U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Software Engineering

ESOF – 3MIEIC05

T1 - History of Software Engineering Assignment

Duarte Nuno Esteves André Lima de Carvalho– up201503661@fe.up.pt

João Álvaro Cardoso Soares Ferreira – up201605592@fe.up.pt

João Augusto dos Santos Lima – up201605314@fe.up.pt

Índice

Conteúdo

No Silver Bullet

"No Silver Bullet" is the title of one of the most influential papers on software development in the past 20 years, written in 1996 by Frederick P. Brooks Jr. Brooks is a celebrated computer architect, computer scientist and software designer, having lead projects at IBM (about which he wrote a book, *"The Mythical Man-Month"*) and won various awards - including the *"National Medal of Technology*" in 1985 and the *"Turing Award"* in 1999. Besides *"No Silver Bullet"* and "*The Mythical Man-Month",* Brooks also authored other peer-review papers such as "*Automatic Data Processing"*, "*Computer Architecture"*, and "*The Design of Design".*

The paper discussed in this assignment, *"No Silver Bullet"*, approaches the reasons behind the difficulty of software development, the explanation of why that fact won't change in the foreseeable future and why there is no one magic solution to all of the problems in this field (the silver bullet to kill the werewolf of software development). According to Brooks, difficulties in development can be divided into two groups: **accidental difficulties,** easier issues to solve that are connected to the production of software, and **essential difficulties**, related to the architecture of a software or system. There is no easy path to solve the latter type, but a path exists.

Accidental difficulties have been overcome as time has went on, through advancements in memory and processing speed and the creation of new high level languages which simplify development.

Concerning the essential problems, **four unique aspects** of development software are highlighted:

Complexity:

Referring to the fact that scaling software for larger challenges is a rather complex task, given that it isn't enough to use the same components with a larger size, as often happens with hardware. When a certain design for software is scaled up, we get a corresponding increase in the quantity of different elements, which in most cases interact with each other in a non-linear way, meaning that the complexity ends up often increasing at a rate above linear.

This heightened complexity can even cause issues in communication among members of the programming team, leading to flaws in the product and additional costs.

Thus, to minimize complexity, its essence is abstracted, unless that complexity is a key part of the essence of the program.

Conformity:

This hardship is connected to software being made to adapt to various types of interface, system or institution.

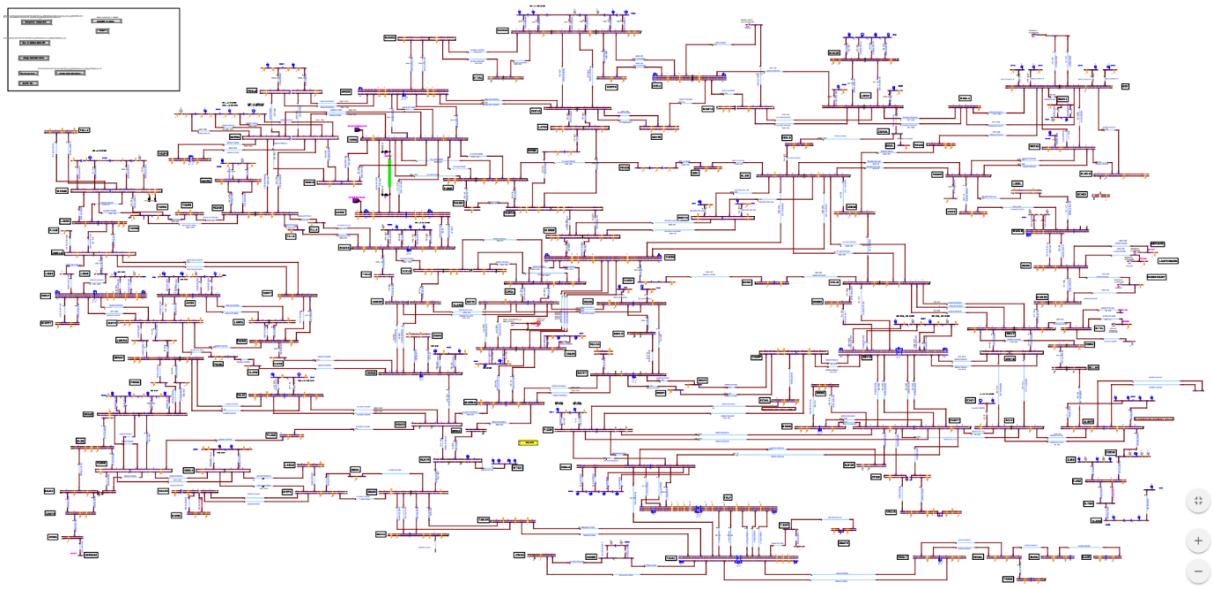
However, in many cases the complexity comes due to the conformity with interfaces other than the one originally intended, as this might not be simplified with a simple redesign of the software.

