PROG 1350 – Software Engineering Fundamentals
Assignment #4 – **Disaster Report**
Due date: April 18, 2019
Student: Lev Potomkin

Table of Contents

Disaster 1: Therac-25 (1985-87)

Background

The Therac-25 was a radiation-therapy medical device, controlled by a computer. It was produced by a company called AECL in 1982. This device was using accelerated electrons to create high-energy beams that can destroy tumors with minimal impact on the surrounding healthy tissue. Relatively shallow tissue is treated with the accelerated electrons; to reach deeper tissue, the electron beam is converted into X-ray photons.

The machine had 3 modes of operation:

1. Electron-beam therapy: low-current beam of high-energy electrons was scanned over the treatment area by magnets.
2. X-ray photon therapy, which delivered a fixed width beam of X-rays produced by 100 times higher current beam of electrons. Emitted X-rays were then passed through a flattening filter and a collimator before hitting patient's tissue.
3. Field light mode: allowed the patient to be correctly positioned by illuminating the treatment area with visible light.

Actual Incident

There were at least 6 incidents during the period from 1985 to 1987. The actual incidents involve delivering high-current electron beam (the one that is 100 times stronger) produced in X-ray mode directly to patients. This could happen for two reasons. First, operator incorrectly selected X-ray mode before quickly changing to electron mode, which allowed the electron beam to be set for X-ray mode without the X-ray target being in place. Second allowed the electron beam to activate during field light mode, during which no beam scanner was active or target was in place.

The high-current electron beam struck the patients with approximately 100 times the intended dose of radiation, and over a narrower area, delivering a potentially lethal dose of beta radiation.

Root Causes

A commission concluded that the primary reason should be attributed to the bad software design and development practices, and not explicitly to several coding errors that were found. In particular, the software was designed so that it was realistically impossible to test it in a clean automated way.

Researchers who investigated the accidents found several contributing causes. These included the following :

- AECL did not have the software code independently reviewed.
- AECL did not consider the design of the software during its assessment of how the machine might produce the desired results and what failure modes existed.
- No error code documentation existed, so when machine malfunctioned, operators ignored the warning and proceeded.
- AECL had never tested the Therac-25 with the combination of software and hardware until it was assembled at the hospital.

The researchers also found several engineering issues:

- The failure occurred only when a particular nonstandard sequence of keystrokes was entered on the VT-100 terminal which controlled the computer.
- The design did not have any hardware interlocks to prevent the electron-beam from operating in its high-energy mode without the target in place.
- The engineers had reused software from older models. These models had hardware interlocks that masked their software defects.
- The hardware provided no way for the software to verify that sensors were working correctly.
- The equipment control task did not properly synchronize with the operator interface task, so that race conditions occurred if the operator changed the setup too quickly.

- The software set a flag variable by incrementing it, rather than by setting it to a fixed non-zero value. Occasionally an arithmetic overflow occurred, causing the flag to return to zero and the software to bypass safety checks.

Possible Prevention

Previous models had hardware interlocks to prevent such faults, but the Therac-25 had removed them, depending instead on software checks for safety.

If the AECL had followed best practices in software design and testing – code review, user documentation, unit, integration and system testing – the incident could have been prevented.

Possible Cause – Requirements

Since it was not the first model (there were Therac-6 and Therac-20), the requirements did not change much. Although the should have changed, since the hardware interlock was removed and instead software handled all errors. It is possible that is was not specified in the requirements, therefore the old software was not changed appropriately.

Possible Cause – Technology

The technology was in place for a several decades and researched by many scientists and engineers before it was put in medical institutions. There were 2 models that operated with the same technology. Moreover, radiation therapy discovered in the beginning of 20^{th} century – so there was plenty of time for polishing the procedure.

Testing

The problem definitely should have been caught during testing. Instead, AECL did not test the software and hardware configuration before the installation on site. Also, the company did not have any automated testing strategies, which is a result of bad design.

