The Journal of Systems and Software 133 (2017) 68–94

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures

W. Eric Wong[a,b,*], Xuelin Li[b], Philip A. Laplante[c]

[a] School of Computer Science and Engineering, University of Electronic Science and Technology of China, China
[b] Department of Computer Science, The University of Texas at Dallas, USA
[c] Software and Systems Engineering, The Pennsylvania State University, USA

ARTICLE INFO

Article history:
Received 8 March 2017
Accepted 23 June 2017
Available online 30 June 2017

Index Terms:
Accidents
Bugged software systems
Software failures
Mishaps
Lessons learned

ABSTRACT

There has been an increasing frequency of failures due to defective software that cost millions of dollars. Recent high profile incidents have drawn increased attention to the risks of failed software systems to the public. Yet aside from the Therac-25 case, very few incidents of software failure causing humans harm have been proven and widely reported. With increased government oversight and the expanded use of social networking for real time reporting of problems, we are only beginning to understand the potential for major injury or death related to software failures. However, debugging defective software can be costly and time consuming. Moreover, undetected bugs could induce great harm to the public when software systems are applied in safety-critical areas, such as consumer products, public infrastructure, transportation systems, etc. Therefore, it is vital that we remove these bugs as early as possible. To gain more understanding of the nature of these bugs, we review the reported software failures that have impacted the health, safety, and welfare of the public. A focus on lessons learned and implications for future software systems is also provided which acts as guidelines for engineers to improve the quality of their products and avoid similar failures from happening.

© 2017 Elsevier Inc. All rights reserved.

1. Introduction

Software is fundamental to our society and pervades our daily lives. Regardless of age, gender, occupation, or nationality, each of us depends on software, either directly or indirectly. Yet the disappointing truth is that software is far from defect-free and large sums of money are spent each year to fix or maintain (defective) software. According to a National Institute of Standards and Technology report (NIST Report, 2002), software bugs cost the U.S. economy an estimated $59.5 billion annually.

Software plays an important role in some of humanity's most complex systems, especially in *safety-critical* application areas such as aeronautics, astronautics, medicine, nuclear power generation, nuclear energy research, and transportation industries. When employed in such systems in these and other industries, software is often responsible for controlling the behavior of electromechanical components and monitoring their interactions. Since most accidents arise due to a lack of understanding of the requirements or in the interfaces and interactions among the components (Leveson, 1995), software quality and reliability can have significant impacts on the overall performance and reliability of safety-critical systems.

Consequently, an important issue that comes up in our research efforts is that of *software safety*. Ideally, software should operate as expected and not contribute to hazards (Leveson, 1995). However, this is not always the case since software systems are getting larger and much more complex with algorithms, internal and external interactions, timing, and general processing than ever before. As a consequence, for example, if system operations render the software to an unsafe state or software operations directly or indirectly present or lead to a system hazard (as in the case of a safety-critical system), the consequences could range from a mere minor performance anomaly to a catastrophic accident with not only loss of money and property, but also possible loss of human life.

Another significant issue is that the later a bug is detected, the more expensive it is to eliminate the bug. Boehm, (1981) studied the cost factors associated with finding and fixing errors with respect to different software life cycle phases. The results showed that the cost to find and fix a bug after delivery can be upwards of 100 times more expensive than resolving it during the early design phase. In other words, the later a bug is revealed, the more activities are required to revalidate the correction. Therefore, improving the quality assurance process during the entire software lifecycle is critical in order to enhance the quality and reliability of software systems and to save time and efforts in debugging the defective software systems.

* Corresponding author.
E-mail address: ewong@utdallas.edu (W.E. Wong).

http://dx.doi.org/10.1016/j.jss.2017.06.069
0164-1212/© 2017 Elsevier Inc. All rights reserved.

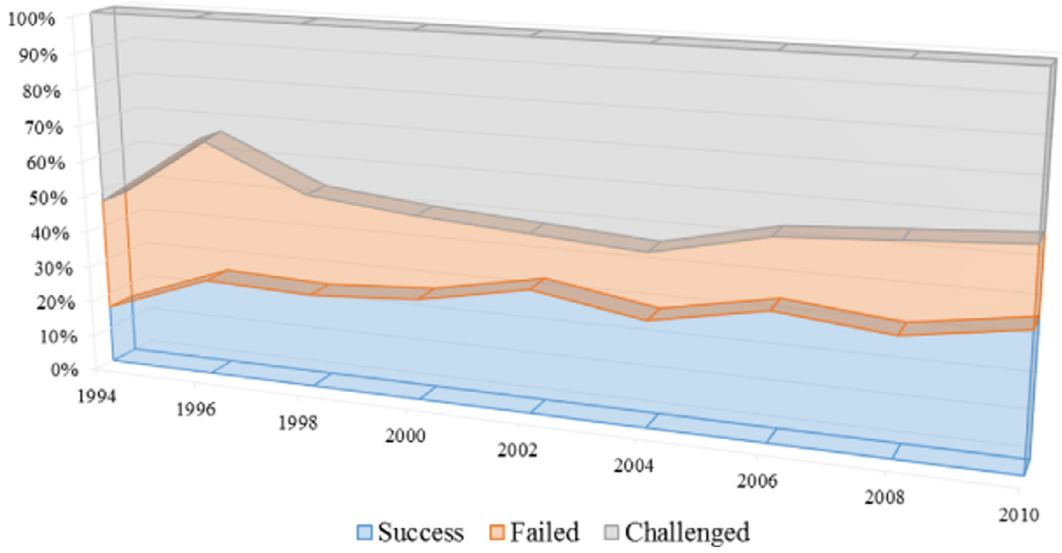

Fig. 1. Software projects outcomes status from 1994 to 2010.

One way to address this is to consider licensing software engineers. Regulation and licensure for engineers has already been established in various areas such as civil engineering, mechanical engineering, electrical engineering, etc. The aim of licensing engineers is to encourage public welfare, safety, well-being and other interests of the general public, and to define the licensure process through which an engineer becomes authorized to practice engineering and/or provide engineering professional services to the public. However, in public discussions pertaining to licensing of software engineers in the U.S., some contributors opine that there exist few actual cases of software failure causing humans harm (Laplante, 2012, 2013). While the community acknowledges high-profile failures like Therac-25, a radiation therapy machine that gave massive overdoses of radiation to certain cancer patients (Leveson and Turner, 1993), some might argue that such dangerous failures are extremely rare, special cases. To determine the veracity of this claim, we analyzed numbers of fatal accidents caused or induced by software and found that software-caused human fatalities do in fact occur with some frequency.

It is known that some system failures are caused or induced by software. For example, Perrow (2008) suggests that catastrophic failures involving significant harm to humans are inevitable as software becomes ever more ubiquitous. Perrow warns that it may be only a matter of time – 5 or 10 years perhaps – before we have a software failure that kills 1000 people or more. As yet, our software systems have proven to be robust, and we have not encountered a software failure of the magnitude Perrow fears. In addition, these researchers tend to overestimate those possible impacts and neglect the fact that the quality and reliability of software also improve. Based on the Standish Group's Chaos Summary (The Standish Group International, Inc. 1999, 2010), the success rate of software projects increased significantly since 1994 to 2010, as illustrated in Fig. 1. Also, the failed rate also decreased.

Moreover, Perrow also interprets certain software failures leading to injury or death as *management failures*. For example:

> Here is a tragic case of a management failure. In the 1991 Gulf War, 28 U.S. troops were killed when the Patriot air defense system missed an incoming Scud missile (E. Schmitt, 1991a,b). The internal counting system has a tiny rounding error that made no difference for slow targets – such as airplanes – but that mattered for missiles if the errors accumulated over an extended operational time. An upgrade to deal with missiles was made, but at one place in the upgraded software, a necessary call to the subroutine was accidentally omitted. This problem was detected, a warning issued, and a software patch dispatched to users more than a week before the incident. Rebooting would correct the error, and only takes a minute. But the battery at Dhahran was in uninterrupted operation for over 100 hours, the discrepancies accumulated and the system failed to intercept the incoming missile. The patch for the system arrived the next day, after a Scud missile has already killed the 28 soldiers.

Similarly, if we accept such an interpretation, the following incident caused by software failure can also be viewed as a management failure:

> 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 (Geppert, 2004; Los Angeles Times, 2004). 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 aircraft in the flight monitored airspace when the primary voice communications system shut down unexpectedly. More precisely, it was a Voice Switching and Control System (VSCS) that failed. 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.
>
> The software in the VSCS system used a 32-bit countdown timer that decremented every millisecond. This allowed for $2^{32}-1$ ms 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} ms (approximately 50 days) to prevent data overload. After this accident, the FAA deployed a software fix to the VSCS which addressed this countdown timer issue.

Based on Perrow's interpretation, not only all software failures (including failure to test sufficiently, failure to build in sufficient fault-tolerance, failure to upgrade the software, etc.) but also hardware failures could be attributed to management failures. Obviously, these management failures are not acceptable explana-

tions since they do not address the root causes specifically which include hardware and/or software as contributing sources of the system failure. Various factors exist in directing the software in these systems to an *unstable* state. These factors cannot be counted as management issues since the real causes for these failures could be effectively unknown, making it difficult to duplicate the issue and find corrections, or to provide insights on how to improve the in design, development, and/or quality assurance processes associated with the software engineering effort.

In order to highlight the ever growing importance of software in the failure analysis of systems, we select a representative set of 59 recent catastrophic accidents and present a brief summary of them and their software-related causes (whether direct or indirect). During this research effort, 104 accidents[1] were examined during the data collection phase and those accidents which are not related to software are excluded from the accidents pool of this study. More importantly, we present the causes of each of the selected accidents, as found by the formal documented accident investigations or by performing our own root cause analysis and discuss any lessons learned. Thus, the contributions of this paper are two-fold: first, the summary of 59 accidents and the underlying causes (either direct or indirect) are presented to emphasize the importance of software especially in the field of safety-critical systems; second, we provide a summary of the lessons learned from these accidents as well as recommendations which we hope will help us better understand the role that software played in these accidents and help us drive appropriate changes to improve our development practices associated with safety-critical software and systems development.

The remainder of the paper is organized as follows: in Section 2, we list several previous works related to our study. A brief summary of accidents caused by software failures is presented in Section 3. In Section 4, the losses induced by the accidents are identified and described. In Section 5, the contributing factors, either software-related or non-software related, are presented. In Section 6, lessons learned from the previous stated accidents are identified and summarized. Threats to validity of this research is discussed in Section 7 and finally our conclusions are drawn in the final section.

2. Related and previous works

In this section, we provide a brief review of publications related to this paper. In Section 2.1, surveys on software accidents are introduced; in Section 2.2, we include the description of several papers concentrating on legal perspectives of licensing software engineers based on accidents incurred by software.

2.1. Surveys on software accidents

Perhaps the most comprehensive source for anecdotal information on software failures is Peter Neumann's *Risk Forum* (Neumann, 2014). First appearing as a regular column in the *ACM Software Engineering Notes*, the forum was later transformed to a searchable database, which contains over 2000 reports of software failures from 1985 to the present. These listings include general discussion of software failures, incident reports, and speculations. Sources of information for these reports include:

- Voluntarily reported failures (e.g., preemptory press releases)
- Corporate documents
- News stories
- Lawsuits
- Government announcements

Rahman et al. (2009) studied 12-years of data from the Risks Forum, focusing specifically on critical infrastructure systems as defined in a congressional mandate (Moteff and Parfmak, 2004), namely:

- IT infrastructure
- Telecommunication infrastructure
- Water supply
- Electrical power system
- Oil and gas
- Road transportation
- Railway transportation
- Air transportation
- Banking and financial services
- Public safety services
- Healthcare system
- Administration and public services

Using the fault classes Rahman, et al. (2014) defined (Table 1), they identified certain "infrastructure failure patterns, propagation, impacts on public life, and historical trends." However, they discussed no specific cases of software failure.

Wallace and Kuhn (1999) examined 342 medical device failures, focusing on "devices that have been recalled by the manufacturers due to software problems." Using U.S. Federal Drug Administration (FDA) data, they characterized the reported failures along thirteen primary symptoms resulting in distributions shown in Table 2.

In his widely cited paper (Charette, 2005), Charette lists 30 non-lethal software "fiascoes" (see Table 3) of cancelled or failed projects that resulted in losses of tens of millions to $4 billion. Charette did not focus on catastrophic failures involving serious injury or loss of life. Rather, he emphasized management failures, specifically poor risk management. Notably, he predicted that the U.S. Healthcare.gov would be a financial failure with the following statement which seems to be quite prescient:

"But this approach is a mere pipe dream if software practices and failure rates remain as they are today. Even by the most optimistic estimates, to create an electronic medical record system will require ten years of effort, $320 billion in development costs, and $20 billion per year in operating expenses–assuming that there are no failures, overruns, schedule slips, security issues, or shoddy software. This is hardly a realistic scenario, especially because most IT experts consider the medical community to be the least computer-savvy of all professional enterprises."

More recently, Wong et al. (2010) reviewed fifteen "recent catastrophic accidents". Some led to loss of human life or serious injury such as the crash of American Airlines Flight 965 on December 20, 1995, the crash of Korean Air Flight 801 on August 5, 1997, the crash of Air France Flight 447 on May 31, 2009, and radiation overdoses administered at the National Oncology Institute of Panama. Others of the "catastrophic failures" did not cause death or injury, at least not directly. For example, the shutdown of the Hartsfield–Jackson Atlanta International Airport on April 19, 2006 due to software caused false alarm of a Transportation Safety Administration (TSA) imaging scanner. Similarly, they describe the possible software causes of the cross-power grid outages in Northeastern US and Southeastern Canada on August 14, 2003. But these and similar incidents may have had second-order effects that led to human suffering or death, though this is not established. In the three named cases described above, the cause of each catastrophic accident also included either a mechanical failure or

[1] The detailed list of these accidents can be retrieved from http://paris.utdallas.edu/accident-survey/.

Table 1
Fault classes related to critical infrastructures.

Fault class	Description
Hardware fault	All fault classes that affect hardware
Software fault	Fault caused by an error in the software system
Human error	Non-deliberate faults introduced by a mistake
Natural fault	Physical faults that are caused by natural phenomena without human participation
Overload	Service demand exceeds the designed system capacity
Vandalism	Sabotage or other intentional damage
Malicious logic fault	These include trojan horses, logic or timing bombs, viruses, worms, zombies or denial of service attack
Authorization violation	Attempt by an unauthorized person to access or damage network resources, but does not exclude the possibility of authorized users who are exceeding their rights. This also includes unauthorized sharing of digital contents, like audio, video or software

Table 2
Wallace and Kuhn Failure Symptom Taxonomy (1999).

Symptom	Description	Percentage of devices showing as primary symptom
Behavior	Device performed an erroneous action	22
Data	Corruption or loss of data	1
Display	Incorrect numbers, text or images shown to users	8
Function	Incorrect calculation or activity	29
General	Not enough information to assign to another category	0
Input	Incorrect information typed by a human, sampled or read from another device	4
Output	Incorrect information sent to another function within the device	19
Quality	User reported “quality” requirements were not met	1
Response	Something happened in the system that should not have	3
Services	An identifiable system service involving multiple functions	10
System	Total system failure	1
Timing	Timing error in a service or device of the system	1
User instruction	Error in a user manual or other documentation	1

Table 3
Notable non-fatal software failures (The original table was published in (Charette, 2005) and reproduced as follows).

Year	Company	Outcome (costs in U.S. dollars)
2005	Hudson Bay Co. [Canada]	Problems with inventory system contribute to $33.3 million[a] loss.
2004–05	UK Inland Revenue	Software errors contribute to $3.45 billion[a] tax-credit overpayment.
2004	Avis Europe PLC [UK]	Enterprise resource planning (ERP) system canceled after $54.5 million[b] is spent.
2004	Ford Motor Co.	Purchasing system abandoned after deployment costing approximately $400 million.
2004	J Sainsbury PLC [UK]	Supply-chain management system abandoned after deployment costing $527 million[b].
2004	Hewlett-Packard Co.	Problems with ERP system contribute to $160 million loss.
2003–04	AT&T Wireless	Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million.
2002	McDonald’s Corp.	The Innovate information-purchasing system canceled after $170 million is spent.
2002	Sydney Water Corp. [Australia]	Billing system canceled after $33.2 million[b] is spent.
2002	CIGNA Corp.	Problems with CRM system contribute to $445 million loss.
2001	Nike Inc.	Problems with supply-chain management system contribute to $100 million loss.
2001	Kmart Corp.	Supply-chain management system canceled after $130 million is spent.
2000	Washington, D.C.	City payroll system abandoned after deployment costing $25 million.
1999	United Way	Administrative processing system canceled after $12 million is spent.
1999	State of Mississippi	Tax system cancelled after $11.2 million is spent; state receives $185 million damages.
1999	Hershey Foods Corp.	Problems with ERP system contribute to $151 million loss.
1998	Snap-on Inc.	Problems with order-entry system contribute to revenue loss of $50 million.
1997	U.S. Internal Revenue Service	Tax modernization effort canceled after $4 billion is spent.
1997	State of Washington	Department of Motor Vehicle (DMV) system canceled after $40 million is spent.
1997	Oxford Health Plans Inc.	Billing and claims system problems contribute to quarterly loss; stock plummets, leading to $3.4 billion loss in corporate value.
1996	Arianespace [France]	System specification and design errors cause $350 million Ariane 5 rocket to explode.
1996	FoxMeyer Drug Co.	$40 million ERP system abandoned after deployment, forcing company into bankruptcy.
1995	Toronto Stock Exchange [Canada]	Electronic trading system canceled after $25.5 million[c] is spent.
1994	U.S. Federal Aviation Administration	Advanced Automation System canceled after $2.6 billion is spent.
1994	State of California	DMV system canceled after $44 million is spent.
1994	Chemical Bank	Software error causes a total of $15 million to be deducted from 100,000 customer accounts.
1993	London Stock Exchange [UK]	Taurus stock settlement system canceled after $600 million[c] is spent.
1993	Allstate Insurance Co.	Office automation system abandoned after deployment, costing $130 million.
1993	London Ambulance Service [UK]	Dispatch system canceled in 1990 at $11.25 million[c]; second attempt abandoned after deployment, costing $15 million[c].
1993	Greyhound Lines Inc.	Bus reservation system crashes repeatedly upon introduction, contributing to revenue loss of $61 million.
1992	Budget Rent-A-Car, Hilton Hotels, Marriott International, and AMR [American Airlines]	Travel reservation system canceled after $165 million is spent.

