Software and System Safety: Promoting a Questioning Attitude

Terry L. Hardy
Great Circle Analytics, LLC
1238 Race Street
Denver, Colorado, USA
thardy@gcirc.com

Abstract

System safety is an accepted approach to help understand and manage hazards and risks in complex systems in order to prevent accidents. Many different industries use system safety analyses and methods to help reduce the potential for harm to people, property, and the environment. When used correctly, system safety methods can provide tremendous benefits, focusing resources to reduce risk and improve safety in complex systems. Because computing systems are increasingly being used to control critical functions and supply safety decision information, software may directly or indirectly contribute to an accident. Therefore, software must be included as part of an organization's system safety efforts to manage hazards and risks. However, for many organizations, software is not effectively incorporated into the system safety process, and questions are not asked about whether the analyses are appropriate for complex, automated systems. This paper will summarize several accident reports and use those reports to illustrate potential failures in the system safety process with respect to software and computing systems. Lessons learned will be discussed, and some essential questions in software safety will be presented. This discussion is intended to provide insights to help promote a questioning attitude that can improve software safety and system safety efforts.

Keywords: software safety, system safety, lessons learned.

1 Introduction

As more advanced technology and automation are used, transportation systems, energy production systems, medical devices, manufacturing processes, and many other systems continue to increase in complexity. These complex systems create safety risks to their operators and to the communities they serve. System safety is an approach to help manage hazards and risks in complex systems.

System safety is often implemented through a system safety process.

A typical system safety process includes the following components:

- Safety planning
- Hazard identification
- Hazard risk assessment and risk decision making
- Risk reduction and hazard controls
- Risk reduction verification
- Hazard tracking, anomaly reporting, and change management

Of particular concern with regard to system safety are the risks related to software and computing systems. Software and computing systems may be safety-critical if they:

- can cause a hazard (for example, if a software command or automated system can inadvertently create the potential for harm),
- control a hazard (for example, if software is needed to prevent a mishap),
- are used in critical calculations or analyses (for example, output from models and simulations),
- are used to test critical systems.

Software includes computer programs, procedures, scripts, rules, and associated documentation and data pertaining to the development and operation of a computer system. Software can be developed by the organization implementing the system or may be purchased as Commercial Off-The-Shelf (COTS) software. Software safety encompasses not just the software but also the computing system. A computing system includes the software and supporting hardware, sensors, effectors, humans who interact with the system, and data necessary for successful operation. Examples of computing systems include Programmable Logic Controllers (PLC) and Supervisory Control and Data Acquisition (SCADA) systems.

In spite of the fact that software is such an important part of complex systems, the analysis of hazards and risks from software has been inconsistent across industries. Hazard and safety analyses have historically been hardware-focused. Therefore, many analysts may not understand how to incorporate software into their system hazard analyses, and evaluators of those analyses may not understand what should be assessed. Organizations may be focused on compliance to regulations, which often do not address software, and therefore those organizations may not properly assess or mitigate software-related risks. As a result, organizations need to increase the attention given to addressing the potential for hazards related to software and computing systems.

2 Lessons Learned: System Safety Process Failures

Although the system safety process is an accepted approach to reducing risk in complex systems, there are a number of ways this process can fail to prevent an accident, especially in systems where software and computing systems are used. The sections that follow present potential failures in implementation of the system safety process, based on review of hundreds of software-related accidents and incidents (Hardy 2012). These sections will use findings from accident reports to provide lessons learned on those process failures. Note that in discussing these accidents this paper does not intend to oversimplify the events and conditions that led to the mishaps. Rarely is there only one identifiable cause leading to the accident. Accidents are usually the result of complex factors that include hardware, software, human interactions, and procedures. The descriptions here are meant to provide examples of where the system safety process failed in some way and to show how software and computing systems can play a role in those accidents. Readers are encouraged to review the full accident and mishap investigation reports to understand the often complex conditions and chain of events that led to each accident discussed here.