Pre-Warning

There was no pre-warning. In fact, the operators initially did not believe that patients were hurt. This is likely due to overconfidence.

Software and Electronics Role

In this accident both the software and hardware were at fault. Software had numerous errors, bugs, poor design and documentation. Hardware was lacking error-protection mechanisms and interlocks as in older versions of the system.

References

N. G. Leveson. (2004). An investigation of the Therac-25 Accidents. Retrieved Apr 17, 2019 from https://web.archive.org/web/20041128024227/http://www.cs.umd.edu/class/spring2003/cmsc838p/Misc/therac.pdf

Wikipedia. Therac-25. Retrieved Apr 17, 2019 from https://en.wikipedia.org/wiki/Therac-25

N. G. Leveson. Medical Devices: Therac-25. Retrieved Apr 17, 2019 from http://sunnyday.mit.edu/papers/therac.pdf

Disaster 2: Mars Climate Orbiter (1999)

Background

The Mars Climate Orbiter was a 338-kilogram robotic space probe launched by NASA on December 11, 1998 to study the Martian climate, Martian atmosphere, and surface changes and to act as the communications relay in the Mars Surveyor '98 program for Mars Polar Lander.

Actual Incident

Mars Climate Orbiter began the planned orbital insertion maneuver on September 23, 1999 at 09:00:46 UTC. Mars Climate Orbiter went out of radio contact when the spacecraft passed behind Mars at 09:04:52 UTC, 49 seconds earlier than expected, and communication was never reestablished. Due to complications arising from human error, the spacecraft encountered Mars at a lower than anticipated altitude and it was either destroyed in the atmosphere or re-entered heliocentric space after leaving Mars' atmosphere.

Root Causes

The primary cause of this discrepancy was that one piece of ground software supplied by Lockheed Martin produced results in a United States customary unit, contrary to its Software Interface Specification (SIS), while a second system, supplied by NASA, expected those results to be in SI units, in accordance with the SIS. Specifically, software that calculated the total impulse produced by thruster firings produced results in pound-force seconds. The trajectory calculation software then used these results – expected to be in newton seconds – to update the predicted position of the spacecraft.

Possible Prevention

The prevention could have been done through proper testing (at integration stage),

software interface checks and validation.

Possible Cause – Requirements

The requirements (Software Interface Specification) were violated by the software supplier (Lockheed Martin) in the implementation of the module that calculated impulse produced by thruster.

Possible Cause – Technology

The Mars missions were done by NASA in the past, so the technology was in place.

Testing

Integration testing could have prevented the incident, but is was not properly done by NASA.

Pre-Warning

The discrepancy between calculated and measured position, resulting in the discrepancy between desired and actual orbit insertion altitude, had been noticed earlier by at least two navigators, whose concerns were dismissed because they "did not follow the rules about filling out [the] form to document their concerns". A meeting of trajectory software engineers, trajectory software operators (navigators), propulsion engineers, and managers was convened to consider the possibility of executing Trajectory Correction Maneuver-5, which was in the schedule. Attendees of the meeting recall an agreement to conduct TCM-5, but it was ultimately not done.

Software and Electronics Role

The faulty part of the system was its software, specifically the interface between to software systems. This caused for incorrect instructions to hardware to be issued during a mission-critical maneuver. In turn, hardware could have had error-protection

mechanisms to prevent the software failure.

References

Wikipedia. Mars Climate Orbiter. Retrieved Apr 17, 2019 from https://en.wikipedia.org/wiki/Mars_Climate_Orbiter

NASA. (1999). Mars Climate Orbiter Mishap Investigation Board Phase I Report. Retrieved Apr 17, 2019 from https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf

CNN. (1999). Metric mishap caused loss of NASA orbiter. Retrieved Apr 17, 2019 from http://www.cnn.com/TECH/space/9909/30/mars.metric.02/index.html

Disaster 3: AeroPeru 757 (1996)

Background

Aeroperú Flight 603 was a scheduled flight from Miami International Airport in Miami, Florida to Comodoro Arturo Merino Benítez International Airport in Santiago, Chile, with stopover in Peru. In was on board of Boeing-757.