[a] Converted to U.S. dollars using current exchange rates as of press time.
[b] Converted to U.S. dollars using exchange rates for the year cited, according to the International Trade Administration, U.S. Department of Commerce.
[c] Converted to U.S. dollars using exchange rates for the year cited, according to the Statistical Abstract of the United States, 1996.

human error. In the case of the radiation overdoses, 23 patients eventually died by September 2005; at least 18 of these deaths were directly attributed to the administered radiation overdoses. Three radiological technicians were indicted for murder for failing to manually verify the required treatment plan; two of them were found guilty who were sentenced to four years in prison and barred from practicing their profession for seven years.

Finally, in a short paper, Zhivich and Cunningham (2009) discuss three cases of catastrophic failures including the Patriot Missile Defense System failure, the Panama City radiation incident, and the Bellingham oil pipeline rupture.

In the 1991 case, a Patriot Missile Defense System failed to intercept an Iraqi missile, which killed 28 American soldiers and wounded many others (E. Schmitt, May 20, 1991a,b). Incident analysis showed that a necessary software patch, which would have caused the system to intercept the incoming missile, had not arrived in time to be installed.

A more recent case of deadly overdoses in radiation treatment occurred at the Instituto Oncologico Nacional in Panama City in 2001 (Borrás, 2006). Treatment-planning software from Multidata Systems International resulted in incorrectly calculated radiation dosages. Twenty-eight patients received excessive amounts of radiation, with fatal consequences for several. Operators triggered the software error by attempting to overcome the system's design limitation in the number and configuration of shielding surfaces used to isolate an area for irradiation. The operators found that by drawing an area with a hole in it, they could get the system to dispense the right dosage in the correct location. However, unknown to them, drawing such a surface in one direction resulted in a correct calculation, whereas drawing the surface in the other direction resulted in an overdose. We can't blame the software alone for these incidents—the operators were supposed to perform manual calculations to ensure that the dosage the software computed was appropriate. They ignored this important check due to lax administrative procedures at the medical institution.

The final case involved the 1999 rupture of an oil pipeline in Bellingham Washington, killing two (Olympic Pipeline explosion). While the pipeline rupture was caused by mechanical failures, it is alleged that a faulty software monitoring system should have indicated the problem before the accident which would have warned the operators to address the mechanical failures before the accident.

Zollers et al. (2004) discuss several interesting cases of catastrophic software failures leading to human injury or death mined from the Software Risks Digest, including a case in which the driver of a Honda CRV trapped in a flood died when his electrically driven windows would not wind down, presumably from a software error. In an Australian case, a boy was killed when he was hit by a truck with defective brakes. The truck Was not supposed to be allowed to be driven. However, a valid registration was granted due to a software failure in the vehicle license registration system. In March 2003, a Royal Air Force Tornado supersonic attack aircraft was struck by a missile fired by a U.S. "Patriot" air missile defense system resulting in the deaths of two flight lieutenants. In May 2004, the United Kingdom defense minister admitted that the tornado software failed to identify itself as friendly and was then classified as an enemy rocket by the Patriot battery, which promptly shot it down.

The Zoller cases are notable in that the fatalities resulted from complex chains of events either started or exacerbated by the software failure, and not directly through a specific single software failure.

Software failures in medical devices seem to pose a greater threat to the public. Wallace and Kuhn (1999) studied FDA data and counted 383 medical instrument recalls that had been caused by specific software problems. Referring to the device recalls on the FDA's list, many of these failures, although not explicitly stated, can be related to software errors (U.S. Food and Drug Administration, 2013).

2.2. Legal perspectives

While software engineers have been debating the relative merits of licensing software professionals for quite some time (e.g. Speed, 1995 and Laplante and Thornton, 2011), it seems that lawyers have spent a great deal of time thinking about the tort and liability issues of software failure. Several significant studies of this aspect of software failure can be found in the literature. Perlman (1998), for example, notes that the once "exotic" uses of the computer are now "an essential part of modern business every day." This means that software is so ingrained in everyday life, that it can and does affect human activities sometimes to the adverse result of deaths or injuries:

> Most computer-related transactions are considered a sale of "goods" and, as a result, are covered under the provisions of the Uniform Commercial Code. The U.C.C.'s warranties, along with the express terms of the contract, preempt the possibility of a negligence claim that might have been brought under a tort theory of liability. Therefore, negligence claims will not commonly be applied to measure the standard of performance between the immediate contracting parties. Moreover, negligence claims are generally not proper where only economic losses are involved. Contracts covering software agreements typically include strategic standard form exculpatory terms such as integration clauses, warranty disclaimers, and provisions commonly limiting remedies to the repair or replacement of defects. The use of these contract provisions shifts the risk of software failure from the seller to the user. (Office of the Under Secretary of Defense for Acquisition September, 1987)

Perlman recounts a set of potential disastrous incidents given by Phillips (1994) including:

- Malfunctioning process control software causing explosions, leaks, etc. at chemical or nuclear plants; example, workers splattered with molten steel
- Malfunctioning software causing truck, train, or plane collisions; example, brake failure of automatic pilot and course direction instruments
- Malfunctioning software used in medical applications such as medical monitoring or diagnosis
- Malfunctioning software causing improper architectural/stress analysis
- Malfunctioning software causing an industrial transport robot delivering parts to a machine to crush an unsuspecting worker

Referring to the following, Perlman appears to be one of the first lawyers to have addressed the issues of professional practice and liability for software engineers or computing professions (Perlman, 1998):

> Case law pertaining to lawsuits involving computer-related disputes demonstrates that the courts have not yet expressly accepted "computer malpractice" as a new tort theory of recovery.

Perlman's observation seems true today; as recently as 2008, the U.S. Supreme Court prohibited patients harmed by defects in FDA-approved devices from seeking damages against manufacturers (Riegel v. Medtronic, Inc., 552 U.S. 312, 2008). This ruling may extend to other critical systems.

Brady (2000) also provides a comprehensive review of tort and legal liability in software failures, but discusses no specific cases or incidents. Similarly, Pinkney (2002) reviews these legal issues and provides a brief summary of several cases. Three of the cases

involve software hacker attacks or failures resulting in significant monetary losses to MCI, Citibank, and Revlon. Other cases involve a cyberattack on a telephone loop at the U.S. Air Force lab in Rome, New York, enabling a further attack on NASA and delaying the launch of the Space Shuttle Atlantis in 1997.

3. Summary of accidents

To better understand the role that software has played in various accidents, we need to first examine some of the technical details of selected sets of public accidents, including the chain of events leading up the event. Understanding the likely actions and causes within the chain of events, we then identify and generalize candidate actions needed to avoid such accidents in the future.

We have categorized our selected publicly reported accidents into two groups based on whether an accident resulted in casualties. Each accident varies by its product architecture, functionality, and technology and will be described based on its location[2] and date, scenario, software-related causes, non-software-related causes, and losses. In selecting these accidents, we removed those accidents from the list that did not have reported clear causes as these could not add to our understanding of specific deficiencies or errors from which lessons-learned could be derived. Also note that the summaries provided below are a brief synopsis of the accidents and are intended to repeat the details of reports; for full accounting of the accident suggest reviewing details of the references provided for each. We start our discussion with accidents involving the loss of life.

3.1. Accidents involving loss of life

AA01: Lufthansa flight 2904 (Lann, 2002; Lufthansa Flight 2904):
Location and Date: Warsaw Chopin Airport, Poland, September 14, 1993.
Scenario: Due to the outdated weather report, the aircraft hit the ground far beyond the normal touch down point. In addition, the computer logic prevents the activation of ground spoilers and engine thrust reversers. The residual length of the runway was not enough to enable the aircraft to stop completely under this situation. The aircraft hit the embankment and an LLZ aerial with the left wing.
Software-related causes:
The minimum compression load on each main landing gear strut to determine whether the aircraft landed or not is set to 6.3 tons in the software for A320-211. However, the condition was not fulfilled until eight seconds after both struts have already touched the ground. After the accident, the minimum compression load was re-configured to 2.0 tons to avoid similar accidents from happening.
Non-software-related causes:
The meteorological conditions were delayed for about three minutes in compared to the real time. However, the wind readout obtained from DFDR did not include the declared time correction.
Losses: Two fatalities (one crew member and one passenger) and 56 injures (51 were seriously injured).

AA02: China airlines flight 140 (China Airlines A300 Flight 140 at Nagoya; Federal Aviation Administration Human Factor Team, 1996)
Location and Date: Nagoya Airport, Japan, April 26, 1994.
Scenario: The China Airlines Airbus A300 flight 140 approaching Nagoya, Japan crashed into a landing zone adjacent to the taxiway of the airport. While the co-pilot was manually flying the aircraft to runway 34 at Nagoya, he inadvertently activated the "Go-Around" mode while descending. This resulted in the consequence that the airplane was nose up in a steep climb, which caused the airspeed dropped drastically. As a result, the aircraft stalled struck the ground.
Software-related causes:
No warning was provided to the pilots when the co-pilot activated the "Go-Around" mode. As a matter of fact, Airbus had the company that implemented the flight control system to produce an update to the system which would deactivate the autopilot "when certain manual controls were applied on the control wheel in 'Go-Around' mode". However, the update was never patched to the aircraft because China Airlines believed that the update was not urgent.
Non-software-related causes:
The co-pilot mistakenly activated the "Go-Around" mode.
Losses: 264 fatalities (15 crew and 249 passengers) and 7 injures.

AA03: American airlines flight 965 (American Airlines Flight 965: Crash on the Mountain 2010; American Airlines Flight 965):
Location and Date: Buga, Colombia, December 20, 1995.
Scenario: The aircraft departed from Miami, Florida, USA and was heading to Cali, Colombia when a navigational error led the plane to crash into a 9800 feet mountain. Twelve seconds prior to the impact, the aircraft's Ground Proximity Warning System was activated and sounded an alarm warning of the imminent threat. However, while the captain and first officer attempted to fly clear of the mountain, previously deployed speed brakes prevented the avoidance of mountain.
Software-related causes:
When programing the navigation computer for the next approach way point, Rozo was identified as "R" in the flight charts accessible to the pilots. However, Colombia had duplicated the identifier for Romeo. Therefore, "ROZO" was used as the identifier for Rozo instead of "R". As a result, Romeo was automatically selected as the next waypoint since Romeo is a larger city than Rozo. This resulted in the consequence that the aircraft flew into a valley and finally incurred the crash.
Non-software-related causes:
The pilots tried to climb clear of the mountain. However, they forgot to disengage the previously deployed speed brakes, which reduced the rate of climbing.
Losses: 264 fatalities (15 crew and 249 passengers) and 7 injures.

AA04: Korean air flight 801 (Korean Air Flight 801; National Transportation Safety Board, 2000; Official Guam Crash Site Information Web Center):
Location and Date: Nimitz Hill, southwest of Antonio B. Won Pat International Airport, August 6, 1997.
Scenario: Pilots of the flight mistakenly picked up a signal as the glide-scope indicator, which turned out to be an irrelevant signal from an electronic device on the ground. This resulted in the abruptly descending of the aircraft and no proper warning was alarmed for the incoming collision. The flight ended abruptly colliding with Nimitz Hill three miles southwest of the landing airport.
Software-related causes:
Before the accident, the landing airport deliberately modified the Minimum Safe Altitude Warning System to reduce

[2] If location is revealed or applicable.

the number of false alarms generated by the system. However, the modified system could not detect an approaching aircraft that was below minimum safe altitude.
Non-software-related causes:
Several other causes can also be identified, such as the captain's poor execution of the non-precision approach, the pilots' fatigue, poor communication between the pilots, and insufficient flight crew training by Korean Air.
Losses: 228 fatalities (14 crew and 214 passengers) and 25 injures.

AA05: Spanair flight 5022 (Levin, 2008; Spanair Flight 5022):
Location and Date: Madrid-Barajas Airport, Spain, August 20, 2008.
Scenario: The flight was scheduled from Madrid-Barajas Airport to Gran Canaria Airport, Spain. However, the aircraft crashed momentarily after taking off. The flaps and slats were not deployed as required for takeoff, which resulted the aircraft rolled to the right and impacted the ground.
Software-related causes:
The Take-off warning system did not work properly. Therefore, no warning was alarmed to remind the pilots of the incorrect configuration.
Non-software-related causes:
The pilots omitted to check the "flap/slat lever and lights" item in the After Start checklist. In addition, the copilot simply repeated that the flaps and slats were properly configured without actually checking them in the Takeoff Imminent verification checklist.
Losses: 154 fatalities (6 crew and 148 passengers) and 18 injures.

AA06: London ambulance service (Dalcher, 1999):
Location and Date: London, UK, October 26, 1992.
Scenario: London Ambulance Service launched the new Computer-Aided Dispatch (CAD) system on October 26, 1992 to replace human dispatchers with the aim of reducing the average emergency-response time. However, problems began to arise after the system was in operation for a few hours. As the system crashed, dispatchers were not able to send ambulances to some locations while several ambulances were sent to the same location. The situation got worse when people expecting an ambulance but not getting one started to call back. The overwhelming amount of calls as well as numerous alert messages resulted in the unhandled old calls completely erased off the screen. Furthermore, operators were not able to empty the queues and the completed jobs could not always be cleared out. As a result, the system was gradually drained of its resources and the LAS chose to terminate the system and switch to semi-manual operation.
Software-related causes:
The CAD system went alive without stress testing. Also, there existed at least 81 known issues when CAD was launched. For example, the system didn't perform properly when handling incomplete data; the user-interface was problematic; a memory leak existed in a portion of code.
Non-software-related causes:
The bad project management by LAS was first to be blamed. The cost of the project was cut and the system could only use the old equipment rather than newer and more up-to-date devices. Then, no backup contingency was in place when the CAD system went wrong. Also, the training for operators was done ten months prior to the system became operational, which resulted in the consequence that the operators were not familiar with the new CAD system.
Losses: 20 people may have died due to the lack of emergency system.

AA07: Therac-25 accidents (Leveson and Turner, 1993):
Location and Date: Marietta, GA; Ontario, Canada; Yakima, WA; Tyler, TX, 1985 to 1987.
Scenario: The Therac-25 was involved in at least six accidents between 1985 and 1987. The patients were given approximately 100 times overdoses of radiation. The overdoses occurred when the high-power electron beam was in action without the beam spreader plate rotated into place. The failure occurred when a particular nonstandard sequence of keystrokes was typed on the system terminal and the interlock that was designed to prevent the overdoses would fail.
Software-related causes:
The new model reused the software designed for previous model, Therac-20. As a matter of fact, each bug contained in the Therac-25 software was also found in the software of the Therac-20. However, Therac-20 had hardware interlocks that masked their software defects; but in Therac-25, the hardware interlock was completely removed.
Other factors such as insufficient integration testing, ambiguous user manuals, and no explanation for the error codes displayed also exacerbated the accidents.
Non-software-related causes:
Personnel from the Manufacture, Atomic Energy of Canada Ltd., as well as operators of the machine did not believe in complaints, which was likely due to overconfidence.
Losses: three patients were dead after receiving excessive radiation. However, the relationship between the deaths and the excessive radiation is not clear.

AA08: Misdiagnosis of Ebola patient (Fox and Johnson, 2014):
Location and Date: Dallas, TX, September, 2014.
Scenario: Thomas Eric Duncan, the first person who died from Ebola in the United States, was mistakenly sent home with apparent symptoms of Ebola early in the morning of September 26. Texas Health Presbyterian Hospital Dallas later announced that flawed software and not human error caused the misdiagnosis.
Software-related causes:
In the electronic health records, the workflows for physicians and nurses are separate, which caused the documentation of the travel history to be inaccessible for physicians. Even though the nurse wrote that Duncan had traveled from Liberia, the doctors who examined him would not notice that from the electronic file. After the accident, Texas Health Dallas quickly fixed the software flaw which prevented further misdiagnoses.
Losses: The patient succumbed to his disease on October 8, 2014. Two nurses were infected by Ebola.

AA09: Miscalculated radiation doses at the National Oncology Institute, Panama (Borrás, 2006):
Location and Date: National Oncology Institute, Panama, August 2000 to March 2001.
Scenario: In March 2001, the Pan American Health Organization (PAHO) deployed an investigation concentrating on the serious radiation overdoses at the National Oncology Institute of Panama (Instituto Oncológico Nacional, ION) between August 1, 2000 and March 2, 2001. The investigation showed that a total of 28 patients were exposed to improper doses ranging from +10% to +105%.
Software-related causes:
The machine allowed a therapist to draw on a screen in order to place metal shields for the protection of healthy tissues from radiation. Though the system only allowed the use of four shields, the therapists at ION tried to trick the software by drawing a large shield with a "hole" in the middle.

However, they failed to find out that the direction of drawing the "hole" could lead to excessive exposure: drawing it in one direction, the correct dose could be calculated; but drawing in another direction, the software recommended twice the necessary exposure. In addition, no warning messages in the program and no manual quality control before the usage also attributed to the radiation overdoses.
Losses: At least 18 deaths were attributed to the radiation overdoses.

AA10: Patriot missile failure (Lum; E. Schmitt, 1991a,b):
Location and Date: U.S. Army barracks at Dhahran, Saudi Arabia, February 25, 1991.
Scenario: On the night of February 25, 1991, a missile fired by the Iraqi forces was not tracked and intercepted by the Patriot missile system; the incoming missile hit a U.S. Army barracks at Dhahran, Saudi Arabia resulting in 28 fatalities.
Software-related causes:
A software error in the Patriot system's radar and tracking software which caused an accumulating small target position error during system operation. Since the Patriot was developed with the assumption that the system would not be running for more than 8 hours at a time, when applied as static defenses, the accumulating error eventually failed to accurately predict the next air target location during computation; this caused the Patriot missile system to not see the incoming missile in time.
Losses: 28 fatalities.