2.1 Failing to plan

System safety efforts must be planned, like any other engineering activity, and then that plan must be followed to be effective. Safety planning includes the planning for the management of system safety and emergency planning in the case where something could go wrong. It is not enough for a plan to exist – the plan must also be effectively implemented, updated, and followed.

On October 26, 1992, the London Ambulance Service (LAS) introduced a new computer aided dispatch (CAD) system to automate call taking, resource identification, and resource mobilization tasks. The automation was intended to improve emergency medical services for the city. At the time, the LAS provided ambulance service to 6.8 million people living in a 600 square mile area, making it the largest ambulance service in the world. The LAS received 2000-2500 calls per days, of which 1300-1600 were emergency calls. Just a few hours after the new computer system was introduced problems began to surface. The system was unable to keep track of ambulances and their locations. Multiple ambulances were sent to the same location in some cases. The system could not keep track of duplicate calls. And the system began to generate so many exception messages that the dispatchers became overwhelmed, and calls were lost. As the system became bogged down the LAS was forced to partially switch back to the manual system. Eight days later the computer system quit working and the LAS had to resort to a completely manual operation. Some estimates stated that as many as 46 people died as a result of the service failures.

An investigation into the incident found multiple causes to the system failures.

- The vendor chosen to build the system was selected primarily on the basis of price, and the vendor's cost estimates were unreasonably low.
- An unrealistic schedule of 11 months from start of development to deployment was placed on the vendor.
- At the time the system went live there were 81 open, known issues and no load testing had been performed on the system.
- Dispatcher training was inadequate.
- The system did not function well when given invalid or incomplete data on positions and statuses of ambulances.
- The user interface was poorly designed and did not respond properly to incorrect user entries.
- A memory leak in a small portion of the code led to the failure of the system eight days after deployment.
- Software requirements were developed without input from key users of the system, including dispatchers and ambulance operators.
- No quality assurance was performed on the software, and configuration management processes were lax.
- The system was overly complex.

The failure of the LAS CAD system was therefore a combination of errors related to safety planning, organizational priorities, safety management, process quality, product design, and product verification (Finkelstein and Dowell 1996).

2.2 Failing to accurately identify what can go wrong

Identifying what can go wrong, also known as 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. Describing what can go wrong can be difficult in complex systems, and identifying hazards takes persistence and creativity. In addition, complex systems using software can fail in complex ways, and some conditions and environments are difficult to postulate.

On February 11, 2003, an employee of the Southern Clay Plants & Pits in Gonzales, Texas was fatally injured while performing maintenance on a reaction tank. The U.S. Mine Safety and Health Administration (MSHA) determined that the cause of the accident was a failure to close and secure a manual gate valve for a steam line and a failure to place the batch PLC in the stop mode. The company was a surface clay mill that purchased clay and blended, refined, milled and processed the material into products used in paints, inks, and grease. On the day of the accident the employee had been informed that there had been a product change in one of the batch processing systems. The employee was assigned to perform cleanup duties on a reactor tank. Two valves controlled steam entry into the tank: a manual gate valve and a butterfly valve with an automatic pneumatic actuator. The PLC controlled the functioning of the batch system based on sensors that monitored material flow. At the time of the accident the PLC was in "slurry hold" mode. In this mode the system was programmed to actuate the steam valve

when the clay slurry level reached 5.5 feet. An aluminum extension ladder used by the employee caused the level sensor to falsely sense that slurry was in the reactor, which resulted in the PLC sending a command to open the steam valve. Because the manual valve had been left open, steam at 350°F then entered the tank, fatally burning the employee (U.S. MSHA 2003a).

2.3 Underestimating risk

After the hazard has been identified there needs to be an understanding of the significance of the potential problem to facilitate safety decision making. Risk assessment helps to understand potential problems and their significance, and helps to prioritize resources to fix the problems identified. The concept of risk includes an understanding of both the severity of the consequences and likelihood of the event. Without a proper analysis of both severity and likelihood it is possible that the risk could be underestimated. A number of accidents involving software and computing systems has shown that risk is frequently underestimated or misunderstood in these systems.

