology that took the designers by surprise. While this case is benign from a failure standpoint, others are not, such as launching the Space Shuttle *Challenger* in temperatures below its tested design limits.

Software and operations failures are obviously due to people that build the software or operate the spacecraft. Hardware failures do not always appear to follow this logic, but in fact they usually do. Many hardware failures are due to improper operation (operating outside the tested environment, as in the *Challenger* scenario) or to weaknesses in the manufacturing processes for the component, which trace back to design flaws or simple operational mistakes, which in turn stem from individual performance or social communicative failures.

Individual performance failures result from the fact that individuals make mistakes. These can be as simple as a transposition of numbers, an error in a computer algorithm, a misinterpretation of data, or poor solder joints in an electronic assembly process (a solderer's mind wanders, leading to a poor solder). Other faults are due to communication failures. These have two causes. The first is miscommunication, when one person attempts to transmit information, but that information is not received properly by another. The attempts to communicate the urgency of the foam impact in the *Columbia* accident are a good example of this type. The second is when there is no communication. In this situation, the information needed by one person exists with another person, but the communication of that information never occurs. In the *Challenger* accident, the data needed to determine the real dangers of low temperatures on O-rings existed, but the communication of that information among the relevant experts never took place on the night of the decision to launch, in part because some of them were absent and in part due to asymmetries in social power (Thiokol engineers would not challenge Thiokol managers that controlled their paychecks, and Thiokol managers would not challenge NASA managers that controlled Thiokol's funding.).

Faults may lead to identifiable symptoms during flight or use (some faults cause no identifiable errors, and the system does not fail). That is, misunderstandings and miscommunications in the design or preparation prior to flight may become manifest when the system is finally used. Any faults in our knowledge are embedded into the system, waiting to appear as errors and failures under the proper circumstances. Once the error or failure occurs, then it is relatively easy to trace back the underlying cause. The problem, however, is to find the problem *before* failure occurs; that is, before the flight or use. Ultimately, this means discovering the underlying individual performance and social communicative faults before they are embedded in the technology, or barring that, discovering them in the technology before operational use, and then ensuring the fault in the system is removed or avoided.

In summary, complexity forces the division of systems into many small parts. This forces the division of the system into small, individually-comprehensible pieces, which in turn requires all of the individuals working on small pieces to communicate with others. Individuals make mistakes, and the social collective of individuals have communication problems, both of which result in faulty knowledge becoming embedded in the system, creating faults, some of which become failures. The challenge is to prevent or find these faults before they create failures.

Failure, Faults, and Anomalies

ISHEM purports to be the discipline that studies prevention and mitigation of failures, and then guides the creation of methods, technologies, and techniques that in fact prevent and mitigate failures. These methods, technologies, and techniques might or might not be classical "technologies." For example, a specialized sensor to monitor fluid flow, connected to a redline algorithm to determine if an error exists,

which in turn is connected to artificial-intelligence-based diagnostic software would be a classical set of health management technologies. However, an effective way to reduce the number of operator failures is education, simulation, and training methods for mission operators or aircraft maintenance crew. Both of these examples portray health management functions necessary for proper system functioning, even though one is a classical technology, and the other is a set of social processes. Any theory of system health engineering and management must be able to encompass the technological, social, and psychological aspects of dealing with failures.

Since failure is the subject matter of the discipline, we must define it for the discipline to have a proper object of study. *Failure is defined as the loss of intended function or the performance of an unintended function.* In this definition, intent is defined by anyone that "uses" or "interacts" with the system, whether as designer, manufacturer, operator, or user. It is crucial to recognize that failure is socially defined. What one user may perceive as normal, another might consider a failure. Thus some failures result from a mismatch between the designer's intent and the operator's actions. The system does precisely what the designer intended, but its behavior is not what the operator wanted. (Campbell et al., 1992, p. 3)

Users can always determine if a failure occurs, because failures create some identifiable behavior related to the loss of some desirable functionality of the system. This undesirable behavior is called a *anomaly* or *error* or *fault symptom*, all of which are synonyms that we define as *a detectable undesired state. The root cause of an anomaly is called a "fault."* Faults might or might not lead to errors or to failures.

Like failures, anomalies and faults are in the eye of the beholder. Someone must decide that a state or behavior is "undesired." In many cases, such as the breakup of Space Shuttle *Columbia* upon re-entry in February 2003, everyone agrees that the behavior of the system was undesired, but there are many situations where minor anomalies occur and there is disagreement as to whether it constitutes an anomaly, or whether it is merely typical and acceptable system behavior. In NASA, these are often referred to as "out-of-family" events. In the *Columbia* and *Challenger* cases, some engineers and managers considered insulation foam falling off the external tank, or O-ring erosion to be anomalies, but after a time were re-classified as normal system behavior, that is "in-family." These warning signs were masked by numerous other problems that seemed more urgent, and then the lack of disastrous consequences led to re-classification of the anomalies to be normal behavior. This is sociologist Diane Vaughan's so-called "normalization of deviance." (Vaughan 1996, Chapters 4 and 5). Recognizing that this is a social process is crucial. Anomalies are not "out there" recognizable to all; they are defined as normal or abnormal by various individuals and groups with often-differing criteria and values. Over the last few decades, research on the nature of technology in the social science community has made this clear, most obviously in the theory of the "Social Construction of Technology." (Bijker et al., 1987)

While these social and individual factors give the impression that there can be an infinite number of possible interpretations of normal and anomalous, in practice the most common interpretations are relatively few. The criteria for discriminating between errors and failures on one hand, and normal behavior on the other, are based on the expected functions of the system in question. Those specifying the requirements for a future system define a set of functions that the system is to perform. In turn, the designers and manufacturers then create a system capable of performing those functions, while the operators use the system to actually perform the function. Over time, the designers or operators may find or create other functions that the system can perform. Failure is defined with respect to those functions, whether old or new, that the system performs. Thus a theory of ISHEM pertains to the success or failure of a system to perform its proper functions.

Mitigation: ISHEM Functional and Operational Architecture

Mitigation forms the operational core of ISHEM, and as such is its most visible aspect. It requires sensors to detect anomalies, algorithms or experts to isolate the fault and diagnose the root cause, and a

variety of operational changes to the system's configuration to respond to the fault. The discrimination of normal versus anomalous behavior must occur on a regular basis in order to ensure proper system operation. Should anomalies occur, the detection and response to those anomalies are dynamic processes that modify internal or external system structures and behaviors so as to minimize the loss of system functionality within other schedule and cost constraints.