Actual Incident

The aircraft took off 42 minutes after midnight (05:42 UTC) on 2 October, and straight away, the Boeing 757 airliner crew discovered that their basic flight instruments were behaving erratically and reported receiving contradictory serial emergency messages from the flight management computer, including rudder ratio, mach speed trim, overspeed, underspeed and flying too low. The crew declared an emergency and requested an immediate return to the airport.

Faced with the lack of reliable basic flight instrument readings, constant contradictory warnings from the aircraft's flight computer (some of which were valid and some of which were not) and believing that they were at a safe altitude, the crew decided to begin descent for the approach to the airport. Since the flight was at night over water, no visual references could be made to convey to the pilots their true altitude or aid their descent. As a consequence of the pilots' inability to precisely monitor the aircraft's airspeed or vertical speed, they experienced multiple stalls, resulting in rapid loss of altitude with no corresponding change on the altimeter. While the altimeter indicated an altitude of approximately 9,700 feet, the aircraft's true altitude was in fact much lower.

The air traffic controller had instructed a Boeing 707 to take off and help guide the 757 back in to land, but before the 707 could do so, the 757's left wingtip struck the water, approximately 25 minutes after the emergency declaration. By the time they realized they were too low, it was too late; the pilots struggled with the controls and managed to get airborne again for 17 seconds, but the aircraft crashed inverted into the water. All 70

passengers and crew died.

Root Causes

Later investigation into the accident revealed that adhesive tape had been accidentally left over some or all of the static ports (on the underside of the fuselage) after the aircraft was cleaned, eventually leading to the crash. Employee Eleuterio Chacaliaza had left the tape on by mistake.

The static ports are vital to the operation of virtually all of those flight instruments that provide basic aerodynamic data such as airspeed, altitude and vertical speed, not only to the pilots but also to the aircraft's computers, which provide additional functions such as warnings when flight characteristics approach dangerous levels. The blockage of all of the static ports is one of the few common-failure modes resulting in total failure of multiple basic flight instruments and as such is regarded as one of the most serious faults that can occur within the avionics systems.

Possible Prevention

It was possible to prevent the disaster either by strictly adhering to the protocols (take the adhesive tape off before flight) by employees. Or, on the software side, the assertion mechanism could have been placed to prevent the take off in the ports are blocked (interlock).

Possible Cause – Requirements

The requirements did not consider the case that caused the incident as a result of poor design of failure modes and their prevention.

Possible Cause – Technology

The technology was not rushed, as airplanes operated fine many decades before the accident.

Testing

This incident could have been prevented by properly testing the system. In this case, the exception test should have been performed to check the condition when the tape is attached or the ports are blocked.

Pre-Warning

There was no pre-warning, although the pilots acknowledged that sensors and devices are behaving weird and inaccurate about half an hour before the incident. Warning could have been made by the employee who had left the adhesive tape on, but no such warning was issued.

Software and Electronics Role

Overall software was operating according to the specification, although it lacked such vital parts as error-protection mechanisms that could have protected the system from the incident.

References

Wikipedia. Aeruperu Flight 603. Retrieved Apr 17, 2019 from https://en.wikipedia.org/wiki/Aeroper%C3%BA_Flight_603

CNN. (1996). Searchers comb Pacific for more bodies after Peruvian crash. Retrieved Apr 17, 2019 from http://www.cnn.com/WORLD/9610/02/peru.plane.crash/index.html

Aviation Safety Network. Accident of the Boeing 757-200 Aircraft. Retrieved Apr 17, 2019 from https://aviation-safety.net/database/record.php?id=19961002-0

and http://www.skybrary.aero/bookshelf/books/1719.pdf

Disaster 4: AT&T network collapse (1990)

