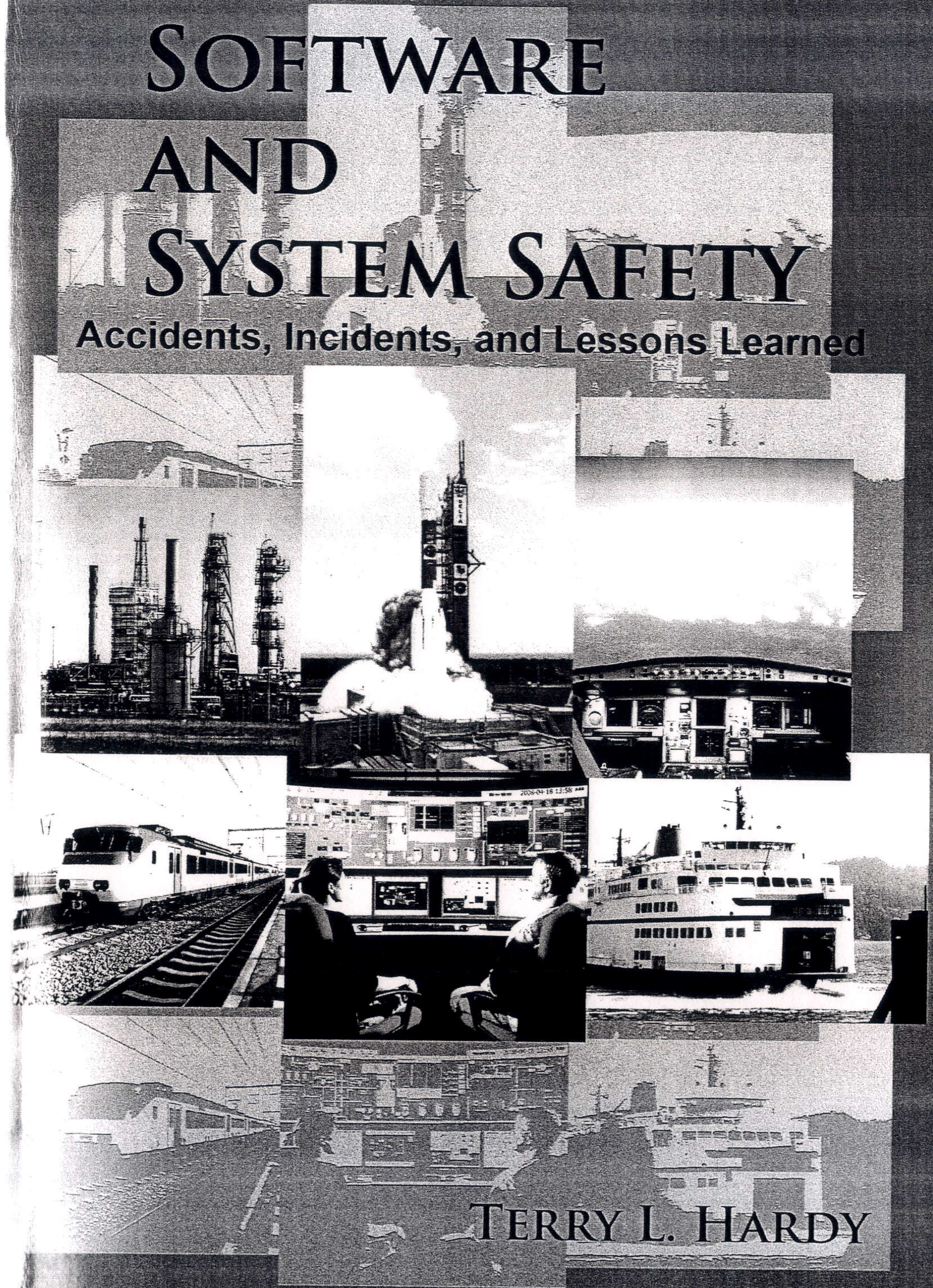
SOFTWARE AND SYSTEM SAFETY

Accidents, Incidents, and Lessons Learned



TERRY L. HARDY

CHAPTER 3

Hazard Identification

A problem well stated is a problem half solved.

- Charles F. Kettering

Judge a man by his questions rather than by his answers.

- Voltaire

On September 17, 2008, four employees of the Saipem company were killed and four others were injured during installation of the Medgaz Trans-Mediterranean pipeline in international waters between Algeria and Spain. The employees were killed when the automated hydraulic pipe handling system released sections of the pipeline, causing pipeline components to fall on the platform where the employees were working. Immediately prior to the accident a power failure had caused operations to stop. To fix the problem, the employees reset the computer system memory and continued with operations. The employees did not realize that the preloaded Erasable Programmable Read-Only Memory (EPROM) of the Programmable Logic Controller (PLC) had been configured to open all clamps after memory reset. When the clamps opened, the pipeline components were released and fell on the employees. The report noted that the practice of allowing personnel on the platform during hazardous operations contributed to the accident. Following the accident the PLC logic was modified to remove EPROM memories from the system and to require confirmation of hazardous sequences by the operator.[1,2]

Hazard identification is arguably the most important part of the safety analysis effort. One could think of the hazard identification step as defining the problem to be solved. If one does not properly identify the problem then it becomes difficult to assess the risk or postulate solutions. Identifying hazards can be a difficult process, and software further complicates this effort through additional complexity and system interactions. Hazard identification usually starts early in the development process, and then continues through development as new information becomes available on the system and its operation. This chapter will discuss key elements in the hazard identification process and will present software safety lessons learned in identifying hazards.

3.1 Hazard Identification Basics

The first step in identifying hazards is to understand the system or operation, including the resources of value to be protected. The system includes hardware, software, processes, humans interacting with the system, and system environments. All of the components of the system should be identified. The boundaries of that system must be understood, including the interfaces between subsystems and external entities. In addition, a design description should be provided. That design description should include the following at a minimum:

- Functional descriptions, indicating the intent of the functions
- Operational descriptions, indicating what the system is supposed to do in all major phases of operation
- Subsystems and components that comprise the system
- Process inputs and outputs
- Support equipment and tools
- Supporting data

Additional items may need to be reviewed to understand the software design and functionality. Some of those items include the following:

- List and description of Computer Software Configuration Items (CSCI): A CSCI is an aggregation of software that is designated for configuration management and treated as a single entity in the configuration management process. These are logical groupings of computer software units.
- List and description of Computer Software Components (CSC): A CSC is a functionally or logically distinct part of a computer software configuration item, typically an aggregate of two or more software units.
- List and description of Computer Software Units (CSU): A CSU is the smallest grouping of software.
- Data dictionaries and documentation of design entity attributes: name, function, dependencies, etc.
- Documentation describing CSCI, CSC, and CSU interfaces.
- Diagrams: architecture diagrams, data flow diagrams, class diagrams, functional block diagrams, etc.
- Documentation of system usage: scenarios, use cases, trade studies, etc.
- Heritage and Commercial Off-The-Shelf (COTS) software documentation.
- Development and test environment documentation.
- External data element documentation.
- Descriptions of hardware-software hybrid devices, such as complex electronics, including functions, interfaces, etc.

Understanding the software functionality is a critical step in software safety. Often this means breaking down the system description into small enough components to analyze functionality. By understanding that system the analyst can begin to discover potential system issues caused by software and computing systems.

Once analysts understand the system then they can begin considering what can go wrong and identifying hazards. There are several approaches that can help visualize hazards. One approach is to use hazardous element checklists based on previous experience. Checklists are available from a number of sources.[3] These checklists can take different forms, such as the following:

- General Hazard Checklists: electrical, explosives, leaks, acceleration, contamination, etc.
- Energy Source Checklists: fuels, propellants, batteries, pyrotechnics, nuclear, etc.
- Space Function Checklists: crew ingress and egress, launch escape, fairing separation, etc.
- General Operations Checklists: welding, cleaning, extreme temperatures, etc.

Another approach to identifying hazards is to use past experience to help drive the analysis. This approach works best when developing systems that are similar to those operated in the past. This approach can include review of previous hazard analyses, and can include operational experience and studies of mishaps and accidents. Design practices, regulations, and standards can also assist in the development of analyses. By understanding an accepted practice one can look for ways in which the new system can fail. These standards include safety criteria, such as the proper way to use relief valves. Regulations can be a good source of potential standards because many prescriptive rules, such as Federal Aviation Administration regulations, came about as a result of an accident or incident.

Still another way to identify hazards is to consider individual failure states.[3] Examples of failure states might include the following:

- Failure to operate (including failure to start or stop operation)
- Operates incorrectly or erroneously
- Operates inadvertently
- Operates at the wrong time
- Receives or sends erroneous or conflicting data

With respect to software, analysts should first consider the multiple ways software can contribute to a system hazard. Note that the term *software-related hazard cause* is used because, as discussed above, software itself does not cause a hazard, but rather it is software operating

within the system that creates the potential for harm. Table 3-1 provides an example of various ways software can contribute to a hazard.

Once the software contributions to system hazards have been considered, specific software error types must also be analyzed. This allows the analyst to identify conditions leading to potential software causes, such as those described in Table 3-1. Table 3-2 provides examples of specific errors to be considered.

Table 3-1. Example Software Contributions to System Hazards.[4]

Description	Examples
Failure to detect a hardware, software, or environmental problem	High tank pressure not detected in a chemical reactor, allowing pressure to increase without automated or manual shutdowns.
Failure to perform a function	Software fails to shut down a pump when tank fill level reaches a maximum, resulting in a toxic chemical tank overflow.
Executing a function out of sequence	FIRE command issued before the ARM command resulting in premature missile launch.
Executing a function at the wrong time or when the system is in the wrong state	Landing gear retract command is issued while the aircraft is still on the ground resulting in damage to the aircraft.
Executing a function incompletely	System startup commanding stops before all valves are completely opened, resulting in a buildup of pressure in a fuel oil line and line rupture.
Executing the wrong function or the right function to wrong system	The system commands power off when the power on command is sent. Command issued to shut down electrical circuit but wrong circuit is shut down, cutting power to critical systems.
Failure to provide information to operators	Failure to issue instructions to train operators to stop resulting in collision with another train.
Providing misleading or incorrect information to operators	Aircraft cockpit displays provide incorrect altitude information when flying in mountainous terrain, creating the potential for controlled flight into terrain.
Models and simulations provide incorrect output	Thermodynamic models underpredict reactor tank temperature rise resulting in tank overpressure upon startup of a new process.

Table 3-2. Specific Software-related Errors and Faults.[5]

Description	Examples
Calculation or computation errors (incorrect algorithms, overflow, underflow, etc.)	The software may perform calculations incorrectly because of mistaken requirements or inaccurate coding of requirements. The algorithm may result in a divide by zero condition.
Data errors (improper data, data stuck on some value, large data rates, no data, data out of order, data corrupt, uninitialized data or variables, etc.)	The software may receive out of range or incorrect input data, no data, wrong data type or size, or untimely data; as a result software could produce incorrect or no output data; or both. A sensor or actuator could always read zero, one, or some other value. The software may be unable to handle large amounts of data or many user inputs simultaneously.
Logic errors (improper command, command out of sequence, failure to issue command, inadvertent mode change, too many parameters passed, command sent to the wrong system, command sent when system is in the wrong mode, etc.)	The software may receive bad data but continue to run a process, thereby doing the right thing under the wrong circumstances. The software may not invoke a routine. A module may be exercised in the wrong sequence, or a system operator may interrupt a process leading to an out of sequence command. System operators may interrupt processes causing a problem in timing sequences, or processes may run at the wrong times.
User interface errors (incorrect, unclear, or missing messaging, poor design and layout, multiple events occurring simultaneously, inability to start or exit processing safely, failure to prevent improper user command)	A message may be incorrect or unclear, leading to the system operator making a wrong decision. An unclear graphical user interface may lead to an operator making a poor decision. A system operator may be unable to start or stop a test of a flight safety system once the automated routines have started.
Software development or test environment errors (improper use of tools, change in operating system or COTS module)	Turning on the compiler option to optimize or debug the code in production software may lead to a software fault. Upgrades to an operating system may lead to a software fault.