ISHEM theory begins with a framework of functions necessary to monitor and manage the health of a dynamic system. These in turn form the "mitigation" aspects of ISHEM, along with active elements of failure prevention (predicting failure based on sensed degradation, and acting to prevent the failure). Figure 1 shows the characteristic looping structure of operational health management functions, which is a reflection of the time-dependent repetitive feedback processes typical of dynamic systems. (Albert et al. 1995)

Any fault or its resulting errors must first be prevented from corrupting or destroying the rest of the system. If this is not done, then the spread of the fault and/or its effects will cause the system to fail. It can also corrupt the mechanisms needed to monitor the system and respond to any problems. Once the fault and its errors are contained, the system must provide data about the anomalous behaviors, and must then determine whether that behavior is normal or anomalous. If it anomalous, then the system can either mask the problem (if there is sufficient redundancy) and continue, or it can take active measures to determine the location of (isolate) the faulty components. Determining the root cause (diagnosis) might or might not be necessary in the short term, but in the long term it is frequently necessary in order to optimize or adjust the system for future functions. Once the possible fault locations are identified, the system can re-route around them, and then recovery procedures can begin. When the system is once again functioning, operators can then take measures to prevent failures from occurring and to optimize system performance. In addition, prognosis methods can predict failures before they occur, allowing operators to replace, repair, or re-route around components before they fail.

This ISHEM functional flow chart provides a basis for understanding the primary characteristics and functions of health management systems. Classical health management technologies and processes for performance monitoring, error detection, isolation, and response, diagnostics, prognostics, and maintainability are all represented and shown to be subsets of the larger flow of ISHEM operations. These functions can be performed by people or technologies or some mixture of the two, making the framework general enough to handle either their social or technological aspects.

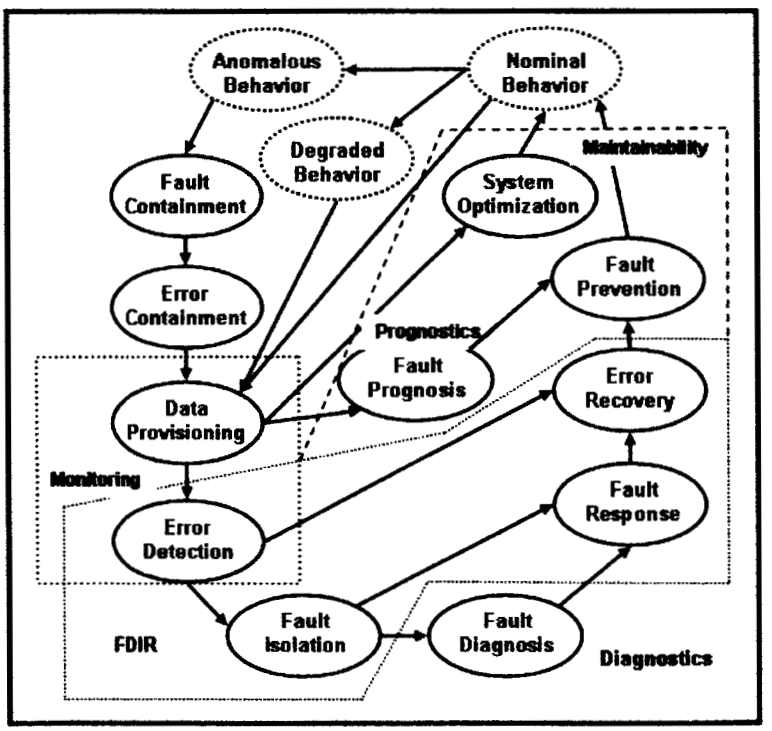


Figure 1: ISHEM Functional Flow

The characteristic looping structure of these functions also has architectural implications, as shown in Figure 2. The looping represents the flow of time required to monitor, detect, isolate, diagnose, and respond to problems within a system. Any health management architecture must take into account the time required to perform these functions, leading to a series of concentric architectural loops, each of which corresponds to a characteristic time available to

perform these functions using combinations of technologies and humans. The fastest loops, generally local to components and subsystems, deal with faults that propagate so fast that computers are unable react quickly enough. The on-board software then deals with faults whose effects propagate more slowly, but typically faster than what humans can handle. Crewed vehicles have the option of on-board human response, which is the third level of response. For situations that can take hours or days to repair, human ground operators can be involved in the fourth level of response. Some of these responses involve changes to the flight system, which in turn affects the test and maintenance equipment. Finally, for expendable launchers or other components that have assembly lines operating to supply many vehicles, flight information is used to modify the manufacturing and test equipment to make the manufacturing processes more reliable for the next generation of vehicles and technologies, and to re-design system elements to remove discovered failure modes. Total Quality Management, for example focuses largely on the improvements to designs and their manufacturing implementation through assembly lines based on operational experience.

The system health management operational architecture shown above is typical for aerospace systems, and in fact, if one deletes the word "vehicle," it is typical for many other kinds of systems as well. (Albert et al., 1995) The components of this architecture are arranged in the looping fashion characteristic of ISHEM functions, as described in the functional flow chart just described. ISHEM functions are then mapped into these architectural elements. There are three primary factors that determine this mapping: time, criticality, and cost.

