Software Engineering Disasters

Jaan Tollander de Balsch

2018-01-19

Mistakes in software engineering can have serious conseqeuences and lead to disasters. The conseqeuences of these disasters can lead to injury and loss of human lives, financial and material losses and security vunerabilities, among many others. Next chapters covers two software disasters from a list by Flynn (2011), reasons behind them and the direct and indirect costs in more detail.

Year 2000 Problem

Year 2000 problem was a bug related to the formatting and storage of calendar data. In earlier days of computers, when memory was very expensive, dates were often shortened using two characters to indicate years, for example date `1978-01-01` would been stored as string `780101` instead of `19780101` using all four characters as the year. Within an organization this could lead to significant savings in storage costs as explained by Kappelman and Scott (1996). For the time this was justifiable optimization, but as the year 2000 approached the saving in storage costs were no longer meaningful. The bugs that occured in programs using two digit year format were either overflow from year 1999 to 1900 or invalid date formatting of the year `2000` as `19100`. This would cause error in date arithmetics and break the assumption that year were ascending. Some problems were also caused by software that didn't recognize the year 2000 as a leap year, written in Wired (2000).

From software engineering perspective one of the causes was *short-sighted temporal requirements*. Many of the programmers who wrote the code for the software at the 90s were not thinking it would still be running in 21th century and even though they most likely understood that the program would not work correctly when year 2000 would arrive. Another perspective is to think the problem as *premature optimization* but due to the significant savings in data storage cost it was rather an informed choice.

Because the bug was well undestood countries and organizations could preparare for the bug in advance or fix them on failure. Although the actual damages were minor the engineering costs to prepare and fix the bugs were estimated to be around 300 billion USD. Although the cost of fixing the bugs were high it resulted in better and more modern software (Carrington 2000).

Therac-25

Therac-25 was a radiation therapy machine produced by Atomic Energy of Canada Limited (AECL). It's name also references to series of radiation therapy incidents that lead to atleast three injurys and three deaths between the years 1985 and 1987 (Rose 1994). In these incidents Therac-25 errorneously administered an overdose of radiation to the patient leading to radiation poisoning. The incidents eventually lead to investigation into Therac-25 and discovery of several flaws in the software design and developement processes that were found to be responsible for incidents.

Two problems were discovered in the software design of Therac-25. First one was caused by a race condition within the operating system when the operator switched between X-ray and electron modes quickly resulting the electron beam to be set to X-ray mode. This would cause the electron beam dose to be approximately 100 times higher than intended dose. Second problem, Yakima software bug, …

Initial investigations blamed the bugs solely on hardware and problems in software design were not considered at all. It was later indentified that the bugs were caused by the software and bad software design practices. For example Therac-25 reused old code from its predecessor Therac-20, which had relied on hardware safety features, which were removed from Therac-25. Developer should not have assumed that reusing old code was safe without the hardware safety features. Also, Therac-25 should not have relied completely on software for safety since complex software is always bound to have some bugs.

- overconfidence
- reliability modeling and risk management

(Leveson and others 1995)

The disaster lead to atleast three deaths and three injurys, and the developement of better safety standards for developing medical software .

Goals for Software engineering Course

1) Understanding software engineering principles, processes and tools and being able to apply them in practice.
2) Being able to design and build larger software that can be used and understood by other people instead of simple code snippets for my own use.
3) Beign able to function in an organization that utilizes software engineering principles.

Bibliography

Carrington, Damian. 2000. "Was Y2k Bug a Boost?" *BBC News.* https://web.archive.org/web/20040422221434/http://news.bbc.co.uk/2/hi/science/nature/590932.stm.

Flynn, Ryan. 2011. "Software Engineering Disaster Hall of Fame." http://www.parseerror.com/bugs/.

Kappelman, Leon, and Phil Scott. 1996. "Accrued Savings of Y2k - Kappelman & Scott." http://www.comlinks.com/mag/accr.htm.

Leveson, Nancy, and others. 1995. "Medical Devices: The Therac-25." *Appendix of: Safeware: System Safety and Computers.*

Rose, Wade. 1994. "Fatal Dose - Radiation Deaths Linked to Aecl Computer Errors." http://www.ccnr.org/fatal_dose.html.

Wired. 2000. "Leap Day Tuesday Last Y2k Worry | Wired." https://www.wired.com/2000/02/leap-day-tuesday-last-y2k-worry/.