Table 3-2. Specific Software-related Errors and Faults (continued)

Description	Examples
Hardware-related errors (unexpected shutdown of the computing system, memory overwrites, data transmission errors due to hardware failure, bus hardware failures, etc.)	Loss of power to the CPU or a power transient may damage circuits.
Virus and other malicious attacks	Malware may infect a computer leading to loss of control of a system.
Process failures (software development, assurance, configuration management, etc.)	Failure to follow development standards leads to improper coding of requirements and computing system errors.

Software contributes to hazards generally through errors of omission (failing to do something it was supposed to do) or errors of commission (doing something at the wrong time or in the wrong order, or doing something that should not have been done). Of these, errors of omission are probably the most prevalent, leading to missing safety requirements. Some errors may be latent and not evident until the system has been operating for a long period of time, when a specific set of conditions triggers a problem. Not all software errors lead to safety issues however, and multiple conditions in addition to software errors typically contribute to accidents. Therefore, it is important to first understand how software can contribute to system hazards before analyzing software and computing system errors.

System safety is more than an examination of components; system safety requires an exploration of the interaction and interrelationships between components and subsystems. Therefore, interfaces must be examined. Software interfaces can include those between software modules, software and hardware, hardware and humans, and so on. In envisioning the ways things can go wrong it is worth considering how data are transferred across those interfaces. For example, some considerations include:[6]

- The software's response to out of range and unexpected values
- The software's response to not receiving values
- The software's response to data arriving when it should not
- Software always following the same path through the code (assuring that the software response is deterministic)
- The software's response to inputs that are not bounded in time, and its response to interrupts
- Task priorities are known, understood, and adhered to
- The software's response to data arriving faster or in higher volume than can be used

Note that all methods of hazard identification have limitations. For example, checklists are never complete, and old hazard reports based on similar systems will rarely capture all of the potential problems in a new system because of changes in design and operation. Therefore, to assure completeness, analysts should use multiple approaches in identifying hazards.

Once the hazards have been identified, those hazards must be described. This description is the first chance an analyst has to articulate the problem. Hazard descriptions are important because the description of the problem will help in defining a solution. A poor description will cause confusion and may lead to an organization solving the wrong problem. If descriptions are too complex they may not be understood when reviewed at a later time. A good hazard description contains three key elements:[7]

- *Source:* an activity or a condition that serves as the root cause
- *Mechanism:* a means by which the source can bring about the harm
- *Outcome:* the harm itself that might be suffered

The order of the *source-mechanism-outcome* elements is not important but they should all be included. These elements correspond to the three components of a hazard. The point here is that a hazard scenario should be developed to identify the situation of concern. Examples of hazard descriptions include the following:

- User display goes dark leading to inability of operator to stop pipeline fuel pump resulting in gasoline pipe rupture and personnel injury.
- Operator inadvertently issues a startup command leading to over pressurization of an oxygen tank resulting in a tank failure and personnel injury.
- Software fails to issue an automated command leading to valve failing to open resulting in loss of reactor cooling.

It is difficult in complicated systems to describe a system hazard in a single statement because there are multiple causes that can lead to the undesired outcome. A separate Causal Analysis may be conducted to determine those individual causal factors and circumstances that bring about the undesirable result. Many hazard reports will show a hazard statement followed by a number of causes. For example, a hazard statement may discuss pipeline rupture during transport of toxic liquids resulting in personnel injury, environmental damage, and facility damage. Causes could include metal fatigue, pump overspeed, external impacts, software inadvertently closing valves, and so on.

Identifying software-related hazard causes can be a difficult process because of the complexity of the system and because software is used in so many parts of the system. For example, software may be used in calculations of critical parameters or to convert data units before the data are provided to a user display. Therefore, it often helps to think it terms of functionality. Top down functional decomposition can be a tremendous help in determining critical functions. Two common terms used are *must work* and *must not work* functions. Must work functions are those aspects of the system that have to work in order to function correctly. The concern of must work functions is reliability, and independent functionality is often used to maintain fault tolerance. Must not work functions are aspects of the system that should not occur if the system is functioning correctly. If they do occur, they could lead to a hazardous situation or other undesired outcome. Fault tolerance of must not work functions is achieved through devices used to prevent an inadvertent or unauthorized event, called *inhibits*. In the hazard identification process, it often makes sense to identify loss of critical functionality (must work functions) and inadvertent activation (must not work functions). Be aware that some functions are considered "must work" when the system is in one state but may be considered "must not work" in other states. For example, the function to retract the airplane landing gear must work while the aircraft is in flight but must not work while the airplane is on the ground. In addition, functions may conflict and therefore cannot be considered in isolation. Must work and must not work functions will be discussed further in Chapter 5.

Hazard identification must consider all operating phases and states. The focus of most hazard analyses tends to be on hazards during operation, such as when a chemical plant is at steady-state operation. However, there are hazards associated with starting up a new process, shutting down a process, maintenance actions during operation, and so on. Not all hazards apply to all phases of operation. For example, hazards related to the software-controlled startup of a reactor with the first introduction of chemicals may be different than hazards related to automated monitoring of corrosive aspects of those chemicals over time. Hazard identification must also consider who or what is at risk. Some hazards may apply to maintenance personnel while others may affect the uninvolved public or hardware assets. The same source and mechanism may also apply to more than one asset at risk – a propellant leak may lead to fire and explosion that can injure on-site workers and result in environmental damage.

It is important to remember that hazard identification, particularly with software and computing systems, is an iterative process. Software functionality will change throughout the development life cycle. Therefore, the hazard analysis must be revisited to assure that software causes and specific software errors have been properly addressed.

3.2 Software-Related Accidents and Lessons Learned: Hazard Identification

There are a number of ways that an organization can fail to properly identify hazards and hazard causes. The following describes a number of lessons learned in the form of assessment flaws and commonly overlooked hazard causes. Accidents are used to illustrate how those flaws in the hazard identification process could lead to an undesirable outcome. The hazard identification lessons learned cover the following areas:

- System Considerations
- Hardware and Environment Considerations
- Personnel and Organizational Considerations
- Hazard Identification Process Considerations
- Software-Specific Considerations

3.2.1 System Considerations

Hazard identification must not only consider components and subsystems. Hazard identification must also address the unique aspects of the system.

The hazard analysis should go beyond analysis of single failures: Skyservice 6315

Accidents typically occur when a number of things go wrong, not just a single failure. A single component or human action should not cause a failure, and these conditions certainly should be eliminated. But, while individual causes are important, analysts should also assess multiple causes and conditions, not just failures. Software-related accidents most often occur because of misunderstandings of software's interaction with the rest of the system. Therefore, an organization should think in terms of scenarios when trying to identify hazards, identifying both direct causes and indirect contributors. Often there is confusion between a failure analysis and safety analysis. Failure analyses tend to be "bottom up," answering the question, "What will happen when a failure or condition occurs?" However, hazard analyses that are organized by failures of systems or components may identify failures that are not truly safety problems. Therefore, analyses can lead to overdesign of the system, providing controls for failures that do not lead to an accident. Worse, by only designing for failures, the hazard analysis might miss conditions that could cause an accident where no component has failed, such as conditions with software or human interactions. In fact, software systems allow organizations to build systems that are so complex that operators may not fully understand subsystem interactions. Therefore, both "top down" and "bottom up" approaches should be used.

On February 15, 2001, Skyservice flight 6315, an Airbus A330, was flying from Medan, Indonesia to Jeddah, Saudi Arabia when the aircraft experienced an engine failure. The pilots shut down the engine per the appropriate checklist, and they discharged a fire bottle to extinguish a potential fire. The pilots then diverted to Sri Lanka and successfully completed a landing with no injuries to passengers. Although the pilots performed admirably, they did experience some control difficulties when diverting the aircraft. Following the loss of the engine, the pilots tried to engage the autothrottles. However, the autothrottles refused to engage, preventing the pilots from using the auto thrust functionality. This situation added to the pilot workload and created some confusion because it was contrary to their training. The Transportation Safety Board of Canada (TSB) found in its investigation that the engine failed as a result of stress corrosion fractures of the second-stage turbine blades. This was the second engine failure to occur on this aircraft, the first being 10 days earlier; both failures were the result of stress corrosion. The failure of the auto thrust was the result of an error in the Flight Management, Guidance, and Envelope Computer (FMGEC) software. Auto thrust engagement was controlled by the FMGEC, which received relevant signals from each engine full authority digital engine control (FADEC). If the FMGEC detected an engine-out situation, it was supposed to authorize the auto thrust on the operating engine. However, the failed engine was apparently windmilling after the failure, and the FMGEC interpreted this rotation to mean that the engine was still operating. The FMGEC software logic was such that if it thought both engines were operating, but one was not responding properly, auto thrust could not be authorized. Airbus reconsidered the logic following this incident.[8]

Hardware, software, human, process, and environmental common cause conditions should be considered: Oconee Nuclear Station

A common cause failure is a failure of two or more components, subsystems, or structures due to a single specific event which bypasses or invalidates redundancy or independence. An example might be a computer system that has a backup server to protect data; a flood could be a common cause that disables both the primary and the backup computer system if they are located in the same building. Common cause failures should be considered in hazard identification to assure that risk is not underestimated. An organization should consider not only hardware and environment common cause but also software and process common cause errors, which are frequently overlooked.

On November 7, 2008, reactor Unit 3 of Oconee Nuclear Station in Oconee County, South Carolina experienced an automatic scram, or reactor trip. The reactor protection system operated as expected. The reactor trip was the result of failure of Control Rod Drive (CRD) processors. The CRD processors controlled rod movement, the operator control panel interface, voltage monitoring of single rod power supplies, rod position indication, and power supply voltage monitoring. In normal operation, the CRD received a daily time code from the plant communications system to synchronize the date and time for multiple systems. The reactor Unit 3 cable room receiver obtained the time code and then sent this signal to the CRD. On the day of the incident, the date stamp sent from the communications system was decoded by the Unit 3 receiver with zeros for the day of year for unknown reasons. While zeros were allowed for the time, the CRD was unable to decode zeros for day of year (the expected range was 1 to 366). This problem affected all CRD processors. As a result of the inability of the software to process an out of range value, both the primary P1 and redundant P2 processors completely stopped. Without the processors, the reactor went into a fail safe mode, and the automatic scram was initiated. In addition, the CRD operator control panel lost all input. The software was updated to include range checking on the day of year field following this incident.[9]

Hazards during the transition between phases and states, including new process startup, should be considered: *SBS Typhoon*

Many accidents occur in the transition between operational phases, rather than when the system is up and running in "steady state" mode. For example, airplane crashes occur most often during takeoff or landing, which are transitions between the flight phases. Accidents can occur during startup or shutdown, especially in new or untried processes, often from a combination of factors not expected in normal operation. Starting up a new process or upgrading an existing system can be especially hazardous because changes to design or operations may be made in real time to meet schedule pressures, potentially introducing new hazards.