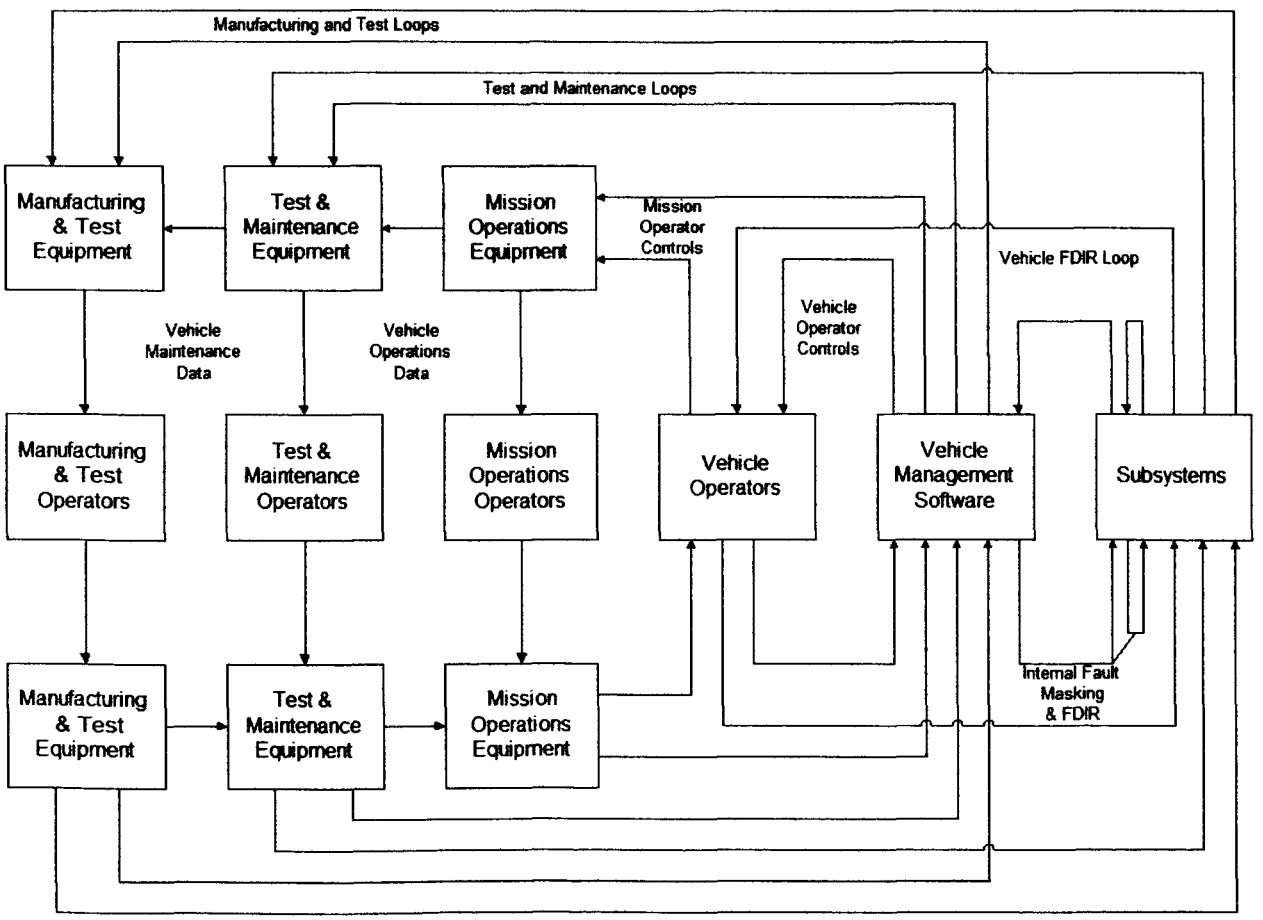


Figure 2: System Health Management Operational Architecture

Time is crucial, because if the fault detection, isolation, and response (FDIR), along with subsequent re-planning, do not occur quickly enough, then a fault may lead to system failure. The actual time required for each loop depends both on the nature of the fault, but equally important, how quickly the fault or its symptoms spread beyond the point of origination. FDIR loops must be significantly faster than the characteristic time for that fault's propagation. The following table shows the order-of-magnitude propagation times based on the physical or logical propagation mechanisms. These also apply to the times required for each element of an FDIR loop, and by summing them, for the overall FDIR time available. Architecturally, designers must create FDIR mechanisms faster than the characteristic propagation times of the faults in question. In many cases, this implies creation of fault and error containment zones that ensure a fault or its symptoms cannot propagate past a pre-determined point.

Along with the propagation time, the *criticality* of a fault dramatically affects a designer's architectural choices. For a fault to even be noticed, it must create an error or symptom that is detectable. If fault symptoms are never detectable, then it follows logically that the fault is unimportant. This is because if a fault *is* important, it *must* manifest itself in a symptom related to a function of the system in question. Put another way, if a fault never compromises a system function, then its existence is either completely invisible to users, or is visible but irrelevant, since it will never lead to system failure. This situation implicitly provides evidence that the designers created irrelevant functions in the system, because if a device on the system fails, but its failure is irrelevant, what relevant function did it perform to begin with? A properly designed, near-optimal system will have only components that contribute to system functions, and thus component failures *must* degrade system functions in some manner.

Function	**Physical Mechanism**	**Characteristic Time**
Electrical Power	Electron transport	1-10 milliseconds
Attitude Control	Thruster impulse or reaction wheel acceleration	50-500 milliseconds
Spacecraft Thermal Control	Radiative Heat Transfer	Minutes to hours
Human autonomic response	Biochemically-induced electrical signals	500 milliseconds – 1 second
Human decision-making	Verbal and visual signals between humans, and brain physiology	Minutes to days
Data computation	Electron transport and processor cycle times	10-100 milliseconds
Planetary probe radio data transfer	Electromagnetic waves	Seconds to hours

Figure 3: Typical Functions, Mechanisms and Characteristic Times

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 fault manifests itself. Virtually all design faults behave in this manner by their nature, but physical component failures can act similarly. The classic example of this is where 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.

There are other limitations to this theoretical near-100% detection probability. First, practicalities of cost may make it prohibitive to monitor all appropriate behaviors. Second, every added hardware sensor itself may fail. Third and most importantly, the symptoms of faults and the consequences of the fault may be such that by the time a fault symptom becomes visible, the system has already failed, or the time between detection of a fault symptom to system failure is so fast that nothing can be done about it. In such cases, the fault acts much like a latent fault that does not manifest any symptoms until the fault actually

occurs. So even though it is a certainty that any fault we care about will create symptoms that we can detect, this is no particular cause for celebration if the system is doomed by the time we can see the symptom.

Of course, some functions are crucial for basic system operation, while others provide performance enhancements or margins. Failures of enhancing components will degrade system performance now or in the future, but will not cause total system failure. An example is the failure of an unused chunk of memory. Built-in-Test may well detect such a failure, and the resulting actions ensure that the software never uses the location, thus reducing memory margins. Failures of components necessary to basic system operation will lead to failure of some or all functions. For some systems, function failures can lead to injury or death to humans. These are the situations in which the discipline of "safety" comes into play. Safety and health management are related but not identical, because there exist situations in which a system performs properly but still remains hazardous to humans (military weapons, for example), while there are many fault situations in which human safety is unaffected (a robotic probe fails in deep space).

