2010 Fourth International Conference on Secure Software Integration and Reliability Improvement

Recent Catastrophic Accidents: Investigating How Software Was Responsible

W. Eric Wong, Vidroha Debroy, Adithya Surampudi & HyeonJeong Kim
Department of Computer Science
University of Texas at Dallas
{ewong,vxd024000,avs091020,hj.kim}@utdallas.edu

Michael F. Siok
Lockheed Martin Aeronautics Company
Fort Worth, Texas, USA
mike.f.siok@lmco.com

Abstract

Areas crucial to life such as medicine, transportation, nuclear-energy research and industry, aeronautics, and others, all make use of software in one way or another. However, the application of software to such domains means that the software may now become safety-critical such that an error in the software or an error in its use could have devastating consequences. This paper reviews 14 recent accidents, several of which resulted in the loss of life in addition to time and money, and identifies the role(s) that software played as an important causative factor. The useful lessons which can be learned from the accidents are also presented, which can then act as principles and guidelines to avoid the recurrence of similar accidents in the future.

Keywords: Software safety, catastrophic accidents, mishaps, safety-critical software systems

1. Introduction

Software systems are actively used today in safety-critical areas such as aeronautics, astronautics, medicine, nuclear power generation and nuclear research, transportation, etc. When employed in such systems, software is often responsible for controlling the behavior of electromechanical components and monitoring their interactions in addition to other tasks such as user interface management, computer administration, and others. Since most accidents arise in the interfaces and interactions among the components, software plays a direct and important role in system safety [12].

Consequently an important issue that comes up is that of *software safety* – which means that the software should execute within a system context without contributing to hazards [12]. However, should the software operation directly or indirectly lead to a hazard in the case of a safety-critical system, then the consequences of the hazard realization could be catastrophic. By catastrophic we imply that the damage is not just restricted to financial losses, or losses in terms of time or property, but rather may also include the loss of life.

This paper selects a representative set of 14 recent catastrophic accidents and presents a brief summary of them and their software-related causes (whether direct or indirect). We also explore the extent of the damage incurred in the case of each accident, and more importantly, identify valuable lessons that can be learned from them. By presenting the cases of such accidents to the reader, we hope to highlight the ever growing importance of software safety both as a field of research and education, and as an aspect of software development that needs more attention in practice.

Thus, the contributions made by this paper are as follows:

- We highlight the importance of software safety, especially in the case of safety-critical systems, by analyzing 14 recent catastrophic accidents and underscoring the software-related causes.
- For each accident we identify useful lessons that can be taken away to avoid their recurrence.

The remainder of this paper is organized in the following manner. In Section 2, we categorize and present a detailed review of the circumstances surrounding each chosen accident and identify the software-related causes for each. Section 3 then identifies the lessons that can be learned from the accidents followed by a discussion in Section 4 of issues and concerns relevant to this paper. Finally, conclusions and suggested follow-on research is presented in Section 5.

2. Recent Catastrophic Accidents Involving Safety-Critical Systems

Each accident presented in this section is summarized individually. Yet, to facilitate discussion and apply an order to the presentation, we also categorize the accidents based on whether the damage incurred involved the loss of life or not. Note that there may be several other ways to classify or categorize these (and other similar) accidents based on various criteria. Further discussion on this is presented in Section 4.3.

2.1 Accidents Involving the Loss of Life

The accidents presented in this section are those that resulted in loss of life in addition to significant financial and/or property loss incurred.

Miscalculated Radiation Doses at the National Oncology Institute in Panama [2,5]: Panama's largest radiation therapy institution is the National Oncology Institute (Instituto Oncológico Nacional, ION). In March 2001, Panama's Ministry of Health asked the Pan American Health Organization (PAHO) to investigate some serious overreactions among cancer patients undergoing specific radiation therapy treatment at ION. A total of 56 patients were treated for specific cancers within the target time period and were identified in the ION radiation dosage study group. From this study group it was determined that 28 patients were likely given radiation doses exceeding their required dosage. By

August 2005 (4.5 years later), 23 of these 28 patients died. It was reported that it was unclear whether these patients would have died from their cancers with or without treatment. However, at least 18 of the patient deaths were attributed directly to radiation overdose.

After a careful investigation, three main causes for these accidental overdoses were identified: (1) an unclear manual for using the radiation mechanism, (2) no warnings in the software program for the improper usage, and (3) no manual quality control before usage. The treatment planning system software allowed a radiation therapist to draw on a computer screen the placement of metal shields (called "blocks"). Investigation revealed that the software allowed the use of at most four shield blocks, but the doctors at ION wished to use five. The doctors thought they could get around this software-imposed constraint by drawing all five blocks as a single large block with a hole in the middle. What the doctors did not realize was that the software provided different results in this configuration based on how the "hole" was drawn. Drawing it in one direction, the correct dose would be calculated; drawing it in another direction, the software would recommend twice the necessary exposure.

Loss of Precision Error in Patriot Missile Defense System at Dhahran, Saudi Arabia [20]: The Patriot is a surface-to-air defense missile system used by the United States Army to protect against cruise missiles and medium to high altitude aircraft. During the Gulf war in the early 1990's, Patriot missile systems were deployed at strategic locations to defend key assets, military personnel, and citizens against Scud missiles launched by Iraqi forces. The Patriot works by locking its radar onto an incoming target and relaying the signals to a computer at a control station that tracks the target's speed, trajectory, and computes a predicted course. Using a series of complex split-second computations, the computer calculates when to launch its missiles and, in the case of Scuds, fires two Patriot missiles each with a 200-pound conventional warhead traveling at about 2,000 miles an hour at each Scud.

