Software is ubiquitous. They could be seen across all corners of the globe. The software industry forms the very foundation on which our modern civilization is built. They have become the engine driving mankind toward the future. Nevertheless, software is still manmade products, which are prone to errors and flaws. A malfunction software may be cause serious disruption to systems from small ones like household computers to large complexes such as the stock market. This essay will discuss 2 software engineering “disasters”, as well as the writer’s personal opinion on the Software Engineering course.

One of the costliest disasters in the history of software engineering is the maiden flight of Ariane 5, Flight 501, which is the European Space Agency’s newest un-manned satellite-launching rocket. The launch took place on 4 June 1996 at ELA-3 in French Guiana. The long-waited launch ended in disastrous failure as multiple errors occurred in software design. According to Gleick (1996), Ariane 5 reused software originally created for its predecessor Ariane 4, but without proper protection an integer overflow happened, which led to an exception handled inappropriately—halting the whole inertial navigation system that otherwise would have been unaffected. Just after 37 seconds after taking off, the rocket departed from its intended flight path, began to fracture due to high aerodynamic forces, and finally self-destructing by the automated system. It took the European Space Agency 10 years and US$7 billion to produce Ariane 5, and the failed launch itself reportedly resulted in a loss of more than US$370 million (Dowson, 1997).

The failure is primarily attributed to software error. Nuseibeh (1997) noticed that data conversion from a 64-bit floating point number to a 16-bit signed integer value overflowed due to the increased horizontal acceleration, resulting in a hardware exception. Range checks for this variable were eliminated for efficiency reasons, although conversions of other variables in the code were protected. The exception brought the reference platforms to a halt, culminating in the flight's annihilation.

Nevertheless, it was found that there exist other factors that contributed to the disaster. Le Lann (1997) identified several system design and management issues plagued Ariane 5. During the development of the inertial reference system software for Ariane 4 and Ariane 5, it was decided that it was not necessary to protect the inertial system computer from being rendered inoperable by an excessive value of the horizontal velocity variable, as was the case with several other alignment software variables. It was not completely understood or studied when this design choice was made as to what values this specific variable may take when the alignment software was permitted to run after lift-off.

Another software engineering disaster that costs human life was the Therac-25, computer-controlled radiation therapy machine produced by Atomic Energy of Canada Limited (AECL) in 1982. Between 1985 and 1987, it was involved in at least six mishaps in which patients received severe radiation overdoses. It occasionally provided its patients radiation dosages hundreds of times higher than usual due to concurrent programming errors (race conditions), causing in death or serious damage (Leveson & Turner, 1993). The accidents also damaged the long-established reputation of AECL as a producer of peaceful nuclear technology applications.

Instead of using actual hardware, the Therac-25 used software controls to transition between modes. Separate circuits were utilized to check radiation intensity in previous generations, as well as hardware interlocks to guarantee that spreading magnets were appropriately positioned. In principle, using software instead would minimize complexity and production costs (Lynch, 2017).

In 1983, the Therac-25 entered service. According to Leveson (1995), there were no issues for several years and thousands of patients. On June 3, 1985, a lady was being treated for breast cancer. She'd been given 200 rad of radiation absorption in the form of a 10 MeV electron beam. When the machine turned on, the patient felt a great amount of heat. She had been burnt by between 10,000 and 20,000 rad, which was unknown at the time. The woman survived, but the radiation caused her to lose her left breast and the use of her left arm. Multiple incidents followed suit, which resulted in the death of 4 other patients.

Just like the Ariane 5, the cause of Therac-25’s accidents is not simply a specific coding error, but a combination of mistakes in engineering and management. Leveson (1995) found out that AECL neglected many important assessments and involved in fraudulent practices. The bug was later fixed, but it still had claimed 4 lives before the changes were made.

Similar disasters related to software engineering could be avoided by applying correct techniques and measures. Sommerville (2015) stated that we should make effective use of existing resources, which means that, where appropriate, we should reuse software that has already been developed rather than write new software. Although software components are meant to be reused, we have to adopt a selective and adaptive approach to software repurposing, with rigorous testing in a controlled environment. It is not wise to assume that reused software is safe:

‘A naive assumption is often made that reusing software or using commercial off-the-shelf software will increase safety because the software will have been exercised extensively. Reusing software modules does not guarantee safety in the new system to which they are transferred...’ (Leveson, 2017).

For me, the Software Engineering course provide a great deal of useful knowledge, which motivates me to put in more effort for the course. My learning goals so far are:

* **Learn software engineering techniques.** Professional software development should employ defined techniques and methods, rather than simply writing lines of codes. This will make software programs more economical, easier to test and maintain.
* **Learn software engineering processes.** Professional software development should take specific steps and procedures to ensure a unified, quality end products.
* **Understand software project management.** This is one of many paths that I would love to follow in the future. Through the last few years, I have developed a set of soft skills that focus on leadership and management. By enrolling in this course, I hope I could further pursue my desire.

**Reference list**

GLEICK, J., 1996. A Bug and A Crash*. New York Times Magazine.* 01 December. Available from:

<http://www.maths.mic.ul.ie/posullivan/A%20Bug%20and%20a%20Crash%20by%20James%20Gleick.htm>

DOWSON, M., 1997. The Ariane 5 Software Failure. *Software Engineering Notes*. **22**(2), pp. 84. Available from: https://doi.org/10.1145%2F251880.251992

NUSEIBEH, B., 1997. Ariane 5: Who Dunnit?. *IEEE Software*. **14**(3), pp. 15–16. Available from: <https://doi.org/10.1109%2FMS.1997.589224>

LE LANN, G., 1997. An Analysis of the Ariane 5 Flight 501 Failure – A System Engineering Perspective. *Proceedings of the 1997 international conference on Engineering of computer-based systems (ECBS'97).* Monterey, CA, 1997.

LEVESON, N.G., TURNER, C.S., 1993. An Investigation of the Therac-25 Accidents. *IEEE Computer*. **26**(7), pp. 18–41. Available from: https://doi.org/10.1109%2FMC.1993.274940

LYNCH, J., 2017. The Worst Computer Bugs in History: Race conditions in Therac-25. *Bugsnag Blog.* 19 September. [viewed 18 September 2021]. Available from: https://www.bugsnag.com/blog/bug-day-race-condition-therac-25

LEVESON, N., 1995. *Medical Devices: The Therac-25*. Boston: Addison-Wesley.

SOMMERVILLE, I., 2015. *Software Engineering.* 10th. Harlow: Pearson Education.

LEVESON, N., 2017. The Therac-25: 30 Years Later. *Computer*. **50**(11), pp. 8–11. Available from: https://doi.org/10.1109%2FMC.2017.4041349