Criticality generally refers to a scale of possible ramifications of a fault. The most critical ramifications can cause loss of human life, and these rank "highest" on a criticality scale. At the other end of ramifications are those faults that are merely a nuisance, leading to slight degradation of current or potential future performance. In between these extremes are a variety of possibilities. These include: losses of margins against future failures (losing one of two strings of a dual-redundant system, for example), significant losses of performance that leave other functions relatively unaffected (such as degradation of a science instrument leading to loss of science data or loss of a high-gain antenna, leaving only a low-gain antenna available), etc.

The criticality of a fault frequently depends on the function the system is performing at the time of the fault. For example, a fault occurring during an orbit insertion maneuver is far more likely to lead to total system failure than that same fault occurring during a benign several-month cruise phase. Similarly, if a fault occurs and it is detected while an aircraft is on the ground, it is often less likely to cause significant damage or danger than if that same fault occurs in flight.

Combining the time and criticality leads to an important theoretical construct: *time-to-criticality* (TTC). One of the most important factors in mapping a system function to an architectural design is that function's TTC, which itself depends on the mission mode (the changing functions of the system as it performs various tasks). As noted previously, if a fault does not affect a system function, it is irrelevant. If it does affect a function, then the TTC determines the ramification of the fault, as well as how long it will take for that ramification to occur. These two factors then determine the speed of the physical mechanism needed to respond to a loss of that function, along with the type of response necessary. If the fault can cause loss of the entire system, and there is no time to allow an on-board computer response, then the designers must create hardware mechanisms that prevent, mask, or immediately respond to a fault, without awaiting any computer computations. If a fault is relatively benign in the near-term, but degrades performance in the future, then providing an on-board mechanism to detect the problem and relay it to ground-based humans to determine the proper course of action is appropriate.

Prevention: ISHEM in the System Life-Cycle

Failure prevention requires not only active means to detect and respond to anomalies, but also measures in the design, manufacturing, and operational processes to eliminate the existence of certain potential failure modes. It is quite common for misunderstandings in design to lead to architectures and subsystems to contain failure modes that could have been removed entirely. Prevention methods include a host of means, from designing out failures, to improved means of communication to reduce the chances of misunderstandings or lack of data, to inspections and quality control to catch parts manufacturing or software problems before they get into the final system, to operational mechanisms to avoid stressing vulnerable components. It is generally far more cost-effective to design out problems at the very start of a program, and to design in appropriate mitigation methods, than it is to patch a host of operational

mitigation features into a badly-designed system. Failure prevention is thus largely an issue of appropriate design processes, much like systems engineering.

The design process is about envisioning a goal along with an idea for the mix of technologies and human processes that can achieve that goal, and progressively elaborating these concepts until it can take shape into physical artifacts interacting with humans to perform the function originally intended. To deal with the potential of the artifact's later failure, designers must understand where failures are likely to originate within the design process, and also to progressively better understand the ramifications of faults in the system as the idea moves from conception to reality. Thus as the system's requirements, architecture, and components become elaborated, we must analyze and understand how failures arise and propagate within the requirements, architecture, and components.

The concept of operations defines the essential functions that the system must perform, along with how humans will interact with the system's technologies to perform those functions. This in turn leads to an initial determination of dependability requirements, which typically include fault tolerance and redundancy levels, quantitative reliability specifications, maintainability (typically through mean-time-to-repair requirements), and safety.

Typically, faults are considered only after the system's architecture has been conceived and the components selected or designed. The usual reason for waiting to consider faults later is that failure modes, effects, and criticality analyses (FMECA) cannot be done until there are components available, upon which the analyses operate. The problem with this strategy is that by that time, many faults have already been designed into the system. What is needed is a way to analyze faults and failures before the specific hardware and other components are specified.

	Initial Requirements	Conceptual Design	Preliminary Design	Detail Design	Fabrication and Test	Deployment & Operations
Quantitative Requirements	• Reliability Allocation • Availability • Margin Philosophy • Time to Criticality	• MTTR Req't • System TTC Timing Req'ts • Margin Allocations	• Subsystem TTC Timing Req'ts • Margin Req'ts • Reliability Req'ts	• Final Margin & Reliability Req'ts	• Requirements Updates	• Requirements Updates
Qualitative Requirements	• System FT Req'ts • System FA Req'ts • Isolation • Fault Classes for FT	• Subsystem FT Req't • Subsystem Functional Fault Req't	• Fault Injection Req'ts • SW, HW, Operations Req'ts	• Final System/ Subsystem/ Component Req'ts	• Requirements Updates	• Requirements Updates
Fault Set Definition	• Fault Classes • Major Implementation Fault Types (Engine Out, Electronics, ...)	• Subsystem Functional Faults (Top Down)	• Preliminary FMEA (Bottom Up) • Preliminary Fault Set Reduction for Fault Injection	• Final FMEA • Fault Set Reduction for Fault Injection	• Updates to Fault Set	• Updates to Fault Set
Fault Analysis & Modeling	• System Cost / Reliability Trades • Testability Analysis	• System Interaction TTC Analysis • Functional Fault Matrix • Initial Behavioral Model	• Detailed Sys. Mod. • Detailed Rel. Anal. • Simulation with Fault Injection • Cost / Reliability Anal. for Params.	• Simulation with Fault Injection • False Alarm Analysis	• Fault Injection into As Built System • System Characterization • Model Updates	• System Characterization • Model Updates • Fault and Contingency Analyses
System Design	• Initial System Concept • Operations and Maintenance Concepts	• Initial Subsystem Concept • ECR/FCR Definition at Function Level	• Detail ECR/FCR • Parameter, Algorithm, and Sensor Selection	• Final Design • Threshold Determination	• Design Feedback • Threshold Adjustment from System Characterization	• System Characterization • Design Updates • Contingency Plans
Verification & Validation	——	• V&V Plan Draft for SHM • Allocation of V&V Methods: Test, Analysis, Proof, Simulation	• Incorporate Prelim FMEA into V&V • Define Fault Inject Techniques • Proof of Key Algorithms	• Test Procedures • V&V by Analysis, Simulation, Test, and Formal Proof	• Subsystem and System Testing Under Stressing Conditions & Fault Conditions	• Testing Updates