Changeability:

One of the positive characteristics of software is that it is relatively easy and fast to alter it. A negative repercussion of that fact is systematic pressures for it to be frequently updated and changed. Naturally, all popular software undergoes alterations and is adjusted to the most recent technologies – there is a cultural matrix in which a software is entangled with plenty of factors to consider, such as users, laws, applications and hardware. These factors are always changing, which forces the software to adapt.

Invisibility:

Software isn't a visible, tangible physical object – it's information, often abstract, due to which its representation to people not involved in its development can often reveal itself daunting and overly complicated. "As soon as we attempt to diagram software structure, we find it to constitute not one, but several, general directed graphs superimposed one upon another". This is a setback to both envisioning the program with one mind, but to communicate it among a group.



Solutions

For accidental difficulties:

High level languages: These have increased the production, interpretation, simplicity and compressibility of new software massively. These kinds of languages also decrease a lot of the accidental complexity that a software would end up having without it.

Unified coding environment/re-using code: This solution removes additional difficulties by supplying integrated libraries with unified file formats, making the re-use of functions in different programs something easy and effective.

For essential difficulties:

Evolution of object oriented and other high-level languages: The further abstraction of modern high level languages exponentially increases the accessibility and usability. With the hardware being less of a setback, a focus on low-level is less of a necessity and languages can afford to abstract further.

Artificial Intelligence: The development of new systems such as facial recognition, voice recognition, other emulations of human capabilities and eventually surpassing that with “automatic” programming, where the software itself defines the parameters and rules, is going to dramatically change what we understand by the limits of software.

Program verification: A big part of the efforts in modern programming is debugging and bug fixing, but easily understandable systems for verifications and automated tests can dramatically reduce the amount of time spent identifying and fixing errors.

Incremental development: According to Brooks, the hardest part of building a software is deciding exactly what to build. Due to this, one of the most effective practices to adopt is growing software incrementally and using tools that support iterative development.

Foster great designers: With the ease of programming, more of a focus should be put on the designers whose creativity will shape future software, and eventually future software practices. While most quality software in the past has been built by large teams, passionate singular designers often create the most “inspired” programs and original solutions for problems.

Promising attacks on the conceptual essence

One of the most important parts of software development is the client interaction, despite the client usually not being entirely sure of all that they intend on getting from the final product, much less able to provide the details.

As a solution, programs that facilitate quick prototypes and even graphical programming could be a way to get the client’s opinion without having to fully flesh out the software, leading to a more satisfied customer and less work-hours and modifications to a program that was once thought complete.

Conclusion

There is currently no magical silver bullet in sight that can fix the most essential problems in software development and design, but there have been plenty of advancements that aid programmers and companies in reducing production time and cost, allowing them to be more efficient and have more ease in their creation.

Case 1 – WannaCry

"WannaCry" was a ransomware attack that occurred in 2017, taking advantage of an exploit in the Server Message Block of older Microsoft Operative Systems. It was a software that would "infect" computers and prevent the users from accessing their file, while also using that same system to re-transmit itself.

The attack infected over 230,000 computers in more than 150 countries, getting the hackers £108,953 in bitcoins and total damages ranging from hundreds of millions to billions of dollars, showing just how widespread the damage of one exploit in a widely-used system can be. It also affected services such as the UK's NHS and Spain's Telefónica.