On the night of February 25, 1991, a Patriot missile defense system operating at Dhahran, Saudi Arabia failed to track and intercept an incoming Scud missile. This Scud subsequently hit a U.S. Army barracks killing 28 soldiers and injuring another 98.

The Patriot battery at Dhahran failed to track and intercept the Scud missile because of a software problem in the weapons control computer. This problem led to an inaccurate tracking calculation (i.e., a loss of precision error) that grew worse the longer the system operated without a reset. At the time of the incident, the Patriot battery had been operating continuously for over 100 hours. By then, the small inaccuracy was serious enough to cause the system to target the wrong place for the incoming Scud. Once identified, the error was repaired and the revised software deployed to all Patriot Missile Defense Batteries..

American Airlines (AA) Flight 965 Accident [14]: On December 20, 1995, AA965 departed from Miami International Airport bound for Cali, Colombia. At 9:40pm, just five minutes before its scheduled arrival, the plane went down in the Andes, crashing into the west slope of a mountain. The crash led to a total of 159 deaths and completely destroyed the airplane (a Boeing 757). Only four of the passengers and a dog survived. This was the first U.S.-owned 757 accident and resulted in the highest death toll of any accident in Colombia. The death toll was also the highest of any accident involving a Boeing 757 at the time and was the deadliest air disaster involving a U.S. carrier since the downing of Pan Am Flight 103 on December 21, 1988.

The radar for the Cali, Colombia air traffic control system had been damaged in a guerrilla attack in 1992, so controllers were unable to use it to track the approach of the 757 aircraft. The pilots relied on a series of radio beacons which had been placed in the area to serve as navigational aids through the canyons and mountainous terrain. Having already been programmed with the location of these beacons, the onboard flight management systems should have guided the pilots from their point of origin to the terminal at the Cali airport.

The user interface for the flight management system allowed the pilot to enter the name of a destination. If the initial instead of the full name of the destination was entered, the flight management system would select the destination which had the highest radio frequency among all the destinations which started with that initial. Once the destination had been selected, the flight management system did not provide any selection feedback.

The circumstances leading to this accident involved the pilots programming the navigational computer with the location of Rozo (a waypoint on their approach). Although this waypoint was designated "R" on their charts, the same identifier had also been used for a second radio beacon (Romeo) near Bogotá. When a crewmember changed the approach course to the airport, he input only "R" instead of "Rozo" in the computer. The flight management system interpreted this as a request to set a course to the "Romeo" waypoint which has a higher radio frequency than the one at Rozo. When two waypoints with the same initial were nearby, the FMS automatically chose the one with the higher frequency but listed the others. The pilot selected the first "R" on the list, unwittingly setting a course for Bogotá, which caused the autopilot to execute a wide semicircular turn to the east and eventually to the accident site.

Korean Air Flight 801 Accident [9,15,18]: On August 5 1997, Korean Air Flight 801 departed from Kimpo International Airport for Seoul, South Korea. Korean Air Flight 801 was a scheduled commercial flight between Seoul, Korea and Guam. On board were 254 people – 2 pilots, 1 flight engineer, 14 flight attendants and 237 passengers. The flight experienced some turbulence along the way but the flight was otherwise uneventful until the jet prepared to land. At 1:42am, the aircraft struck Nimitz Hill, about 3 miles (5 kilometers) short of the runway at an altitude of 660 feet (201 meters). Of the 254 people on board, 228 were killed; 23 passengers and 3 flight attendants survived the accident with serious injuries. The impact and post-crash fire destroyed the aircraft. In 2000, a lawsuit was settled in the amount of $70,000,000 on behalf of 54 families.

The National Transportation Safety Board (NTSB) determined that the probable cause of the accident was the captain's failure to adequately brief and execute a non-precision localizer-only approach. A contributing cause was the first officer's and flight engineer's failure to effectively monitor and cross-check the captain's execution of the approach. In addition, the captain's fatigue and Korean Air's flight crew training were listed as contributors. The crew had been using an outdated flight map, which stated that the minimum safe altitude for a landing aircraft was 1,770 feet (540 meters) as opposed to 2,150 feet (656 meters). Flight 801 had been maintaining 1,870 feet (570 meters) prior to landing. Bad weather in the area was also a contributing factor. Although Flight 801 likely exited a heavy rain shower shortly before the accident, the flight crew was still not able to see the airport due to the presence of another rain shower located between Nimitz Hill and the airport.

The identified software-related cause was the FAA's intentional inhibition of the Minimum Safe Altitude Warning (MSAW) system. The MSAW system was developed by the FAA in response to the NTSB Safety Recommendation A-73-46. However, a 54-nm inhibit zone was setup at the Guam International Airport to alleviate frequent nuisance warnings; this was allowed by policy. By inhibiting these warnings close to the airport, the usefulness of the MSAW system in providing minimum safe altitude over terrain warnings was diminished. Simulations by NTSB/FAA indicated that, without the inhibition, MSAW would have generated an alert 64 seconds before impact which would have been sufficient for the controller to advise Flight 801 and give the aircrew an opportunity to take appropriate actions to avoid the crash.

2.2 Accidents that Did Not Involve the Loss of Life

The accidents presented in this section are those that resulted in significant loss of time and money, yet fortunately the loss of life was avoided.