Figure 4: ISHEM in the System Life Cycle

This can be accomplished through a "functional fault analysis" in which time-to-criticality plays an essential role in designing for dependability. The analysis takes the initial system architecture, and posits the failure of the function performed by that architectural element. Since each element consists of known physical processes, and since the connections between architectural components are physical or logical, the both the failure of components and the propagation of the fault symptoms from the component failure can be analyzed. These determine timing, redundancy, and fault and error containment requirements for the system, and an allocation of health management functions to various system control loops or to

preventing the fault occurring by use of large design margins. The TTC analysis largely defines the top-level roles of humans, computers, and other technologies in dealing with faults. From these allocated roles flow the specific sensors, algorithms, and operator training necessary to monitor and respond to system health issues.

Once the roles of humans and machines are defined and the dependability requirements levied, the system and subsystem engineers can go about their typical design processes, which are augmented by institutional arrangements to enforce dependability standards in design, and not merely in manufacturing or verification and validation. Testability tools and analyses can greatly aid the selection of sensors and other related issues.

A "health management engineer" (HME) position created at the system level significantly aids dependability design. This engineer works alongside the chief engineer and the system engineer. 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 health management 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.

The functional fault analysis also provides a starting point for more in-depth Failure Modes, Effects and Criticality Analyses, Risk Management Analyses and related analyses, which in turn provide the data on fault symptoms needed to test the system. Health management systems by their nature do little besides monitoring until faults occur, at which point the relevant humans and machines spring into action. The only way to test health management systems is to create fault conditions that will stimulate the algorithms or the humans into executing contingency procedures. It is well-known that mission operations training for human space flight relies on simulation of failures, which forces the mission operations team to work together under stressful conditions to solve problems. The same holds true for robotic missions. In both cases, simulated failures are injected into the system, which includes both the real or simulated flight vehicle and the operators (and crew, if applicable) of that vehicle. The FMECAs provide many or all of the symptoms used to simulate faults, which then test both the flight hardware and software, as well as the operators and crew. To inject faults, the test and mission operations systems must be able to simulate fault symptoms as well as nominal component behavior.

Cost is an important factor in the design process for dependability as for any other system feature. While it might be technically feasible to create design fixes to various faults, in some cases it may be cost-prohibitive to do so. In these cases, the solution may well be to take the risk of failure, after doing appropriate statistical and physical analyses to assess the risks, and weigh those versus the cost of a design solution. In other cases there may be a range of potential solutions that can mitigate various levels of program and system risk. A common solution is to use operational or procedural fixes to a problem. Thus a spacecraft may have a thermal fault that can be operationally mitigated by ensuring the spacecraft never points its vulnerable location at the Sun. Cost versus statistical reliability tradeoffs often help to make specific design decisions of this sort. Another crucial cost issue is to automate the transfer of design knowledge from ISHEM-related designs and analyses for use in procedural or automated mitigation, such as contingency plans and artificial intelligence-based models for diagnosis and prognosis.

ISHEM Design and Operations Principles

The previous sections describe the fundamentals of a theory of ISHEM. However, translating these theoretical ideas into concrete design principles and processes is an effort that will take many years and experience with implementing these ideas. This section begins the process of moving from theory to principles that can form a basis for action. It collects a number of promising avenues for further research and application, as opposed to a complete set of principles deductively derived from the theory, in the hopes that researchers, designers, and operators can use these as stepping stones to move the theory and the applications forward.

One of the major problems facing system designers and operators is the problem of "preparing for the unforeseen." The small but growing body of literature on how and why technologies fail makes it clear that failures often occur due to some unforeseen circumstance or implication, either internal or external to the system. If failures were easy for humans to predict, then they would be quite rare. Unfortunately, even in those cases where there is evidence of impending failure, a variety of factors cloud human abilities to perceive or understand the signals of that impending failure. This reality means that designers and operators must somehow prepare for the unexpected and recognize that the problem that is likely to happen will often be one that nobody considered, and that never showed up in any analysis or FMEA. Since these unexpected failures are by definition unanticipated, the FMEAs used as the basis for fault injection and testing do not include them. Nor will system models and simulations necessarily include these faults. The system's responses, whether by hardware, software, or people or some combination, cannot, therefore, be anticipated.

Put in other terms, *humans frequently create systems whose behavior, particularly in fault cases, is so complex that their creators cannot fully predict it.* Unlike nature, which is a reasonably stable entity that scientists can study over the course of centuries in the knowledge that it does not change very much, every human-engineered system is unique, with behaviors that change with each change in design or component. A launch vehicle that seems reliable in the present may become more unreliable in the future due to design changes or changes in its operational environment, such as the retirement of experienced operators.

The biggest fear of any engineer or operator of a complex system is the fear of what s/he does not know. What subtle design interaction has gone unnoticed for years, ready to strike in the right operational circumstance? What aspect of the external environment has not been anticipated, leaving the system to cope with it in unexpected ways? What nagging minor problem is actually a sign of a much bigger problem just waiting to happen? The potential fault space is essentially infinite, and there is no way, even in principle, to be sure that all significant contingencies or problems have been anticipated. In fact, there is a significant probability that they have not.

It is possible in principle to detect the existence of any "significant" fault. *Since any fault of significance must manifest itself with a symptom that affects a system function, fault detection can approach 100% by monitoring all relevant system functions.* Although we cannot define all the things that might go wrong, we can in principle determine what it means for the system to behave properly. Designers and operators should be able to define limits of proper functioning for all relevant system functions, and then detection mechanisms can be designed into the system to monitor those functions. Fault detection need not worry about all of the possible ways in which a function can go awry—a task with no knowable bounds—it merely needs to determine deviation from nominal functioning, which is a finite problem. This is the basis for the theory of parametric fault detection, which compares actual performance to expected performance, seeking a residual difference that may indicate a fault. As noted earlier, the existence of *latent faults and time criticality issues negate some of the potential benefits of the ability to detect faults*. The field of prognostics is dedicated to detecting small clues in current behavior that lead to prediction of future failure, and is hence one means to deal with latent faults.