On January 19, 1995, an X-31 U.S. government research aircraft was destroyed when it crashed in an unpopulated area just north of Edwards Air Force Base while on a flight originating from the NASA Dryden Flight Research Center, Edwards, California. The crash occurred when the aircraft was returning after completing the third research mission of the day. The pilot safely ejected from the aircraft but suffered serious injuries, including two fractured vertebrae and a broken ankle and rib. A mishap investigation board studying the cause of the X-31 accident concluded that an accumulation of ice in or on the unheated Pitot-static system of the aircraft provided false airspeed information to the flight control computers. The resulting false reading of total air pressure data caused the flight control system to automatically misconfigure for a lower speed. The aircraft suddenly began oscillating in all axes, pitched up to over 90 degrees angle of attack and became uncontrollable, prompting the pilot to eject. The mishap investigation board also faulted the safety analyses, performed by Rockwell and repeated by NASA, which underestimated the severity of the effect of large errors in the Pitot-static system. Rockwell and NASA had assumed that the flight software would use the backup flight control mode if this problem occurred, and this in itself would reduce the risk. The mishap investigation board noted that probability and severity were confused in this safety analysis; just because the risk assessment concluded that the probability of total pressure being lost was low did not mean that the consequences were any less severe. This risk assessment resulted in a failure to recognize the safety-criticality of the Pitot tube and thus a failure to perform testing using both nominal and off-nominal conditions. (Haley 1995).

2.4 Overestimating the effectiveness of safeguards

If we simply identified the hazard and assessed the risk we would do little to improve safety. It is the implementation of safeguards (hazard controls) and designing safety into the system that reduces the risk. However, these controls must be appropriate for the hazard considered and they must be effective. Ineffective controls may provide a false sense of security, and may not work when needed. Automated systems may have weaker controls than thought, especially if human interaction is required. In addition, hazard controls themselves could introduce new, unforeseen hazards.

On February 18, 2009, an employee was fatally injured at the Ravensworth Coal Preparation Plant reject waste bin in the Hunter Valley region of New South Wales, Australia. The accident occurred when 10 tons of waste rock were inadvertently released from the reject bin and fell onto the cabin of the employee's truck. At the Ravensworth Coal Preparation Plant, raw coal was extracted from the mine and usable coal was separated from waste rock. The waste rock was transferred approximately 2 kilometres on conveyers to the reject bin. The waste rock was then loaded from the reject bin onto trucks and hauled away. The process of loading the trucks with waste rock was controlled by a PLC system. The PLC system included truck detection sensors, traffic lights, bin capacity sensing, and remote control, hand-held transmitters used by the truck drivers. On the day of the accident the truck driver drove his truck under the reject bin delivery chute. A signal was sent from the handheld remote control to command the chute to open. The accident report stated that it was not clear whether the signal was sent inadvertently or intentionally. Opening the chute required that two of three lines of truck detection sensors be blocked in addition to a command from the remote control to assure that the truck was in the correct location. Each sensor line contained three sensors, and all three sensors had to be blocked for the entire line to be considered as blocked. At the time of the accident the truck was obscuring one line of sensors, and a second line of sensors was obscured by dirt on the lenses and therefore was not working correctly. Because two of the sensor lines were blocked and the remote control signal had been sent, the PLC automatically opened the reject bin chute door and dropped 10 tons of material on the truck cab before the driver had safely cleared the chute, resulting in the fatal injury (State of New South Wales 2010).

2.5 Failing to verify that safeguards actually work

Once the control strategy has been identified and implemented, those controls should be validated and verified. Validation determines that the correct system is being built and verification determines that the design solution has met all the safety requirements. Verification normally includes analysis, test, inspection, and demonstration. Experience has shown that verifications that are performed using improper assumptions or are conducted under conditions that are different from those in operation can lead to an underestimation of risk. Of special concern in software is the failure to test using sufficient off-nominal conditions and considering hardware failures and improper inputs.