Shutdown of the Hartsfield-Jackson Atlanta International Airport [3]: Hartsfield–Jackson Atlanta International Airport is one of the world's busiest airports, both in terms of passengers and number of flights. It serves as a major hub for travel throughout the Southern United States. The airport has 151 domestic and 28 international gates. To ensure security, all the passengers must go through a checkpoint and have their carry-on luggage screened by an X-ray machine before they are allowed to board their aircraft. The alertness of the security personnel screening luggage is tested by the periodic display of suspicious devices on the X-ray machine displays, followed by a message indicating that the particular display was only a test.

On April 19, 2006, an employee of Transportation Security Administration (TSA), U.S. Department of Homeland Security, identified the image of a suspicious device but did not realize it was part of the routine testing for security screeners. The software failed to indicate such a test was underway. As a result and per procedure, the airport authorities evacuated the security area for two hours while searching for the suspicious device manually, which they could never find. This evacuation occurred at peak travel time which caused delays of more than 120 flights. In addition, many passengers were inconvenienced; many had to rush to reach their departure gates or reschedule flights due to this evacuation.

This false alarm was due to a software error which failed to alert the security screeners that the suspicious device was only a test image. Normally the software flashes the words "This is a test" on the screen after a brief delay.

Loss of Communication between the FAA Air Traffic Control Center and Airplanes [6,13]: On Tuesday, September 14, 2004, the Los Angeles International Airport and other airports in the region suspended operations due to a failure of the FAA radio system in Palmdale, California. Technicians on-site failed to perform the periodic maintenance check that must occur every 30 days and the radio communications system shut down without warning. The air traffic controllers lost contact with the aircraft when the primary voice communications system shut down unexpectedly. More precisely, it was a Voice Switching and Control System (VSCS) that failed.

Air traffic controllers use a touch-screen interface to the VSCS to select a phone line to connect to other controllers or to a radio frequency to talk to flight crews. When the VSCS failed, the backup system that was supposed to take over in the event of a primary system failure also crashed. Fortunately, the Collision Avoidance Systems (CAS) onboard the commercial aircraft in-flight helped to avert catastrophe during this outage. The CAS interrogated the transponders of nearby aircraft and if danger of a collision was detected, one of the pilots was instructed by the system to climb and the other to descend. This communications outage disrupted about 600 flights (including causing 150 flight cancellations), impacting over 30,000 passengers. Flights through the airspace controlled by the Palmdale, CA facility were either grounded or rerouted elsewhere.

An error in the VSCS compounded by human error was ultimately responsible for the three-hour radio breakdown. The replacement for the older VSCS system needed to be reset approximately every 50 days to prevent data overload. The software used a 32-bit countdown timer that decremented every millisecond. This allowed for 2^{32}-1 milliseconds worth of timer ticks after which the counter reached zero and the system could no longer decrement the time; the software then shut down. The manual radio system reset was needed before the counter reached 2^{32} milliseconds (approximately 50 days). A field technician failed to perform the radio reset and the VSCS subsequently shut down. The backup to this system also failed.

Learning from this experience, the FAA deployed a software fix to the VSCS which addressed this countdown timer issue.

Mishap Involving NASA's Demonstration of Autonomous Rendezvous Technology (DART) [17]: On April 15, 2005, the Demonstration of Autonomous Rendezvous Technology (DART) spacecraft was successfully launched from the Western Test Range at Vandenberg Air Force Base, California.

DART was designed to rendezvous with and perform a variety of maneuvers in close proximity to the Multiple Paths Beyond Line of Sight Communications (MUBLCOM) satellite without assistance from ground personnel. DART performed as planned during the first eight hours through the launch, early orbit, and rendezvous phases of the mission, accomplishing all objectives up to that time even though ground operations personnel noticed anomalies with the navigation system. During proximity operations, however, the spacecraft began using much more propellant than expected.

Approximately 11 hours into what was supposed to be a 24-hour mission, DART detected that its propellant supply was depleted and it began maneuvers for departure and retirement. Out of 27 defined mission objectives, DART met only 11. NASA declared a "Type A" mishap and convened a Mishap Investigation Board (MIB). These measures are taken for any mishap resulting in a mission failure and a loss of more than one million US dollars in government funds. This mishap category requires the most detailed level of investigation.

None of the 14 requirements related to the proximity operations phase, the critical technology objectives of the mission, were met. The MIB found that DART had also collided with MUBLCOM 3 minutes and 49 seconds before its retirement. The MIB determined the underlying causes for this collision based on hardware testing, telemetry data analysis, and simulations. From the investigation, the MIB developed two timelines for the study: one for DART's premature retirement and another for DART's collision with MUBLCOM.

Software-related causes played a significant role in this incident. The premature retirement of DART occurred due to a cycle of computational "resets" by the software throughout the mission which triggered excessive thruster firings and an unexpected rapid fuel depletion. Also incorrect velocity measurement from the primary GPS receiver was introduced into the software's calculations during each reset. A software fix for this known "bug" had not been applied to the on-board software by the DART team either prior to or during the mission.

Loss of the Mars Polar Lander [8,16]: The Mars Surveyor '98 program was comprised of two spacecraft launched separately: the Mars Climate Orbiter and the Mars Polar Lander (MPL). The two missions were designed to study the Martian weather, climate, and water and carbon dioxide budget in order to understand the reservoirs, behavior, and atmospheric role of volatiles and to search for evidence of long-term and episodic climate changes.