The main vulnerability that was exploited is named *"EternalBlue"* and consists of a mishandling of specially crafted packages by Window's Sever Message Block (SMBv1), opening up the possibility that an attacker could execute code on the target computer. This exploit was initially discovered by the U.S. National Security Agency (NSA) in early 2017, allowing them to release a patch to fix this exploit in March 2017. Along with the patch, they also released a bulletin detailing the issue and announcing the release of the patch for all supported versions of Windows at that time. Two months later, "*WannaCry"* took advantage of the fact that a lot of Windows users had not yet applied the patch and exploited the still open vulnerability, leading Microsoft to release an emergency patch the following day. Along with that, *"WannaCry"* was able to execute and replicate itself due to the *"DoublePulsar"* exploit, connecting to other computers on the same network or random ones online.

Microsoft claimed the NSA also had a responsibility for the attack as they didn't provide the information on the flaws as soon as they found it, not allowing them to put out a patch in time, due to the NSA's policy of "stockpiling vulnerabilities". However, as we'll see, there is also a lot that Microsoft could've done to prevent this from happening.

Figure 1 - WannaCry Ransom Note

Main problems:

Delayed patching:

The main issue with a patch is that it (generally) forces the system that it's being applied to to restart. This means that not only can everyday users put it off as an annoyance, it makes it harder for computers integrated into larger networks that require them to be on to be patched (such as is the case with the NHS).

Lacklustre general protection:

Aside from the exploit, the method WannaCry used to approach its victims was akin to a phishing scam that could spread itself, which a lot of these computers would still be exposed to. After all, the user still had to play a part in the file that infected the system, and additional protections need to be put in place for these kinds of scenarios, as the end user is ultimately the easiest point of entry for any attack or exploit.

Prevalence of legacy software:

Legacy software is still very commonly used by companies due to a variety of factors: cost of purchase, the ease of use due to it already being established, a temporary halt associated with any major change and a mentality of "if it isn't broken, don't fix it". This includes Windows XP (released over 17 years ago, in August 2001) which isn't supported by Microsoft anymore but is still in use in 42 NHS trusts and 52% of businesses. Solutions:

Make updates and patches that deal with exploits to be of utmost priority while also attempting to minimize the impact of system's downtime, installing during periods of low or near to no activity.

Use proactive defences that block any attempt at exploits, such as what was done by Norton and Symantec Endpoint Protection, using techniques such as Advanced Machine Learning, Intelligent Threat Cloud and IPS Network-based protection.

Push for companies to update software, showing them the cost of not updating can be much higher than they expect and working with corporations to make integration of new technology easier and more seamless.

Sources:

https://motherboard.vice.com/en_us/article/jpgb3y/nhs-hospitals-are-running-thousands-of-computers-on-unsupported-windows-xp

https://www.forbes.com/sites/robertbowman/2014/05/12/windows-xp-is-extinct-so-why-are-so-many-companies-still-on-it/#66d75be66977

https://www.techrepublic.com/article/report-52-of-businesses-still-running-windows-xp-despite-support-ending-in-2014/

https://www.symantec.com/blogs/threat-intelligence/wannacry-ransomware-attack

Case 2 - Therac-25(1985-1987)

Figure 2 - Therac-25 Machine

Description

Therac-25 was a computer-controlled radiation therapy machine produced by Atomic Energy of Canada Limited (AECL) in 1982, and it was an evolution of previous units (Therac-6 and Therac-20). This machine had two modes:

- Direct electron-beam therapy, in which a narrow low-current beam of high-energy electrons was scanned over the treatment area by magnets.
- Megavolt X-ray (or Photon) therapy, that after a target has been selected would deliver a fixed wide beam of X-rays, with much more energy, then passing the emitted X-rays through both a flattening filter and a collimator.

Impact

This machine is linked to at least 6 radiation overdoses, two of them resulting in patient deaths.

