Software Engineering Disasters

January 2020

Software engineering disasters happen mainly due to buggy code during development, incomplete validation or improper risk management. There have been major failures due to buggy software, raising the question whether we should trust the software or not. Following are three examples of such failures:

Examples

MARINER 1, 1962

MARIN 1 was the first Mariner mission, aimed to perform a Venus flyby. It was intentionally destroyed 293 seconds after its launch because of an unscheduled maneuver(NSSDCA 1962). The error resulted due to incorrectly coding a handwritten formulae in computer code. The computer code missed a necessary hyphen in the data-editing program which resulted in sending incorrect signals to the spacecraft. As a result, the spacecraft received erroneous steering commands which threw the spacecraft off the course. The loss is reported to be approximately $18.5 million in 1962(Wikipedia contributors 2019). This error could have been avoided by thoroughly testing the software in a simulator or formally verifying the specifications. Though, MARINER 2, which was launced shortly after MARINER 1, successfully accomplished the given task.

AT&T Network Collapse, 1990

On January 15, 1990, AT&T network remained collapsed for around 9 hours, causing a loss of more than $60 million to AT&T alone(Burke 1995). Over 50% of the calls through AT&T failed, causing a loss to airlines, hotels, taxi rentals etc. services which relied on telephone. The main cause of this major failure was a software update. The network had 114 switches at different locations in USA. Working of these switches is well explained in (Burke 1995). A software update to speed up the network became the reason of the breakdown. A bug was found in a break statement located in an if-else statement of a C language program-Burke (1995). Even though the software had been vigorously tested, this one line bug remained undetected. This lead to shutting down all the 114 switches

because all of them had the same software. Hence, the whole telephone network of AT&T was interrupted. According to Burke (Nov. 1995), this could have been avoided by using a more structured programming language with stricter compiler.

Avoding possible World War 3, 1983

This one is an interesting case. In Sept. 1983, a Soviet early warning system sent an alert at midnight, saying that America had launched five missiles on Russia. But a duty officer Lt. Col. Stanislav Petrov felt that this was a false alarm, because if America had to launch a nuclear attack on Russia, it would have sent more than just five missiles. So Petrov reported it as a false alarm to his seniors. Later it turned out that it was actually a false alarm, went off because of sun's reflection off the cloud tops. If Petrov had taken this as a true signal, then this could have been a beginning of a new war but fortunately nothing bad happened. It wasn't actually a software bug though, but a bug in the sensor system. (Washington Post 1999, p. A19)

Such software engineering disasters can be avoided by:

- detailed and precise specifications: be precise in what we want from the software, what functions we want and necessary to include etc.
- formal verification (though for highly complex systems, it might be unfeasible): formally verifying each property that we expect the software to satisfy. But if a software is very complex, than it will have many states. So formally verifying it may be computationally very expensive.
- using strict compilers (less flexible ones): as in AT&T case, a more structured programming language with a strict compiler can be helpful, at least in debugging part (which takes most of the time in software development).
- don't follow "if it ain't buggy, don't fix it". At least this was the case with Ariane 5.

Learning goals from the course

I aim to learn following things from this course:

- To get a bigger picture of software development:
  In university's courses, I have mainly wrote codes for assignments or small course projects. But real life software are much more bigger and complex than ones I have worked on.
- What to build?
  Once we have the requirements, how to decide which kind of software

would be suitable, a mobile app or a web app, or a desktop application? And what functions to keep and what not to keep. Hopefully this will become more clear with the course.

- How to start a software from scratch?
  When we have an idea of what to work on, it's often difficult for me to decide where to start. So I expect to learn the hierarchy/steps of development process for software. And basic stuff like how to build a skeleton of the software first and then adding other functionalities.

References

Burke, D. (1995), 'All circuits are busy now: The 1990 AT&T long distance network collapse', `http://users.csc.calpoly.edu/~jdalbey/SWE/Papers/att_collapse`.

NSSDCA (1962), 'Mariner 1', `https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=MARIN1`.

Washington Post (1999), 'I had a funny feeling in my gut', `https://www.washingtonpost.com/wp-srv/inatl/longterm/coldwar/shatter021099b.htm?` [By David Hoffman].

Wikipedia contributors (2019), 'Mariner 1 — Wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=Mariner_1&oldid=926740859`. [Online; accessed 27-January-2020].