MPL, with two Deep Space 2 (DS2) microprobes was launched on January 3, 1999 – 23 days after its partner, the Mars Climate Orbiter. All three components of the MPL were mounted to a shared cruise stage which provided Earth communications, power, and propulsion support services for the trip to Mars. The length of the planned MPL mission after landing was 90 days; the DS2 mission was two days. The probes were to be released from the cruise stage after the lander–cruise stage separation, plummeting to the surface to impact about 60 kilometers from the MPL landing site. After an eleven month hyperbolic transfer cruise, MPL reached Mars on December 3, 1999. However, NASA lost contact with the spacecraft just prior to its scheduled atmospheric entry and communication was never regained. The total loss was $120 million (not including the launch vehicle and two DS2 microprobes) of which $110M was allocated to spacecraft development and the other $10M to mission operations.

Given the total absence of telemetry data and no response to any of the attempted recovery actions, an exact cause, or causes, of failure could not be determined. Nevertheless, a special review board appointed by the Jet Propulsion Laboratory (JPL) Director identified that the most probable cause of the loss of MPL was a premature shutdown of the descent engines. It was assumed that one of the MPL's leg contact sensors tripped during descent which signaled a false indication that the spacecraft had landed. This, in turn, resulted in the spacecraft shutting down its descent engines. At least one of the magnetic sensors attached to the landing legs generated a spurious touchdown indication. The software, intended to ignore touchdown indications prior to the enabling of the touchdown sensing logic, was not properly implemented and the spurious touchdown indication was retained. In addition, the touchdown sensing software was not tested with MPL in the final flight software configuration prior to its use on the mission.

Loss of the Mars Climate Orbiter [19]: Companion to the MPL (discussed above), the Mars Climate Orbiter was also part of the Mars Surveyor' 98 program. Mars Climate Orbiter was launched on December 11, 1998. After a brief cruise in Earth orbit, the spacecraft was put into trans-Mars trajectory; about 15 days after launch, the largest Trajectory Correction Maneuver (TCM) was executed using the spacecraft's hydrazine thrusters. During cruise to Mars, three additional TCMs were performed.

The Mars Climate Orbiter was intended to enter an orbit at an altitude of 140 – 150 kilometers (approx. 460,000 - 500,000 feet) above the Martian surface. However, a navigation error caused the spacecraft to reach as low as 57 kilometers (~190,000 feet). The spacecraft was likely destroyed by atmospheric stresses and friction at this low altitude. This malfunction was due to programming error; imperial units (pound-seconds) were used instead of the metric units (newtons) in the navigation software. The computer controlling the spacecraft's thrusters had underestimated the thruster forces by a factor of 4.45 (i.e., one pound-second is equivalent to 4.45 Newtons). This error may have accounted for some of the TCMs required during transit to Mars. A final, optional engine firing to raise the spacecraft's path relative to Mars before its arrival was considered by the mission team but was not performed. The orbiter is thought to have entered the Martian atmosphere at too steep an angle leading to the loss.

Although the software was adapted from an earlier Mars Climate Orbiter project, some testing was not performed prior to launch and navigational data generated by the software was not cross-checked during flight. The total cost of the loss was about $85 million (not including the launch vehicle) of which $80M was for spacecraft development and $5M was for mission operations.

Misplacement of a Satellite by Titan IV B-32/Centaur Launch Vehicle [11,22]: On April 30, 1999, a Titan IV B was launched with a mission to place a Milstar satellite in geosynchronous orbit. The rocket was launched from the Cape Canaveral Air Station (CCAS) in Florida. The launch vehicle is configurable and can be launched with no upper stage or with one of two optional upper stages each providing greater and varied capability. The two types of upper stages are the Centaur Upper Stage and the Inertial Upper Stage; the Centaur upper Stage was used in this mission.

The first stages of flight proceeded without incident with the Centaur separating from the Titan IV B as planned and starting its main engines for the first burn phase. However, after the initial stages of flight, the vehicle began to roll unexpectedly. The Centaur stabilized itself during the subsequent coast phase, but expended 85 percent of the Reaction Control System (RCS) propellant.

During the second burn phase, the vehicle again began to roll, eventually losing pitch and yaw control as well. Due to the premature depletion of RCS propellant, the attitude control system was unable to stabilize the vehicle. Because of the anomalies during the Centaur's flight, the Milstar satellite was placed in an unusable low elliptical orbit, failing to reach the desired geosynchronous orbit. The entire mission failed due to this misplacement; the cost of this lost mission was about $1.23 billion. In addition, NASA had to face severe criticisms due to this mishap being the third straight Titan IV failure at that time.

The Accident Investigation Board concluded the root cause of the Titan IV B-32 mission mishap was the failure of the software development, testing, and quality/mission assurance processes used to detect and correct a human error in the manual entry of a constant. The process allowed for a single point of failure when generating mission critical data.

Loss of Contact with the SOHO [21,23]: The Solar and Heliospheric Observatory (SOHO) was deployed on December 2, 1995 and began normal operations in May 1996 to study the Sun. SOHO is a joint project between the European Space Agency (ESA) and NASA and is also a component of the International Solar Terrestrial Program (ISTP). The three main scientific objectives of the SOHO mission are: (1) investigation of the outer layer of the Sun, (2) making observations of solar wind and associated phenomena, and (3) probing the interior structure of the Sun. SOHO also provides the primary source of near-real-time solar data for space weather observation and prediction.