On February 26, 2011, the supply ship *SBS Typhoon* inadvertently made contact with two other ships in Aberdeen Harbour, United Kingdom, the safety vessel *Vos Scout* and the supply vessel *Ocean Searcher*, causing structural damage and deck equipment damage. Only minor injuries were reported. *SBS Typhoon* was conducting functional trials of a newly installed Dynamic Positioning (DP) system when the accident occurred. The DP system was an upgrade to the existing DP system, which had been susceptible to single-point failures. Just prior to the accident, the crew was preparing to use the new DP system when the chief officer had moved the "manual/DP/joystick" mode selector to "DP" mode. In this mode the system should have issued commands for propulsion depending on inputs from the crew along with position and environmental sensor information. However, when the chief officer changed to DP mode, the system unexpectedly applied full ahead pitch to the Controllable Pitch Propellers (CPP). The chief officer tried to change the mode back to manual, but the ship continued to move ahead. The ship stopped when the chief officer pressed the emergency clutch disengage button, after *SBS Typhoon* had struck *Vos Scout* and *Ocean Searcher*.

The investigation into the *SBS Typhoon* accident was conducted by the Marine Accident Investigation Branch (MAIB). The investigation found that, prior to delivery, the replacement DP system had completed a Factory Acceptance Test (FAT) in accordance with the manufacturer's FAT procedure checklist. However, during that FAT the technician conducting the test did not configure the system properly. When the system was installed in *SBS Typhoon* another technician failed to identify that the configuration was in error because, according to the report, he did not refer to the specification for the correct configuration and he made an incorrect assumption that if the system passed FAT it must have been correctly configured. Because of the improper configuration, the DP system interpreted inputs from the chief officer as full pitch ahead, resulting in the uncontrolled and unexpected movement of the vessel. The investigation report noted that the test of the new system was poorly planned and

managed. The investigation report stated that, "No consideration was given by any of the involved parties to the risks of connecting an unproven control system to rotating propulsion plant. No trial prerequisites were considered, so no effective control measures were imposed." For example, the emergency stop systems were not tested prior to the operation, and those systems did not operate effectively when needed.[10]

The software and system must start up and shut down in known, safe states: Yakima Blade Mill; Afghanistan Friendly Fire

When starting software and computing systems, the hardware should be in a state understood by the operators. For example, fill valves may be automatically opened on a storage tank when the software starts, and analysts should determine whether this is a desired state. Shutdowns should be orderly and controlled, and the system should be designed so that abrupt shutdowns (which may occur with power interruptions) do not hinder the safety of the system.

On January 8, 1997, an employee of Central Pre-Mix Concrete in Yakima, Washington was fatally injured when the blade mill in which he was working inadvertently started up. In this operation, sand and gravel were transported by over-the-road trucks from an off-site pit to the plant for washing and screening. The finished product was sold, or used to supply the company's ready mix operation located adjacent to the wash plant. The mill was used to pre-condition aggregates and was over 22 feet long, 7 feet wide, and 10 feet high. The blades on the mill were 3 feet in diameter. The mill was controlled by a PLC. At the time of the accident the employee was thawing frozen material inside the blade mill and replacing broken and worn paddle tips. Other workers were replacing a faulty breaker while this employee worked inside the mill. In the week prior to the accident an internal heat problem had caused a breaker to trip, resulting in loss of power to the plant components. The faulty breaker controlled a number of smaller breakers, including those for PLC power. The faulty breaker was replaced and reset while the employee was in the mill, and resetting the breaker caused the PLC to power up. Unfortunately, the PLC was programmed to start up in a mode that ran the blade mill. The PLC logic had been modified in October 1996 to address issues associated with power losses so that the mill would not entirely shut down during power surges and loss. However, this modification resulted in power being unintentionally returned to components after any restart, whether for power failure or for maintenance. The employee was fatally injured from the impact of the large blades after the inadvertent startup.[11]

On December 20, 2001, U.S. Army Special Forces soldiers were killed and 20 others injured in a "friendly fire" incident during the war in Afghanistan. At the time of the accident the Special Forces unit had called for an air strike on a Taliban outpost near Kandahar. To assist in the air strike the combat controller used a Precision Lightweight Global Positioning System (GPS) Receiver, also known as a "plugger." The plugger calculated the latitude and the longitude of the enemy position, which could then be relayed to the aircraft. Minutes prior to the air strike, the controller had used the plugger to obtain the coordinates of the Taliban outpost and had relayed that position to the bomber. Then, just before the B-52 was to arrive to drop a satellite-guided bomb, the batteries died on the plugger. The controller quickly replaced the batteries, but unbeknownst to him the plugger started up again not with the coordinates of the Taliban position but with the coordinates of his own location. He relayed the position a second time to the B-52, as was required before a strike, but this time with the wrong coordinates. The bomb was then dropped, striking the Special Forces unit.[12]

3.2.2 Hardware and Environment Considerations

It can be difficult to identify all hardware and environmental hazards that can be associated with computing systems. The following are examples of some frequently overlooked hazards.

Hazards related to induced environments should be considered: Galileo Spacecraft

Induced environments are those stresses put on the system as it operates. Induced environments can include shock, vibration loads, impact loads, pressure, and heating. These induced environments can create additional hazards that must be addressed. For software and computing systems some unique considerations include noise related to electronic inputs, radiation effects from other hardware, and system effects that could cause computer hardware damage such as those above.

On October 18, 1989, the Galileo spacecraft was dispatched from the Space Shuttle Atlantis on its mission to explore Jupiter. In system level testing prior to the mission, the spacecraft software identified multiple Attitude and Articulation Control Subsystem (AACS) checksum errors that were not indicative of actual errors. A checksum is a fixed size data value computed from a block of data; a checksum is used to detect errors during data transmission or storage. These anomalous checksum errors were caused by both AACS memories placing data on the data bus at the same time (bus contention). The bus contentions were caused by electromagnetic coupling within the AACS intra-subsystem harness while simultaneously accessing both AACS memories. The data were inducing noise on the address lines which caused the software to think that there were memory conflicts. The investigation noted that the problem probably should have been caught in subsystem testing, but the simulators used in the subsystem tests were of limited fidelity and therefore were unable to find the problem. The report suggested that the impact of simulator limitations should be thoroughly understood and documented.[13]

Hazards related to natural environments should be considered: Birgenair 301

Natural environments can be the source of a number of problems that must be considered. Natural environments can include humidity, wind, radiation, temperature, lightning and even flora and fauna. Although most natural environments may be known, the ability of systems to withstand these environments may be overestimated. Consideration should be given not only to extreme environments but combinations of normal environmental conditions. Natural environments can have direct and indirect effects on computing systems.

On February 6, 1996, Birgenair flight 301, a chartered Boeing 757-225, crashed into the sea shortly after takeoff from Puerto Plata's Gregorio Luperón International Airport in the Dominican Republic. All 189 persons on board died in the crash. During takeoff the captain noted that the Air Speed Indicator (ASI) was not working properly. The captain decided to continue the flight because the co-pilot's ASI was functional. While climbing, the ASI read 350 knots, causing the autopilot and autothrottle to react by increasing the pitch-up attitude and reducing power to try to slow the aircraft (the actual speed at this time was about 220 knots). The crew then started to observe contradictory warnings from the flight control system, receiving rudder warning and excessive airspeed advisories followed by a stick shaker warning. The crew finally realized that the ASI readings were unreliable, and that the autopilot was slowing them to a stall condition. The crew disconnected the autopilot and applied full thrust, but the actions were not enough to prevent the airplane's impact with the water. It is believed that the incorrect air speed readings were the result of obstructed Pitot tubes. The aircraft had been left outdoors for at least 20 days prior to flight. It is believed that in that time an insect called the black and yellow mud dauber wasp had nested inside the Pitot tubes, preventing them from functioning properly. Without the Pitot tubes the flight software could not perform accurate calculations of air speed. The accident report stated that the probable cause was the crew's failure to recognize the activation of the stick shaker as a warning of imminent entrance to the stall, and the failure of the crew to execute the procedures for recovery from the onset of loss of control. The nonfunctioning Pitot tubes were identified as a contributing factor.[14]

The analysis should address whether the software properly handles spurious signals and missing data: Northwest Airlines 1142; Limerick Generating Station

Spurious signals and lost data can occur throughout operations, often due to mechanical failures, electrical failures, and poor data transmission. Recovery from such transients should be addressed. For example, a transient input such as a voltage spike could result in modified or unexpected system inputs, or missing data could result in an unexpected mode change, and the software must detect and recover from such inputs.

On April 27, 1995, Northwest Airlines flight 1142 experienced an uncommanded roll during a flight from Detroit, Michigan to Baltimore, Maryland. No one was injured in the incident, and the pilots were able to maintain control and land safely following the incident. On April 28, 1995, the same airplane, operating as Northwest Airlines flight 115, also

experienced an uncommanded roll following takeoff from Minneapolis-St. Paul International Airport. Again, no injuries were reported and the crew was able to control the aircraft and land safely. The airplane was an Airbus A320-211 aircraft which used a flight control system that featured fly-by-wire technology. Inputs from the flight crew were transmitted electronically rather than mechanically to flight control surface actuators. The aircraft was equipped with two sidesticks in the cockpit for control. The sidestick movements were detected with potentiometers linked to the flight control computers. Testing after the incident showed evidence of short-duration voltage spikes when the sidestick was in the neutral position. The voltage spikes were the result of loss of contact between the wiper and the track inside the potentiometer. The loss of contact was in turn due to a build-up of lubricating grease in the track. This voltage spike was interpreted by the Elevator Aileron Computer (ELAC) as a command for an airplane roll. Longer duration spikes would have triggered a fault, and the system would have switched to the backup ELAC. A service bulletin was written to modify the software to identify a voltage spike as an invalid command to minimize the likelihood of an uncommanded roll. In addition, a service bulletin was written to add a resistor to the sidestick potentiometer to reduce the magnitude of the voltage spike.[15]

On April 20, 1999, reactor Unit 1 at the Limerick Generating Station in Limerick County, Pennsylvania experienced a loss of feedwater. As a result the reactor was scrammed and the reactor was shut down. The plant safety system responded as designed to a loss of feedwater flow. The loss of feedwater was the result of a series of events. A breaker that supplied position indications to the inlet valves of the Deep Bed Condensate Demineralizer System (DBCDS) experienced spurious tripping. The logic in the DBCDS PLC interpreted the spurious signal as a loss of position indication on the valves, and the PLC was programmed to close all outlet flow control valves in this condition. Closing all outlet valves led to an increase in differential pressure across the deep bed demineralizers, which should have led to the opening of bypass valves to relieve the pressure. However, the pressure rise was so rapid that the bypass valves could not open, and the valves experienced a thermal overload. With no water flow, all three reactor feed pumps tripped and ceased to function. The loss of the pumps led to a drop in the reactor water level, critical to protection of the reactors. The water level eventually fell to a point where an automatic scram was initiated and all control rods were inserted. After this incident the breakers causing the spurious signal were replaced and the PLC logic was modified to allow the outlet flow control valves to remain open in the event of spurious inputs.[16]

The analysis should address how the software recovers from power outages and power transients: Scandinavian SK 682

Power loss and electronic circuit failures should not result in an unsafe system state. The software should assure that the system reverts to a safe state, and the software should be designed so that the system recovers to a safe state when power is restored.