On November 16, 2000, the Space Technology and Research Vehicles (STRV) microsatellites STRV 1-C and

STRV 1-D were launched on an Ariane 5 launch vehicle. STRV 1-C was intended to perform accelerated life testing of new components and materials in the high radiation environment of geosynchronous transfer orbit. STRV 1-D carried additional experiments. Two weeks after launch STRV 1-C displayed control problems; STRV 1-D exhibited the same problems a few days later. Eventually, both spacecraft lost communications with the ground. Investigations after the loss of the spacecraft found that a software error provided continuous current, instead of a short pulse, to latching relays. The continuous current heated the relays and degraded their insulation, which resulted in a short circuit that disabled the main receiver. A secondary receiver existed for redundancy, but this secondary receiver had been isolated by a trip switch. The trip switch required a ground command to be reset, and this could not be done without communications through the primary receiver. The problem was traced to a software specification that did not incorporate a requirement to command the relays by pulse. The problem was not found on the ground because the test software drove the relays with pulsed signals (Harland and Lorenz 2005).

2.6 Inadequate hazard tracking and anomaly reporting processes

Accident analyses often show that clues existed before the mishap occurred. Such clues frequently take the form of anomalies observed during the life cycle of a project. Therefore, learning from failure is critical to improving safety and preventing accidents. Anomalies discovered in the life cycle development must be properly reported to learn from those problems. In addition, a closed loop root cause and corrective action process must be in place to translate the documented anomalies into safety actions. That process must assure that hazard reports are re-evaluated as problems are found.

On August 12, 1998, the Titan IV A-20 launch vehicle lifted off from Florida. The rocket was carrying a classified National Reconnaissance Office payload. Approximately 40 seconds into flight the launch vehicle pitched down and began to break up, then automatically destroyed itself when the Inadvertent Separation Destruct System initiated the destruct sequence as soon as one of the solid rocket motors separated from the core booster. The payload was lost, although there were no injuries as a result of the accident. The accident investigation board found that exposed wires shorted during flight, causing an intermittent outage of the Missile Guidance Computer (MGC), which in turn lost the signal to the Inertial Measurement Unit (IMU) used to guide the rocket. The MGC recovered power, but the IMU then provided a false indication that the launch vehicle had pitched up and to the left (it had in fact been flying on the correct course). To compensate for the perceived pitch up, the MGC commanded the launch vehicle to pitch down and to the right. The aerodynamic stresses from these movements exceeded the structural margins of the launch vehicle and the rocket began to break up, ultimately destroying itself. The accident investigation board did not identify the source of the wire damage leading to the short circuit. However, the board reviewed historical records and identified hundreds of wiring faults and defects at the factory that were later discovered by inspection, and found previous incidents of short circuits while in flight. The board noted that the guidance system design was a causal factor because the timing signal from the MGC to the IMU was unable to withstand power transients that could reset the computer (U.S. Air Force 1999).

2.7 Failing to adequately manage change

While change is a normal part of the engineering process, there is no such thing as a minor change with respect to software safety. All changes to safety-critical systems must be evaluated because even minor changes can have major safety impacts. This typically means that organizations must have robust change management and configuration management systems, and changes must be factored back into the hazard analysis.