Originally planned as a two-year mission, SOHO continues to operate after more than ten years in space. In October 2009, a mission extension lasting until December 2012 was approved. The 610 kg SOHO spacecraft is positioned in a halo orbit around the Sun-Earth Lagrange L1 point, the point between the Earth and the Sun where the balance of the Sun's gravity and the Earth's gravity is equal to the centripetal force needed for an object to have the same orbital period in its orbit around the Sun as the Earth, with the result that the object will stay in that same relative position.

Flight controllers at NASA Goddard Space Flight Center (GSFC) lost contact with SOHO in the early morning hours of June 25, 1998. Control of SOHO was eventually recovered 3 months later; SOHO was then re-oriented towards the Sun on September 16 and returned to its normal operations on September 25. However, though control was eventually recovered, the one billion dollar spacecraft was temporarily at stake and there was financial expense associated with recovery operations to establish contact with and return the lost spacecraft to operation.

The incident was preceded by a routine calibration of the spacecraft's three roll gyroscopes (gyros). The incorrect diagnosis of a Gyro B fault and a rapid decision to send a command to turn it off in response to unexpected telemetry readings was a contributing factor to this incident.

With respect to the software-related causes, a modified command sequence in the onboard control software was missing a critical function to reactivate one of the gyros (Gyro A). The absence of this function caused the failure of an Emergency Sun Reacquisition (ESR) process and set in motion the chain of events ending in the loss of telemetry from the spacecraft. Furthermore, another error in the software improperly left Gyro B in high gain mode which generated the false readings that led to the initial inappropriate triggering of the ESR.

Loss of Ariane 5 – Flight 501 [1]: Ariane 5 was an expendable launch system designed to deliver payloads into geostationary transfer or low Earth orbits. On June 4, 1996, the Ariane 5 rocket departed on its maiden flight.

37 seconds after lift-off at an altitude of about 3,500 meters, the Ariane 5 rocket veered off its flight path, broke up, and exploded. The Ariane 5 had undergone a decade of development at an expense estimated at $7 billion. The rocket and cargo losses were valued at $500 million.

Shortly after lift-off, incorrect control signals sent to the Ariane 5 engines caused them to swivel. The resulting stresses on the rocket led to a breakup and onboard systems automatically triggered the self-destruct. This malfunction could be directly traced to a software failure. The cause was a program segment that attempted conversion of a 64-bit floating point number to a 16-bit signed integer. The input value (a 64-bit floating point number related to the horizontal velocity of the rocket with respect to the platform) was larger than 32,767 and outside the range that could be represented by a 16-bit signed integer, so the conversion failed due to an overflow. The software error arose in the active and backup computers at the same time, resulting in the shutdown of both and subsequent total loss of attitude control. Software for the Ariane 5 was reused from the Ariane 4 project. The Ariane 5 was a faster rocket than its predecessor wherein the noted error would now more likely occur when it would not have in the earlier rocket.

Emergency Shutdown of the Hatch Nuclear Power Plant [4,10]: The Edwin I. Hatch nuclear power plant was recently forced into an emergency shutdown for 48 hours, after a software update was installed on an office computer.

The mishap occurred on March 7, 2008 after an engineer installed a software update on a computer operating on the plant's business network. The software update was designed to synchronize data on both the business system computer and the control system computer. According to a report filed with the NRC, when the updated computer rebooted, it reset the data on the control system causing safety systems to errantly interpret the lack of data as a drop in water reservoirs that cool the plant's fuel rods. As a result, automated safety systems at the plant triggered a controlled emergency shutdown.

The Southern Company (which managed the technology operations for the plant) stated that the plant's emergency systems performed as designed and that at no time did the malfunction endanger the security or safety of the nuclear facility and surrounding area. They explained that company technicians were aware that there was full two-way communication between certain computers on the plant's corporate and control networks. However, the engineer who installed the update was not aware that the software was designed to synchronize data between the business system computer and the control system computer and that a reboot in the business system computer would force a similar reset in the control system computer.

Estimating a loss based on electricity prices, the total cost to purchase electricity from external sources during the 48-hour shutdown period could be approximately $5 million. This does not include other operation-related losses.

Power-Outage across Northeastern U.S. and Southeastern Canada [7]: On August 14, 2003 in the heat of the summer at around 4pm EST, parts of the Northeastern United States and Southeastern Canada experienced widespread blackouts. Specifically among the states affected were New York, New Jersey, Vermont, Michigan, Ohio, Pennsylvania, Connecticut, and Massachusetts. Among the major urban agglomerations touched by the electrical power outage in the United States were New York City, Albany, and Buffalo in New York, Cleveland and Columbus in Ohio, and Detroit in Michigan. Ottawa and Toronto in Canada were also affected.

The blackouts resulted in the shutdown of nuclear power plants in New York and Ohio and affected airport operations in the affected states. More than 50 million people were impacted by the outages; the total loss was estimated at $13 billion.

The outage began due to some routine system troubleshooting. A technician disabled a software trigger that launched a 'state estimator' every five minutes; this is normal procedure. Unfortunately, the technician forgot to turn the trigger back on after finishing with the maintenance and the system did not provide a reminder to do so.

With the trigger disabled, the power system began to set off a series of alarms, slowly at first and then faster as the system failure became more widespread. A race condition in the system that set off the alarms developed and caused the alarm software process to lock up. Alarms that were to be sounded began to queue up with no software process to handle them. Because of this the workers in the control room were not alerted to the impending power losses. Eventually this queuing incident halted the server where the process was hosted. An image of the locked-up program and its queue of alarms were moved to a backup server and an automated text was sent to a technician. Since an exact copy of the locked up program was used on the backup server, it did not take long for that server to fail in the same way.