AA11: Olympic pipeline explosion:
Location and Date: Bellingham, Washington, June 10, 1999.
Scenario: On June 10, 1999, a gasoline pipeline operated by Olympic Pipeline Company exploded in Bellingham, Washington's Whatcom Falls Park. The gasoline vapors exploded at 5:02 PM, and a fireball was sent down to the Whatcom Creek and resulted in death of three people.
Software-related causes:
A three-year investigation pointed out that a series of failures, including a faulty computer system, are to be blamed for the deaths. The main computer system failed due to unknown reason. In addition, the backup computer lagged in coming back on line, leaving a gap without any protection for as much as 14 minutes. During the computer glitch, a block valve south of Bellingham began to oscillate and finally closed, building up pressure close to the pipe's design capacity inside the line.
Non-software-related causes:
The employees were not adequately trained by Olympic Pipeline. Also, a broken pressure relief valve was not identified during regular maintenances.
Losses: Three fatalities.

AA12: V-22 Osprey helicopter crash (Berler; Gross, 2004):
Location and Date: Jacksonville, North Carolina, December 11, 2000.
Scenario: A V-22 had a flight control error, taking the lives of all on board and resulting in the grounding of the Osprey fleet.
Software-related causes:
The incident was found to be caused by a combination of software anomaly and a hydraulic system failure. The crashed aircraft experienced a leak in the hydraulic system and the pilot pressed the system reset button several times to compensate. However, investigators revealed a glitch in the plane's software that caused the plane to decelerate with each press of the button, which made the accident more likely.
Losses: Four fatalities.

AA13: Early releases of inmates in Washington State (Berman, 2016; Kaste, 2016):
Location and Date: Washington state, starting from 2002.
Scenario: Since 2002, the Department of Corrections in Washington state has given more than 3000 inmates early releases. The number could go as high as 3200. A computer glitch is to blame for these early releases.
Software-related causes:
In 2002, the state Supreme Court ruling required the Department of Corrections to apply good-behavior credits to the inmates' sentences. However, a computer glitch existed in the system which gave prisoners more credits than expected.
Losses: Four fatalities.

AA14: Deepwater horizon oil rig explosion (Shafer and Laplante, 2010):
Location and Date: Gulf of Mexico, Louisiana, April 2010.
Scenario: An explosion occurred on the Deepwater Horizon oil rig operated by BP Petroleum and its subcontractor Transocean. The explosion killed 11 workers, destroyed and sank the rig, and caused millions of gallons of oil to pour into the Gulf of Mexico, about 40 miles off of the Louisiana coast.
Software-related causes:
At a government hearing into the disaster, the chief electronics technician testified that the drill monitoring and control systems crashed several times and blue screens appeared on the oil rig's computer screens. With the systems crashed, the driller could not have access to the crucial data about what was going on in the well.
Non-software-related causes:
Other factors such as failed valves, dodgy cement, and misinterpretation of pressure test also lead to the disaster.
Losses: 11 fatalities and the largest environmental catastrophe in American history.

3.2. Accidents that did not involve the loss of life

AB01: Ariane 5 (Ariane 5; Ariane 501 Inquiry Board, 1996):
Location and Date: Kourou, French Guiana, June 4, 1996.
Scenario: The maiden flight of the Ariane 5 heavy-lift rocket launcher lasted for only about 37 s before the rocket departed its flight path and exploded.
Software-related causes:
At an altitude of about 3500 m, a conversion of a large 64-bit floating point number to a 16-bit signed integer (a value which could not be properly represented within this signal range and was an unprotected condition causing a processor halt) caused incorrect engine nozzle control deflection signals to be sent to the Ariane 5 engines, inducing high angle of attacks, large aerodynamic loads on the structure, and eventual booster separation which caused the self-destruction sequence to be triggered. The software design of the Ariane 4 was reused nearly 'as is' in the Ariane 5. The Ariane 5 boosters had more power than the earlier model – additional thrust and vibrations contributed to the conditions that led to the unhandled exception and processor halt.
Losses: $7 billion for ten years' development and a cargo valued at $500 million.

AB02: Solar and Heliospheric Observatory (SOHO) (SOHO Mission Interruption Joint NASA/ESA Investigation Board, 1998a, 1998b; Solar and Heliospheric Observatory; Weiss et al., 2001):

Location and Date: Gulf of Mexico, Louisiana, June 25, 1998.
Scenario: On December 2, 1995, the Solar and Heliospheric Observatory (SOHO) was deployed from the Atlas IIAS launch vehicle. The SOHO was the first ever mission to study the outer layer of the Sun, solar wind, and interior structure of the Sun. SOHO operated normally until May of 1998 when it was in its extended mission phase. On June 25, 1998, SOHO lost its lock on the Sun, which resulted in the loss of contact of the spacecraft for nearly three months.
Software-related causes:
On June 25, 1998, during the planned phase of calibration and space reconfiguration, a pre-programmed command sequence issued from the ground system which contained a logic error caused one of the three gyroscopes to be mistakenly left in the high gain setting. Additional actions by the ground crew to diagnose and recover the satellite initiated the chain of events that lead to the eventual loss of attitude control and telemetry.
Losses: Though the spacecraft continued to work after the recovery, only one gyro remained operational. One month later after the recovery, the last gyro failed on December 21, 1998. A lot of man powers were dedicated to the task of developing a new gyroless operations mode for SOHO.

AB03: Misplacement of Milstar satellite (USAF Accident Investigation Board):
Date: April 30, 1999.
Scenario: The Milstar satellite was launched on April 1999 using the Titan IV (401) rocket to provide secure and jam-resistant worldwide communications for the United States Air Force. There were six satellites to be deployed in geosynchronous orbits. However, only five of them were successfully placed in orbit with one of them deployed in a lower, non-operational orbit. The vehicle carrying the failed satellite experienced unexpected rolls and then loss of attitude control during its flight. The loss of attitude control causes excessive thruster firing depleting on-board fuel; the satellite was unable to be placed in the desired orbit.
Software-related causes:
An incorrect value of the roll-rate filter constant was assigned, which resulted in the unexpected rolls and loss of attitude.
Losses: The entire mission failed due to the misplacement of the satellite. The cost is about $1.23 billion. In addition, NASA had to face severe criticisms due to this mishap, which was the third straight Titan IV failure.

AB04: Mars Climate Orbiter (Mars Climate Orbiter Official website; Mars Climate Orbiter Mission Failure Investigation Board):
Location and Date: Space near Mars, September 23, 1999.
Scenario: The Mars Climate Orbiter was one of the two NASA spacecraft in the Mars Surveyor '98 program. The robotic space probe was launched on December 11, 1998 to study the Martian climate, atmosphere, and surface changes. It was also responsible for the communications relay for Mars Polar Lander which was to be introduced later. However, on September 23, 1999, communication with the Mars Climate Orbiter was lost; it was determined that the vehicle arrived too close to the planet and was likely destroyed by Martian atmospheric stresses on the spacecraft or lost to space.
Software-related causes:
The navigation malfunction was due to an error in units of measure within the software. Specifically, the software on the ground station generated thrust instructions with Imperial measures of pound-seconds (lb-s) while the Mars Climate Orbiter was programmed to accept metric units of newton-seconds (N-s); the subsequent navigation errors and failure of the ground crew to recognize the recurrent trajectory errors were due to software design error, placed the spacecraft in an incorrect trajectory to safely reach planetary orbit.
Losses: The total loss is $85 million of which $80 million was the spacecraft development and $5 million for mission operations.

AB05: Mars Polar Lander (JPL Special Review Board March 2000; NASA Science):
Location and Date: Surface of Mars, December 3, 1999.
Scenario: The Mars Polar Lander (MPL) was the other one of two NASA spacecraft in the Mars Surveyor '98 program. The Mars Surveyor '98 Lander was launched on January 3, 1999 to study the soil and the climate of a region near the South Pole on Mars. Due to a communication failure with the Mars Climate Orbiter which has been described previously above, the MPL used the Direct to Earth (DTE) technique to communicate with NASA. However, on December 3, 1999, during the descent phase of the Mars Polar Lander, the lander lost communication with Earth and never reestablished the communications link.
Software-related causes:
It was impossible to determine the exact cause of the failure. The most probable cause of the MPL mishap was a premature shutdown of the descent engines probably due to software misinterpreting switch bounce during descent after extension of the landing legs of the spacecraft. The spacecraft would have computed a touchdown when in fact it was still airborne. As a consequence, MPL crashed about 40 m above the surface and was presumed destroyed on impact with the surface.
Non-software-related causes:
Other factors such as failed valves, dodgy cement, misinterpretation of pressure test also led to the disaster. Also, budget and schedule pressure was also another cause for the inadequate testing and other quality assurance activities.
Losses: The total cost was $120 million of which $110 million was the spacecraft development and $10 million for mission operations.

AB06: Mars Global Surveyor (Cain, 2006):
Location and Date: Space near Mars, November 2, 2006.
Scenario: The Mars Global Surveyor was launched in November 1996. It operated for ten years and provided global mapping of the Martian surface, atmosphere, magnetic field, and interior. Its mission was extended until communication was lost with NASA on November 2, 2006.
Software-related causes:
The loss of contact was caused by a flaw in a parameter update to the spacecraft's system software. The spacecraft held two identical copies of system software. However, two different parameters were updated to those copies by two independent operators. In order to correct the mistake, another corrective update was performed. However, the update command was written to the wrong memory address which further induced the accident.
Non-software-related causes:
Lack of communications between the operators is another cause of the incident.
Losses: No remarkable loss could be identified, since the spacecraft has been in working status much longer than expected.

AB07: 2015 Servile Airbus A400M Atlas Crash (Servile Airbus A400M Atlas crash, 2015, Hepher):
Location and Date: La Rinconada, Spain, May 9, 2015.
Scenario: An Airbus A400M Atlas cargo plane crashed during a test flight on May 9, 2015. The mishap was caused by a "quality issue in the final assembly" of the aircraft engine.
Software-related causes:
Reuters (Hepher) reported that a file storing torque calibration parameters for each engine was accidentally wiped from three of the aircraft's four Engine Control Units (ECU). As a result, three of the four aircraft engines unexpectedly shut down during the test flight leading directly to the loss of aircraft.

AB08: Chemical bank ATM failure (Hansell, 1994; Reuter News Service, 1994; Yates, 1994):
Location and Date: New York, February 1994.
Scenario: Chemical Bank offices around New York received numerous phone calls about the double-posted withdrawals which caused 15 million dollars to disappear from customers' accounts. The discrepancy was soon discovered and all the money was returned to customers' accounts.
Software-related causes:
Chemical Bank installed an updated computer program in its data center. This update contained a programming error that would process all withdrawals and transfers twice when customers used an automated teller machine (ATM) or made a purchase at retail stores.
Losses: The direct monetary losses of Chemical Bank were not revealed. However, efforts were made to fix and return the money to the accounts of customers. In addition, customers might lose their confidence in the bank.

AB09: AT&T network outage (Burke, 1995; Elmer-Dewitt, 1990; Peterson, 1991):
Location and Date: New Jersey, January 15, 1990.
Scenario: The AT&T operation center in New Jersey, had a significant increase of warning signals that appeared across its 72 video monitoring screens. Those screens displayed operational information about AT&T's worldwide telephone switching network. Only about fifty percent of the placed phone call made it through AT&Ts network.
Software-related causes:
After nine hours, AT&T technicians were able to identify the root cause which was a single line of code (misplacement of a BREAK statement) in the 4ESS operating software system introduced by a software update aiming to improve the network's performance.
Losses: AT&T alone lost from $60 to $75 million in unconnected calls.

AB10: AT&T SS7 Software patch failure (Neumann, 1994; Sterling, 1993):
Location and Date: Washington, D.C., Los Angeles, San Francisco, and Pittsburgh, July 1, 1991.
Scenario: One and a half years after the AT&T Eastern USA telephone system outage of January 1990, on July 1, 1991, various metropolitan areas had serious outages of AT&T telephone service. Computer-software collapses in telephone switching stations disrupted service, and Washington, D.C. (6.7 million lines), Los Angeles, San Francisco, and Pittsburgh (1 million lines) were all affected for several hours.
Software-related causes:
These problems were eventually traced to a glitch in an untested software upgrade. A single mistyped character in the April software upgrade patch, a "6" instead of a "D" in a single line of code, was blamed to be the main cause of the accident.
Losses: Overall twelve million people were left without service. Although exact monetary losses are unknown, many AT&T customers were disgruntled and might have considered switching to other service providers.

AB11: AT&T CRM upgrade problem (Federal Communications Commission, 2002; Koch, 2004; Cowley, 2003):
Date: 2003.
Scenario: AT&T made a software upgrade to its Customer Relationship Management (CRM) system. The CRM software deals with account management functions; for example, tracking orders that were made and handling requests for customer service plan changes. However, customer service representatives were no longer able to create or have access to customer relationship accounts.
Software-related causes:
Due to a glitch in the process arising from the different interpretations of data standards between two manufacturers (TSI and NeuStar), the interface bug caused the AT&T CRM system to crash. Later, it was determined that schedule pressure lead to insufficient testing of the software upgrade.
Losses: AT&T lost a significant number of potential new customers.

AB12: DART mishap (Croomes, 2006):
Location and Date: Space, April 15, 2005.
Scenario: The Demonstration of Autonomous Rendezvous Technology (DART) satellite was expected to rendezvous with and perform various maneuvers near the orbiting Multiple Paths, Beyond-Line-of-Sight Communications (MUBLCOM) satellite, and do these maneuvers autonomously. Part way through the mission, the DART satellite began using an excessive amount of propellant and depleted the on-board propellant supply earlier than expected. The DART satellite then collided with the MUBLCOM satellite. As a consequence, the DART satellite retired prematurely and initiated its preprogrammed departure without completing its main mission objectives.
Software-related causes:
The root cause of the propellant loss was the incorrect velocity measurement from the primary GPS receiver which was incorrect by about 1.9 feet per second. This previously known bug resided in the software but the fix was never implemented in the final code loaded onto the spacecraft. Also, an additional feature in the computational logic known as "gain" was set at an appropriate level which caused the logic to trust the computed values more than it reasonably should.
Non-software-related causes:
The project was selected as a high-risk, low-budget technology demonstration under NASA Research Announcement (NRA), in which case detailed design decisions about how to meet the requirements were left to the contractors.
Losses: More than $1 million.

AB13: Loss of voice communication between FAA Air Traffic Control Center and airplanes (Geppert, 2004; Los Angeles Times, 2004):
Location and Date: Los Angeles International Airport, September 14, 2004.
Scenario: The Los Angeles International Airport and other airports in that region experienced suspended air traffic control operations due to a failure of an FAA radio system located in Palmdale, California. Field technicians failed to

perform the required reboot of this radio system which was needed every 30 days – this resulted in the shutdown of radio system without warning.
Software-related causes:
Though the FAA attributed the accident solely to human error, there was a design error that made the manual reboot necessary in the first place. Following this incident, the radio was replaced with the upgraded unit which eliminated this design error.

AB14: Shutdown of the Hartsfield-Jackson Atlanta International Airport (Hartsfield-Jackson Atlanta International Airport; Schneier, 2006):
Location and Date: Hartsfield-Jackson Atlanta International Airport, April 19, 2006.
Scenario: An employee of TSA identified the image of a suspicious device which was likely to be a bomb but did not realize it was part of the routine testing for security screeners. To ensure the security of the airport, the screener and his supervisor manually rechecked all the bags on the conveyor belt but failed to find anything resembling what was on the screen. Due to this, the airport decided to evacuate the security area for two hours.
Software-related causes:
Though it can be interpreted as a human error, the lack of a warning message was also indicated as the cause of the mishap.
Losses: Delay of more than 120 flights.

AB15: London tube closure (Thurston, 2006):
Location and Date: London, UK, November 2006.
Scenario: In November 2006, the installation of new software caused the widespread delay to the London Underground. The new computer software with the revised timetable information broke down the system when it started.
Software-related causes:
Though it can be interpreted as a human error, the lack of a warning message was also indicated as the cause of the mishap.

AB16: Cancellation of London Stock's Taurus project (Bacon, 2013; London Stock Exchange, 2012; TAURUS):
Location and Date: London, UK, Early 1993.
Scenario: Taurus, which stands for Transfer and Automated Registration of Uncertificated Stock, was an ambitious effort aiming to turn stock trading into a paperless transaction system. However, in the early 1993, the London Stock Exchange (LSE) announced cancelation of this project because Taurus was eleven years late and 13,200% over the budget with no concrete solutions.
Software-related causes:
The main cause was the use of Vista Software for database management which could not handle batch processing. LSE tried to rewrite almost 60% of the Vista Software for database management to deal with Vista's inability to handle batch processing. This development introduced numerous hidden bugs and long project delays.
Non-Software-related causes:
Other factors can also be considered as causes attributing to the failure. The risk assessment for the project failed to acknowledge the failure and did not cancel the project at the very first moment. The design of the project was questionable due to its complexity and lack of well-organized requirements.
Losses: Taurus stock settlement system canceled after $600 million is spent.

AB17: Outages in Tokyo Stock Exchange (Brooke, 2006; Leyden, 2005; Tokyo Stock Exchange):
Location and Date: Tokyo, Japan, November 1, 2006.
Scenario: The outage of the Tokyo Stock Exchange took place on November 1, 2006. Due to the unexpected outage, the exchange operations were delayed for three hours.
Software-related causes:
This outage was caused by bugs in the newly installed transaction system software developed by Fujitsu. The system was designed to handle higher trading volumes than the previous system. Flawed instructions from Fujitsu updating the system caused the entire outage of Tokyo Stock Exchange.
Losses: Exchange operations were delayed for three hours.

AB18: Money loss at Tokyo Stock Exchange (Typing Error Causes $225 M Loss at Tokyo Stock Exchange, 2005):
Location and Date: Tokyo, Japan, December 8, 2005.
Scenario: Mizuho Securities Co. lost at least 27 billion yen (approximately $225 million) on a stock trade. The trouble began with a typing error by a trader. Instead of selling one share at 610,000 yen ($5041), the trader sold 610,000 shares at one yen (less than a penny). Two minutes later, the trader discovered the error and tried to revoke the trade. However, after three failed attempts, the trade could not be revoked.
Software-related causes:
Investigations indicated that the software system suffered from a glitch that under certain circumstances traders could not cancel trades. The bug was introduced in an update of the system; regression testing may have discovered the bug. Unfortunately, regression testing has never been performed adequately.
Losses: A loss of $225 million.