On October 24, 2002, a grinder exploded at the Foreman Quarry and Plant in Foreman, Arkansas. An operator was killed when flammable waste fuel covered him and ignited. The operator had started the pump for solid waste fuel processing when the accident occurred. The U.S. MSHA stated that the cause of the accident was that the safety monitoring system designed to shut off the waste fuel system pump had not been maintained so that it functioned properly. The Foreman Quarry and Plant, operated by Ash Grove Cement Company, mined limestone and processed it for use in Portland cement. Kilns were used in the processing, and these kilns were heated by burning coal, natural gas, and liquid waste fuel. The liquid waste fuel was delivered by truck or railcar and pumped into large storage tanks. From the storage area it was pumped through a grinder to reduce the particle size of the solids in the fuel. Two independent systems monitored and controlled the waste fuel delivery. A Foxboro Intelligent Automation Distribution Control System (I/A DCS) monitored and recorded normal operating parameters. The Foxboro also issued audible and visual alarms that were available at the plant control room. A PLC provided basic start up and shutdown of the system and responded to commands from the Foxboro. On the day of the accident the Foxboro sensed that the fuel delivery pressure was low, apparently due to blockage in the line. As designed, the Foxboro sent a command to the PLC to shut down the pumps. However, the PLC failed to respond and the pumps kept running. Three months prior to the accident this PLC had been installed; this was supposed to be a simple replacement of an older PLC of similar capability. However, the Foxboro had not been connected to the newer PLC, and the connections remained to the older non-functioning PLC. The system had never been tested with the new PLC. A test had been scheduled three days prior to the accident but had been aborted when a pump failed during the test; the test had never been rescheduled. The accident report stated that the blockage may have broken free just prior to the accident. With the pumps running, the pressure elevated significantly and a “water hammer” effect caused overpressurization in the system at the grinder. The grinder was torn loose from its base, spraying fuel and pulling loose a 480-volt cable that ultimately served as an ignition source (U.S. MSHA 2003b).

2.8 Weak safety culture

Most accidents are the result of a confluence of factors, and not just the result of failures of components or systems. Since the greatest threats to safety often originate in organizational issues, many industries have begun to realize that making the system safer requires improvements in the organization's safety culture. However, not all organizations have been successful in improving and maintaining organizational safety.

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 U.K. Marine Accident Investigation Branch (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 (U.K. MAIB 2011).

3 Overall Software Safety Lessons Learned

The accidents and incidents described here illustrate that there are significant challenges in the software safety discipline, and that organizations often fail to perform effective software safety efforts as part of an overall system safety approach. Some broad lessons learned that emerge from the examination of hundreds of accidents (Hardy 2012) include the following.

- *Decisions made in the acquisition and planning phases of development can profoundly affect safety***.** Planning typically involves trade-offs between many different facets of the program, including cost, schedule, performance, and safety. Poor planning can lead to unexpected safety consequences, and many safety decisions are actually made in the planning and acquisition phase. However, software safety personnel are often not included in early phases of a program when those critical decisions are being made. In addition, adequate resources may not be allocated to the software safety effort. This can result in a failure to perform hazard analyses and identify safety requirements early in the program when these activities provide the most impact.
- *Communication barriers between software engineers, hardware engineers, safety personnel, and management are common*. No one person can fully understand a complex system, especially one with software. Therefore, multiple individuals and organizations must interact and trade information to effectively reduce risk. This means that different parts of the organization must learn to speak each other's language. Communications between customers and suppliers must also be open and frequent. Misunderstandings and miscommunication are often contributors to accidents. Some of those misunderstandings come from inadequate requirements management efforts.
- *Software hazard causes are oversimplified or focused only on failures***.** Review of a number of hazard reports from different organizations has shown that software hazard causes and controls often do not provide sufficient detail or clarity. Software causes may be generically stated as "software error," instead of defining specifically the software functionality that can lead to an undesirable outcome. The focus is often on failure of the functionality to work, but other causes, such as inadvertent operation, may be ignored. Interfaces, especially those between software and hardware, may be misunderstood, and interactions between components are not explored. The software hazard analyses may not pay enough attention to those cases where the software works exactly as intended, but the implemented functionality is unsafe.
- *Risks may be underestimated and optimistically evaluated***.** Risk assessments allow organizations to make decisions about uncertain futures given existing knowledge. Assessing the risk of software-related systems presents challenges in large part because the evaluation of the likelihood of the hazard is difficult. Instead of using that limitation as an opportunity to carefully consider many different risk factors, organizations instead may create optimistic projections of what they want to happen. Or they may equate past success with low risk, ignoring the fact that testing and operations cannot feasibly consider all combinations of possible inputs.
- *Hazard controls may rely on good software processes and testing*. System safety efforts should follow the design order of precedence, where the first approach is to try to design out

the hazard or minimize the risks through design selection. Software is no different in this regard. Yet organizations may still focus on quality control and quality assurance efforts, such as focusing on good software processes or extensive unit testing, to prove that the design is safe. However, software processes and testing will not prevent an accident if the software design is flawed with respect to safe system operations.