Isolating the location of a fault is in principle more difficult, but is generally eased by the practical limitations on the number of possible components that can be electronically or mechanically switched. The so-called "line replaceable unit" or LRU, is the level of component at which maintenance personnel can replace a unit, or in the case of robotic spacecraft, that can be electronically or functionally routed around. From a practical standpoint, it does not necessarily matter if one can isolate the fault to a specific chip, if one can only switch an entire computer processor in which the chip exists. In practice, a typical procedure to determine where a fault exists is simply to keep swapping components until the system starts to function, and assuming that the last swap switched out the faulty unit. Isolation to the LRU can be, and often is a finite and straight-forward process under the assumption that only one fault exists in the system at a given time. However, the existence of latent faults that manifest themselves only when another fault occurs cannot be discounted. This complicates matters, and has caused the complete failure of a number of systems. Another complication is when the root cause of the fault is not any single component, but

rather the interactions between components. In these cases, and also in cases where there is no unit that can be readily replaced, it is less crucial to isolate failures than it is to know of their existence and find other means to mitigate them in the current or in future systems.

Diagnosing the root cause of these faults is not so easy, and in principle cannot be determined in all cases. Determining how to best operate a system in the future often requires knowing the specific root cause of a fault in the present. When the system is in deep space, for example, it is sometimes impossible to determine the exact cause of a fault with certainty, due to a lack of data and inability to directly inspect or test the spacecraft. In these cases, operators determine the set of possible causes, and then determine future actions based on the possibility it could have been any of them. Even in ground-based situations where the device in question can be torn down, tested, and inspected, finding the root cause is often quite difficult, as the component in which the fault occurs almost always has been built by another organization that could be in a different country. The root cause of the fault may well be in an assembly line or with the procedures or performance of a specific person or machine. In addition, it is often difficult or impossible to create the environment in which a fault occurred, making it difficult or impossible to replicate. The most important thing is to ensure system functionality. That is always aided by proper diagnosis of root cause, but it can nonetheless often be accomplished even when the root cause cannot be determined.

Behind many of these difficulties lies the problem of complexity. Complexity is a feature that relates to human cognitive and social abilities, and hence solutions to the problem of complexity must be tailored to and draw from those same human abilities. While it is often stated that computers can resolve the problem of complexity, this is not strictly true. Only if computers can compute, create and/or present information in a way that makes it easier for humans to understand systems and their operations, will they assist humans in dealing with complexity. A simple example is the use of computer graphics to portray information, as opposed to many pages of text or a hexadecimal readout of computer memory. Humans frequently find a graphical representation easier to comprehend, even though this is not the optimal representation for computers, which ultimately store data in serial digital fashion using binary operations. The presentation of the data in a so-called "user-friendly" form makes all the difference.

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 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*. Secondly, *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.

Complexity also relates directly to knowledge. Something is "too complex" when our knowledge about them is incomplete or hard to acquire. As previously described, the technologies we create merely embody the knowledge of those who create them. Gaps or errors in our knowledge lead to faults in the devices we build. If we do not find the gaps and errors in our knowledge before we build a device, then the interconnection of the various components will in some cases lead to immediate failure upon connection (interface failures), or in other cases leads to failures under special circumstances encountered only later in operation. System integration, which is the process of connecting the parts to make the whole, is when many failures occur precisely because many miscommunications of or inconsistencies in our knowledge show up when we connect the parts together. The *parts fail when connected because the knowledge they represent is inconsistent or fallacious*.

This leads to a fundamental principle. Since technologies are nothing but embedded knowledge, the only way to determine if a fault exists is by comparing the existing knowledge with another *independent* source of knowledge. When components are hooked together for the first time during integration, this

compares the knowledge of one designer with that of another, and mismatches result in errors appearing at this time. Redlines placed in on-board fault protection software are often generated by different processes than those used to design the system they are trying to protect. When the source of knowledge is the same for a design and the test, then both can be contaminated by a common assumption underlying both, allowing a "common mode fault" to slip by unnoticed. Since an independent source of knowledge is needed, this generally requires a different person or group from the original designer, and this in turn requires communication.

The end result of these insights is that *dependable operation of a system requires communication processes to compare independent knowledge sources for all critical flight elements and operations.* Even when this is done, it does not guarantee success, as it is always possible that some faults will remain undetected because there remain common assumptions with the "independent" knowledge sources, or there are situations that none of the knowledge sources considered. The only remedies are to find yet more independent knowledge sources to consider the system and its many possible behaviors, and to give existing knowledge sources more time to consider the possibilities and ramifications.

Interestingly, complete independence of knowledge is impossible. Someone that has sufficiently different background to have "complete" independence of knowledge will by definition know 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 *all* of the same assumptions, background, and training. Someone with complete independence will have *none* of the assumptions, background, and training of the organization they are trying to verify. How, in that case, will they have any knowledge of the organization or devices if they know *nothing* about it? They will be useless in verifying the operation or device.

Knowledge independence does not and cannot mean complete independence. It means that some commonalities must be eliminated, but others must remain to allow for any kind of verification. This is a conundrum that cannot be evaded. The solution appears to be to have different kinds of verification, with different people having different backgrounds, each of which has some commonality with the item and organization in question, but collectively having many differences. Thus another principle is that *it is impossible to attain complete knowledge independence for system verification.*

The principle of knowledge independence, and its corollary stating impossibility of complete independence are used quite frequently, though not described quite in this manner. Testing of all kinds is a means of verification because it is a means to use another mechanism and set of knowledge to interact with the system. The test subsystem itself embodies knowledge of the system, as well as the simulated faults. Analysis is where either the designer or someone else uses a different method to understand the behavior of the system than the original design itself. So too is inspection, where an inspector visually (or otherwise) searches for flaws using his or her knowledge of what s/he should see. Finally, even design mechanisms like redline tests or triple modular redundant voting are independent tests of behavior. Testing an in-flight behavior means comparison to some other assessment of in-flight behavior, whether it is an *a priori* analysis leading to a redline boundary placed into a software parameter, or two processors being compared to a third. Trying to find a command error before upload to a spacecraft depends on humans reviewing the work of other humans, or computer programs searching for problems using pre-programmed rules for what normal (or abnormal) command sequences should appear. In all cases, independent knowledge sources come into play. Another way of viewing knowledge independence is to realize that it is another way of saying that we use redundant mechanisms to check any other mechanism, whether by humans or by machines programmed or designed by humans.