AB19: Toronto Stock Exchange failure (Modine, 2008):
Location and Date: Tokyo, Japan, December 18, 2008.
Scenario: Toronto Stock Exchange (TSX) shut down 18 min after opening and continued to remain un-operational for the entire day. The failure that caused the shutdown was discovered — only part of the users were receiving data about trading activities.
Software-related causes:
Although specific details were not given, the company stated that the failure was due to a "network firmware issue" which "resulted in complications with data sequencing". This indicates that the software may have been poorly integrated with the networking infrastructure or inadequately tested.
Losses: TSX shut down 18 min after opening and remain un-operational for the entire day.

AB20: Standards & Poor's erroneous alert message on France's credit rate (Charette, 2011; Neumann and Lamar, 2011):
Date: November 10, 2011.
Scenario: The rating company Standards & Poor's (S&P) sent out an erroneous alert message to subscribers that the company had downgraded France's triple-A credit rating. It took nearly two hours before S&P sent out another note stating that the alert was in fact a "technical error".
Software-related causes:
Following the publication by S&P of a review of all of its Banking Industry Country Risk Assessment (BICRA) rankings, the BICRA ranking for France was changed to "N/A" (not available) on the Global Credit Portal page. The system mistakenly interpreted this change as a "downgrade" and triggered the erroneous message sent to subscribers.

AB21: Security flaws in medical devices (Greenberg, 2014; Storm, 2014):
Date: October 22, 2014.
Scenario: Reuters reported that the Department of Homeland Security (DHS) was investigating 24 cases of suspected cybersecurity flaws in medical devices and hospital equipment.
Software-related causes:
DHS was concerned that these devices might be controlled remotely and become the cause of severe issues, such as instructing an infusion pump to overdose or forcing a heart implant to deliver a deadly jolt of electricity.

AB22: Emergency shutdown of the Hatch Nuclear Power Plant (Edwin I. Hatch Nuclear Power Plant; Krebs, 2008):
Date: March 7, 2008.
Scenario: The Edwin I. Hatch Nuclear Power Plant was forced into an emergency shutdown for 48 hours. The accident occurred at Unit 2 started after a software update was applied to a computer operating the plant's business network.
Software-related causes:
The computer which caused the shutdown was used to monitor chemical and diagnostic data from one of the facility's primary control systems. The update was intended to synchronize data on both the business and the control system. After the update was patched, the updated computer rebooted and reset the data on the control system, causing the safety systems to errantly interpret the lack of data as the sign of a drop in water reservoirs. To ensure the safety of the nuclear plant, automated safety systems triggered the shutdown.
Losses: A total loss of $5 million

AB23: Power outage across the Northeastern U.S. and Southeastern Canada (Jacobs, 2013; Northeast Blackout of 2003):
Location and Date: Parts of the Northeastern and Midwestern United States and Ontario, Canada, August 14, 2003.
Scenario: parts of the Northeastern United States and Southeastern Canada experienced widespread power blackouts after 4:10 p.m. EDT. Some power supply was restored by 11 p.m. But some places did not get their power back until two days later. The time for restoration for some remote areas even tripled.
Software-related causes:
A software bug known as a race condition existed in the energy management system. The bug was triggered at a control room of the FirstEnergy Corporation, located in Ohio. The bug crashed the control room's alarm system for over an hour. During this time, operators were unaware of the need to redistribute the power supply in order to remain balanced loads. Therefore, a manageable local blackout cascaded into massive distress on the electric grid.
Non-software-related causes:
The overloaded transmission lines at a generating plant in Eastlake, Ohio, came in contact with overgrown trees. Therefore, these lines went out of service and their loads tripped to other transmission lines. However, other lines were not able to bear the loads and started tripping their relays. Once multiple trips occurred, some generators lost part of their loads and tripped out of the grip to prevent further damage.
Losses: The blackout affected more than 50 million people and cost a total of about $13 billion.

AB24: Nissan Leaf Software Glitch (Nissan Leaf recalled for software glitch, 2011; O' Dell, 2011):
Date: 2011.
Scenario: The Nissan Leaf owners suffered from the "failure to start" issue in 2011. When the air conditioning in the vehicle was activated, the increased demand in power triggered a high voltage alert. When this alert was present, a user could continue driving, but once the car was turned off, it would not start again.
Software-related causes:
An incorrectly programmed control software and a hypersensitive sensor were to blame for the issue. After identifying the issue, Nissan ceased delivery of newly ordered vehicles. Additionally, they sent engineers to individual dealerships of the affected cars to manually reprogram the control software while the company worked to develop a "solution package".
Losses: Man powers to reprogram the control software.

AB25: Failure of Windows Genuine Advantage (Lake, 2010; Microsoft; MSDNArchive, 2007):
Date: August 24, 2007.
Scenario: Windows Genuine Advantage is a program by Windows that determines whether your copy of Windows is a legitimate copy. Due to a human error in August 2007, thousands of people who had legitimate copies of Windows were accused of having a pirated copy.
Software-related causes:
The incident was caused by the mistakenly sent preproduction code to production servers. The production servers had not yet been upgraded with a recent change to enable stronger encryption/decryption of product keys during the activation and validation processes. As a result, activation and validation requests from genuine windows systems were declined by the production servers. Though Microsoft repeatedly emphasized that the incident was due to "nothing more than human error", it is worth noting that no appropriate warning or reminder in the process of sending code to the production servers. After the incident, Microsoft added checkpoints before changes can be made to production servers.
Losses: Reputation of Windows damaged due to the increasing dislike of their anti-piracy tool.

AB26: Failure of Oxford Health Plan's Bill and Claim system (Freudenheim, 1997; Hammonds, 1997; The Harder They Fall, 1997):
Date: 1996.
Scenario: Oxford Health Plans Inc. introduced a new billing and claim system on September 1996 in order to deal with the expected increase in annual enrollments to the plan. However, the system was faulty and misled Oxford into overestimating the revenues and membership while underestimating the cost that they have to pay the doctors and hospitals.
Software-related causes:
The system could not handle data with any error. For example, once the system detected a single mistake in a 1000-member account, it kicked out the entire group.
Non-software-related causes:
Little efforts were spent on the testing of the new system, which resulted in the system to be inadequately tested. Also, Oxford tried to transit too many customers by one operation under the condition that no backup plan was available.
Losses: Some customers canceled their policies, which cost Oxford $42 million. In addition, problem with incorrect estimates for Medicare patients and the advance payments that Oxford paid out led to another $51.9 million loss for Oxford.

AB27: Security flaw in Maximo Heart Monitor (Feder, 2008; Hacking the Human Heart, 2010):

Date: 2008.
Scenario: In 2008, a team of researchers from the University of Washington and the University of Massachusetts dedicated a great amount of time and $30,000 worth of lab equipment to investigate the security of the Maximo heart monitor and other medical devices.
Software-related causes:
They concluded that not enough attention was being given to the security process when creating such medical systems and the result could impose potential danger on users.

AB28: Cancellation of mission by U.S. Air Force F-22 raptors (Hill, 2007; Johnson, 2007; Wastnage, 2007):
Date: February 2007.
Scenario: In February 2007, the U.S. Air Force's F-22 Raptor, one of the leading fighter jets in the world, faced computer systems failure flying from Hawaii to Kadena Air Base in Japan. When the aircraft crossed the International Date Line (IDL), multiple computer systems such as fuel system, navigation system, and part of communication system went offline and failed on board. The crew attempted numerous times to reboot the system without success. The aircraft returned to Hawaii. If weather condition were bad or the F-22 Raptors were separated from the refueling tankers that guided them back to safety, they may have gotten into serious trouble.
Software-related causes:
Though the details of the bug were not publicly revealed, Air Force Major General Don Shepperd stated that "it was a computer glitch in the millions of lines of code, somebody made an error in a couple lines of the code".
Losses: The mission was canceled and Air Force lost 48 h of time and work hours for engineers to fix the code that was flawed.

AB29: Intel Pentium FDIV Bug (Janeba, 1995; Price, 1995):
Date: August 24, 2007.
Scenario: Intel's Pentium microprocessor chip made in 1994 contained a floating-point unit (FPU), also called a math coprocessor that contains the instructions to divide floating-point numbers. These instructions made the chips much faster for complex numerical calculation. However, the Pentium chips had errors on their FPU instructions for division, which led to incorrect outputs for specific floating-point numbers.
Software-related causes:
The root cause was the omission of five entries out of about a thousand values in a stored table needed by the built-in floating-point unit.

AB30: USS Yorktown carrier shutdown (Slabodkin, 1998):
Location and Date: Coast of Cape Charles, VA, September 1997.
Scenario: The Yorktown Smart Ship lost control of its propulsion system when bad data was fed into its computers during maneuvers off the coast of Cape Charles, Va. Although the USS Yorktown had been considered a success in reducing manpower, maintenance, and costs, an incident where the ship was dead in the water for hours has taught a lesson to the US Navy.
Software-related causes:
The "divide by zero" overflow that caused the system failure did not happen in actual combat, but this engineering local area network casualty generated a debate about possible solutions to prevent similar failures from happening in the future.
Loss: Engineers' efforts and work hours to fix the bug.

AB31: GPS dark up (Associated Press, 2010; Elliott, 2010):
Date: January 11, 2010.
Scenario: As many as 10,000 U.S. military GPS receivers were affected and were useless for at least two weeks.
Software-related causes:
Air Force inspectors discovered a glitch in the patch aiming to update the ground control stations that rendered dark the graphics of up to 10,000 GPS receivers for at least two weeks. The Air force acknowledged that no tests were performed on those receivers being affected. As a result, the Air Force started to require more tests on GPS receivers for a broader sample of military and civilian models.
Loss: Engineers' efforts and work hours to fix the bug.

AB32: FoxMeyer ERP System failure (Fox-Meyer Drugs, 2012; Scott, 1999):
Date: August, 1996.
Scenario: In August of 1996, FoxMeyer, once was a drug company with annual sales of about five billion US dollar, filed for bankruptcy, and was sold to a competitor for only 80 million soon after. Four years' prior, the corporation chose to make a large investment in an Enterprise Resource Planning (ERP) system in order to provide efficient distribution of prescription drugs. Despite high hopes for the ERP project, it turned out to be a complete failure. The system had significant data errors and couldn't handle the drug distribution correctly. In addition, the project was way behind the schedule and over the budget.
Software-related causes:
Both poor planning and lack of performance contributed to the failure of the software's integration and resultant monetary loss that FoxMeyer suffered. The software package SAP R/3 was originally designed for manufacturing companies instead of wholesalers, which lacked in the ability to fulfill the need for product distribution.
Non-Software-related causes:
FoxMeyer failed to allow sufficient time for the implementation and testing. In addition, no end user was involved in the project planning and employees had no chance to express their business needs.
Loss: A loss of $65 million invested in the project; The company went bankrupt soon after.

AB33: Nike's supply-chain software failure (Bousquin, 2001):
Date: August, 1996.
Scenario: Nike hired i2, a supply-chain software vendor, to create a system that would predict the demand of their products. The project was started in June 2000 and Nike intended to make use of the supply-chain software in order to better plan and control the production of existing products. However, the supply-chain software did not reach the expectations of Nike in performance nor functionality. Though designed to aid in speedily forecasting market changes, the system from i2 suggested producing too many shoe styles that were declining in popularity and too little popular models. The defective software system led to significant lost in sales and decreased stock prices.
Software-related causes:
The demand prediction application was poorly integrated with the supply chain planner. The two applications used unique business rules and data formats. Operators even had to download the data from the demand prediction application and manually reload the data into the supply chain planner. Second, i2 constantly stated that the major cause of the failure was the over-customization of their products, which was entirely due to the fact that Nike refused to follow the implementation methodology provided by i2.

Non-software-related causes:
Such a complex system was not adequately tested and went live just one year after launching the project. An analyst at Credit Suisse First Boston stated that because of the complexity of the project, "he would not have been surprised if to test the system for three years while keeping the older system running."
Loss: Nike lost $100 million in lost sales, on top of the $40 million cost of the project. Profit per share was 35 to 40 cents, which was a significant decline from their previous 50 to 55 cents per share.

AB34: Hershey Foods Corporation ERP implementation failure (Gilmore, 2009; Koch, 2002):
Date: August, 1996.
Scenario: Hershey Foods Corporation decided to replace its legacy management system with the new Enterprise Resource Planning (ERP) software. SAP AG's R/3 ERP suite along with companion software from Manugistics and Siebel were chosen. IBM Global Services was responsible for the integration of the software provided by the three different vendors. However, some of the modules were delayed and were implemented three months later than expected. This delay caused the late shipment of several consignments and incomplete deliveries.
Software-related causes:
Poor project management and communication were the root cause of the failure. First of all, the lack of communication between technical personnel and on-site operators caused the consequence that ERP system could not identify specific storage locations. Since Hershey occasionally store products in rented temporary spaces and unused rooms in the factories, these locations were not taken into account by the new system and therefore the products stored there were not listed as available for distribution.
In addition, Hershey ignored the recommended 48-month implementation time and insisted the 30-month turnaround. The limited implementation time resulted in the lack of adequate testing of the system. Moreover, Hershey tore down its old logistics system to make way for the new one, which resulted in the lack of backup plan when something went wrong.
Non-software-related causes:
The employees were inadequately trained on how to use the new software. The transfer took place during peak season and little time were spared for the endeavors.
Loss: Annual revenues for 1999 were $150 million less than that for 1998. In addition, the stock price of Hershey dropped 8%.

AB35: Hewlett Packard ERP system migration failure (Wailgum, 2009):
Date: May 2004.
Scenario: In May 2004, Hewlett Packard Corp. began migrating one of its largest divisions to a centralized Enterprise Resource Planning (ERP) system manufactured by SAP AG. However, due to a series of "small problems", the migration turned out to be a failure. 20 percent of customer orders for HP servers stopped in their tracks due to data modeling issues between the legacy system and the SAP system. The data models of the two systems differed from each other which resulted in the consequence that the SAP system was not able to process some orders for customized products.
Software-related causes:
The root cause for the failure was the problems regarding data integrity and interface. Data from the legacy system was rejected by the SAP system, which required a lot of manual intervention.
Non-software-related causes:
Lack of adequate testing played an essential role in the failure due to aggressive planning of the project. In addition, the communications among the teams developing the system were also blamed as another cause of the failure.
Loss: The project alone cost HP $160 million in order backlogs and lost revenue. In addition, HP's servers and storage group had a $400 million revenue decrease, which was partially blamed on the failure.

AB36: Cancellation of Avis PeopleSoft ERP system (Best, 2004):
Date: 2004.
Scenario: In 2004, car rental firm Avis Europe announced the termination of its new ERP system due to "substantial delays and consequently higher cost due to a number of problems with design and implementation".
Software-related causes:
A large number of problems existed in the system which were due to the design and implementation.
Non-software-related causes:
Poor project planning and management was the root cause for the failure.
Loss: A loss of $54.5 million.

AB37: Prius software Glitch (Kanellos, 2005):
Date: 2004 and 2005.
Scenario: the 2004 and 2005 Priuses can make a warning light on without any cause. In addition, the gas engine might be shut down altogether. As stated in (Kanellos, 2005), 68 incidents have been reported to the National Highway Traffic Safety Administration.
Software-related causes:
Though the detailed causes for the incidents were not publicly revealed by Toyota, it is reasonable to conclude that the incidents were due to computer glitches since mechanics need to re-program the electronic control module when owners brought their cars back.
Loss: The incidents caused significant reputation loss of Toyota Prius, especially in a stage that customers were still not convinced of the reliability of these new electronic vehicles. Man powers were also required to solve the issue.

AB38: CIGNA's CRM system data migration failure (Bass, 2003):
Date: January 2002.
Scenario: In January 2002, CIGNA HealthCare Corporation experienced a "near disaster" while migrating its members to its new claims processing and customer service processing systems. Due to technical "glitches", many customers were unable to obtain health coverage or receive correct quote. Customer service representatives weren't capable of providing appropriate assistance.
Software-related causes:
The root cause for the incident was the system's inability of dealing with large numbers of account migration at the same time. Before the mass migration, CIGNA tested the functionality by migrating relatively small groups (10 to 15,000) of user accounts. Encouraged by the early success, CIGNA migrated 3.5 million customers in one batch. Unfortunately, the attempt failed completely.
Non-software-related causes:
The pressure from CIGNA had an impact on the overall quality of the system as well as the testing process.

Loss: The CRM system was successfully launched after the issues fixed. However, company membership fell from 13.3 million to 12.5 million which was directly related to these customer service problems. In addition, its stock fell 40 percent after the failure.

AB39: Enron payouts to employees stalled (Hays, 2007):
Date: 2007.
Scenario: In 2007, more than 20,000 former Enron employees were told that they were either overpaid or underpaid because of a math error. Due to the faulty computer software used to calculate allocations, 7700 people received too much money while 12,800 didn't receive enough.
Software-related causes:
The accident has been attributed to a computer glitch in the software calculating the payouts.
Loss: The amount of overpayment that couldn't be recovered amounted to $9.15 million, or $11.2 million with interest.

AB40: Atlanta water bill spike (Zamost and Phillips, 2011):
Location and Date: Atlanta, Georgia, 2007 to 2009.
Scenario: From 2007 to 2009, the city of Atlanta experienced a problem with its customer water usage and billing. For example, bills of $150 to $250 skyrocketed to $1000 to $1200.
Software-related causes:
When the water meters reached 450, the software started to miscalculate customer usage leading to the overcharges.
Loss: The city faced countless lawsuits and meetings with city officials complaining the situation. Altogether $466,368 credits were issued to residents. In addition, there was a loss of trust between the city and the residents.