- *There may be a failure to ask "what if the hazard controls don't work?"* Organizations may implement what appear to be effective controls, but then do not take the analysis any further. While organizations certainly understand that those controls may fail, they do not take the next step and ask what happens if they do not perform their function or perform the function incorrectly. Organizations make optimistic assumptions about the ability of the system, including hardware, software, and humans, to come to the rescue when the undesired event happens.
- *Testing tends to focus on functional operation and not off-nominal conditions***.** Testing can be expensive, and most organizations are limited in the resources they can apply to testing. Therefore, the focus is naturally on making sure the system meets the requirements. This is necessary, but not sufficient. Many accidents have shown what can happen if testing does not include off-nominal scenarios and abnormal conditions. Testing should not just address what is required but also include what can go wrong.
- *Testing may not provide information on subsystem and component interactions***.** Software and computing system accidents occur most often because of unanticipated interactions, not because the software was poorly coded. A number of accidents have occurred when no component failed in the conventional sense, but the interaction of components caused a system failure. Therefore, a significant focus of safety testing must be on a fully integrated system, with testing of end-to-end events. That testing must include stressing of the software, and should include interactions of the software with hardware, humans, and environments. Many verification efforts however fail to perform sufficient integrated system testing, or include operator interaction in that testing.
- *Anomalies may not be factored into the design or hazard analysis*. Learning from failure and problems is essential to safety. These problems provide clues of accidents yet to come. Therefore, software problem reports, like those of hardware, should be part of a larger root cause and corrective action system. These problem reports should include issues found during actual operation. Yet organizations do not always take these problems seriously or use those problems to look for issues that could lead to system failure. Software does not have to be perfect to be safe, and not all errors impact safety. But errors in safety-critical functions should be investigated and corrected. Conversely, organizations may incorrectly assume that a lack of anomalies or mishaps implies that the system is safe; in fact, latent errors could exist, and these errors may contribute to an accident.
- *Software change management and hazard analyses processes may not be integrated***.** Engineering by its very nature is an activity that requires change, and changes occur in the hardware, software, processes, and organizations throughout development and into operation. While a number of organizations may have strong configuration and change management practices, those practices do not always integrate with the hazard analysis process. Hazards may fall through the cracks if those processes are not integrated.
- *Human-software interactions have significant safety implications that are often underestimated***.** Humans interact with hardware and software in positive and negative ways. Organizations may not understand the importance of human-software interactions or pay as much attention as they should to displays and control panels. In addition, they may make changes to user interfaces and information flow on critical systems without adequate assessment. Organizations may count on operators saving the day when a bad day occurs in complex, software-intensive systems, but they may not provide proper tools and training to enable operators to perform those safety-critical functions.
- *Support software may be as critical to safety as control software but may not be included in safety analyses***.** The focus of most software safety efforts is naturally on software that directly controls an operation. But software and computing systems show up in many different parts of the system, and this support software may turn out to be safety-critical. Support software, including models and simulations, may be just as hazardous as controlling software, but it is often not thoroughly examined.
- *Hazard analyses and safety systems may not be updated using operations and maintenance experience***.** It is usually during the initial operating phases that the most is learned about the system. However, organizations may fail to feed what is learned in operations and maintenance back into their safety analyses.

4 Promoting a Questioning Attitude in Software and System Safety

It is important to promote the use of system safety methodologies and analyses. It is difficult to understand and then decrease the risk of complex technologies without the use of a structured approach to identifying and controlling hazards. However, as discussed above,

lessons learned from past accidents and experiences point to the importance of cultivating and encouraging a questioning attitude toward all aspects of the system safety process, especially where software and computing systems are important for safety. Implementation failures can occur in any of the system safety process steps. We should use lessons learned such as those described in this paper to help us understand how previous efforts failed to prevent accidents, and how our own efforts might be similar. We should require compelling evidence before concurring with the analysis.