Background

Normally, AT&T's long-distance network was a model of reliability and strength. On any given day, AT&T's long-distance service, which at the time carried over 70% of the nation's long-distance traffic, route over 115 million telephone calls. The backbone of this massive network was a system of 114 computer-operated electronic switches scattered across the United States. These switches, each capable of handling up to 700,000 calls an hour, were linked via a cascading network known as Common Channel Signalling System #7. When a telephone call was received by the network from a local exchange, a switch would scan a list of 14 different possible routes to complete the call. At the same time, it passed the telephone number to a parallel signalling network that checked the alternate routes to determine if the switch at the other end could deliver the call to it's local company. If the destination switch was busy, the original switch sent the caller a busy signal and released the line. If the switch was available, a signal-network computer made a reservation at the destination switch and ordered the destination switch to pass the call, after the switches checked to see if the connection was good. The entire process took only four to six seconds.

Actual Incident

It took only slightly longer to slow the entire network to a crawl. Working backwards through the data, a team of 100 frantically searching telephone technicians identified the problem, which began in New York City. The New York switch had performed a routine self-test that indicated it was nearing its load limits. As standard procedure, the switch performed a four-second maintenance reset and sent a message over the signalling network that it would take no more calls until further notice. After reset, the New York switch began to distribute the signals that had backed up during the time it was off-line. Across the country, another switch received a message that a call from New York was on

it's way, and began to update it's records to show the New York switch back on line. A second message from the New York switch then arrived, less than ten milliseconds after the first. Because the first message had not yet been handled, the second message should have been saved until later. A software defect then caused the second message to be written over crucial communications information. Software in the receiving switch detected the overwrite and immediately activated a backup link while it reset itself, but another pair of closely timed messages triggered the same response in the backup processor, causing it to shut down also. When the second switch recovered, it began to route it's backlogged calls, and propagated the cycle of close-timed messages and shut-downs throughout the network. The problem repeated iteratively throughout the 114 switches in the network, blocking over 50 million calls in the nine hours it took to stabilize the system.

Root Causes

The cause of the problem had come months before. In early December, technicians had upgraded the software to speed up processing of certain types of messages. Although the upgraded code had been rigorously tested, a one-line bug was inadvertently added to the recovery software of each of the 114 switches in the network. The defect was a C program that featured a break statement located within an if clause, that was nested within a switch clause.

When the destination switch received the second of the two closely timed messages while it was still busy with the first, the program should have dropped out of the if clause, processed the incoming message, and set up the pointers to the database. Instead, because of the break statement in the else clause, the program dropped out of the case statement entirely and began doing optional parameter work which overwrote the data. Error correction software detected the overwrite and shut the switch down while it could reset. Because every switch contained the same software, the resets cascaded down the network, incapacitating the system.

Possible Prevention

Although the report states that the update was tested, since the accident happened, it was not good enough. Proper testing and code review could have prevented the incident (at the level of unit tests), since the error was in a single routine. Secondly, following best practices regarding multi-threaded environment could have eliminated the error by placing synchronization locks on important variables and logic.

Possible Cause – Requirements

Requirements for the overall system were correct, as it has operated for years before the accident. Requirements for the particular update are also considered to be fine, since they only contain performance enhancements and no functionality checks.

Possible Cause – Technology

The system has been operating long before the incident happened, although the multi-threaded programming was only starting to be incorporated in large software systems and not a lot of best practices existed to properly deal with asynchronous errors.

Testing

The report states that the update was tested “rigorously”. However, the bug still made its was to production. Thus, the conclusion is - testing was not properly done.

Pre-Warning

These was no warning issued regarding the incident.

Software and Electronics Role

The core of the problem was located in the software update that was issued. Hardware operated according to the specification, in particular the error-correction mechanism that shut down the system.

References

Jizz. (2001). The AT&T Network Crash of 1990. Retrieved Apr 17, 2019 from https://everything2.com/index.pl?node_id=1204697

D. Burke. (1995). All Circuits are Busy Now. Retrieved Apr 17, 2019 from http://users.csc.calpoly.edu/~jdalbey/SWE/Papers/att_collapse