On July 13, 2003, Scandinavian Airlines flight SK 682 traveling from Rome to Copenhagen was forced to make an emergency descent due to a sudden drop in cabin pressure. The crew followed correct emergency procedures and made a successful emergency diversion to Zurich. No significant injuries occurred to passengers or flight crew. The Swiss Air Accident Investigation Bureau stated that the loss of cabin pressure was due to an internal short circuit in a computer which led to a shutdown in the air conditioning system. The short circuit was due to a combination of power transients that culminated in the failure of the circuit. This short circuit led to an incorrect ground discrete signal being sent to the Aircraft Condition and Reporting System Management Unit (ACARS MU) auto shutdown function. This incorrect signal was received by ACARS software, which then commanded the left and right air-conditioning system pressure regulator valves and flow control valves to be closed, resulting in the cabin pressure drop. The report stated that, "This failure occurred due to an insufficient risk analysis being performed prior to the MD-80 ACARS MU Mini ACMS [Aircraft Condition Monitoring System] wiring installation."[17]

Margins must be adequate on memory, CPU usage, etc.: Browns Ferry Nuclear Power Plant Unit 3

Unexpected data rates or throughput have caused numerous problems. Margins should be defined and evaluated throughout development and operation. The system and software should be designed to prevent

performance degradation caused by factors such as memory overload and counter overflow. Testing and monitoring for memory leaks should be conducted to identify the potential for unexpected behavior. Load testing should be performed to determine if the system can be overloaded with data.

On August 19, 2006, operators at Browns Ferry nuclear power plant in northern Alabama manually scrammed Unit 3 after the loss of both reactor recirculation pumps. The operators were following proper procedures; loss of the pumps meant that water flow to cool the reactors was reduced, potentially leading to reactor instability. Investigation into the incident found that Variable Frequency Drive (VFD) controllers for the pumps were unresponsive. In addition, the condensate demineralizer primary controller (essentially a water softener for the nuclear plant) had failed at the same time as the VFDs. The condensate demineralizer primary controller was a programmable logic control system connected to the Ethernet-based plant computer system network. The VFDs were also connected to this same network. The VFD and the condensate demineralizer controller utilized company proprietary software and were microprocessor based. Investigators at the plant determined that the root cause of the shutdown was excessive traffic on the network. Investigators believed that the condensate demineralizer controller malfunctioned and began spewing excessive data over the network. The excessive traffic essentially locked up the system, resulting in unresponsive controllers; these conditions were repeated in tests following the shutdown. As stated by the U.S. Nuclear Regulatory Commission, "all network devices must allocate time and resources to read and interpret each broadcasted data packet, even if the packet is not intended for that particular device. Excessive data packet traffic on the network may cause connected devices to have a delayed response to new commands or even to lockup, thereby, disrupting normal network operations." The company instituted network firewalls to limit traffic following the incident.[18]

Memory management should be addressed: Mars Rover Spirit

Memory modifications may occur due to radiation-induced errors, uplink errors, configuration errors, human errors, or other causes. Inadvertent memory modification can produce unexpected results. Computing systems should be designed to detect inadvertent memory modification and recover to a known safe state.

The National Aeronautics and Space Administration (NASA) Mars Exploration Rovers, Spirit and Opportunity, landed on Mars on January 4 and 25, 2004, respectively. On January 21, 2004, Spirit abruptly ceased communications with mission control. When contact was re-established, mission control found that Spirit could not complete any task that requested memory from the flight computer. Examination of the problem showed that the file system was consuming too much memory, causing the computer to reset repeatedly. The root cause of the failure was traced to incorrect configuration parameters in two operating system software modules that controlled the storage of files in memory. Effects of overburdened memory were not recognized or tested during ground tests. Mission operations personnel recovered Spirit by manually reallocating system memory, deleting unnecessary files and directories, and commanding the computer to create a new file system. Although the rover was recovered, the malfunction took 14 days to diagnose and fix, thereby compromising mission objectives. A post-anomaly review showed that memory management risks were not understood. In addition, schedule pressures prevented extensive testing and understanding of software functions.[19]

3.2.3 Personnel and Organizational Considerations

The effects of personnel and organizational issues may be overlooked in the hazard identification process. The following are examples of lessons learned in human and organizational hazard causes.

Hazards related to human-software interfaces should be considered: Air Inter 148; Regina Ethane Release

Humans interact with hardware and software in a variety of ways. Therefore, hazards related to human-software interactions must be considered. In determining whether the system is robust, consideration should be given to the following factors:

- The system demands must be compatible with human limitations
- People can perform tasks reliably under adverse conditions
- Humans are kept informed of the system conditions
- Humans can readily take actions in abnormal situations
- The system can recover from human error

The human-computer interface must also be carefully considered. In particular, the Graphical User Interface (GUI) can be a significant source of hazards. Thought should be given to the amount of information provided to an operator to avoid information overload. Some specific considerations include the following:

- Computer systems should minimize the potential for inadvertent actuation of hazardous operations.
- Only one operator at a time should control safety-critical computer system functions.
- Software should provide confirmation of valid command entries, data entries, or both, to the operator.
- Software should provide feedback to the operator that indicates command receipt and status of the operation commanded.
- Safety-critical commands which require several seconds or longer to process should provide a status indicator to inform the operator that processing is occurring.
- Software should provide the operator with real-time status reports of operations and system elements.
- All messages, including error messages, should be unambiguous, and unique error messages should exist for each type of error.
- The system should ensure that a single failure or error cannot prevent the operator from taking actions to put the system in a safe state.
- The system should not inadvertently bypass operator control of safety-critical functions.

On January 20, 1992, Air Inter flight 148 crashed in the Vosges Mountains while circling to land at Strasbourg-Entzheim Airport in France. A total of 87 people of the 96 on board were killed in the crash. The aircraft, an Airbus A320, was equipped with an autopilot system that was new to the pilots flying the aircraft. The accident investigation found that the airplane crashed because the pilots had left the autopilot in the incorrect mode and had set an improper descent angle. The autopilot was equipped with Heading mode and Vertical Speed mode. On approach the pilots mistakenly entered a value of "-33" for a 3.3 degree angle of descent and put the system in Vertical Speed mode when it should have been in Heading mode. In Heading mode this incorrect input would have corresponded to 800 feet/minute rate of descent, but in Vertical Speed mode it meant a descent rate of 3300 feet/minute. The mountains were in clouds above 2000 feet, so it is unlikely that the flight crew would have recognized their rapid descent. The aircraft was not equipped with a Ground Proximity Warning System (GPWS) so the crew did not receive any warning before the aircraft impacted the mountains. It was not known why the crew did not recognize their errors, but recommendations following the crash included modifications to capture and warn the pilots of such errors.[20]

On May 10, 1994, a release of ethane occurred in the Cochin Pipe Lines Ltd. System near Regina, Saskatchewan, Canada. The ethane release led to a fire that destroyed communication links between the Regina Division terminal and the Amoco Canada Petroleum Company Ltd. control center in Fort Saskatchewan, Alberta. The fire was allowed to self-extinguish. The TSB determined that the probable cause was a lack of preventative maintenance on a densitometer pump that resulted in bearing wear and damage to the containment shell which then ruptured. The pump ensured that an adequate flow of product passed through the densitometer; the pump had not been inspected or maintained for five years. At the time of the ethane release, alarm messages had been issued at the control center in Fort Saskatchewan, but these were interpreted as problems with a PLC. Such PLC problems had been seen in the past, so the alarms were ignored. In addition, the Supervisory Control and Data Acquisition (SCADA) system was not operating properly. A display window of the Pipeline Model Application System (PMAS) software component was not scrolling

forward as new alarms were generated, and the audible alarm function of the SCADA system had been disabled. The PMAS was designed to assist in leak detection, running mathematical simulations of the pipeline. The PMAS displayed trend data critical for identifying leaks and other problems. Because the display window was not scrolling forward properly, trend data were not available and the operators were unaware of a problem. The reason for the loss of scrolling could not be determined. The SCADA alarm system had been disabled because many changes were being made to the system and operating conditions at the time of the accident, and those changes would have led to many nuisance alarms.[21]

Hazards related to schedule and cost pressures should be considered: V-22

Schedule and cost overruns may lead to a number of conditions with potentially hazardous consequences, including the following:

- Reduced testing
- Shortcuts in the development process
- Reduced system functionality
- Stressed employees who make mistakes or who may leave the effort entirely
- Programs deployed with unknown problems

NASA has found that a number of mission failures resulted in part from highly complex systems developed on short timelines. Cost and schedule are always factors in the development of complex systems, but they must be balanced with safety.

On December 11, 2000, four Marines were killed in the crash of a V-22 aircraft near Camp Lejeune, North Carolina. According to published reports, the left-hand nacelle titanium hydraulic line developed a leak during flight. As a result, the hydraulic line reservoir was depleted, and the V-22 experienced a total loss of the hydraulic system. The pilots responded to the master alert and primary flight control system annunciation by depressing the flight control alert and resetting the flight controls. When the primary flight control reset button was pressed in accordance with established procedures, a software anomaly caused significant pitch and thrust changes in both prop rotors. These fluctuations resulted in decreased airspeed; reduced altitude; and incorrect pitch, roll, and yaw motions that eventually accompanied increasing rates of descent and angle of attack. Essentially, the vehicle swerved out of control, stalled, then crashed. The crew had pressed the reset button to reset the primary flight control logic 8 to 10 times in an attempt to reset the system and regain control during the emergency. However, each primary flight control system reset aggravated the situation until the aircraft entered a stall condition. The software anomaly was found in a previously overlooked path in the flight control laws associated with the propeller rotor governor. This anomaly allowed large torque and RPM changes to be introduced when multiple failure conditions existed. These anomalies, attributed to software, were due in part to inadequate testing. In its report to the Secretary of Defense in its review of the V-22 program, the GAO noted that, "Our review of the V-22 program, which is already in low-rate initial production, revealed that the Department planned to proceed with a full-rate production decision without knowing whether new technology could meet Marine Corps requirements; whether the design would work as required; or whether the design could be produced within cost, schedule, and quality targets. This knowledge is lacking in part because of inadequate test and evaluation. Specifically, developmental testing was deleted, deferred, or simulated in order to meet cost and schedule goals."[22, 23]

Hazards related to organization and management issues should be addressed: *Ever Excel*

Organization and management issues have been identified as factors in many accidents. Even well-designed systems and seemingly effective hazard controls can fail under political, economic, and operational pressures. In addition, changes in management can lead to an increase in risk as critical functions and oversight are modified or eliminated. An organization's safety culture therefore plays a key role in assuring that risk is reduced to an acceptable level. A failure to consider these factors may result in an incomplete analysis and a misunderstanding of the risk. Some specific considerations include, but are not limited to, poor planning, improper goals and objectives, poor communication among team members,

inadequate record keeping, insufficient human resources, poor risk decision making, poor scheduling, failure to investigate incidents, hazard controls relying only on procedures, and inadequate test scheduling and resources.