In the interests of better understanding the scale of the blackout, we take a brief look at the infrastructure affected by the blackout. With the power fluctuations on the grid, power plants automatically went into "safe mode" to prevent damage in the case of an overload, which in turn put much of the power normally available off-line until those plants could be slowly taken out of "safe mode." In the meantime, homes and businesses both in the affected and surrounding areas were asked to limit power usage until the grid was back to full power. Several areas lost water pressure. Transportation, communication, and oil and gasoline services were also greatly affected. Looting and civic disturbances were reported in areas such as Ottawa, Ontario and Brooklyn, New York.

3. Lessons Learned

In this section we present the lessons learned from the occurrence of these accidents and the knowledge we can take away from them to prevent their recurrence. Each accident has its own unique and important lessons; however, we choose to present the lessons in this section in a collective form (from what we consider most significant to least). Even though each lesson may be based on a particular accident, this does not imply that the lesson is restricted to that accident alone.

A lesson may be useful in many scenarios (i.e., may avoid multiple accidents) and at the same time multiple lessons may be learned from the same accident. These lessons therefore, act as generally applicable guidelines and principles.

1) **Testing for the bad is as important as testing for the good**: For safety-critical software, in addition to testing what the software should do, test also what the software should not do (i.e., which scenarios should not occur but if they do, test for specific behavior returning the system to a safe state). Instead of using default system exception handlers which may simply result in system shut down, consider customized handlers for various exceptions in order to ensure the safe deterministic operation of critical software systems. Examples evidenced where testing for what the software should not do may have helped are the miscalculated radiation doses at ION and the Ariane 5 incident [1].

2) **Be wary of change impact – small changes can sometimes have big consequences**: When changing any part of a safety-critical software system (no matter how small the change), examine and test how that change will affect the safety of the overall system, i.e., function and exception, interface behavior, timing and others. While this impacts analysis may not uncover every likely issue, it provides a systematic process for dealing with the effects of change to a safety critical software system. An example of small changes affecting critical operations is evidenced in the unintended emergency shutdown of the Hatch nuclear power plant for a business computer update [4,10]. Another supporting example is the Korean Air Flight 801 accident where the MSAW system was inhibited [9,15,18]. A third example is the loss of contact with the SOHO [21,23] where a modified command sequence was missing a critical function to reactivate a

gyro. Also, even well-tested, proven software must still be thoroughly retested and rechecked when reused, adapted, and deployed to a different environment. An example of this is the loss of the Mars Climate Orbiter [19] and the Ariane 5 [1]. Finally, any changes made to the software, and especially safety-critical software, should be properly researched and documented. While this seems like a trivial statement, oftentimes the documentation is forgotten when coupled with aggressive schedules, cost targets, and competing resources. An example of this is the mishap involving NASA's DART [17] where a known fix was not applied to the flight software.

3) **Always have a backup, but there are no guarantees**: For systems where a high degree of safety is of utmost concern, redundancies should always be considered. An example of this is the loss of communication between the FAA Air Traffic Control Center and commercial aircraft [6,13]. As important as this is however, it is also important to note that if a system fails due to a software fault, any identical redundant system using the same software and interfaces to the same equipment will likely fail due to the same software fault. As a result, "*redundancy*" (namely, having an identical backup system which runs the exact same software or using an identical software program used as a backup to a primary program) may not always be an appropriate preventative measure. An example issue with software backup is the power-outage across Northeastern U.S. and Southeastern Canada [7] where the alarm process failed on the primary and backup servers.

4) **User-awareness is critical. Never take safety for granted**: In the case of safety-critical systems, one cannot assume that a computer automatically ensures safe system operation. When computing systems provide interaction with the environment, inputs are read and computer responses selected coincident with prior programming. When human users interact with computer systems, the humans are responsible for maintaining adequate situational awareness taking responsibility and accountability for the state changes of the system in which the context is taking place. Accidents can occur with safety critical systems in this case due to human distraction or lack of situational awareness within the context of system decision. An example of this loss of situation awareness is evidenced with the American Airlines Flight 965 accident [14] where the selection of the wrong waypoint allowed the aircraft to steer into the path of terrain. Another example is the miscalculated radiation doses at the National Oncology Institute in Panama [2,5].

5) **Don't leave well enough alone**: When a system has a known problem that could affect its safe operation, it is never a good idea to let the issue persist indefinitely. Instead of relying on an improvised workaround, the error in the software should be corrected as soon as possible and updated software tested, released, and deployed to avoid a potential crisis. An example of this is the loss of communication between the FAA air traffic control center and aircraft [6,13] due to a field representative not resetting software in the VCSC before the internal clock reached 0. In this case, a manual procedure allowed the software operation to continue until a repair could be effected, but the repair did not actually occur until after the incident had already occurred.