AB41: McDonalds' Innovate project failure (Gallagher and Barrett, 2003):
Date: 2007 to 2009.
Scenario: McDonalds' Innovate Project was one of the most expensive and extensive information technology projects. The project's goal was to establish a real-time global network to link over 30,000 stores worldwide to their headquarters by using the Intranet. However, McDonalds' canceled the project whose costs reached $170 million.
Non-software-related causes:
Failure of the project was mainly due to the bad management during the entire process of development. Lack of communications and insufficient leadership have been identified as the two major causes. In addition, lack of adequate testing also played a role in the failure. Furthermore, the project was pushed too hard, which resulted in the consequence that the multiple phases project went live in a short period of time.
Loss: A loss of $170 million.

AB42: Sydney Water's CIBS project failure (Krigsman, 2007):
Date: 2007 to 2009.
Scenario: Sydney Water pursued litigation against PricewaterhouseCoopers over the failed Customer Information Billing System (CIBS) project. Sydney Water stopped the CIBS project following lengthy delays and cost overruns
Software-related causes:
The failure was attributed to the poor management and governance from Sydney Water.
Loss: The failed project cost more than $135 million.

AB43: Confirm project failure (Bacon, 2013 ; Confirm Project):
Date: 1992.
Scenario: The CONFIRM system was a very ambitious project which aimed to create a global distribution system used by airline, rental car, and hotels. In 1988, AMR, Marriott, Hilton Hotels Corp., and Budget Rent-A-Car made contracts to complete the system at an expected cost of $55 million. However, the system never went live as the project turned out to be much more complex than the partners had expected. In April 1992, CONFIRM failed tests at Los Angeles-based Hilton, just three months before the system was intended to go live.
Software-related causes:
The main cause of the failure was the difficulties of dealing with the interfaces between two mainframes. The system required application-to-application communication channels between the two mainframes for nearly 60 applications. However, the users found that the system's user interface, transaction processing application, and the database did not communicate effectively with one another. In addition, data in the databases was difficult to recover in the event of crashes.
Loss: After three and a half years and $125 million, the CONFIRM project was canceled.

AB44: Trips reservation system failure (Tomsho, 1994):
Date: 1993, 1994.
Scenario: In 1993, Greyhound Bus Lines implemented a complex software system called "Trips" to help with bus reservations, ticketing, and the arrangement of buses and drivers. However, after Trips was launched in the summer of 1993, there were immediately problems such as repeatedly system crashes and doubled ticketing time. Again in 1994, the system went down again during an urgent bid to increase ridership. Buses and drivers were in short supply in many cities which resulted in some terminals swamped with frustrated passengers that could not get on the bus.
Software-related causes:
The complexity of the Trips system was the root cause for the crash in 1993. Also, lack of load testing played important roles in both two failures. What's more, the system was too quickly to go live, though it was recommended to be ready in several phases.
Loss: A revenue loss of $61 million.

AB45: FAA's advanced automation system failure (Cone, 2002):
Date: 1994.
Scenario: The Advanced Automation System (AAS) was intended to provide a complete overhaul of the nation's major air traffic control computer systems. However, the project failed after $1.5 billion of hardware and software turned out to be useless.
Software-related causes:
The software project was way too complicated.
Non-software-related causes:
Under estimation of the resources required, and poor management can also be identified as causes.
Loss: A loss of $1.5 billion.

4. Losses

The loss affected by an accident is often correlated to its severity. This section describes the various kinds of losses incurred due to these accidents. The losses have been classified as fatal and non-fatal. Table 4 lists all the fatal accidents involved in our study, which helps to understand the combined impact of failed software critical system on humans. Non-fatal losses, listed in Table 5, describe accidents involving loss of money, time, or reputation (good-will).

Table 4
Losses of notable fatal software failures included in our study.

Year	Accident	Fatalities
1985–87	Therac-25 Accidents*	Not clear
1991	Patriot Missile Failure	28
1992	London Ambulance Service	30
1993	Lufthansa Flight 2904	2
1994	China Airlines Flight 140	264
1995	American Airlines Flight 965	159
1997	Korean Air Flight 801	228
1999	Olympic Pipeline Explosion	3
2000	V-22 Osprey Helicopter Crash	4
2002	Early Releases of Prisoners in Washington	4
2008	Spanair Flight 5022	154
2010	Deepwater Horizon Oil Rig Explosion	11
2014	Misdiagnosis of Ebola Patient	1
2000–01	Miscalculated Radiation Doses at the National Oncology Institute, Panama	18

* Number of fatalities is not clear based on several reports, including the most widely cited one by Leveson and Turner, 1993

Special notice should be given to the security flaws in Maximo heart monitors. Even though research of the Maximo heart monitor indicated that up until that point, no fatalities had been caused by hackers, the project concluded that it was entirely possible. By creating a device with little security, medical companies were allowing potential access to a patient's lifeline - possibly resulting in death.

5. Causes

Accident causes help one understand the underlying events that must be prevented to avoid the mishaps in the future under the same conditions. This section examines some causes of the accidents presented in this paper. We have classified the accident causes by 'software-related' and 'non-software-related' in order to distinguish between the technical and non-technical factors involved in the mishap. Also included are short descriptions of why some of the representative cases are caused by the corresponding cause.

5.1. Contributing factors: software related

Software related factors describe the role of software in an accident and the causes of failures. These causes include faults in the software, improper design of the system/software, and other software related issues that caused the accident.

5.1.1. C01: Faulty value computation

Although this kind of error has been reduced dramatically due to modern tools that perform more testing and validation of the system, it should still be considered one of the major causes of accidents.

1. AB01 was one such accident in which an operand fault occurred while converting a 64-bit floating point Horizontal Bias (HB) value to a 16-bit signed integer. The Système de Référence Inertielle or Inertial Reference System (SRI) computed the faulty value just before ignition on the platform and the On-Board Computer (OBC) computed an error output, which altered the nozzle direction drastically and led to the flight departure and eventual self-destruction of the rocket.
2. Similarly, AB03 also occurred due to feeding of the wrong roll rate filter constant to the software file in the Inertial Measurement System. The incorrect value caused the launch vehicle to move unexpectedly, burning more fuel than expected. As a result, the Milstar Satellite was placed in a lower, unusable orbit.
3. In AB04, the Mars Climate Orbiter lost communication with Earth due to a faulty value computed by the development team. The crew used Imperial measurement terms to calculate the minimum altitude at which the spacecraft should orbit around Mars and came up with a value of 110 pound-seconds. However, the spacecraft system software was designed to take instruction in terms of Newtons-seconds, the metric unit of force. As a result, the spacecraft computer calculated an incorrect value which directed the spacecraft to a lower altitude in the Martian atmosphere destroying the spacecraft.
4. In AB12, the measured velocity from the primary GPS receiver was offset about 0.6 m per second. Whenever the estimated and measured positions diverged beyond acceptable limits, the software would reset and replace the estimated position and speed with the measurements from the primary GPS receiver. Since the primary GPS receiver measurements were not accurate, the estimated and measured positions would eventually diverge and trigger another reset. This problem led to an excessive use of the thrusters and early depletion of fuel.
5. In AB30, the ship lost control of its propulsion system because its computers were unable to divide by zero. The Yorktown's Standard Monitoring Control System administrator entered zero into the data field for the Remote Data Base Manager program, causing the database to overflow and crash the LAN consoles and remote terminal units.

5.1.2. C02: Security

Security has long been recognized as a key aspect of software quality; this has become more important as software is becoming more ubiquitous. However, either failures to fulfill requirements or omissions of requirements can lead to security vulnerabilities which result in severe consequences, such as threats to human life, loss of significant amounts of assets or resources, and/or loss of reputation. Below are several examples related to security vulnerabilities:

1. Security flaws in AB27 were the result of poor design. Companies put little effort in security because potential "hacking" did not seem like a relevant issue; research conducted in 2008, however, indicated otherwise. A heart monitor, which administers levels of either electric shock could be intercepted by an outside party when connected to the Internet. This potential attack would be almost indistinguishable from the patients' poor pre-existing conditions. In other words, the deaths that could be triggered would look like standard cases of diabetic coma or cardiac arrest; however, they would actually have been triggered by a breach in the device's security and loss of device control.
2. In AB21, Security flaws in the 24 medical devices investigated by Department of Homeland Security (DHS) could be utilized

Table 5
Losses of notable non-fatal software failures included in our study.

Year	Accident	Outcomes	Losses
1990	AT&T Network Outage	AT&T alone lost from $60 to $75 million in unconnected calls.	$60–$75 million
1991	AT&T SS7 Software Patch Failure	12 million people in various metropolitan areas had serious outages of telephone service.	
1992	Confirm Project Failure	A loss of $165 million.	$165 million
1993	Trips Reservation System Failure	A revenue loss of $61 million.	$61 million
1993	Cancellation of London Stock's Taurus Project	Taurus stock settlement system canceled after $600 million is spent.	$600 million
1994	Intel Pentium FDIV Bug	Intel's customers lost trust in Intel due to this incident.	
1994	Chemical Bank ATM Failure	$15 million dollar lost from customers' accounts.	
1996	Ariane 5	A decade of development cost of $7 billion and a cargo with a value of $500 million.	$7 billion and a cargo with a value of $500 million
1996	FoxMeyer Enterprise Resource Planning System Failure	A loss of $65 million invested in the project; The company went bankrupt soon after.	$65 million
1997	Failure of Oxford Health Plan's Bill and Claim System	A loss of $3.4 billion in corporate value.	$3.4 billion in corporate value
1997	USS Yorktown Carrier Shutdown	Engineers' efforts and work hours to fix the bug.	
1998	Solar and Heliospheric Observatory (SOHO)	The Spacecraft's entire value of $1 billion.	$1 billion
1999	Misplacement of Milstar Satellite	A loss of $1.23 billion.	$1.23 billion
1999	Mars Climate Orbiter	A loss of $85 million.	$85 million
1999	Mars Polar Lander	A loss of $120 million.	$120 million
1999	Hershey Foods Corporation ERP Implementation Failure	A loss of $151 million.	$151 million
2000	Nike's Supply-Chain Software Failure	A loss of $100 million.	$100 million
2002	CIGNA's CRM System Data Migration Failure	A loss of $445 million.	$445 million
2002	Sydney Water's CIBS Project Failure	At least a loss of $61 million.	$61 million
2002	FAA's Advanced Automation System Failure	A loss of $1.5 billion.	$1.5 billion
2003	AT&T CRM Upgrade Problem	AT&T lost a significant number of potential new customers.	
2003	Power Outage of Northeastern US and Southeastern Canada	The blackout affected more than 50 million people and cost a total of about $13 billion.	$13 billion
2003	McDonalds' Innovate Project Failure	A loss of $170 million.	$170 million
2004	Loss of Voice Communication between FAA Air Traffic Control Center and Airplanes	FAA Air Traffic Control Center lost voice contact with more than 400 airplanes they were tracking over the southwestern United States.	
2004	Hewlett Packard ERP System Migration Failure	A loss of $160 million.	A loss of $160 million
2004	Cancellation of Avis PeopleSoft ERP System	A loss of $54.5 million	A loss of $54.5 million
2005	DART Mishap	"Type A" (mission failure exceeding a government loss of $1 mission).	More than $1 million
2005	Prius Software Glitch	Work hours for engineers to examine the cars and reprogram the electronic control module. Also the reputation of Toyota and the trustworthy for the new type of vehicle.	
2005	Money Loss at Tokyo Stock Exchange	A loss of $225 million	A loss of $225 million
2006	Mars Global Surveyor	A loss of $154 million and the science.	$154 million
2006	Shutdown of Hartsfield-Jackson Atlanta International Airport	The airport falsely evacuated the security area for two hours; flights were backed up for days	
2006	London Tube Closure	Widespread delay to the London Underground.	
2006	Outages in Tokyo Stock Exchange	The exchange stopped operating for several hours during two outages.	
2007	Failure of Windows Genuine Advantage	Reputation of Windows damaged due to the increasing dislike of their anti-piracy tool.	
2007	Cancellation of Mission by U.S. Air Force F-22 Raptors	Air Force lost 48 hours of time and work hours for engineers to fix the code that was flawed.	
2007	Enron Payouts to Employees Stalled	7700 people were overpaid; 12,800 didn't receive enough.	$9.15 million, or $11.2 million with interest
2007–2009	Atlanta Water Bill Spike	Residents experienced unusual peaks in their water usage.	Reputation loss and man powers to recalculate the water usage.
2008	Emergency Shutdown of the Hatch Nuclear Power Plant	The nuclear power plant was forced to shut down for 48 hours, which caused $2 million loss.	$2 million loss
2008	Toronto Stock Exchange (TSX) Failure	TSX shut down 18 minutes after opening and remain un-operational for the entire day.	
2008	Security Flaw in Maximo Heart Monitor	The devices could impose potential danger on users due to security flaws.	
2010	GPS Dark Up	The problem rendered around 10,000 U.S. military GPS receivers useless for days.	
2011	Standards & Poor's Erroneous Alert on France's Credit Rate	Euro weakened against the dollar and French bond prices fell.	
2011	Nissan Leaf Software Glitch	Man powers to reprogram the control software.	Man powers to reprogram the control software
2014	Security Flaws in Medical Devices	Devices might be controlled remotely and became the cause of severe issues.	

Table 6
Summary of causes and their corresponding accidents.

Causes	Accidents
C01: Faulty value computation	AA07, AA10, AB01, AB03, AB12, AB29
C02: Security	AB21, AB27
C03: Inadequate design logic	AA01, AA02, AA04, AA06, AA08, AB05, AB13, AB20, AB32
C04: Compatibility	AA03, AB19, AB33, AB35, AB43
C05: Coding mistakes	AA11, AA12, AA13, AB02, AB04, AB06, AB07, AB08, AB09, AB10, AB15, AB17, AB22, AB23, AB24, AB28, AB30, AB37, AB39, AB40
C06: Complexity	AA06, AB16, AB32, AB33, AB44, AB45
C07: Improper reuse	AA07, AB01
C08: No warning	AA09, AA14, AB14, AB18, AB25
C09: Schedule pressure	AB05, AB44, AB45, AB47
C10: Faulty operations	AA01, AA02, AA03, AA05, AB13, AB25
C11: Conflict and insufficient manual	AA07, AA09
C12: Ignoring warnings	AA02, AA07
C13: Failure to communicate	AB04, AB06, AB11, AB32, AB34, AB35, AB41
C14: Failure of SQA and testing	AA04, AA08, AA09, AA10, AB01, AB08, AB09, AB10, AB11, AB12, AB15, AB16, AB17, AB19, AB22, AB33, AB34, AB35, AB36, AB44, AB45
C15: Human variables	AA04, AB18, AB25
C16: No backup plan	AA11, AB25, AB34, AB43

to remotely control the devices and induce a deadly jolt of electricity.

5.1.3. C03: Inadequate design logic

Incomplete logic behind the software system design can also lead to severe accidents. For safety-critical systems, due to the complexity of the design as well as to the complicated environment in which the systems perform, it is difficult in some cases to take every aspect of the functioning process into consideration.

1. In AB05, the spacecraft monitored the landing sensor as the only indicator of ground contact. However, the engineers failed to notice that the legs of the spacecraft could depress and release momentarily due to the Mars' atmosphere. Therefore, the spacecraft shut off the engines while too far above the Mars surface.
2. Though the crash of AA02 can be attributed to China Airlines' failure in upgrading their on-board computer system, the logic of the system was not able to consider specific scenarios and it finally cost the aircraft.
3. In AA03, the Flight Management System (FMS) interface was partly to blame due to the chosen implementation of the destination confirmation, which ultimately led the pilot to selecting the wrong city and a dangerous flight path.

5.1.4. C04: Compatibility

Software forms the essential core of many systems, especially some safety-critical systems. However, flawed interactions among software modules or between software and hardware components often result in unplanned behavior which can lead to serious mishaps with these systems.

1. In AB24, the Nissan Leaf had a flaw related to compatibility between the controller software for the air conditioning unit and the voltage sensor responsible for activating the failsafe mode. The failsafe mode was selected when the combination of these components created an incorrect reading of voltage which prevented the car from restarting.
2. In AB11, the failure of AT&T's CRM system upgrade was due to interoperability problems that occurred when AT&T chose to switch its system for handling Local Number Portability (LNP) from TSI to NeuStar. TSI and NeuStar had different interpretations of the Wireless Intercarrier Communications Interface Specifications (WICIS), which led to the inability of TSI's system to communicate with NeuStar's leading to the failure of the CRM.
3. In AB33, the supply-chain software developed by i2 also suffered from compatibility issue between two separate applications. The two applications used unique rules and data formats which resulted in the consequence that operators had to download the data from one application and reload the data into another.

5.1.5. C05: Coding mistakes

Coding mistakes (also called programming errors or errors committed during the code implementation stage) are some of the most commonly observed software-related causes of the previously described accidents (see Table 6). Even though these mistakes could be eliminated by peer reviews, static code analysis, test, and verifying and validating the system thoroughly, sometimes it is just not possible to root out all programming errors due to project time and budget constraints.

1. In AA06, a programming error in the software left a memory leak within the system, which slowly drained the available memory until the system had no more resources and halted.
2. In AB08, the code was written to send an electronic copy of each ATM withdrawal and transfer to the computer system that processes paper checks. Withdrawals were deducted from the bank's customer accounts once by the ATM and then again by the check system.
3. In AB09, the failure was caused by misplacement of a break statement in the code.

5.1.6. C06: Complexity

Software system functionality has expanded dramatically in recent decades (Davis et al.). As a result, the complexity of software has also become a key issue in determining the operational quality of the software. The more complex the software, the more likely it is that it will contain faults (Wong et al., 2016). Some examples:

1. A significant incident caused by increased complexity was evident in AB16. The project was way too complicated and involved stakeholders from various organizations. In order to meet the requirements from all parties, the complexity went way beyond control and the project eventually failed.
2. In the case of AB33, the system was not originally designed for the shoe and apparel industry. This led Nike to needing i2, the software vendor, to make a number of customizations to the software base. However, neither Nike nor i2 had the ability to properly manage the large and increasingly complex system even though the concept of the system was "*wonderful and compelling*" (Bousquin, 2001).

5.1.7. C07: Improper reuse

Software reuse has become a major trend in software engineering (Bauer and Vetro, 2016). Undoubtedly, reusing that software which have been proven to be robust on previous models can help us reduce the overall development costs. However, improper reuse could be attributed to as the causes for failures of software systems.