On April 21, 2010, the chief engineer on the container ship *Ever Excel* died when he became trapped between the top of the ship's passenger lift and the edge of the lift shaft. According to the MAIB, at the time of the accident the ship was undergoing a routine compliance inspection in Kaohsiung, Taiwan. The second engineer was unable to open the lift shaft doors to complete the inspection. The chief engineer tried to solve the problem and entered the lift car, climbed through an escape hatch, climbed on top of the lift car, and closed the hatch. The second engineer incorrectly believed that the chief engineer had set the controls to manual mode to take control of the lift car. Therefore, the second engineer released the emergency stop button then turned the reset key attached to the lift door. By closing the emergency hatch door the chief engineer had disabled the first safety barrier, an interlock that would not allow the lift to operate with the door open. The second engineer removed the second safety barrier, the emergency stop, by releasing the emergency stop and resetting the system. As a result, the lift returned to its normal automatic operating mode, and the lift automatically moved upwards, trapping and asphyxiating the chief engineer. The MAIB report noted that the crew had failed to follow manufacturer-suggested procedures in performing lift maintenance. The report also stated that the crew was unable to release the chief engineer after the accident and damaged the lift because they had not practiced emergency operation of the lift. In addition, the report identified a weak safety culture in the organization, stating, "It was evident that completing the task was considered more important than working safely." The report went on to state that communications were poor, risk assessments were not completed, there was little feedback provided to the crew on safe procedures, the company did not make use of previous accident and incident reports, and auditing was ineffective.[24]

Hazards related to communications and training should be considered: Knoxville Pipeline Rupture

Accidents can result because people may not be provided with critical information, the information might be sent at the wrong time, the information could be provided to the wrong person, or the recipient may misunderstand what is being communicated. Miscommunication is more likely when distractions and irrelevant information are present, such as under a stressful emergency situation. The potential for poor communication must be addressed and mitigated. Training is critical to help anticipate and respond to abnormal conditions. Training is also critical to assure that operators understand computer systems and displays. Poor training can result in a failure to respond properly to both nominal and off-nominal conditions.

On February 9, 1999, a pipeline operated by Colonial Pipeline Company carrying diesel fuel ruptured in Knoxville, Tennessee. Approximately 53,550 gallons of diesel fuel were released from the rupture. At the time of the leak the pipeline controller was using a SCADA system to operate the 10-inch diesel pipeline, and he had just completed a delivery of fuel. The SCADA system recorded a loss in pressure of 19 psi but did not issue an alarm because this pressure drop was not significant enough to trigger an alarm. The controller started another delivery when he noticed that the meter indicated no flow and did not show a normal pressure rise. The controller decided to restart the pipeline without first communicating with senior officials. After 4.5 hours the local fire department reported the leak to the operator and the controller shut down the diesel flow. The NTSB stated that the probable cause of the accident was pipe rupture due to environment-induced cracking and low fracture toughness of the pipe. Contributing to the severity of the accident was the failure of the controller to determine from the SCADA system data that a leak had occurred, and his actions to restart the pipeline. After the accident the company updated its operating procedures and training with regard to pipeline anomalies. The company also improved its automated leak detection system.[25]

3.2.4 Hazard Identification Process Considerations

The hazard identification process itself may be flawed, leading to a hazardous condition. The following are examples of concerns in the hazard identification process.

The hazard analysis must be updated as the design changes: Joseph M. Farley Nuclear Plant

Engineering by its very nature is an activity that requires change. As the development life cycle proceeds, the design often changes and new hazards are uncovered and some hazards may no longer be relevant. If the hazard analysis is not updated as development proceeds then resources may be expended on previously identified hazards that may no longer be relevant, and new hazards may not be discovered as the design matures. An organization should update hazard analyses at all major reviews and when significant system changes occur.

On May 28, 2000, reactor Unit 1 at the Joseph M. Farley Nuclear Plant near Dothan, Alabama experienced an automatic scram when the steam generator level reached a low level set point. All safety systems functioned as designed during the event, according to the report. The root cause of the event was traced to a software change made in 1995. The 1995 software change was made to the Digital Electrohydraulic Control (DEHC) system to reduce the vulnerability of that system to momentary electrical spikes and transients. Originally, the software automatically reset the valve position to zero following a turbine trip from electrical transients if the system was operating in manual mode. In the 1995 change, that function was removed so that the valves remained in their previous position following power outages. However, this change was not properly documented. In the event in 2000, operators manually shut down a turbine, and they expected that the turbine would coast to a stop. However, the turbine slowed to 20 RPM and stabilized. The operators decided to cycle the turbine governor valves to 3% open and then close them again to stop the turbine. To cycle the valves manually, the operators switched the DEHC system from Operator-Auto mode to Turbine Manual Mode. According to procedures, this mode should only have been selected if Operator-Auto mode failed for some reason, but personnel did not follow this procedural guidance. The operators opened the valves 3% then they manually shut down the turbine. Because of the previous software change, the valves remained open at 3% even though the operators thought they were completely closed. The ultimate outcome was that the steam flow was greater than the feed flow, and this mismatch led to the reactor shutdown.

The investigation found that the cause of the Joseph M. Farley Nuclear Plant incident was personnel error as a result of 1) a valve cycling operation performed with inappropriate procedural guidance with regard to the operating modes of the system, and 2) a prior software change made to the turbine control system with inadequate documentation describing the design change and its operational impact. Without the documentation, personnel preparing operator procedures and training did not recognize that system operation was affected by the change. Without such procedures and training, operations personnel made inappropriate decisions that led to the scram. Corrective actions included improved documentation, procedures, and training.[26]

The hazard analysis must be updated as the processes or procedures change: *Collaroy*

As systems become operational, changes are often made in processes and procedures. Operational modifications may not only change the nature of the known hazards, but they can also introduce new hazards. The hardest to control are those procedural changes that are gradual. Small changes may be made to the operation as more is learned until the resulting processes (and associated hazards) are much different than those originally envisioned. If the reasons behind the original procedures are not documented, then changes may be made to processes or procedures without understanding the potential to increase risk. Note that operational changes can include change to personnel, contractors, and management structure.

On March 4, 2005, the ferry *Collaroy* operated by the Sydney Ferries Corporation collided with a wharf in Sydney Cove, Australia. There were no passengers on board at the time, and the crew was not injured. The ferry received minor damage, but the backboards at the wharf were extensively damaged. The collision occurred when the master of the vessel was not able to stop the ferry. The primary control of the propulsion system failed,

and backup systems were inoperative as well, leading to loss of control. Warning systems were also inoperative. *Collaroy* was equipped with a propulsion control system that relied on four PLCs. Two PLC units were assigned to each propeller such that there was always one main PLC and one backup. The system was designed such that if one of the PLCs failed the control system automatically switched over to the backup unit. At the time of the loss of control, one of the PLCs had failed because of a failed electronic circuit in a logic card on the PLC. The system should have then reverted to the backup PLC. However, the Australian Office of Transport Safety Investigations team discovered that the backup PLCs were not turned on. Because the backup systems were not turned on there was no warning alarm of a PLC failure. Without a backup system, the propulsion control was lost upon failure of the primary PLC. The ferry company informed the investigators that prior to the accident they had experienced faults in the PLC electronic card circuitry. These faults occurred because of repeated cycling from being turned on and off, which heated up the circuits. These faults resulted in loud warning alarms, which became a nuisance to the crew. Therefore, the crew modified the procedures, using an alternate PLC startup procedure. But according to the investigation report this alternate procedure may have fooled the crew into thinking that the backup units were on when they had not started. The report called for improved risk assessments of the propulsion control system and additional crew training for emergency situations.[27]

The hazard analyses should consider software requirements as a cause: Mars Polar Lander

System failures can often be traced to incorrect and incomplete requirements and specifications. Often key requirements needed for safety are omitted because of misunderstandings in how the system operates. A significant effort should be in place to assure that requirements are clear, correct, complete, consistent, and testable. Requirements come from many sources, and may be technical or process safety requirements; technical requirements describe what the software is supposed to do while process requirements speak to the way in which software is developed. Technical requirements can be functional requirements, data requirements, and interface requirements. Technical software safety requirements should include the modes of operation under which they are valid, and any modes in which they are not applicable. The assumptions used in the requirements and design should also be clearly stated; a design that is based upon unrealistic or optimistic assumptions may be high risk. Requirements analysis such as traceability analyses and criticality analyses should be part of an effective requirements management process.

The Mars Polar Lander (MPL) spacecraft was launched on a mission to the planet Mars on January 3, 1999. Upon arrival at Mars, communications ended according to plan as the vehicle prepared to enter the Martian atmosphere. Communications were scheduled to resume after the Lander and the probes were on the surface. However, repeated efforts to contact the vehicle failed, and eventually the program managers declared the spacecraft to be lost. The cause of the MPL loss was never fully identified, but the most likely scenario was that the problem involved deployment of the three landing legs during the landing sequence. Each leg was fitted with a Hall Effect magnetic sensor that was designed to generate a voltage when the leg contacted the surface of Mars. A command from the flight software was to shut down the descent engines when touchdown was detected by this sensor. The MPL investigators believed that when the landing legs deployed from the spacecraft, software interpreted spurious signals from the motion of the vehicle as valid touchdown events. The software, upon receiving these signals, then prematurely shutdown the engines at 40 meters above the surface of Mars, and the spacecraft crashed into the surface of Mars.

Prior to launch designers knew of a possible failure mode whereby the sensors would falsely detect that the vehicle had touched down. However, the software requirements did not account for this failure mode and therefore the software did not ignore spurious signals prior to landing. Although the MPL failure report noted that the verification and validation program was well planned and executed, the report also stated analysis was often substituted for testing to reduce costs. Such analysis may have lacked adequate fidelity to identify this system failure scenario. Also, the touchdown sensing software was not tested with the Lander in the flight configuration. The MPL investigators specifically recommended that system software testing include stress testing and fault injection in a suitable simulation environment to determine the limits of capability and search for hidden flaws.[28]

3.2.5 Software-Specific Considerations

There are a number of concerns specific to the use of software that should be considered in hazard identification. The following are examples of those software-specific concerns.

Analyses should consider what happens if the software locks up: Bellingham Pipeline Rupture

Software freezing and locking up the computer can lead to unexpected consequences. The hardware should be in or revert to a known, safe state if this occurs. For example, an analysis may determine that drain valves could automatically close when the software locks up, leading to overfilling conditions in a chemical processing system. Consideration must be given to fault management of such situations. Watchdog timers are sometimes used to trigger a system reset if the main program fails to perform some action. Note that non-safety-critical code may cause the computer to lock up, and that can still be a concern if safety-critical software resides on the same processor and therefore becomes unusable because the system is not functional.

On June 10, 1999, a 16-inch-diameter pipe carrying gasoline ruptured in Bellingham, Washington. The ruptured pipeline, owned by Olympic Pipe Line Company, released approximately 237,000 gallons of gasoline into a nearby creek, according to the NTSB. That gasoline then ignited, burning approximately 1½ miles along the creek. Three people died in the fire, including two 10-year-old boys playing alongside the creek. One home and the Bellingham water treatment plant were also damaged in the accident.