6) **Success during simulations does not imply success during operation**: Whenever possible, safety-critical software should undergo integration and final testing using real equipment rather than simulations alone. If a system is to be validated using simulation or other analyses, care must be taken to ensure that the models underlying the analysis are suitably accurate and well-tested. Also, simulated stress testing and fault injection (i.e., using inputs just outside allowed range values, using unallowed inputs, etc.) should be an integral part of the software testing process to discover latent faults, establish the limits of the system, and verify deterministic behavior in the presence of faults. The loss of the Mars Polar Lander [8,16] illustrates this lesson where a false landing indication on a sensor may have caused the spacecraft loss on landing maneuver. A system with well-defined limits of operation could fail when it is operated outside of those limits. Just because an anomaly does not show up in thousands of hours of testing, it in no way implies that it may not show up in practice. An example of this is the lack of precision software bug that led to the Patriot Missile system failure at Dhahran, Saudi Arabia [20]. The error would not have occurred if the system was reset before the accumulated error would be large enough to cause the malfunction [20].

7) **Never pick user-convenience over software safety**: While functionality of a system that makes its use more convenient for users is desirable, it needs to be carefully analyzed and scrutinized to ensure this new functionality does not result in a new unmitigated potential risk. For safety-critical systems where new functionality must integrate with or provide new safety-critical functionality, an impact analysis should be performed to quantify the extent to which the current system is affected prior to effecting required changes. An example of a user interface providing undocumented behavior is the miscalculated radiation doses at the National Oncology Institute in Panama [2,5] where the physicians tried to define a 5-block mask instead of working within the limits of the software. Another example is the American Airlines (AA) Flight 965 [14] accident involving the selection of waypoint mechanization of the flight management system software.

8) **Software may not always be used in the same way it is tested; improper use may lead to catastrophic consequences**: Software (especially safety-critical software) should be designed to avoid implementation or usage ambiguities. Although flexible software may be desired, the implementation must be accomplished such that when users run the software based on their own

mental assumptions, the software operates as expected and when it does not, appropriate warning messages are displayed to remind users of the potential risk of such exercises. In the case of the miscalculated radiation doses at the National Oncology Institute in Panama [2,5], the physicians used the software in a way that that caused the system to generate a lethal radiation dosage. No warning messages were generated to alert the system user of the result. Yet, the radiation treatment system allowed the selection to be made.

9) **Having a clearly defined system boundary is essential**: System boundaries should be carefully defined and interfaces setup appropriately such that systems external to the safety-critical system can be appropriately supported within a managed risk approach. Providing this managed interface provides a mechanism to allow communication between the systems if warranted while restricting operations between the systems that could adversely affect safety-critical operations. The emergency shutdown of the Hatch nuclear power plant [4,10] is an example of this system boundary being sufficiently broad to allow an operation outside the system boundary (i.e., a business computer) to adversely affect the operation of the power plant.

10) **Safety-critical software should not be overly-sensitive to erroneous data**: There are times within a computer system when inputs to an operation may fall outside the designed boundaries of the expected input parameters. In computer systems where sufficient exception handling is not customized by the programming team, resultant operation may not be deterministic under certain circumstances. For safety-critical systems, deterministic behavior is expected. In that regard, development testing should be structured to ensure that signals (i.e., variables, decision constructs, etc.) are tested in normal and abnormal range conditions to ensure deterministic system behaviors. This lesson is learned based on the mishap involving NASA's Demonstration of Autonomous Rendezvous Technology (DART) [17] where successive system resets eventually depleted the spacecraft resources.

11) **Whenever a critical software system is using *sample* data for testing purposes, the system should alert the end users.** Sample data is typically used in computer systems to demonstrate correct operation of the system under known input conditions when it is inconvenient or not generally possible to use live data instead. Sample data is used to this end as representative data that one may expect to experience in operation with the system. This sample data should be carefully chosen and a thorough testing effort should be conducted to ensure that appropriate alert messages are displayed indicating a test in progress or that sample data is being used. In the Hartsfield-Jackson Atlanta International Airport [3] incident, baggage screeners looking for contraband items were misinformed by the computer system that a suspicious item was detected. Whether this incident was caused by a software bug or not, appropriate messages should have prevented this mishap.

12) **Engineers developing safety-critical software must have a comprehensive understanding of the overall software development process**. While this statement seems obvious, all too often the details of the software development process to most go unchallenged and in some cases, unknown. Engineers charged with the development of safety-critical software, software engineers, quality assurance engineers, systems engineers, and others, should be familiar with the overall software process and for those where it matters, deeply knowledgeable of the mechanics of lower level processes and workings of tools. Further, the process should be followed and audited. Performance to the defined software development process provides assurance to the product development team that the engineering development of the software is managed. Following the process is of critical importance; there are many examples of projects not following their software process. Within the accidents and mishaps reported here, the misplacement of a satellite by a Titan IV B-32/Centaur Launch Vehicle [11,22] serves as one example of the consequence of not performing required process work in developing and testing the filter constants for the flight software.

4. Discussion

In this section we discuss the reasons behind the choice of the 14 accidents presented in this study and why they were categorized as they were. Additionally we also address the issue that software is typically not the sole cause of the accidents or mishaps in which software was identified as a causal factor. Software behavior may, however, be in the chain of events that ultimately leads to the mishap or accident.

4.1 Rationale Behind Choice of Accidents Studied

No paper regardless of breadth or depth can hope to cover every accident that may be relevant to the subject matter presented herein. The accidents that have been chosen for analysis were selected on the basis of several criteria˜ the extent of the damage, the extent to which software played a contributing cause, and the recentness of the accidents. While this paper does not claim to be a complete listing of such accidents, it does however claim to provide a representative picture of how faulty software, or software that has been used in a faulty manner, can lead to loss of time, money, and gravely so, under certain circumstances – loss of life as well.