1. In AB01, the software used for Ariane 5 space shuttle was entirely reused from Ariane 4. However, no simulation testing was performed to ensure the reused software's performance on newly designed hardware platform.
2. In AA07, the software for Therac-25 was reused from Therac-20. However, the blocking shield in Therac-20 has been removed from Therac-25. Therefore, patients were imposed to excessive radiations when a software failure occurred with no shield in place.

5.1.8. C08: No warning or error messages provided

The warning or error messages are of great importance to operators. These messages are vital for operators to gain the current status of the systems. However, in some cases, no proper warnings were provided, which resulted in several failures that could have been prevented in the first place.

1. The accident board for AA02 concluded that there was inadequate warning or recognition capability for the flight crew to be alerted to the out-of-trim condition (Federal Aviation Administration Human Factor Team, 1996).
2. In AA05, no warning was provided to remind the pilots of the not-deployed flaps and slats.
3. In AB25, no warning signs were provided when the operators did tasks which should be taken care of very seriously.

5.2. Contributing factors: non-software related

Non-software related factors, whether related to the development or the execution of a project, can also be attributed to the cause of an accident. This section investigates such scenarios and examines how each named factor may have contributed to the failure of the system and provides some insight into avoidance of future accidents.

5.2.1. C09: Schedule pressure

Most projects have a scheduled deadline for completion. It is reasonable to have predefined milestones and a clear deadline for such complex projects; however, an inappropriate schedule will cause pressures which can lead not only to insufficient testing of the systems at the back end of the project but also to reduced overall product quality. Sometimes there is no flexibility in these schedule, such as launch windows for inter-planetary spacecraft.

1. In AB05, engineers focusing on the Mars Polar Lander overlooked the false indication of ground landing by the spacecraft due to significant schedule pressure. It has been stated in the report that the micro probes vibration, which resulted in shutdown of the system just before the landing, was identified during testing. However, this problem was never rectified.
2. This same general situation can also be found in AA06. Despite the ambitious scope of the second attempt of their project, completion was expected within a span of eleven months. This significantly smaller time frame stands in contrast to the three-year time span allowed for their less complex first attempt. The contract with Solo Electronics was signed four months before the expected delivery date, despite the fact that System Options needed the communication equipment from Solo Electronics in order to interface with its software. In the end, the deadline was forced to be extended. This choice also affected project costs.
3. A similar cause can also be found in AB12. The navigation logic's gain setting was changed very close to the planned launch, but the pressure to stay on schedule seemed to be the main reason that they opted to not test the change in the gain.

5.2.2. C10: Faulty operation of the system

Certain accidents have occurred due to incorrect decisions made by human operators during a certain critical point of the operational process. A single bad decision could possibly result in a complete disaster; however, the accident could have been avoided at that critical juncture.

1. One of the contributing causes of the AA01 was the faulty decision concerning the airplane. The pilots at the controls decided to continue to land in spite of an excessive tailwind and ground speed. This decision resulted in two consequences: they could not control the aircraft due to a delay in the braking system, and the aircraft landed beyond the normal touchdown point on the ground. As a result, the Airbus collided with the earth bank.
2. The loss of the Solar and Heliospheric Observatory (SOHO) was also due to a rapid faulty decision regarding the system. There were three Gyros (A, B, and C) used in this satellite system. The decision was made to deactivate Gyro A in order to conserve satellite life, but this was not communicated to the ESR mode of the spacecraft. Later, Gyro B was wrongly identified as faulty and turned off when the altitude error drove the system to ESR mode while integrating with Gyro A results. This resulted in the loss of altitude control as well as the loss of telemetry and power control.
3. In AA04, an incorrect decision to modify the airport's Minimum Safe Altitude Warning System (MSAW) was responsible for the aircraft's collision with Nimitz Hill. The Guam airport chose to limit the number of false alarms caused by the system by modifying the system, which inhibited the program from detecting an approaching plane that was below new lower safe altitude. Therefore, airport operations failed to notify Flight 801 of potential danger. Additionally, the choice to conduct the flight without the Guam Instrument Landing System was a risk that helped lead to the accident. As a result, the pilots of Flight 801 executed an incorrect technical procedure using a non-precision approach to landing and prematurely descending the plane into the terrain.

5.2.3. C11: Conflicting and insufficient manual

In some cases, it is not the software itself that is faulty, but rather the instructions on how to operate it. System manuals can become confusing or contain outdated information, which can lead to an incorrect assumption about the software and could lead to a misuse of the software; this has been shown as causes or contributors of some accidents.

1. In AA03, the naming conventions of the navigational charts were inconsistent with those of the flight management system. When the crew looked up the waypoint "Rozo," the chart indicated the letter "R" as its identifier. The FMS, however, had the city paired with the word "ROZO." As a result, when the pilot entered the letter "R," the system did not know if the desired city was Rozo or Romeo. It automatically picks Romeo, which is a larger city than Rozo, as the next waypoint.
2. In AA04, the crew aboard Korean Air Flight 801 was using an outdated flight map, which provided an incorrect value for the Minimum Safe Altitude. Since this given altitude was lower than the correct value, the plane was flying at a height much lower than that required for safe flight in the area.

5.2.4. C12: Ignoring warnings

Deliberately ignoring a warning given by a system could result in a poor outcome. When someone operating a software related system chooses to dismiss system-generated cautions and warnings, the potential for a mishap rises. Examples show that ignoring these warnings could make the difference between life and death.

1. In AA01 incident, the pilots ignored the wind shear warning and made a decision to "Land" instead of "Go-Around".
2. In the AA06, the second project, the LAS had received frequent warnings. A number of letters from computer consultancies and safety experts warned that the system was severely flawed. However, the complaint letters and concerns were ignored by the LAS executive management and were only addressed with excuses.
3. In the failure of AB32, the company chose not to listen to early warnings given by a consulting firm in Chicago, which strongly advised the drug company against the using a particular software product because it would not meet their expectations. However, FoxMeyer decided to keep their choice based on the software product's reputation. Ignoring this warning caused the loss of millions of dollars and, in the end, was a significant contributor to the bankruptcy of FoxMeyer.
4. In the case of AB33, the vice president of marketing for i2 mentioned that i2 had always recommended that their customers deploy their software system through various stages. However, Nike insisted on deploying it simultaneously to thousands of suppliers and distributors.

5.2.5. C13: Failure to communicate

In certain projects, there are cases in which the development teams, management teams, and the customers do not communicate effectively with each other, which results in some confusion and certainly some lack of clarity. Effective communication between different roles among different teams not only reduces uncertainty but also improves performance. This is a fundamental factor in the whole life-cycle of the software under development, a human-intensive effort.

Several incidents of record could be attributed to failure of communications to some extent. For example:

1. In AB06, the lack of communication between two operators of the spacecraft independently updated the two copies of the on-board system. The conflict was identified as the root cause for the incident.
2. In AB04, the investigation board indicated that some communications channels among project engineering groups were too "informal" (Isbell and Savage).
3. In AB11 and AB35, inconsistencies in data model between two sub-systems resulted in the two incidents.
4. In AB34, the lack of communication between technical personnel and on-site operators caused the consequence that ERP system could not identify specific storage locations.

5.2.6. C14: Failure of software development, testing and quality assurance process

A number of accidents were likely caused by issues in the development of software and quality assurance. It is critical to review our software development work in a systematic manner to ensure the software under development is checked thoroughly and that software product quality reaches a satisfactory level. However, in reality, some software is not adequately tested before deployment leaving latent defects to be encountered later.

1. The most notable incidents caused by inadequate testing and quality assurance processes among the accidents reviewed are the AB01 and AB03. The Ariane 5 rocket launch of 1996 only stayed operational for forty seconds before it was destroyed down range. Had there been more rigorous review process, system safety hazard analyses, tests, and/or real environment simulation, the accident may have been avoided.
2. The misplacement of the AB03 was due to a failure of the software quality assurance process; the wrong roll rate filter constant led to the crash. This error should have been found during the software reviews, testing, and/or system verification.
3. Other cases such as the failure of AB05, AA06, AB08, and AB15 all share similar causes, as insufficient workforce and project time led to short-cuts taken in the software development process, significantly reducing the effort focused on testing and quality assurance.

5.2.7. C15: Human variables

While different technical reasons for an accident exist, the variable of human activity must be taken into consideration:

1. In AA04, a contributing factor to the fatal incident was overlooked yet preventable – fatigue (National Safety Transportation Board, 2000). The pilot was not operating under the proper mental conditions, and thus, he was unable to control the aircraft in the appropriate manner.
2. For AB25, one member of the Windows Genuine Advantage team accidently sent pre-production code to the production servers. Since the production servers did not yet have the ability to enable stronger encryption or decryption of product keys during activation and validation, the production servers declined the valid product keys (MSDNArchive, 2007).
3. In AB11, there had been various rumors of future outsourcing and layoffs. This affected the project team's productivity due to a loss in morale. Some of the employees were searching for new jobs instead of focusing on the work assigned to them (Koch, 2004).

5.2.8. C16: No backup plan

Several projects were created with the intention to replace their current systems, as companies tend to throw out their old systems in favor of new ones. This leads to various problems, since the new system may be faulty, and unpredictable issues may arise.

1. In AB11, there was no backup plan in the case of an upgrade failure. AT&T was unable to roll back to the previous version of the system, since the "project managers didn't preserve a sufficient portion of the previous version to make this possible" (Koch, 2004).
2. Another example is AB34. The company did not have a contingency plan in place when the ERP system was being implemented. When the project eventually failed, the company did not have any backup and soon went bankrupt.
3. Similar cause can also be identified in AB43. The data was impossible to be recovered when a crash happened.

In the table below, we have categorized all the accidents discussed in this paper based on the causes identified. Note that one specific accident could appear in different rows due to the fact that multiple causes can be identified for the corresponding accident.

6. Lessons learned

In this section, we present the lessons learned from the occurrence of the previously described accidents in order to help us prevent the same or similar accidents from happening in the future. Instead of summarizing the lessons from each accident, we present the lessons in three perspectives: managerial, technical, and sociocultural, respectively.

6.1. Managerial

With the development of software systems, they are becoming increasingly large when compared with those decades ago. Therefore, issues related to software project management have been revealed as the root cause for most of the failed projects which have been canceled due to the fact that they could not fulfill customers' expectations. As it has been stated in (Office of the Under Secretary of Defense for Acquisition, 1987), "the task force is convinced that today's major problems with military software development are not technical problems, but management problems." Therefore, it is of great value that managers of large-scale software systems pay more attentions to the management of their project under development and lay a solid foundation for their projects to be successful.

(1) Properly estimate the time and costs

It is important to estimate the time and costs for specific software projects. First, a good estimation helps managers make appropriate decisions in various situations and keep track of the direction a project is moving forward to. Second, a blue print could be given to managers and engineers on how long the project should take and how much money it would require. Third, the estimation could provide hints on how well the project is progressing. In addition, project baselines as well as time-phased budgets are largely dependent on how good the estimation is conducted.

However, cost and schedule overrun have become major issues in the industry of software development (Ewusi-Mensah, 1997). Though it is understandable for companies to desire completion of a software product with minimum cost and time, the development of software should not be underestimated in its complexity. One has to admit that even though budget and schedule pressures can help shorten the development cycle as well as reduce the money spent in the process, the development and testing effort for a specific project may also be degraded, which could significantly lower the quality of the software produced. Moreover, there is no tangible, accurate estimator for software development cost or time. Thus, projects should be expected to encounter various situations that will slow down development and raise the cost. In the Mars Polar Lander mission, the cost-risk tradeoff should have been carefully monitored even though costs were tightly constrained. However, an insufficient amount of time and staffing led to fewer system checks. Furthermore, for the DART mishap, despite the fact that the modification was made with the launching date approaching, the change still should have been tested, since it did influence the quality of the product.

Usually several categories of approaches can be applied in the process of estimating the time and costs:

- Estimating by analogy are based on the comparison between the proposed project and previously completed similar projects. Actual data from the completed projects are extrapolated to estimate the proposed project. This method can be used either at system-level or at the component-level. However, disadvantages also exist. For example, the estimation is partly determined by how well the proposed project is described and the similarity between projects is evaluated. Also, the approach is based on the assumption that efforts of the proposed project are known based on previous similar projects.
- *Top-down estimates* are achieved by someone who uses his/her knowledge and/or experience to determine the duration of the project as well as the total costs. However, these estimates are usually from top managers who may not have enough knowledge of how the project is actually implemented. In addition, no time and efforts for specific tasks are considered. One example is the AB16. The estimates of the project were almost useless since no engineer was actually involved in the estimating process. As a result, the project went way over budget and experienced long delays in delivering the project.
- Though the top-down approaches suffers from these shortcomings, they are useful in the early stage of software project planning. From a strategic level, these approaches can help managers evaluate various proposals for software projects and select the better ones for further consideration. Several top-down approaches could be considered, such as Consensus Methods (Fink et al., 1984), Ratio Methods (Shwartz et al., 1995), Apportion Methods (Fink et al., 1984), etc.
- *Bottom-up estimates* use the estimated cost of each software components and then combine the results to arrive at an estimated cost of overall project. It aims at constructing the estimate of a system from the knowledge accumulated about the small software components and their interactions. The leading method using this approach is COCOMO's detailed model (Boehm, 1981). Compared with those top-down approaches, these approaches are more stable and can provide more accurate estimates. However, shortcomings also exist. The time and costs for system-level are sometimes ignored, and necessary information may not be available in early phases.
- *Algorithmic methods* have also been applied in this area such as Putnam model (Gupta et al., 2016), Function Point Analysis Based Methods (Gupta et al., 2016), etc.

(2) Improve operational procedures and processes and produce accurate documentations

Appropriate procedures and processes for operating the system need to be followed properly and thoroughly so that testers and users of the system will be able to test or operate the system correctly. Though it may be preferable to use flexible software that meets the requirements of different groups of users, implementation of the software, especially for situations that are safety-critical, should be designed to avoid ambiguity. In addition, proper warnings and messages regarding the misuse of the system should be displayed to the end users whenever potential risks exist. The most notable case is the AA09, in which the physicians applied the software in a way that was inappropriate. Another example is the accident of AA03, in which the system displayed no message regarding the next waypoint selected by the pilots where there were multiple cities with similar names.

Operators should also be more carefully trained and bear in mind that the recommended procedures must always be performed. For example, in the accident of AA05, the pilots ignored the pre-flight checklist and just simply repeated "checked" during the procedure. Therefore, they failed to realize that the flats and slats were not deployed for takeoff.

In addition, manuals should be written thoroughly and kept up to date with the latest model of the system, as this is the foundation to ensure correctness of the operations performed by the end-users. Even though operators are more likely to proceed based on the training they received and won't turn to the manuals or other documentations for help, bad things could happen once they read the manuals or other documentations which happened to be questionable. This lesson can be learned directly from both the AA01 and AA03 accidents. For the former accident, the aircraft operation manual contained conflicting information, such as the appropriate speed to use to counter wind shear; for the latter, the inconsistency of navigational charts was a contributing factor that resulted in wrong selection of the next waypoint. The same applies to the crash of AA04, which was caused by the usage of an outdated version of the flight map.

(3) Manage the risks properly during the implementation

Bad risk assessment and management could bring extremely negative impacts to projects under development. Due to the instability of factors involved in the development of software projects,

managers and engineers should put more attentions to the potential risks and figure out a way to manage them. In addition, it will also provide valuable recommendations for decision-making. An example of bad risk assessment is AB16. Without an appropriate risk assessment, the managers failed to cancel the project when there was no chance that the project could succeed. More and more money and man powers were spent on the project instead.

Normally, the risk assessment and management are performed in the following steps (Project Management Institute):

- Risk identification, which aims to analyze the project and the potential risks underneath;
- Risk assessment based on the risks' severity, likelihood of occurrence, and the degree of controllability;
- Risk response development, which contains the actions required to develop a contingent and actionable plan to reduce the possible impacts brought by the risks.
- Risk response control, which is mainly focusing on implement the strategy and plans achieved in the previous steps and make decisions (i.e., change management strategy, continue/cancel the project., etc).

Several most commonly seen risks have also been discussed in (Boehm, 1991), which could act as the guidelines for managers and engineers to refer to and keep an eye on during their development of large-scale software projects.

(4) Keep a clear vision of the overall software development process

While this seems obvious, in some cases this lesson was clearly violated, such as in the London Ambulance Services outage. Project managers should have been deeply knowledgeable about the mechanism of the entire system as well as the tools used in the process. In addition, the development process should have been thoroughly audited to ensure proper management of the process. In this way, not only can the quality of the system be assured, but also the possibility of completing the development process within a predefined time and budget limit will be significantly increased.

6.2. Technical

Based on the descriptions we provided in previous sections, a large number of accidents and project failures happened due to technical issues, such as flawed program code, faulty instructions, inconsistencies of interfaces between modules and/or sub-systems. In this sections, we provide the lessons learned and recommendations on how to resolve these issues.

(1) Quality assurance process should be performed more thoroughly

A large number of accidents and failures discussed previously (i.e., AA04, AA05, AA06, and AA07) have been attributed to the lack of proper quality assurance and adequate testing. Software Quality Assurance (SQA) involves the entire software development process – monitoring and improving the process, making sure that any agreed-upon standards and procedures are followed, and ensuring that problems are found and dealt with (Hower).

SQA is essential to every software development process (Rosenberg and Gallo, 2002). SQA improves development to deployment time, reduces the number of incomplete or missed deadlines, and reduces time spent on making sure that requirements, design, code, and documentation look the same by ensuring consistency without doing it all manually. Using SQA audits to evaluate that guidelines have been met before advancing in the development process is essential. Also, audits can be used to show stakeholders that continuous progress is being made on a project.

The following aspects should be emphasized with respect to the application of SQA techniques in real-life projects:

- Standards and procedures. Every organizations or companies should establish a set of standards and procedures since they serve as the foundation of the software being developed. These standards and procedures can be formed from unique perspectives, such as requirements, design, code, documentation, and so on. Examples can be found at (NASAb).
- Audit. Audit is the major technique used in the SQA process. It should be carried out routinely throughout the entire life cycle of software projects, especially for those large-scale systems with functionalities closely related to safety and/or security. The purpose of using an audit is to assure that proper control procedures are being followed, that required documentation is maintained, and that the developer's status reports accurately reflect the status of the activity. An example guideline for audit can be found in (NASAa).
- Tools. Various tools can be used to help the SQA process. These tools include QA C++, Jtest, Sablime Configuration Management System, and so on.