The NTSB investigated the accident and determined that the probable cause of the rupture was damage to the pipe during a modification project performed in 1994. This damage weakened the pipeline, making it susceptible to rupture under increased pressure in the pipe. The NTSB also stated that inspections of the pipeline during the project were inadequate and did not identify and repair damage. The report noted that in-line pipeline inspection data should have prompted the company to excavate and examine that section of pipeline, but the company failed to perform such work after reported anomalies. The SCADA system computers also played a role in the accident. The SCADA system was used for operation of the pipeline, for example to open and close valves remotely as required or to operate pumps as needed. Just prior to the accident the operator was preparing to initiate delivery of gasoline to ARCO's Harbor Island terminal in Seattle, diverting delivery from another facility. During the process of switching delivery destinations, the pressure in the pipeline began to increase, which is a normal condition but one that required the operator to start a pump to reduce pressure. However, when the operator tried to start that pump, the SCADA system failed to execute the start command issued by the operator. The operator soon found that the SCADA system was unresponsive to any commands, something that had never happened before. The report stated that, "Had the controller been able to start the pump at Woodinville, it is probable that the pressure backup would have been alleviated and the pipeline operated routinely for the balance of the fuel delivery." Instead, the pressure in the pipe increased, and the increased pressure likely caused the damaged pipe to rupture.

The cause of computer system failure was likely a change made to the system database just prior to the accident. The NTSB accident report stated that the SCADA system administrator entered new records into the live database at the time of the accident. The system administrator however did not check the records or test the system software to see if those changes introduced any problems. The computing system problem could not be replicated after the accident and therefore the cause of the anomaly could not be definitively identified. The report stated, "The Safety Board concludes that, had the SCADA database revisions that were performed shortly before the accident been performed and thoroughly tested on an off-line system instead of the primary on-line SCADA system, errors resulting from those revisions may have been identified and repaired before they could affect the operation of the pipeline."[29]

Mode transition should be considered: Tarom 381; *West Navion* Drilling Ship

Transitioning to the correct software mode is often a critical part of the system operation. There are a number of different definitions in the literature, but for the purposes of this book a *mode* is one of several alternative conditions or methods of operation of a device. Modes can be thought of as mutually exclusive sets of behavior. For example, software on a weather satellite may have one mode for launch, another for on-orbit science operations, and another safehold mode when anomalous conditions are identified, each mode with different functions and responses to input. If a mode change is required as part of the operation, then the analyst should show failure to transition modes or inadvertent mode change as a hazard sub-cause. The analyst should also specify the "triggers" that will allow the software to make that mode change, and provide the level of fault tolerance on those triggers and their independence. For example, sensor limits may result in a predefined switch from automatic to manual mode in an airplane's guidance system. In addition, computing systems should clearly indicate to the operator when mode changes have occurred, and they should be designed to prevent confusion over which mode the system is in. If the mode change is performed at the wrong time or without notification to the operator then the results may be catastrophic.

On September 24, 1994, Tarom flight 381 traveling from Bucharest, Romania to Paris-Orly Airport in France stalled and went into a dive while preparing to land. The pilots were able to recover and land the aircraft safely. The accident investigation found that the direct causes of the accident were actions by the pilots and a lack of understanding of the automated flight system. As the aircraft approached the runway the pilot noticed that the aircraft was not captured by the glideslope automatically. Therefore, he disconnected the autopilot and kept the autothrottle in operation. The pilot selected a flap position of 20°, and the crew encountered a nose-up effect resulting from increased thrust. In this flap position the aircraft speed was slightly above the level where the system invoked automatic speed protection. This meant that the autoflight system changed from VS mode to LVL CHG mode without informing the pilots. In LVL CHG mode, altitude change was controlled automatically. In this mode the autothrottle commanded maximum thrust and commanded a pitch up to meet that thrust if the altitude selected on the Flight Control Unit (FCU) was greater than that of the aircraft. The pilots had entered an altitude of 4000 feet as the altitude to reach in the event of an aborted approach, and because the aircraft was at 2000 feet, the autothrottle commanded an increase in thrust and pitched the aircraft up to climb to 4000 feet. The pilot then accidentally trimmed the aircraft to its electrical stop at 13° nose up, which resulted in an out of trim situation. The aircraft climbed rapidly and eventually stalled. When the angle of attack sensors were disturbed, the autothrottle automatically disabled and the flight crew was able to regain control of the aircraft. The investigation report noted that the approach was too rapid due to a late start in the descent, which contributed to the accident. Other contributors included inadequate crew resource management, premature selection of the go-around altitude and improper configuration of the slats and flaps (which led to activation of the speed protection), and the crew's difficulty in understanding the functions of the autothrottle and overspeed protection.[30]

On November 10, 2001, the Dynamic Positioning (DP) system on the *West Navion* drilling ship inadvertently changed modes from automatic to manual heading control mode. The DP system determined the position of the ship and allowed it to be properly positioned over a well during operations using navigation satellites and sea-bed transponders to guide it. A helicopter had just landed on the ship and was being refueled at the time of this inadvertent mode change. Neither the pilot of the helicopter nor the ship's crew was aware that this mode change had occurred. The system provided no indication in manual mode that the ship was changing heading. The only indication to the crew that the system was in manual mode was the absence of a heading window on one of the DP screens. The ship began to drift following the mode change, and the force of the ship's movement combined with high winds caused the helicopter to fall over on its side. The co-pilot of the helicopter was standing next to the aircraft when it toppled over, and he was severely injured by flying debris as the helicopter's main rotors broke apart upon impact with the helideck. The helicopter was significantly damaged in the accident. According to the U.K. Air Accidents Investigation Branch (AAIB) report, it is believed that a software error had occurred that led to the loss of heading control, although the investigation could not pinpoint the exact cause. Because the mode change could have been easily implemented with a single operator action via a "one touch" switch, the possibility existed of an operator accidentally initiating a mode change. A similar incident had occurred on October 12, 2001, where the DP had a loss of heading mode change. However, the company had not fully investigated the incident and had not identified the root cause. The company had been considering installation of a "double touch" switch to prevent inadvertent mode changes on the DP system following the October 12 incident, but that modification had not been completed prior to the accident on November 10.[31]

Consideration should be given to the order of commands and out of sequence inputs, commands, and events: CryoSat Satellite

The sequence of commands and inputs can be critical for safety. For example, two-step commanding is a necessary part of the control strategy where human interaction is required. However, the order of commands can matter (e.g., a FIRE command issued before an ARM command should be ignored). The system should be designed to reject out of sequence commands if they could introduce a hazard. Also, the analyst should consider whether command timeouts are needed – if the ARM command is issued, there should be some time limit after which an operator cannot issue FIRE in order to prevent the operator from forgetting about that ARM command. Note that in some emergency or contingency operations two step commanding may not be appropriate – this should be analyzed on a case-by-case basis. In addition, if input from a sensor arrives early, late, or not at all the software should be able to identify and safely handle this situation. For example, the software could be designed to set the earliest and latest time when data are allowed.

On October 8, 2005, a Russian-built Rockot launch vehicle, carrying the CryoSat satellite, blasted off from Russia's northwestern Plesetsk Cosmodrome. Analysis of the telemetry data indicated that the first stage performed nominally. The second stage performed nominally until main engine cut-off was to occur. The second stage main engine failed to shut down at the proper time, and continued to operate until depletion of the remaining fuel. As a consequence, the second stage did not separate from the third stage, and the third stage engine did not ignite. This lack of engine capability resulted in unstable flight, causing the vehicle flight angles to exceed allowable limits. The on-board computer automatically ended the mission at 308 seconds into flight. For the second stage shutdown to succeed, pressurization of the low-pressure tank of the third stage had to have been completed before issuance of the shutdown command. The failure analysis showed that the command to shut down the second stage engine was generated correctly. However, the completion time for the pressurization sequence was erroneously specified; therefore, pressurization completed after the shutdown command was generated. This failure case had not been identified in development and was not tested. No built-in tests existed for the pressurization time.[32, 33]

Analysts must understand the required performance in terms of speed, accuracy, and precision: Saint-Clet Pipe Rupture

A failure to understand required computing system performance may lead to an unsafe condition. For example, performance issues could prevent timely receipt of commands, or there may be a misunderstanding of how often critical information must be updated on a display in order for an operator to make a safety decision. One particular issue that must be considered is data latency. Latency is the time interval during which new information will not change the outputs. This latency interval is affected by the hardware and software design, and may be important if critical information is being updated but the sampling rate is too slow to accept the new information and do something with it. Data age should also be considered – not all input values are valid forever, and data age limits should be assessed.

On December 7, 2002, Trans-Northern Pipelines Inc. was delivering petroleum products from the Montreal, Quebec refining basin to the Ottawa, Ontario terminal storage facility when a pipe ruptured and approximately 32 cubic meters of low sulfur diesel was released to the Saint-Clet, Quebec drainage area. There were no injuries reported. The TSB investigated the accident and found that the rupture occurred because an automatic Cornwall Take-Off Valve (TOV) closed in an uncommanded operation. The closure of the valve created a pressure surge in the line, leading to rupture of the pipe. The pipe that ruptured had pre-existing

cracks, likely from unauthorized third-party construction activity near the pipe. The TSB stated that had the system been equipped with flow control or overpressure protection the pipeline may not have ruptured. The TOV closure and the subsequent high pressure resulted from the command center operator's attempts to maximize flow rates into the station. He had set system parameters to operate near the pressure where the TOV would close automatically and therefore only minor pressure perturbations were needed for the valve to close. Contributing to the accident was the configuration of the SCADA system computers used to operate the system. According to the TSB the SCADA system was gathering system events data at 5-second intervals, but recording events data for trending purposes only at 15-second intervals; therefore, key events were not being recorded. This meant that the operator may not have seen momentary spikes in the pressure and therefore could not identify hazardous conditions. "The discrepancy between SCADA data and observed data means that the SCADA system indicates that operations are conducted in a safe manner while, in fact, the pipeline is being exposed to higher operating stresses that increase the risks of pipe rupture."[34]

Task prioritization, resource contention, and timing should be considered: Mars Pathfinder

Where a multitasking system is employed, tasks must be scheduled and resources must be allocated to tasks based on the scheduling frequency required for the tasks, the criticality of the tasks, and the resources used by the tasks. This must be done in such a way that the response priority is based on risk – the higher the risk the higher the priority. The analyst should determine whether the prioritization scheme can be unexpectedly bypassed or resource contention could be an issue. The analyst should also identify where timing is critical for tasks, and determine the time needed to complete the task. For example, the analyst should determine what happens if data arrives early, late, or not at all. The analyst should ask what could cause time-dependent data commands to be delayed. Timing may also affect error response – some delays may be needed to assure graceful mode transition. Analysts should also identify whether "deadlocks" exist (two or more processes each waiting for the other to release a resource) or whether there are any conditions where a module could corrupt or interfere with other modules or system performance, such as "thrashing" (two or more processes accessing a shared resource). Race conditions should also be considered; a race condition is a sequencing fault where independent tasks execute or initialize data out of sequence.