Most importantly, we should ask critical questions about the overall software and system safety process. By asking focused questions we can challenge assumptions. Such questions can stimulate thinking and get people to open up about the risks. Good questions allow us to view the system holistically, rather than just as the sum of its parts. Examples of such questions include the following:

- Do plans reflect how business is really done? Are plans reviewed? Do plans have unrealistic schedules or resource allocations? Is software part of that planning? Poor or unrealistic plans may reflect an organization that does not truly place a priority on safety activities.
- Is there a convincing story that the safety analysis is complete and thorough, and that software's contributions to hazards have been identified? Did the analyst use multiple tools (fault tree, hazard analysis, etc.) to perform the analysis? Were checklists, accident reports, previous experience, or a combination of those employed? Failure to show that the problem is being looked at from multiple perspectives could be an indication that there are holes in the analysis and that significant problems may not be identified.
- Are the reports detailed enough? Are causes descriptive? Does the logic make sense and is it complete? Do controls match up with the causes, showing a one-to-one or many-to-one relation? Lack of detail could be an indication of insufficient knowledge of the system, or lack of information on the system.
- Are the hazard controls primarily procedural rather than design changes, safety features or devices? Is there an overreliance on humans and software to "save the day"? Overreliance on operational controls may indicate a weak safety design.
- Can the control strategy actually be implemented and verified? Is the control strategy so complex that it will be impossible to determine whether it will work when needed? Is the control truly effective? Are controls truly independent? Complex controls or overlapping control strategies may be an indication of a weak safety design.
- Has the risk assessment truly considered the worst case? What is the basis for the likelihood levels? Has the risk assessment considered lower severity but higher likelihood cases? Is the risk analyzed by cause and by phase? Failure to provide good answers to these questions indicates a potential misunderstanding of the risk.
- Are problems found in test and design included in the hazard reports and factored into the design? Failure to incorporate problems and corrective actions is an indication of the potential to miss serious design flaws.

These questions help to identify whether the system safety process is robust. However, we must also ask questions related specifically to the use of software and computing systems in complex systems. The best questions come from real-world examples of accidents where software has been a contributor. Some examples of questions are as follows, and others can be found in Hardy (2011).

- Have safety-critical software, commands, and data been identified?
- Do hazard controls for software-related causes combine good practices and specific safeguards?
- Do standards exist for software peer reviews and other design reviews?
- Is software and system testing adequate, and do tests include sufficient off-nominal conditions?
- Is the computing system design overly complex?
- Is the design based on unproven technologies?
- What happens if the software locks up?
- Are the sensors used for software decisions fault tolerant?
- Has software mode transition been considered?
- Has consideration been given to the order of commands and out of sequence inputs?
- Will the software and system start up and shut down in a known, safe state?
- Are checks performed before initiating hazardous operations?
- Will the software properly handle spurious signals and power outages?

These are by no means all the questions a decision maker should ask, and positive answers to these questions provide no assurance that an accident will be prevented. These questions should encourage critical thinking and generate additional safety questions to provide further insight on system risk. A failure to ask these questions could mean that the potential for an accident is higher than we had assumed.

We also have a responsibility as system safety practitioners to share our doubts and questions with decision makers to allow them to understand what we do not know and where uncertainties exist. In particular, we should:

- *Avoid oversimplifying the potential hazard causes*. Identifying hazard causes in complex, automated systems can be a difficult process, and decision makers should be made aware of the challenges in performing this activity.