Aerospace systems frequently operate correctly because many of the design, development, manufacturing, and operational processes actually compare independent knowledge sources through communication processes. Systems management, which is the management system developed within the U.S. Air Force and NASA in the 1950s and 1960s, developed to deal with the technical, political, and economic issues of spaceflight. Systems engineering developed in the same time and for similar reasons. (Johnson 1997) These managerial processes primarily use social means to check for technical problems. To the extent that they actually compare independent knowledge sources and provide sufficient time to

those sources to consider all possibilities, they prevent many failures. However, to the extent that these processes have become bureaucratized, which is necessary to ensure that beneficial practices are passed along to the next generation, the very processes of standardization create common beliefs that undermine the independence and alertness needed to find problems.

This leads to another principle: *bureaucracy is needed to ensure consistency of dependability processes, but human cognitive tendencies to lose focus during repetitive actions and to suppress the reasoning behind bureaucratic rules leads create conditions for human errors*. 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 tiny 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, which excel at repetition. Unfortunately, this is not always possible. Humans must have some mind-stimulating activities to maintain proper awareness. The solution is almost certainly related to 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 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. We need knowledge based both on empirical experience (simulations) and fundamental principles that allow operators to reason through failure issues, both as designers and operators.

To summarize, the most significant aspects of ensuring dependable system design and operation relate to the uncertainties in our knowledge of that system, and our human inability to maintain proper focus. The faults that lead to failures are frequently unanticipated, unanalyzed, and not modeled. Unfortunately, many others are simple, yet remain undetected due to human limitations. The first strategy to solve this problem is to simplify the system as much as possible, which means dividing the system into comprehensible chunks needed for individual comprehension, and then defining clean interfaces between them, which minimizes the chances of social miscommunication. Then the system must be analyzed and verified by comparison with independent knowledge sources. Unfortunately complete independence is impossible, even in principle. Nonetheless, a strategy of using multiple knowledge sources is nonetheless crucial to detect failures before operational use of the system. Actual operational use is, of course, the ultimate test of knowledge. In this case, exposure to the environment and to the system's human operators will unearth those problems not found earlier. Maintaining proper focus to detect and resolve problems before they lead to failure requires a balance between repetition and consistency on one hand, and originality and creativity on the other.

Overview of ISHEM Research and Practice

Although the ISHEM label is somewhat new, design engineers and system operators have created many methods for preventing and mitigating faults, while researchers have been developing a variety of technologies to aid the practitioners. In addition, other disciplines have begun assessing the problem of system failure and conversely, the issue of system health from their disciplinary or problem-based perspectives. This collection of papers is organized into several groups to reflect the current state of the art both in theory and in practice.

At the top level, this paper, along with others on the current ISHEM state-of-the-art, the system life cycle, and technical readiness assessment describe top-level issues that affect both research and practice in all of the other disciplines. They provide theoretical and practical frameworks in which to place the other research and application areas.

The next set of papers, on knowledge management, economics of systems integration, high reliability organizations, safety and hazard analysis, verification and validation and human factors, each describe cognitive and social issues of integrating humans and machines into dependable systems.

Another major way of viewing ISHEM is to review what has been done in practice in major application areas. For aerospace, this means understanding the nuances of how ISHEM is designed into commercial and military aircraft, rotorcraft, robotic and human-occupied space vehicles, launchers, armaments and munitions, and the ground operations to operate these diverse kinds of systems.

Similarly, but in a more disciplinary fashion, these systems are built from subsystems, each of which has its own nuances. Thus power systems have similar issues whether for spacecraft or commercial aircraft. Other typical subsystems with unique ISHEM features include aircraft and spacecraft propulsion, computing, avionics, structures, thermal and mechanical systems, life support, and sensors.

Finally, researchers and system specialists have devised a variety of methods that apply to specific portions of the ISHEM functional cycle. Diagnosis and prognosis are the most obvious. However, there are several others: quality assurance, probabilistic risk assessment, risk management, maintainability, failure assessment, failure data collection and dissemination, physics of failure, and data analysis and mining.

Conclusion

The complexity of the systems we now create regularly exceeds our ability to understand the behavior of our creations. This results in a variety of dangerous, costly, and embarrassing failures. One contributing cause for these failures is the lack of any comprehensive discipline to understand the nature of our engineering systems, the roles of our human cognitive and social abilities in creating them, and the resulting faults and failures that ensue.

Integrated System Health Engineering and Management is a comprehensive umbrella for a variety of disparate methods that have developed over decades to prevent and mitigate failures. We have outlined here the beginnings of a theory and some principles to undergird ISHEM practices and technologies, so as to aid in the implementation of ISHEM 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.

Acknowledgements

Thanks to Phil Scandura for helpful comments regarding the definition of failure, aircraft health management and the historical context of ISHEM. Andrew Koehler provided thoughtful ideas regarding complexity and causality. Serdar Uckun correctly pointed out the complexities of a system's interactions with its external environment, and the relationship of prognostics to fault latency.

Bibliography

[Albert et al. 1995] Albert, Jeffrey, Dian Alyea, Larry Cooper, Stephen Johnson, and Don Uhrich, May 1995. "Vehicle Health Management (VHM) Architecture Process Development," *Proceedings of SAE Aerospace Atlantic Conference*, Dayton, Ohio.

[Bijker, et al., 1987] Bijker, Wiebe E., Thomas P. Hughes, and Trevor Pinch, eds. 1987. *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology*. Cambridge, Mass.: MIT Press.

[Campbell et al. 1992] Campbell, Glen, Stephen B. Johnson, Maxine Obleski and Ron L. Puening. 14 July 1992. *System Health Management Design Methodology*, Martin Marietta Space Launch Systems Company, Rocket Engine Condition Monitoring System (RECMS) contract, Pratt & Whitney Corporation, Purchase Order #F435025.

[Johnson 1997] Johnson, Stephen B. 1997. "Three Approaches to Big Technology: Operations Research, Systems Engineering, and Project Management," *Technology and Culture* 38, no 4: 891-919.

[Johnson 2002a] Johnson, Stephen B. 2002. *The United States Air Force and the Culture of Innovation 1945-1965*. Washington, D.C.: United States Air Force and Museums Program.