The Mars Pathfinder spacecraft was launched on December 4, 1996, by NASA on a Delta II launch vehicle. Its mission was to land and deploy the first roving probe on another planet. Mars Pathfinder landed on July 4, 1997, and its rover, Sojourner, began exploration soon thereafter. On September 27, 1997, all communication was lost with Sojourner. Then the system began encountering periodic total system resets, resulting in lost data and a failure to receive ground commands. The problems were due to a software fault known as priority inversion. In priority inversion, a high priority task is waiting for a resource from a lower priority task. On Mars Pathfinder and Sojourner, data and commands to control the devices had to pass through an information bus in the computer. In order to prevent communication conflicts, the data and commands had to take turns. Because some data and commands were more important than others, a priority system was set up of low, medium, and high priority tasks. When a specific task was running the system blocked other tasks from taking over. At one point in the mission a low-priority meteorological task was running, and a medium-priority communications task tried to interrupt. The system performed the correct task prioritization, but at the same time the system also prevented the high-priority information bus task from running because the lower-priority tasks had not completed. After a predetermined time had passed, a watchdog timer went off informing the system that the high-priority bus task had not been running for a while. The system assumed that an error had occurred and initiated a total system reset. The NASA flight team was able to replicate the problem on the ground and uploaded a software patch that solved the problem. NASA had seen similar behavior in pre-flight testing, but the phenomenon was never reproduced, so engineers had assumed it was due to a hardware glitch.[35]

The software-software interfaces should be evaluated: STS-49

Inputs and outputs to software modules should be defined in design documentation. In addition, parameters necessary for status, error handling,

and error recovery should be specified. Interfaces should be designed to minimize failure propagation through design features such as exception handing and parameter validation. Consideration should be given to parameter type mismatch, parameter order mismatch, and parameter number mismatch.

On March 14, 1990, the INTELSAT VI communications satellite was launched on a commercial Titan 3 launch vehicle. However, the second stage failed to separate from the satellite during flight, and the satellite was stranded in the wrong orbit. NASA launched the Space Shuttle *Endeavor* on mission STS-49 on May 7, 1992, to salvage the mission by installing a new second stage motor to enable the satellite to achieve its proper orbit. To do so the Space Shuttle had to first rendezvous with the INTELSAT satellite. The first attempts by the Space Shuttle to rendezvous with the stranded satellite failed. A problem was identified with the software used to calculate the trajectories to perform the rendezvous; the investigators found a precision mismatch in a software library function. Rounding errors in computer software can originate because computers can only represent numbers using a limited number of significant digits. Many computers have the capacity of representing numbers as single or double precision. A single precision can represent data to about 7 decimal places, and a double precision to about 15 decimal places (at the cost of increased computing resources). In the case of STS-49, the software routine used to calculate rendezvous firings, called the Lambert Targeting Routine, failed to converge on an acceptable solution that would allow a rendezvous. This failure occurred because of a mismatch in precision of state-vector variables and the limits used to bound the calculation. The state-vector variables described the position and velocity of the Space Shuttle, and these variables were double precision, while the limit variables were single precision. While the difference was small, the errors introduced by this mismatch were just enough to prevent the algorithm from converging. The flight and ground crews were able to devise a workaround to successfully complete the rendezvous, and the software problem was fixed for later flights.[36]

The analysis should address unused or "dead" code: Ariane 5

Often organizations will reuse software for various projects. The reused software could contain code that is logically excluded from execution so that it will not execute for the new use ("dead" code) and unused but active code (code that can be accessed but is not intended to be used). The use of software containing dead and unused code can introduce unnecessary complexity, and that code could be inadvertently executed. Where possible dead code and unused code should be avoided, but in some cases additional risk may be introduced in modifying the software to eliminate that code. Therefore, the operator should perform an analysis to identify any dead and unused code and then identify risks and mitigations.

On June 4, 1996, the Ariane 5 launch vehicle veered off course and broke up approximately 40 seconds into launch. The vehicle started to disintegrate because of high aerodynamic loads resulting from an angle of attack greater than 20 degrees. This condition led to separation of the boosters from the main stage, in turn triggering the self-destruct system of the launcher. This improper angle of attack was caused by full nozzle deflections of the solid boosters and the Vulcain main engine. The on-board computer software commanded these nozzle deflections based on data received from the active Inertial Reference System. Ultimately, these improper deflections resulted from specification and design errors in the Inertial Reference System software, including improper error handling. The specific error was in a single line of code that attempted to load a 64-bit number into a 16-bit location, causing overflow. Reused software from the Ariane 4 program, including the exception handling code used in the Inertial Reference System, contributed to the failure. The source of the fault occurred in a function that was not required for Ariane 5, but rather was a function carried over from the Ariane 4 software that should not have been used by the system. Therefore, protections were not implemented against an excessive horizontal velocity component input to this function, according to the accident report. No end-to-end tests were conducted to verify that the Inertial Reference System and its software would behave correctly when subjected to the countdown sequence, flight time sequence, and trajectory of Ariane 5.[37, 38]

Hazards related to computer viruses, malware, worms, and other attacks should be considered: Davis-Besse Nuclear Power Plant; Spanair 5022

A computer virus is a computer program that can copy itself and infect a computer by attaching itself to an existing program. Malware is software that can secretly access a computer, but cannot copy itself. A computer worm is a type of malware that is self-replicating but does not attach itself to an existing program. All these intrusions and others can cause unexpected and unwanted computer operation, including disruption of network traffic, corruption of critical files, data loss, and so on. The result can be loss of safety-critical functions, inadvertent commanding, incorrect display messaging, and so on. Computer security tools such as anti-virus and anti-malware software must be used, and processes and procedures must be in place to prevent inadvertent infection of computers.

On January 25, 2003, the Davis-Besse nuclear power plant near Sandusky, Ohio was infected with the Slammer worm. This worm caused a traffic overload on the Davis-Besse computer network. Because of the overload the Safety Parameter Display System (SPDS) was unavailable for almost five hours. The SPDS monitored critical safety systems at the power plant, including coolant systems, core temperature sensors, and external radiation sensors. Those systems were necessary for safe operation of the plant. In addition, the congestion created by the worm affected the Plant Process Computer, another monitoring system. At the time of the incident the plant was offline to fix a problem in the plant's reactor head, and therefore the worm presented no safety concerns. In addition, analog backup systems were available if needed. However, had the plant been in operation then plant personnel may have been overburdened and additional safety concerns may have arisen. The worm had entered the Davis-Besse network first through a contractor's unsecured network, then through a T-1 line acting as a bridge between the contractor and the Davis-Besse corporate network. The T-1 line was later found to bypass the plant's firewall, which had been programmed to block a worm from entering.[39]

On August 20, 2008, Spanair flight 5022, a McDonnell Douglas DC-9-82, crashed on takeoff from Barajas Airport in Madrid on a flight to Gran Canaria, Spain. A total of 154 people on board died, and 18 others were seriously injured. The investigation found that the takeoff took place with the flaps and slats retracted. This was an improper configuration for takeoff, and likely led to the crash. Standard operating procedures included checklists to prepare for takeoff, including setting the wing configuration. It was not clear why the pilots did not complete this checklist, but they may have been interrupted when they had to return the airplane prior to flight to fix a malfunctioning sensor. The pressures to meet schedule may have led to a failure to strictly follow procedures. The investigation found that the electronic system to notify the crew of an improper configuration, the Takeoff Warning System (TOWS), did not activate. A synthetic voice should have notified the crew that the flaps and slats were not properly configured. Although the cause of the TOWS failure could not be determined, authorities suspected that the TOWS may not have activated because the central computer was infected with malware. The malware could have entered the system through a third-party device such as a Universal Serial Bus (USB) flash drive or through a remote Virtual Private Network (VPN) connection.[40, 41]

Hazard analyses must consider errors related to improper calculations and algorithms: GEOSAT Follow-On (GFO) Spacecraft

Errors in calculations and computations can create unexpected results. Calculations can be incorrect for many reasons, including inaccurate requirements and poor implementation. Safety analyses should assure that algorithms and calculations are correct.

The GEOSAT Follow-On (GFO) spacecraft was launched on February 10, 1998, from Vandenberg Air Force Base in California on a Taurus rocket. GFO was a United States Navy program to launch a series of satellites to maintain continuous ocean observation. Once the spacecraft separated from the launch vehicle it began to tumble instead of achieving the correct attitude. Analysis of the motion equations programmed into the vehicle found that the momentum and torque were being applied in the wrong direction in the Attitude Control System (ACS). In the development process a sign on a coefficient had been inverted, leading to the forces being applied in the wrong direction. This ACS sign error was corrected by uplinking modified ACS control loop parameters in a flight software data table. The satellite recovered and was operational following the fix.[13]

Hazard analyses must not only consider software but also data: SAS Charter Flight

Often the safety effort concentrates on the software but does not include the external data provided to that software. External data can come from many sources, and easily-modifiable configuration data are commonly used in software systems. Examples include day-of-launch guidance parameters for a rocket, track layout and car identification information used in a railway command and control system, or piping configuration information used in a chemical plant control system. Data errors include omission of important information, duplicated entries, incorrect labeling, data type errors, value errors, and so on. Data should be subjected to systematic analyses as part of the overall hazard analysis efforts, the implication of data errors must be understood, and the data should be verified.[42, 43]

On December 7, 2003, a charter flight operated by SAS was accelerating on the runway for takeoff from Göteborg/Landvetter Airport in the Gothenburg region of Sweden when the co-pilot noticed that the aircraft's nose was lifting spontaneously without him moving the control column. The co-pilot reported the situation to the commander. The commander took control and aborted the takeoff, then taxied the airplane to the aircraft terminal building. Had the flight continued the crew may have had difficulty controlling the aircraft in flight.

At the terminal building SAS personnel reviewed available information and found that the aircraft loadsheet which specified the placing of passengers did not correspond to where the passengers were actually sitting. The loadsheet stated that the passengers were evenly distributed, but in fact most passengers were near the back of the airplane. This distribution meant that the center of gravity of the aircraft was inappropriate for safe flight, and the aircraft was "tail heavy."

The incident report from the Swedish Accident Investigation Board stated that the inaccurate loadsheet was the result of shortcomings with the company's computerized systems. Prior to the incident the aircraft had arrived from Salzburg with a full load of 180 passengers. However, at Göteborg 59 passengers disembarked while 121 passengers remained; no new passengers were taken on board. The computerized system used by

SAS for creating the loadsheet was called Passenger and Load Control (PALCO). During passenger check-in, passenger seating information was normally sent to PALCO, and then PALCO generated a loadsheet for the pilots to use to determine center of gravity. Information for this flight was supposed to have been sent from Salzburg to PALCO for the follow-on flight from Göteborg. However, this seating information never arrived at Göteborg because the check-in system at Salzburg was not linked to PALCO. Without such information, PALCO used the default value for passenger seating, assuming an even passenger distribution. PALCO did put a message on top of the loadsheet that said "EVENLY DISTRIBUTED," but the crew failed to notice this message, in part because no crew procedures had been implemented to look for such a message. In addition, PALCO sent a warning to the passenger check information screens in Göteborg, but since no passengers boarded the aircraft at Göteborg no one saw this warning. Therefore, the crew initiated takeoff with information on passenger seating and aircraft center of gravity that did not reflect reality. After the incident the PALCO system software was revised to provide an additional warning if seating information is missing. In addition, the software required a two-step user confirmation process before the loadsheet can be printed.[44]

Hazard analyses must consider logic errors: TransAsia Airways 536

Logic errors can create a number of potential safety issues, including commands issued out of sequence, failure to issue command, inadvertent mode change, command sent to the wrong system, command sent when system is in the wrong mode, and so on. Many times such logic errors are actually the result of properly designed behavior executed in unexpected conditions and environments. Analyses and peer reviews should be conducted to try to anticipate and mitigate such logic errors.