(2) Testing should be careful designed and strictly performed

Testing must be thoroughly performed in every phase of project development in order to ensure the quality of deliverables. Also, it is essential to make sure that testing should be adequate and that software is always tested using real equipment rather than executed in a simulation environment for verification purposes. In the three incidents from AT&T (AB09, AB10, and AB11), the modifications/updates should have been carefully evaluated to determine whether it would impact system reliability. For AB05 accident, simulated stress testing and fault injection should have been an integral part of the software testing process in order to discover hidden faults and to establish the limits of the system. Many of the problems that occurred could have possibly been avoided if thorough software testing had been put in place.

Different techniques can be applied in different stages of project development:

- In the early phase of implementation, unit testing should be carefully performed to ensure that each component and/or subsystems of the system perform as expected (AB04);
- Later in the integration stage, integration testing as well as system-level testing need to be carried out to make sure the system meets the functional requirements defined (AB35);
- Other tests aiming to prove the non-functional requirements are guaranteed satisfactorily. These testing techniques include stress testing (AA06), reliability testing (AB37), security testing (AB21 and AB27), compatibility testing (AB33), fault injection (AB05) and so on;
- Once a system is modified or updated, regression testing is need to ensure no other functionality is affected by the modifications (AB08, AB09, and AB18);
- Even well-tested, proven software must still be thoroughly re-tested and rechecked when adapted and deployed in a different environment (AA07 and AB01). It should not be automatically assumed that new and reused hardware or software components are working before the product is even tested;
- Techniques such as boundary value analysis should also be applied in the testing process (AB04 and AA01).

Also, it is worth noting that various testing standards should be applied in the testing procedure. Each testing standard is fulfilled by different test cases. Therefore, some bugs may be caught by test set satisfying a specific type of testing standard but not by test set achieved by using other standards.

(3) Be prepared for the "bad"

Even though we are always trying our best to prevent bad things from happening, unfortunately they do. Therefore, it is important that preparations are in place in case undesirable things happen. The following aspects should be paid more attention to:

- A ready-in-a-minute backup is available. When making updates or creating a new system to replace the current one, one should keep in mind the importance of having a backup plan in case something goes wrong during the update or deployment of a new system. An example of this is the case of AB11. AT&T should have considered making preparations for a backup plan in order to avoid a huge loss of money and reputation. However, they were pressured to fix the system and stabilize it by the deadline. It is worth mentioning that, due to differences between hardware and software, a simple "redundancy" of the original system (meaning that the backup system runs the same version of software used by the original system) may not be useful in specific circumstances, as the backup system could suffer from the same defects as the original one (AB23). Thus, preparing a proper backup system is essential, especially for those critical systems (AB25).
- In order to ensure the functioning of the system, engineers should design multiple error/failure handling mechanisms. This is even more important for safety-critical systems. Cases like the AB01 clearly demonstrate the importance of customization of failure handlers for various exceptions.
- "Testing for the bad" is as important as "testing for the good". In other words, we should test not only what the system should do but also what it should not do. (AB21 and AB27)

(4) Design flaws should be avoided as completely as possible

A large number of accidents were caused by design flaws, which signifies that there should be a greater effort to avoid them in the future. Several aspects of the software should be considered during the entire software lifecycle, i.e., security, complexity, compatibility, and so on. This requires the whole cycle of software development to be managed systematically and thoroughly. Several accidents have demonstrated the importance of avoiding these defects, such as the security flaws in AB21 and AB27, the crash of AA03, the AB12, etc.

(5) Warnings and/or error messages should be prompted whenever feasible

Warnings and error messages play an important role in notifying the operators that something goes wrong and needs to be taken care of. A number of accidents have been identified to be caused by a lack of appropriate warnings (AA02, AA05, and AB23). Warnings can be used in different scenarios, such as:

- Notify the operators that something goes wrong and needs to be taken care of (AA05);
- Warn the operators that something bad would happen if continue (AA02);
- Remind the operators to be careful when doing the next operations (AB23).

6.3. Sociocultural

Aside from the two aspects discussed above, the sociocultural aspect is also of great value in organizations and companies. More importantly, since the scale of software systems are becoming larger and larger, it is nearly impossible for such systems to be implemented solely by one single company. As a result, more attentions should be paid to not only the sociocultural relationship between groups within an organization but also that among various organizations involved in the implementation and maintenance.

(1) Establish effective communication channels

Communications between groups involved in the project implementation are vital. If an engineer only focuses on the job assigned to him/her and overlooked what others are doing, there are plenty of reasons that doubts should be put on the outcome of their project. Similarly, bad communications between stakeholders and software manufacturers as well as that between different manufacturers can also lead the project to failure.

Therefore, we suggest the following:

- Encourage the communications between developers (AB06). Various methods are helpful in creating an atmosphere that is comfortable for communications, such as arrange more experienced developers sitting next to newbies, organize regular group meetings, encourage the use a sketch board, etc. Also, it is important to inform the colleague who is responsible for the same task with you to avoid possible conflicts in the modifications.
- Establish an open environment for the developing team (AB35). Instead of maintaining a hierarchical organizational structure where the manager makes all the decisions, an open environment is vital in order to ensure opinions from various aspects are considered. Consensus should be made whenever a conflict occurs.
- Keep the stakeholders aware of what is currently going on (AB16). By doing so, the stakeholders and developers will stay on the same page, which is not only important to ensure the needs of stakeholders are met but also essential to relieve developers' pressure from the stakeholders.

(2) Properly and adequately train the employees

Employees not properly trained can be another factor in dealing with the unexpected situations (AA06 and AB34). Therefore, special notices should be paid to the training process, especially when you are introducing a whole new system to replace the legacy one. Below are several recommendations on this item:

- Spend sufficient time in training your employees (AA04);
- Don't train your employees long time when you don't even have access to the new system (AA06);
- Don't try to train your staff in time slots with large amounts of tasks (AB34);
- Always remind your staff that standardized operations must be followed (AA05).

(3) Control the human variables that may impair the project

We have to acknowledge that even the most experienced employees sometimes make mistakes. Therefore, efforts should be made to control these factors in order to limit their impacts to the largest extent. The following aspects are of practical value to managers:

- Keep an eye on the mental and physical conditions of your employees (AA04);
- Proper warning or error messages should be prompted when faulty operations are performed (AA01);
- Methods to retract an operational error should be implemented in case something went wrong (AB18).

7. Discussions

In this section, we discuss several factors which may have an influence on the conclusions we made in this paper.

7.1. Selection of accidents

We have to admit that it is impossible to cover every accident which may be relevant to this paper. In this paper, we select accidents for analysis based on several criteria, such as the extent of the damage, the extent to which software played a contributing cause, and the recentness of the accidents. Instead providing a complete list which contains all the accidents related to software failures, we intend to emphasize the importance of paying more attention to software by illustrating a picture of how defected

software could result in loss of time, money, and more importantly, loss of life. In addition, we do not consider deliberately constructed software failures, such as in 1982 when CIA operatives planted a logic bomb in system controlling the trans-Siberian gas pipeline triggered a massive explosion (no loss of life was reported) (Safire, 2004).

7.2. Contributing causes, not SOLO causes

It is also difficult to know the causal sequence between the software failure and human harm, for example, software caused failure of the electrical grid leading to medical device or transportation system failures. As a result, software failures are not the solo cause of various accidents. However, undoubtedly, software has directly or indirectly played a causal role in each of the accidents discussed in this paper, which is why we emphasize that more attention should be paid to software, especially in safety-critical areas.

7.3. Information sources

We have tried not to bias our description of the accidents. However, due to the fact that we are unable to perform firsthand investigations, the only information source is documents from different sources. As a result, information that are directly related to the accidents may not be available, such as the manufacturers' development plan, quality control process, and so on. Though we are very careful to only report what we can get from these documents, there is no guarantee that the documents themselves are correct. To deal with this problem, we search for multiple confirming sources to alleviate the possible biases. In fact, the same problem exists for all other surveys focusing on analyzing different accidents (Charette, 2005; Geppert, 2004; Laplante, 2013; Leveson and Turner, 1993; Perrow, 2008; Pinkney, 2002).

8. Conclusions

This paper reviews catastrophic accidents which led to losses of life as well as losses of time and money. After identifying the roles played by software in causing the accidents, though we have to admit that most of the accidents were not caused by the software alone (other factors such as human errors, budgets and time limits, etc.), the software has been a key factor in the causal chain of the accidents. It clearly alarms that the importance of software systems grows exponentially due to the fact that these systems have been applied to most of advanced systems, including safety-critical systems.

By identifying the causes and the lessons learned from those accidents, either fatal or non-fatal, similar accidents could be avoided in the future when developing systems that share common features with those failed. More importantly, a more systematically organized process of developing huge software systems should be applied and the verification and validation of safety-critical systems needs to be deployed in order to assure the quality of these software systems.

Although discussions on several accidents, for example the healthcare.gov accident, have started on the risks forum, no one has shown a direct causal relationship to failures of that site and injury or loss of life. Moreover, it is clear that we need to wait to know more. We are only beginning to understand the complex interactions of systems of (software) systems and the ubiquity of embedded software in public infrastructure, transportation systems, consumer products, and more leaves me incredulous that someone who has worked on complex software systems could believe that it is possible to build such systems without faults and risk to the public.

Acknowledgement

The authors would like to give special thanks to Dr. Michael F. Siok from Lockheed Martin Aeronautics Company, Fort Worth, Texas for his help in preparation of this article.

References

"2015 Servile Airbus A400M Atlas crash", Wikipedia. (https://en.wikipedia.org/wiki/2015_Seville_Airbus_A400M_Atlas_crash).
"American Airlines Flight 965: Crash on the Mountain", AviationKnowledge: The First Aviation WikiJournal. October 4, 2010 (http://aviationknowledge.wikidot.com/asi:american-airlines-flight-965:crash-on-the-mountain).
"American Airlines Flight 965″, Wikipedia. (http://en.wikipedia.org/wiki/American_Airlines_Flight_965).
"Ariane 5," Wikipedia (http://en.wikipedia.org/wiki/Ariane_5).
Ariane 501 Inquiry Board, July 1996. Ariane 5, Flight 501 Failure.
Associated Press, June 1 2010. Glitch Reveals Military Reliance on GPS Tech. FOX News http://www.foxnews.com/tech/2010/06/01/glitch-shows-military-relies-gps.
Bacon, F., June 11 2013. Failure Examples of IT Projects. IT Cortex http://www.it-cortex.com/Examples_f.htm.
Bass, A., March 15 2003. Integration Management – CIGNA's Self-Inflicted Wounds. CIO http://www.cio.com/article/2440140/enterprise-software/integration-management---cigna-s-self-inflicted-wounds.html.
Bauer, V., Vetro, A., July 2016. Comparing reuse practices in two large software-producing companies. J. Syst. Softw. 117, 545–582.
Berler, R., "Saving the Pentagon's Killer Chopper-Plane," Wired (http://archive.wired.com/wired/archive/13.07/osprey_pr.html).
Berman, M., February 9, 2016. What Happened After Washington State Accidentally Let Thousands of Inmates Out Early. The Washington Post https://www.washingtonpost.com/news/post-nation/wp/2016/02/09/heres-what-happened-after-the-state-of-washington-accidentally-let-thousands-of-inmates-out-early/?utm_term=.af947e3ee4ed.
Best, J., October 22 2004. Avis Cancels PeopleSoft ERP System. ZDNet http://www.zdnet.com/avis-cancels-peoplesoft-erp-system-3039171024.
Boehm, B.W., 1981. Software Engineering Economics. Prentice Hall.
Boehm, B.W., January 1991. Software Risk Management: Principles and Practices. IEEE Software.
Borrás, C., 2006. Overexposure of radiation therapy patients in Panama: problem recognition and follow-up measures. Revista Panam. Salud Publica 20 (2/3), 173–187.
Bousquin, J., February 27 2001. i2's Software Just Didn't Do It for Nike. TheStreet http://www.thestreet.com/tech/software/1322748.html.
Brady, P.J., 2000. In: Common Law Remedies for Computer Failures, 36. TRIAL, pp. 73–79.
Brooke, J., January 19 2006. After Panic, Tokyo Market Rebounds. New York Times http://www.nytimes.com/2006/01/19/business/worldbusiness/19livedoor.html?pagewanted=all.
Burke, D., November 1995. All Circuits are Busy Now: The 1990 AT&T Long Distance Network Collapse http://users.csc.calpoly.edu/~jdalbey/SWE/Papers/att_collapse.html.
Cain, F., November 10 2006. Contact Lost with Mars Global Surveyor. Universe Today http://www.universetoday.com/926/contact-lost-with-mars-global-surveyor/.
Charette, R.N., September 02 2005. Why software fails. IEEE Spectr. http://spectrum.ieee.org/computing/software/why-software-fails.
Charette, R.N., November 11 2011. S&P's 'technical error' outrages French government. IEEE Spectr. http://spectrum.ieee.org/riskfactor/telecom/internet/sps-technical-error-outrages-french-government.
"China airlines A300 flight 140 at Nagoya," (http://lessonslearned.faa.gov/ll_main.cfm?TabID=3&LLID=64).
"Confirm Project," Wikipedia (http://en.wikipedia.org/wiki/Confirm_Project).
Cone, E., April 9 2002. The Ugly History of Tool Development at the FAA. Baseline http://www.baselinemag.com/c/a/Projects-Processes/The-Ugly-History-of-Tool-Development-at-the-FAA/.
Croomes, S., 2006. Overview of the Dart Mishap Investigation Results. National Aeronautics and Space Administration Technical report.
Dalcher, D., 1999. Disaster in London: the LAS case study. In: *Proceedings of the IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS)*,March 7 – 12, pp. 41–42.
Davis, S.J., MacCrisken, J., and Murphy, K.M., "Economic perspectives on software design: PC operating systems and platforms," *Microsoft, Antitrust and the New Economy: Selected Essays*, Springer, 361–419.
"Edwin I. Hatch Nuclear Power Plant," Wikipedia (http://en.wikipedia.org/wiki/Edwin_I._Hatch_Nuclear_Power_Plant).
Elliott, D., June 1 2010. Glitch Shows How Much US Military Relies on GPS Receivers. USA Today http://usatoday30.usatoday.com/tech/news/2010-06-01-military-gps_N.htm.
Elmer-Dewitt, P., January 29 1990. Ghost in The Machine. (http://content.time.com/time/magazine/article/0,9171,969266-1,00.html .).
Ewusi-Mensah, K., September 1997. Critical issues in abandoned information systems development projects. Commun. ACM 40 (9), 74–80.

Feder, B.J., March 12 2008. A Heart Device Is Found Vulnerable to Hacker Attacks. New York Times http://www.nytimes.com/2008/03/12/business/12heart-web.html?_r=1&.

Federal Aviation Administration Human Factor Team, June 18 1996. Report on: The Interfaces Between Flightcrews and Modern Flight Deck Systems Available at http://www.tc.faa.gov/its/worldpac/techrpt/hffaces.pdf .

Federal Communications Commission, July 16 2002. FCC Extends Wireless Local Number Portability Deadline by 1 Year to November 24, 2003. FCC News.

Fink, A., Kosecoff, J., Chassin, M., Brook, R.H., September 1984. Consensus methods: characteristics and guidelines for use. Am. J. Public Health 74 (9), 979–983.

Fox, M., Johnson, M.A., October 2 2014. Texas Hospital Makes Changes After Ebola Patient Turned Away. NBCNews http://www.nbcnews.com/storyline/ebola-virus-outbreak/texas-hospital-makes-changes-after-ebola-patient-turned-away-n217296.

Fox-Meyer Drugs, September 16 2012. Why Projects Fail (http://calleam.com/WTPF/?p=3508 .

Freudenheim, M., October 28 1997. Billing Problem Leads to Losses For Big H.M.O. New York Times http://www.nytimes.com/1997/10/28/business/billing-problem-leads-to-losses-for-big-hmo.html.

Gallagher, S., Barrett, L., July 2, 2003. McDonald's: McBusted. Baseline http://www.pcmag.com/article2/0,2817,1173628,00.asp.

Geppert, L., November 2004. Lost radio contact leaves pilots on their own. IEEE Spectr. 41 (11), 16–17.

Gilmore, D., May 7 2009. The Top Supply Chain Disasters of All Time. Supply Chain Digest http://www.scdigest.com/assets/FirstThoughts/09-05-07.php.

Greenberg, A., October 22 2014. DHS Investigates Possible Vulnerabilities In Medical Devices, Report Indicates. SC Magazine http://www.scmagazine.com/dhs-investigates-possible-vulnerabilities-in-medical-devices-report-indicates/article/378735/.

Gross, K., September 2004. Dispelling the Myth of the MV-22. Military.com http://www.military.com/NewContent/0,13190,NI_Myth_0904,00.html.

Gupta, S., Tiwari, S., Singh, H., Shukla, A., Raghuvanshi, H., November 2016. A comparison between various software cost estimation models. Int. J. Emerg. Trends Sci. Technol. 3 (11), 4771–4776.

"Hacking the Human Heart: Medical Devices Found Subject to Technical Attack," Future Crimes, March 7, 2010 (http://www.futurecrimes.com/article/hacking-the-human-heart-medical-devices-found-subject-to-technical-attack-2/).

Hammonds, K.H., November 6 1997. Behind Oxford's Billing Nightmare, How a misconceived system cost the health-care giant millions. BusinessWeek http://www.businessweek.com/1997/46/b3553148.htm.

Hansell, S., February 18 1994. Glitch Makes Teller Machines Take Twice What They Give. New York Times http://www.nytimes.com/1994/02/18/business/glitch-makes-teller-machines-take-twice-what-they-give.html?pagewanted=all&src=pm.

"Hartsfield-Jackson Atlanta International Airport," Wikipedia (http://en.wikipedia.org/wiki/Hartsfield–Jackson_Atlanta_International_Airport).