4.2 Software: 'sole cause' versus 'contributing cause'

We also point out that software may not be the sole cause for the accidents and that quite often the causes are a combination of software errors, human mistakes, and/or hardware failures. Factors such as poor communication, negligence, and oversight to name a few, have also been identified as past contributors. However, undoubtedly, software has directly or indirectly played a causal role in each of the accidents discussed in this paper. This is why the tone of the paper has been primarily restricted to the software-related causes of the accidents.

4.3 The Categorization of the Accidents Studied

As stated in Section 2, there are many different ways to classify or categorize the accidents studied and other similar accidents. Examples of such classifications may be based on the financial cost incurred, the nature of the software fault, the area/domain of the accident (such as nuclear research, aeronautics, etc.), and various others. However, such classifications come with a great deal of subjectivity and are open to individual interpretation. An accident may also fit more than one category. To avoid such complications and possible ambiguities, we partitioned the accidents based on whether the cost resulted in the loss of life or not. Not only is this factual and non-subjective, but it also holds additional benefit in that the accidents categorized under *loss-of-life* illustrate the serious nature of the subject matter and the importance of safety in software both in the past and in our future as software becomes even more prolific within our societal products.

5. Summary

This paper reviews 14 recent catastrophic accidents, several of which have led to not just losses in terms of time and money, but also the loss of life. In the case of each accident, the role played by software in causing the accident has been identified. While the accidents may not have been caused by the software alone, and may have been compounded by human error and/or hardware failure, nevertheless software has had a hand in the causal chain of events. This is especially alarming given that software systems are being increasingly used in our day-to-day lives, including safety-critical systems.

By virtue of our review of these accidents, we also identify lessons learned that can act as principles and guidelines so that similar accidents might be avoided in the future. The identified lessons taken together suggest that better verification and validation activities and practices need to be employed more systematically.

Further research needs to be conducted in the area of software safety in order to improve and add to the existing body of knowledge on the subject. Software engineering curriculums need to undergo significant updates in order to better impart the fundamentals of software testing and software safety to students who go on to become our future industry practitioners.

Acknowledgements

This research is supported in part by the Net-Centric Industry University Collaborative Research Consortium (IUCRC) industrial memberships and in part by National Science Foundation (NSF) funds for the net-centric IUCRC and for the University of Texas at Dallas Research Experiences for Undergraduates (REU) grant (CCF-0851848).

References

1. Ariane 501 Inquiry Board, “Ariane 5, Flight 501 Failure,” July 1996
2. C. Borrás, “Overexposure of Radiation Therapy Patients in Panama: Problem Recognition and Follow-up Measures,” *Rev Panam Salud Publica*, 20(2/3):173–87, 2006
3. CNN, “TSA: Computer Glitch Led to Atlanta Airport Scare,” April 21, 2006
4. Environmental Protection agency, “Clear Skies in Georgia”
5. Food and Drug Administration of U.S.A., “FDA Statement on Radiation Overexposures in Panama”
6. L. Geppert, “Lost Radio Contact Leaves Pilots on Their Own,” *IEEE Spectrum*, 41(11):16-17, November 2004
7. Great Northeast Power Blackout of 2003 http://www.globalsecurity.org/eye/blackout_2003.htm
8. JPL Special Review Board, “Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions,” March 2000
9. S. S. Krause, “Aircraft Safety: Accident Investigations, Analyses and Applications,” second edition, McGraw-Hill 2003
10. B. Krebs, “Cyber Incident Blamed for Nuclear Power Plant Shutdown,” Washington Post, June 5, 2008
11. G. L. Lann, “Is Software Really the Weak Link In Dependable Computing?” The 41st IFIP WG 10.4 meeting, Saint John, US Virgin Islands, January 2002 (Workshop on “*Challenges and Directions for Dependable Computing*”)
12. N. Leveson, “Safeware: system safety and computers,” Addison-Wesley, September, 1995.
13. Los Angeles Times, “FAA to Probe Radio Failure,” September 17, 2004
14. M. Nakao, “*Crash of American Airlines Boeing*”, December 1995
15. National Transportation Safety Board, “Abstract on Korean Air Flight 801Conclusions, Probable Cause, and Safety Recommendations,” NTSB/AAR-99/02, August 1997
16. NASA, Mars Polar Lander official website
17. NASA, “Overview of the DART Mishap Investigation Results,” May 2006 (www.nasa.gov/pdf/148072main_DART_mishap_overview.pdf)
18. Official Guam Crash Site Information Web Center (http://ns.gov.gu/guam/indexmain.html)
19. Report: The Mars Climate Orbiter Mission Failure Investigation Board (http://marsprogram.jpl.nasa.gov/msp98/news/mco991110.html)
20. E. Schmitt, “Army is Blaming Patriot’s Computer for Failure to Stop Dhahran Scud,” New York Times, May 20, 1991
21. SOHO Mission Interruption Joint NASA/ESA Investigation Board, “Final Report,” August 1998
22. USAF Accident Investigation Board, “Titan IV B-32/Centaur/Milstar Report” (http://sunnyday.mit.edu/accidents/titan_1999_rpt.doc)
23. K. A. Weiss, N. Leveson, K. Lundqvist, N. Farid, and M. Stringfellow, “An Analysis of Causation in Aerospace Accidents,” in Proceedings of the 20th Digital Avionics Systems Conference, Volume: 1, pp. 4A3/1-4A3/12, Daytona Beach, Florida, USA, October 2001