On October 18, 2004, TransAsia Airways flight 536 overran the runway upon landing at Taipei Sungshan Airport in Taiwan. The aircraft was damaged, but none of the crew or passengers were injured in the accident. The accident investigation found that the root cause of the accident was the failure of the pilots to set the thrust lever on engine 2 to the IDLE position. During landing the Flight Warning Computer (FWC) normally delivered a message just prior to touch down which indicated that

the crew should move the throttle control levers to the IDLE position to take manual control of the thrust for landing. The pilots had set the autobrake mode to the MED position prior to landing, intending to use the automated system to decelerate the aircraft after touchdown. In this case the crew received the warning to set thrust levers to the IDLE position, but only set thrust lever on engine 1 to this position. The on board computer was configured to only extend the ground spoilers when both thrust levers were set to IDLE; therefore, the ground spoilers were not extended to help slow the aircraft down. In addition, the software logic only engaged the autobraking system when the spoilers were extended; therefore, the autobrake function was not activated, unbeknownst to the crew. Also, the warning to the crew to move the thrust lever 2 to the IDLE position stopped after four warnings due to other actions by the crew. The report noted that there was no other way other than this warning for the crew to know that the thrust lever was in the wrong position, so the pilots did not have enough information to troubleshoot the problem.[45]

3.3 Summary

Hazard identification may be one of the most difficult tasks in the system safety process. But hazard identification is critical to the system safety effort, because one cannot analyze or reduce risks if those risks have not been identified. With respect to software, analysts should first consider the ways that software can contribute to a system hazard. Then, software-specific errors should be considered. Because software may be used in many different kinds of functions, analysts should take care to look for all the ways that software can impact safety. Analyses should include support software, software used in monitoring functions, and models and simulations. Hazard identification requires persistence in gathering information based on similar systems designed and operated in the past, and creativity in trying to look for ways a new system can act in unexpected and potentially catastrophic ways. This is especially true where software and computing systems are used.

References

1. Saipem, "Saipem Sustainability Report 2008," 2008.
2. International Marine Contractors Association, IMCA Safety Flash 18/08, December 2008.
3. Ericson, C.A., *Hazard Analysis Techniques for System Safety*, John Wiley & Sons, 2005, pp. 483-495.
4. National Aeronautics and Space Administration, *Software Safety Guidebook*, NASA-GB-8719.13, March 31, 2004.
5. Federal Aviation Administration, *Guide to Reusable Launch and Reentry Vehicle Software and Computing System Safety*, version 1.0, July 2006.
6. Lutz, R.R., "Targeting Safety-Related Errors During Software Requirements Analysis," *J. Systems and Software*, Vol. 34, Issue 3, 1996, pp. 223-230.
7. Swallom, D.W., "Source-Mechanism-Outcome: A Simple, Yet Effective Hazard Description Model," Proceedings of the 24th International System Safety Conference, 2006.
8. Transportation Safety Board of Canada, "Power Loss – No. 2 Engine Skyservice Airlines Inc. Airbus A330-300 C-FBUS Columbo, Sri Lanka 15 February 2001," Report Number A01F0020, March 25, 2003.
9. U.S. Nuclear Regulatory Commission, Licensee Event Report 287/2008-001, January 7, 2009.
10. U.K. Marine Accident Investigation Branch, "*SBS Typhoon*, Contact in Aberdeen Harbour, 26 February 2011," Report No. 13/2011, August 2011.
11. U.S. Mine Safety and Health Administration, Accident Investigation Report: Fatal Machinery Accident, Yakima - Pre-Mix #6, Central Pre-Mix Concrete Company, Yakima, Yakima County, Washington, January 8, 1997, Mine ID No. 45-00995, 1997.
12. Loeb, V., "Friendly Fire Deaths Traced to Dead Battery, Taliban Targeted, but U.S. Forces Killed," *Washington Post,* March 24, 2002.
13. National Aeronautics and Space Administration, "Design Development Test and Evaluation (DDT&E) Considerations for Safe and Reliable Human Rated Spacecraft Systems," Volume 2, NASA/TM-2008- 215126, April 2008.
14. Flight Safety Foundation, "Erroneous Airspeed Indications Cited in Boeing 757 Control Loss," *Accident Prevention*, Vol. 56, No. 10, October 1999.
15. U.S. National Transportation Safety Board, "Uncommanded roll during cruise, Airbus A320-211, April 28, 1995, Report Number CHI95IA342, 1995.
16. U.S. Nuclear Regulatory Commission, Licensee Event Report 352/1999-003, May 19, 1999.
17. Aircraft Accident Investigation Bureau (Switzerland), Final Report by the Aircraft Accident Investigation Bureau concerning the incident to the McDonnell Douglas DC-9-82 aircraft, LN-RML operated by SAS Scandinavian Airlines System under flight number SK 682 on 13 July 2003, Report No. 1865, September 9, 2005.

18. U.S. Nuclear Regulatory Commission, "Effects of Ethernet-Based, Non-Safety Related Controls on the Safe and Continued Operation of Nuclear Power Operations," NRC Information Notice: 2007-15, April 17, 2007.
19. Reeves, G. and T. Neilson. "The Mars Rover Spirit FLASH Anomaly." Paper presented at the IEEE Aerospace Conference, Big Sky, Montana, March 2005.
20. Job, M., *Air Disaster*, Volume 3, Aerospace Publications, 1998.
21. Transportation Safety Board of Canada, "Ethane Release and Fire, Amoco Canada Petroleum Company Ltd., Regina Diversion Terminal, Mile Post 445, Cochin Pipeline, Regina, Saskatchewan, 10 May 1994," Report Number P94H0018, May 19, 1995.
22. Murray, B., "Corps Cites Software Failure in Osprey Crash." *Federal Computer Week,* 9 April, 2001.
23. U.S. General Accounting Office, "Defense Acquisitions: Readiness of the Marine Corps' V-22 Aircraft for Fill-Rate Production," GAO-01-369R, February 20, 2001.
24. U.K. Marine Accident Investigation Branch, "Report on the investigation into the fatal accident to the chief engineer in the lift shaft on board *Ever Excel* in Kaohsiung, Taiwan on 21 April 2010," Report No 6/2011, May 2011.
25. U.S. National Transportation Safety Board, "Hazardous Liquid Petroleum Products Pipeline Rupture, Colonial Pipeline Company, Knoxville, Tennessee, February 9, 1999," Report No. NTSB/PAB-01-01, March 28, 2001.
26. U.S. Nuclear Regulatory Commission, Licensee Event Report 348/2000-006, June 27, 2000.
27. Office of Transport Safety Investigations, "Collision of the Manly Ferry *Collaroy* Number 3 West Wharf, Circular Quay, 4 March 2005," OTSI File Ref: 03545, November 25, 2005.
28. National Aeronautics and Space Administration, Jet Propulsion Laboratory, Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions, JPL D-18709, 2000.
29. U.S. National Transportation Safety Board, "Pipeline Rupture and Subsequent Fire in Bellingham, Washington, June 10, 1999," Pipeline Accident Report NTSB/PAR-02/02, October 8, 2002.
30. Bureau Enquêtes-Accidents (BEA, France), Report on the incident on 24 September 1994 during approach to Orly (94) to the Airbus A310 registered YR-LCA operated by TAROM, YR-A940924A, February 2000.
31. U.K. Air Accidents Investigation Branch, Report on the accident to Eurocopter AS332L Super Puma, G-BKZE on-board the *West Navion* Drilling Ship 80 nm west of the Shetland Islands on 10 November 2001, Aircraft Accident Report 3/2004, May 2004.
32. Briggs, H., "CryoSat Rocket Fault Laid Bare." *BBC News*, October 27, 2005.
33. EUROCKOT Launch Services GmbH, "CryoSat Failure Analyzed – KOMPSAT-2 Launch in Spring 2006," Eurocket Press Release, December 21, 2005.
34. Transportation Safety Board of Canada, "Refined Product Pipeline Rupture, Trans-Northern Pipelines Inc. 273.1-millimetre-diameter Mainline Kilometre Post

63.57 Near Saint-Clet, Quebec 07 December 2002," Report Number P02H0052, May 10, 2005.

35. Durkin, T., "The Vx-Files: What the Media Couldn't Tell You About Mars Pathfinder," *Robot Science & Technology*, Issue 1, 1998.
36. Goodman, J., "Lessons Learned from Seven Space Shuttle Missions, NASA/CR-2007-213-697, January 2007.
37. Lions, J. L., *Ariane5: Flight 501 Failure Report by the Inquiry Board*, European Space Agency, 1996.
38. O'Halloran, Colin, et al., "Ariane 5: Learning from Failure." Proceedings of the 23rd International System Safety Conference, August at San Diego, California, 2005.
39. Poulsen, K., "Slammer Worm Crashed Ohio Nuke Plant Network," *Security Focus*, August 19, 2003.
40. Comision De Investigacion De Accidentes E Incidentes De Aviacion Civil (CIAIAC), "Accident Involving Aircraft McDonnell Douglas DC-9-82 (MD-82), registration EC-HFP, Operated by Spanair, at Madrid-Barajas Airport on 20 August 2008," Interim Report A-032/2008, August 4, 2009.
41. Meredith, L., "Malware Implicated in Fatal Spanair Plane Crash," *TechNewsDaily*, August 20, 2010.
42. Faulkner, A., "Is It Data or Is It Software," 19th International System Safety Conference, Huntsville, AL, September 2001.
43. Faulker, A., and N. Storey, "Data: An Often-Ignored Component of safety-Related Systems," MoD Equipment Assurance Symposium (ESAS02), 2002.
44. Accident Investigation Board (Sweden), "Uncommanded rotation, Incident involving aircraft LN-RPL at Gothenburg/Landvetter Airport, O County, Sweden, on 7 December 2003," Report RL 2005:20e, September 29, 2005.
45. Aviation Safety Council (Taiwan), "GE 536 Occurrence Investigation Report: Runway Overrun During Landing On Taipei Sungshan Airport, TRANSASIA AIRWAYS FLIGHT 536, A320-232, B-22310, October 18, 2004," Report No. ASC-AOR-06-03-002, 2006.

CHAPTER 4

Hazard Risk Assessment

Progress always involves risks. You can't steal second base and keep your foot on first.

- Frederick B. Wilcox

The only virtue of being an aging risk manager is that you have a large collection of your own mistakes that you know not to repeat.

- Donald Van Deventer

On February 25, 2009, Turkish Airlines flight 1951 crashed while attempting to land at Amsterdam Schiphol Airport on a flight from Istanbul, Turkey. Nine people were killed in the accident. The investigation by the Dutch Safety Board found that as the aircraft was landing the left radio altimeter suddenly indicated a value of negative 8 feet. This erroneous value was passed onto the autothrottle computing system. The autothrottle was part of the automatic flight system and included a computer used to regulate engine thrust by moving thrust levers. The autothrottle received radio height to perform its function primarily from the left radio altimeter. If the left radio altimeter height reading was determined to be unusable, a warning should have been issued and the computer should then have used the right radio altimeter input. However, in this case the software did not respond appropriately, and the computer used the erroneous signal to perform subsequent actions. The safety board found that the software responded to negative input values as "non computed data," and such input automatically activated the "retard flare" mode of the system. In this mode the engine thrust was reduced to idle, causing the