The Problem

As previously mentioned this machine was an evolution of two previous machines, Therac-6 and Therac-20, , and all of them chared the same source code. All three machines used a PDP-11 computer, but Therac-6 and 20 were designed to operate as standalone devices, so they didn't need that computer.

When used in manual mode, a radiotherapy technician would manually set up various parts of the machine. On the Therac-6 and 20, hardware locks prevented the operator from doing something dangerous, say selecting a high power electron beam without

the x-ray target in place. Attempting to activate the accelerator in an invalid mode would trigger a protector, bringing everything to a halt.

For Therac-25, Atomic Energy of Canada Limited switched to another design, relying in computer control only. Furthermore they decided to remove all the hardware locks, which meant that everything depended on software.

Bugs

Four bugs were detected that could hurt patients:

1. One shared variable was used both for analyzing input values and tracking turntable position. Quickly entering the data on the terminal could, therefore, result in leaving the turntable in the wrong position, thanks to a race condition.

2. It took about 8 seconds for the bending magnets to set in place. If the operator changed the beam type and power within that time and moved the cursor to the final position, the system would not detect those changes.

3. Division by the value of the variable controlling the beam power in some cases led to a zero-division error and, as a result, power increase up to the largest value possible.

4. Setting a (one-byte) Boolean variable to “true” was done through the “x=x+1” command, so pressing the “Set” button would result in the system failing to identify the message about incorrect turntable position 1 time out of 256.

Source: https://hownot2code.com/2016/10/22/killer-bug-therac-25-quick-and-dirty/

Case 3 - Heathrow Terminal 5 2008

Description

On the 27 of March of 2008, the terminal 5 of the Heathrow was going to open. The total cost of the build was 8.5 billion Euros. Unfortunately the new baggage handling system was suspended causing flight to be cancelled, luggage delayed and long queues.

The sophisticated check-in program was supposed to handle 12,000 bags an hour. This software would have been the largest baggage handling system in Europe for a single terminal. It was designed by an integrated team from BAA, Ba and Vanderlande Industries of the Netherlands. It also had 12 transfer break lines and 132 check-in desks for easy flow of large number of passengers. Luggage was automatically read, screened and sorted to their final build location.

The software costed around 285 million Euros. It connected 9000 devices , 2100 PC's and involver 180 IT suppliers and run 163 IT systems.

Problems

- During the testing of the baggage system, there were installed software filters to prevent specimen messages generated by the baggage system to be delivered to the live systems in the other terminal. But they let this testing software after the terminal opened preventing the system from receiving information about bags transferring to British Airways from other airlines. These unrecognized bags were sent to manual sorting storage facility.

- An incorrect configuration stopped the data from the baggage-handling system to the baggage reconciliation system. After a week the system failed for a whole day making bags to miss their flights because the system told the staff that they had not been security screened.

- There were errors in the transmission of British Airways flight data to the British Airports Authority making the system not recognize a proportion of the bags. Also the lack of server capacity in the terminal intensified the problem.

As there problems happen, many bags went unrecognized by the system, flight were missed and then needed to be re-booked. The software froze after not being able to handle all the re-booking from the missing flight making the staff switch of the automatic re-booking system.

At 5pm on the first day of opening, the terminal couldn't accept more baggage. Passengers how want to take the flight need to go without their bags. All bags needed to be manually sorted. About 23205 bags were sorted until 31 March were the software filter was removed.

Consequences

About 500 flight were canceled and 23000 bags were mishandled making a total loss of 18 million Euros.

British Airlines puts failure in inadequate system testing caused by delays in the construction work. Construction was scheduled to finish on 17 September but delays meant they could only start testing until 31 October. BA had to reduce the scope of system trials because staff could not enter the Terminal 5 site

Sources:

https://www.computerweekly.com/news/2240086013/British-Airways-reveals-what-went-wrong-with-Terminal-5

https://www.zdnet.com/article/it-failure-at-heathrow-t5-what-really-happened/

https://www.nytimes.com/2008/03/27/world/europe/27iht-heathrow.5.11482919.html

http://news.bbc.co.uk/2/hi/7318568.stm#graphic