Hays, K., July 3 2007. Data Error Stalls Enron Payouts to Employees. Houston Chronicle http://www.chron.com/business/enron/article/Data-error-stalls-Enron-payouts-to-employees-1839108.php.

Hepher, T., "Exclusive: A400M Probe Focuses on Impact of Accidental Data Wipe." (http://www.reuters.com/article/us-airbus-a400m-idUSKBN0OP2AS20150609).

Hill, B., February 26 2007. Lockheed's F-22 Raptor Gets Zapped by International Date Line. DailyTech http://www.dailytech.com/Lockheeds+F22+Raptor+Gets+Zapped+by+International+Date+Line/article6225.htm.

Hower, R., Software QA and Testing Resource Center. Web site: http://www.softwareqatest.com/.

Isbell, D. and Savage, D., "Mars Climate Orbiter Failure Board Releases Report, Numerous NASA Actions Underway in Response," Available at http://mars.nasa.gov/msp98/news/mco991110.html.

Jacobs, M., August 8 2013. 13 of the Largest Power Outages in History – and What They Tell Us About the 2003 Northeast Blackout. Union of Concerned Scientists http://blog.ucsusa.org/2003-northeast-blackout-and-13-of-the-largest-power-outages-in-history-199.

Janeba, M., 1995. The Pentium Problem http://www.willamette.edu/~mjaneba/pentprob.html.

Kanellos, M., October 14 2005. Software Glitch Stalls Some Toyota Hybrids. CNET http://news.cnet.com/Software-glitch-stalls-some-Toyota-hybrids/2100-11389_3-5895574.html.

Kaste, M., January 1 2016. 2 Prisoners Mistakenly Released Early Now Charged in Killings. National Public Radio http://www.npr.org/2016/01/01/461700642/computer-glitch-leads-to-mistaken-early-release-of-prisoners-in-washington.

Koch, C., April 15 2004. Project Management: AT&T Wireless Self-Destructs. CIO http://www.cio.com/article/2439700/project-management/project-management–at-t-wireless-self-destructs.html.

Koch, C., November 15 2002. Supply Chain: Hershey's Bittersweet Lesson. CIO http://www.cio.com/article/2440386/supply-chain-management/supply-chain---hershey-s-bittersweet-lesson.html.

Johnson, M.D., February 20, 2007. Raptors arrive at Kadena. The Official Web Site of Kadena Air Base. (http://www.kadena.af.mil/news/story.asp?id=123041749 .).

JPL Special Review Board, March 2000. Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions.

"Korean Air Flight 801″, Wikipedia. (http://en.wikipedia.org/wiki/Korean_Air_Flight_801).

Krebs, B., June 5 2008. Cyber Incident Blamed for Nuclear Power Plant Shutdown. Washington Post http://www.washingtonpost.com/wp-dyn/content/article/2008/06/05/AR2008060501958.html.

Krigsman, M., December 1 2007. PricewaterhouseCoopers (PwC) Sued by Sydney Water over IT Failure. ZDNet http://www.zdnet.com/blog/projectfailures/pricewaterhousecoopers-pwc-sued-by-sydney-water-over-it-failure/506.

Lake, M., September 9 2010. 11 Famous Software Bugs. Computerworld UK http://www.computerworlduk.com/in-depth/infrastructure/3239009/11-famous-software-bugs/?pn=4.

Lann, G.L., January 2002. Is software really the weak link in dependable computing? Workshop on Challenges and Directions for Dependable Computing.

Laplante, P.A., February 3 2012. Answers to FAQs about Software Licensing. The Institute http://theinstitute.ieee.org/ieee-roundup/opinions/ieee-roundup/answers-to-faqs-about-software-licensing.

Laplante, P.A., November 25 2013. Misconceptions About Licensing Software Engineers. The Institute http://theinstitute.ieee.org/ieee-roundup/opinions/ieee-roundup/misconceptions-about-licensing-software-engineers.

Laplante, P.A., Thornton, M., July 2011. When do Software Systems Need to be Engineered. Today's Engineer http://www.todaysengineer.org/2011/Jul/licensure.asp.

Leveson, N., September 1995. Safeware: System Safety and Computers. Addison-Wesley.

Leveson, N.G., Turner, C.S., July , 1993. An investigation of the therac-25 accidents. IEEE Comput. 26 (7), 18–41.

Levin, A., August 25 2008. Spanair Crash Probe to Look at Pilot Reactions. USA TODAY.

Leyden, J., November 25 2005. Fujitsu Execs Take Pay Cut After Tokyo Exchange Crash. The Register http://www.theregister.co.uk/2005/11/25/stock_exchange_glitch_fall_out/.

"London Stock Exchange – Taurus," Why Projects Fail, September 14, 2012 (http://calleam.com/WTPF/?p=3474).

Los Angeles Times, "FAA to probe Radio Failure," September 17, 2004.

"Lufthansa Flight 2904," Wikipedia (http://en.wikipedia.org/wiki/Lufthansa_Flight_2904).

Lum, A., "Patriot Missile Software Problem." (http://sydney.edu.au/engineering/it/~alum/patriot_bug.html).

Mars Climate Orbiter Official website (http://marsprogram.jpl.nasa.gov/msp98/orbiter/).

Microsoft, "Description of Windows Genuine Advantage (WGA)" (http://support.microsoft.com/kb/892130).

Modine, A., December 19 2008. Canada's Biggest Stock Exchange Back After Day-Long Shutdown. The Register http://www.theregister.co.uk/2008/12/19/tsx_bites_it_for_a_day.

Moteff, J., Parfmak, P., October 2004. Critical Infrastructure and Key Assets: Definition and Identification. Congressional Research Service, Library of Congress.

MSDNArchive, August 29 2007. So What Happened?. Genuine Windows Blog http://blogs.msdn.com/b/wga/archive/2007/08/28/so-what-happened.aspx.

National Transportation Safety Board, 2000. Controlled Flight into Terrain, Korean Air Flight 801, Boeing 747-300, HL7468, Nimitz Hill, Guam, August 6, 1997 *Aircraft Accident Report*, NTSB/AAR-99/02.

NASA, "NASA-GB-A301: Software Quality Assurance Audits Guidebook" (https://www.hq.nasa.gov/office/codeq/doctree/nasa_gb_a301.pdf).

NASA, "NASA-STD 8739.8: Standard for Software Assurance" (https://www.hq.nasa.gov/office/codeq/doctree/87398.htm).

NASA Science, "Mars Polar Lander" (http://nasascience.nasa.gov/missions/mars-polar-lander).

National Safety Transportation Board, 2000. Aircraft Accident Report NTSB/AAR-00/01.

Neumann, J., Lamar, M., November 11 2011. S&P 'Oops' on Rating of France Is Probed. The Wall Street Journal http://online.wsj.com/article/SB10001424052970204224604577030083804142906.html.

Neumann, P.G., October 28 1994. Computer-Related Risks, first ed. Addison-Wesley Professional.

Neumann, P.G., January 31 2014. The Risks Digest: Forum On Risks To The Public In Computers And Related Systems. retrieved http://catless.ncl.ac.uk/Risks .

"Nissan Leaf recalled for software glitch," CBSNews April 21, 2011 (http://www.cbsnews.com/news/nissan-leaf-recalled-for-software-glitch/).

NIST Report, May 2002. Software Errors Cost U.S. Economy $59.5 Billion Annually NIST Planning Report 02-3.

"Northeast Blackout of 2003," Wikipedia (http://en.wikipedia.org/wiki/Northeast_blackout_of_2003).

O'Dell, J., April 12 2011. Nissan Leaf Quality Glitch Detected. AutoObserver http://www.edmunds.com/autoobserver-archive/2011/04/nissan-leaf-quality-glitch-detected.html.

Office of the Under Secretary of Defense for Acquisition, September 1987. Report of the Defense Science Board Task Force on Military Software Available at http://repository.cmu.edu/cgi/viewcontent.cgi?article=1946&context=isr .

Official Guam Crash Site Information Web Center. (http://ns.gov.gu/guam/indexmain.html).

"Olympic Pipeline explosion," Wikipedia (https://en.wikipedia.org/wiki/Olympic_Pipeline_explosion).

Perlman, D.T., 1998. Who pays the price of computer software failure? Rutgers Comput. Technol. Law J. (Summer) 383–415.

Perrow, C., 2008. Software Failures, Security, and Cyberattacks May Manuscript draft.

Peterson, I., February 16 1991. Finding Fault: The Formidable Task Of Eradicating Software Bugs. The Free Library http://www.thefreelibrary.com/Finding fault: the formidable task of eradicating software bugs.-a010381574.

Phillips, D.E., November 1994. When Software Fails: Emerging Standards of Vendor Liability Under the Uniform Commercial Code.

Pinkney, K.R., 2002. Putting blame where blame is due: software manufacturer and customer liability for security-related software failure. Albany Law J. Sci. Technol.
Price, D., April 1995. Pentium FDIV flaw-lessons learned. IEEE Micro 15 (2), 86–87.
Project Management Institute, "A Guide to the Project Management Body of Knowledge," fifth Edition.
Rahman, H.A., Beznsov, K., Marti, J.R., May 2009. Identification of sources of failures and their propagation in critical infrastructures from 12 years of public failure reports. Int. J. Crit. Infrastruct. 5 (3), 220–244.
Report by the Mars Climate Orbiter Mission Failure Investigation Board (http://marsprogram.jpl.nasa.gov/msp98/news/mco991110.html).
Reuter News Service, February 19 1994. ATM Glitch Sucks Millions from N.Y. Bank Accounts. Deseret News http://www.deseretnews.com/article/337438/ATM-GLITCH-SUCKS-MILLIONS-FROM-NY-BANK-ACCOUNTS.html?pg=all.
Rosenberg, L.H., Gallo, A.M., 2002. Software quality assurance engineering at NASA. In: Proceedings of Aerospace Conference March 9-16, vol. 5, pp. 2569–2575.
Safire, W., February 2 2004. The Farewell Dossier. New York Times http://www.nytimes.com/2004/02/02/opinion/the-farewell-dossier.html.
Schmitt, E., May 20 1991. AFTER THE WAR; Army Is Blaming Patriot's Computer for Failure to Stop the Dhahran Scud. New York Times http://www.nytimes.com/1991/05/20/world/after-war-army-blaming-patriot-s-computer-for-failure-stop-dhahran-scud.html.
Schmitt, E., May 20 1991. Army is Blaming Patriot"s Computer for Failure to Stop Dhahran Scud. New York Times.
Schneier, B., April 21 2006. Software Failure Causes Airport Evacuation.
Scott, J.E., 1999. The FoxMeyer drugs' bankruptcy: was it a failure of ERP? *Americas Conference on Information Systems*, August 13-15.
Shafer, D., Laplante, P.A., 2010. In: The BP Oil Spill: Could Software be a Culprit, 12. IT Professional, pp. 6–9.
Shwartz, M., Young, D.W., Siegrist, R., 1995. The ratio of costs to charges: how good a basis for estimating costs? Inquiry 32 (4), 476–481 -1996.
Slabodkin, G., July 13 1998. Software Glitches Leave Navy Smart Ship Dead in the Water. GCN http://gcn.com/articles/1998/07/13/software-glitches-leave-navy-smart-ship-dead-in-the-water.aspx.
SOHO Mission Interruption Joint NASA/ESA Investigation Board, 1998. Final Report August http://soho.esac.esa.int/whatsnew/SOHO_final_report.html .
SOHO Mission Interruption Joint NASA/ESA Investigation Board, 1998a. Preliminary Status and Background Report July http://umbra.nascom.nasa.gov/soho/prelim_and_background_rept.html .
"Solar and Heliospheric Observatory," Wikipedia (http://en.wikipedia.org/wiki/Solar_and_Heliospheric_Observatory).
"Spanair Flight 5022″, Wikipedia. http://en.wikipedia.org/wiki/Spanair_Flight_5022.
Speed, J.R., November 1995. What do you mean I can't call myself a software engineer? IEEE Softw. 16 (6), 45–50.
Cowley, S., November 7 2003. Upgrade Glitch Downs AT&T Wireless' CRM System. InfoWorld http://www.infoworld.com/article/2675442/networking/upgrade-glitch-downs-at-t-wireless–crm-system.html.
Sterling, B., November 1 1993. The Hacker Crackdown: Law and Disorder on the Electronic Frontier. Bantam Books.
Storm, D., October 22 2014. DHS Investigates 24 Potentially Deadly Cyber Flaws In Medical Devices. Computerworld http://www.computerworld.com/article/2837413/security0/dhs-investigates-24-potentially-deadly-cyber-flaws-in-medical-devices.html.
"TAURUS (Share Settlement)," Wikipedia (http://en.wikipedia.org/wiki/TAURUS_(share_settlement)).
"The Harder They Fall," The Economist, October 30, 1997 (http://www.economist.com/node/104702/).
The Standish Group International, Inc., 1999. Chaos: A Recipe for Success.
The Standish Group International, Inc., 2010. Chaos Summary for 2010.
Thurston, R., November 24 2006. Software Failure Causes Tube Closure. ZDNet http://www.zdnet.com/software-failure-causes-tube-closure-3039284859/.
"Tokyo Stock Exchange," Wikipedia (http://en.wikipedia.org/wiki/Tokyo_Stock_Exchange).
Tomsho, R., October 20 1994. How Greyhound Lines Re-Engineered Itself Right into a Deep Hole. Wall Street Journal http://business.baylor.edu/Charles_Davis/courses/acc5317/articles/toms1094.htm.
"Typing Error Causes $225 M Loss at Tokyo Stock Exchange," Fox News, December 09, 2005 (http://www.foxnews.com/story/2005/12/09/typing-error-causes-225m-loss-at-tokyo-stock-exchange.html).
U.S. Food and Drug Administration, 2013. 2013 Recalls, Market Withdrawals & Safety Alerts http://www.fda.gov/Safety/Recalls/ArchiveRecalls/2013/ucm20035840.htm?Page=1.
USAF Accident Investigation Board, "Titan IV B-32/Centaur/Milstar Report".
Wailgum, T., March 24 2009. 10 Famous ERP Disasters, Dustups and Disappointments. CIO http://www.cio.com/article/2429865/enterprise-resource-planning/10-famous-erp-disasters–dustups-and-disappointments.html.
Wallace, D.R., Kuhn, D.R., November 1999. Lessons from 342 medical device failures. In: Proceedings of the 4th IEEE International Symposium on High-Assurance Systems Engineering. Washington, D.C., pp. 123–131.
Wastnage, J., February 14 2007. Pictures: Navigational Software Glitch Forces Lockheed Martin F-22 Raptors Back To Hawaii, Abandoning First Foreign Deployment To Japan. Flightglobal http://www.flightglobal.com/news/articles/pictures-navigational-software-glitch-forces-lockheed-martin-f-22-raptors-back-to-hawaii-212102/.
Weiss, K.A., Leveson, N., Lundqvist, K., Farid, N., Stringfellow, M., October 2001. An analysis of causation in aerospace accidents. In: Proceedings of the 20th Digital Avionics Systems Conference, vol. 1. Daytona Beach, Florida, USA pp. 4A3/1-4A3/12.
Wong, W.E., Debroy, V., Restrepo, A., January , 2010. The role of software in recent catastrophic accidents. IEEE Reliab. Soc. 2009 Annu. Technol. Rep.
Wong, W.E., Gao, R., Li, Y., Abreu, R., Wotawa, F., August 2016. A survey on software fault localization. IEEE Trans. Softw. Eng. 42 (8), 707–740.
Yates, R.E., February 18 1994. Gasps Automatic As ATM Glitch Shrinks Account Balances. Chicago Tribune http://articles.chicagotribune.com/1994-02-18/business/9402180194_1_chemical-bank-atm-president-of-electronic-banking.
Zamost, S., Phillips, K., March 2 2011. Skyrocketing Water Bills Mystify, Anger Residents. CNN http://www.cnn.com/2011/US/03/01/water.bills.war.
Zhivich, M., Cunningham, R.K., March 2009. The real cost of software errors. IEEE Secur. Privacy 7 (2), 87–90.
Zollers, F.E., McMullin, A., Hurd, S.N., Shears, P., 2004. No more soft landings for software: liability for defects in an industry that has come of age. Santa Clara High Technol. Law J.

W. Eric Wong received his M.S. and Ph.D. in Computer Science from Purdue University. He is a full professor and the founding director of the Advanced Research Center for Software Testing and Quality Assurance in Computer Science, University of Texas at Dallas (UTD). He also has an appointment as a guest researcher with National Institute of Standards and Technology (NIST), an agency of the US Department of Commerce. Prior to joining UTD, he was with Telcordia Technologies (formerly Bellcore) as a senior research scientist and the project manager in charge of Dependable Telecom Software Development. In 2014, he was named the IEEE Reliability Society Engineer of the Year. His research focuses on helping practitioners improve the quality of software while reducing the cost of production. In particular, he is working on software testing, debugging, risk analysis/metrics, safety, and reliability. He has very strong experience developing real-life industry applications of his research results. Professor Wong is the Editor-in-Chief of IEEE Transactions on Reliability. He is also the Founding Steering Committee Chair of the IEEE International Conference on Software Quality, Reliability, and Security (QRS) and the IEEE International Workshop on Program Debugging (IWPD).

Xuelin Li graduated from Beihang University with a B.E. degree in Software Reliability Engineering. He is currently on the PhD track under the supervision of Professor W. Eric Wong at the University of Texas at Dallas. Research interests include software testing, software fault localization, and software complex networks.

Dr. Philip A. Laplante is Professor of Software and Systems Engineering at The Pennsylvania State University. He received his B.S., M.Eng., and Ph.D. from Stevens Institute of Technology and an MBA from the University of Colorado. He is a Fellow of the IEEE and SPIE and has won international awards for his teaching, research and service. Since 2010 he has led the effort to develop a national licensing exam for software engineers. He has worked in avionics, CAD, and software testing systems and he has published more than 33 books and 250 scholarly papers. He is a licensed professional engineer in the Commonwealth of Pennsylvania. He is also a frequent technology advisor to senior executives, investors, entrepreneurs and attorneys and actively serves on corporate technology advisory boards. His research interests are in software testing, requirements engineering and software quality and management. Prior to his appointment at Penn State he was a software development professional, technology executive, college president and entrepreneur.