- *Do not downplay uncertainties, especially with likelihoods*. Obtaining credible reliability estimates for software may not be possible for new systems, and qualitative risk assessments should be supplemented with analyses of other factors such as complexity, maturity, degree of system testing, and so on.
- *Do not self-censor, especially with respect to hazard controls*. When safety practitioners are aware that a contract has been issued which limits the choices of hazard controls, it is natural to eliminate options from consideration. However, the decision maker should be aware that such choices are being made.
- *Provide alternatives, but discuss the tradeoffs in risk*. Rather than simply saying "no" to an activity, safety practitioners should provide the decision maker with options, then clearly describe the risk of each option.
- *Discuss the limitations of the testing and verification efforts*. It is practically impossible to test every possible combination of software inputs, or test every possible hardware or software configuration to be used. Decision makers should be made aware of these limitations.
- *Be clear about the effects of failures and changes during development and the potential for increased risk*. Problems discovered during development and in operation, and changes resulting from problem fixes and upgrades, can have major impacts on safety.
- *Use accidents and incidents to provide support for safety conclusions*. Decision makers will respond more favorably to our conclusions if there is concrete evidence to back up our claims. We should use available accident and incident reports to provide that evidence. These "stories" will also resonate better than statistics with decision makers in making our case.

It is up to all stakeholders to look for those conditions that could lead to an accident and to recognize that the worst can happen. This means we should all express concerns about safety management and engineering when necessary based on our knowledge, experience, and judgment, and based on lessons learned from accidents. We must ask questions to understand the potential for harm, to understand the steps taken to assure that the risks have been reduced, and to assure that there is proof that hazard controls are effective. And we must openly and honestly communicate what we do not know. We will never eliminate risk, nor do we want to. Without risk there is no reward. But it is up to all of us to promote and encourage a questioning attitude to ensure that we are knowledgeable of those risks and to assure that the risks have been appropriately reduced.

5 Summary

System safety can provide immense benefits to any industry, especially those designing, building, and operating complex systems using software and computing systems. By proactively identifying hazards, assessing and characterizing risks, and taking actions to reduce those risks, organizations can prevent accidents and reduce the potential for death, injury, property damage, and environmental impacts. However, poor system safety analyses can result in precious resources being used on low risk activities while larger risks are ignored. When applied inappropriately, system safety methods can lead to overconfidence and result in an underestimation of certain important risks. System safety efforts should be promoted and advocated, but we should also promote a questioning attitude to further the discipline. We should understand the ways that these analyses can provide misleading results, especially in software-intensive systems, and we should examine the ways in which risk can increase by the actions we take. Lessons learned in the form of accidents and experiences in implementing the system safety process should be used to fuel those questions. It is through a questioning attitude that system safety and software safety efforts can accomplish their main goal -- preventing accidents.

6 References

Finkelstein, A., and J. Dowell (1996): "A comedy of errors: the London Ambulance Service case study," 8th International Workshop on Software Specification and Design.

Haley, D. (1995): "Ice Cause of X-31 Crash," National Aeronautics and Space Administration Dryden Flight Research Center, Edwards, California, NASA Press Release 95-203.

Hardy, T.L. (2011): *Essential Questions in System Safety: A Guide for Safety Decision Makers*, AuthorHouse.

Hardy, T.L. (2012): *Software and System Safety: Accidents, Incidents, and Lessons Learned*, AuthorHouse.

Harland, D.M., and R.D. Lorenz (2005): *Space System Failures: Disaster and Rescues of Satellites, Rockets, and Space Probes*, Praxis Publishing.

State of New South Wales (2010): "Fatality involving David Hurst Oldknow Ravensworth Underground Mine Coal Preparation Plant Reject bin 802 18 February 2009," May 2010.

United Kingdom Marine Accident Investigation Branch (2011): "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.

United States Air Force (1999): "Titan IVA-20 Accident Investigation Board Summary," January 15, 1999.

United States Mine Safety and Health Administration (2003a): "Report of Investigation: Fatal Other Accident (Steam Burns), February 11, 2003, Southern Clay Plants & Pits Southern Clay Prod. Inc., Gonzales, Gonzales County, Texas," Mine I.D. No. 41-00298.

United States Mine Safety and Health Administration (2003b), "Report of Accident: Exploding Vessels Under Pressure Accident, October 24, 2002, Foreman Quarry and Plant, Ash Grove Cement Company, Foreman, Little River County, Arkansas," Mine I.D. No. 03-00256.