[Johnson 2002b] Johnson, Stephen B. 2002. *The Secret of Apollo: Systems Management in American and European Space Programs*. Baltimore: The Johns Hopkins University Press.

[Johnson 2003] Johnson, Stephen B. 2003. "Systems Integration and the Social Solution of Technical Problems in Complex Systems," in Andrea Prencipe, Andrew Davies, and Michael Hobday, eds. *The Business of Systems Integration*. Oxford: Oxford University Press, 2003. pp. 35-55.

[Vaughan 1996] Vaughan, Diane. 1996. *The Challenger Launch Decision: Risky Technology, Culture, and Deviance at NASA*. Chicago: University of Chicago Press.

[Webster's 1991] *Webster's Ninth New Collegiate Dictionary*. 1991. Springfield, Massachusetts: Merriam-Webster, Inc., Publishers.

Introduction to Integrated System Health Engineering and Management in Aerospace

Dr. Stephen B. Johnson
NASA Marshall Space Flight Center
sjohns22@uccs.edu

Outline of Talk

- Definitions
- Operational & Design Theory
- Principles

Integrated System Health Engineering & Management

- *ISHEM = the processes, techniques, and technologies used to design, analyze, build, verify, and operate a system to prevent faults and/or mitigate their effects*
- Technical, individual, and social aspects
- Synonym: Dependable System Design and Operations
- "Dependability"

Complexity

- Beyond the capability of any one person to understand or keep track of all details
  - Heterogeneous (power, propulsion, etc.)
  - Deep: requires many years of study to master
  - Scale: the system requires so many components that it is impossible for any one person to keep all in mind
  - Interactivity: interactions between internal components, and with the external environment are "messy"

Implication of Complexity

- By definition, beyond what any one person can master (our cognitive abilities are limited)
- REQUIRES communication among individuals
- Implication:
  - Engineering of a "complex" system requires excellent communication and social skills

Failure

- "A loss of intended function or performance of an unintended function."
  - Can be designer's or user's intent
- Failure is both individually and socially defined
  - "in the eye of the beholder"
  - Some "failures" are considered normal by others

Faults and Errors

- **Fault: The physical or logical cause of an anomaly.**
  - **The "root cause", can be at various levels**
  - **Might or might not lead to "failure"**
- **Anomaly (error): A detectable undesired state.**
  - **The "detector" must ultimately interpret the "state" as "undesirable"**
  - **Can be user, designer, others**

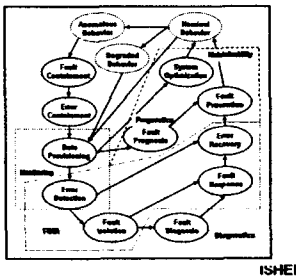
Causes of Faults and Failures

- **Individual performance failure (cognitive)**
  - **Lack of knowledge (unaware of data)**
  - **Misinterpreted data**
  - **Simple mistakes (transposition, sign error, poor solder, etc., usually from human inattention)**
- **Social performance failure (communicative)**
  - **Miscommunication (misinterpretation)**
  - **Failure to communicate: information exists, but never got to the person or people who needed it**

Embedded Knowledge

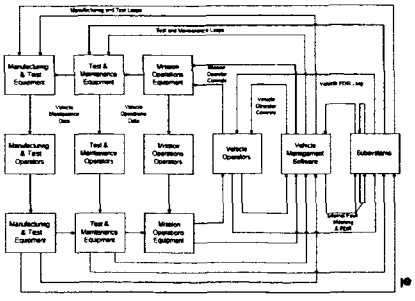
- **Technologies are nothing more than "embedded knowledge"**
- **Technologies embody (incarnate) the knowledge of their creators**
- **"Faults" result from flaws in the knowledge of the creators, OR mismatch in understanding between creators and users**
  - **Cognitive or Communicative!**

ISHEM Functional Relationships



- **Circular, "closed-loop" relationships**
- **Hints at the physical architecture**

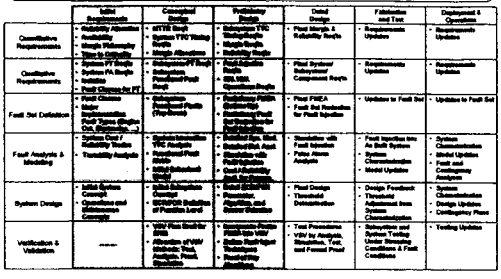
ISHEM Operational Architecture



Typical Functions, Mechanisms, and Characteristic Times

Function	Physical Mechanism	Characteristic Time
Electrical Power	Electron transport	1-10 milliseconds
Attitude Control	Thruster impulse or reaction wheel acceleration	50-500 milliseconds
Spacecraft Thermal Control	Radiative Heat Transfer	Minutes to hours
Human autonomic response	Biochemically-induced electrical signals	500 milliseconds – 1 second
Human decision-making	Verbal and visual signals between humans, and brain physiology	Minutes to days
Data computation	Electron transport and processor cycle times	10-100 milliseconds
Planetary probe radio data transfer	Electromagnetic waves	Seconds to hours

ISHEM in the System Life Cycle



Principle of Knowledge Redundancy, and Limits

- Checking for failure or faults requires a separate, independent, credible knowledge source
- Commonality means that reviewers share common assumptions with the reviewed
- Independence means reviewers share nothing in common with the reviewed
- Complete independence neither possible nor desirable

Clean Interfaces

- Desired and sometimes required
- Reduce the "interactivity" between components
- Reduce the interactivity of the people and organizations designing and operating the components
- Simplifies communication, reduces chance for miscommunication!

Bureaucracy and "Situational Awareness"

- Bureaucracy needed to institute and repeat processes for dependability
- Bureaucratization: repetition and suppression or forgetting of reasons behind the rules leads to inattention or misunderstanding, and hence to faults
- Must foster individual "awareness" within the bureaucracy... create bureaucracy to fight the deadening effect of bureaucracy!

Conclusion

- NASA has a "culture problem" that leads to occasional failures
- The problem is social and cognitive as well as technical
- ISHEM to be the overarching theory over the technical, social, and cognitive aspects of preventing & mitigating failure
- We are working to install / instill ISHEM into the new Vision for Space Exploration