Software Engineering Assignment

Ning Li Jan, 2022

Assignment-1

Q1: Many modern applications change frequently before they are presented to the end user and then after the first version has been put into use.
Suggest a few ways to build software to stop deterioration due to change.

Answer:

Many modern applications change frequently before they are presented to the end user and then after the first versions have been used. The few ways to build software to stop deterioration due to change would be:

1)Gather the required information.

2)Designer and customer define the overall objectives for the software. Identify the known requirements.

3)After building a prototype the developer uses an existing program fragment, this will help the working program to complete quickly.

4)To maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations.

5)Documents should be developed in a timely manner, to do this documentation standard are defined and mechanisms are established.

6)Review works done up to a particular stage.

7)There should be a backup person for every critical team member.

8)Check whether the risk aversion steps are being properly applied or not. Check whether the necessary information for future risk analysis is necessary to collect.

Q2: Search and read one of software failure event, such as

- Therac-25
- Ariane-5 rocket
- NASA Mars Phobos 1
- .....

Based on your investigate, describe the event and your understanding:

1) event description
2) failure reason and your own thoughts/analysis
3)software development process improvement solution

Answer:

Ariane 5

On June 4th, 1996, the very first Ariane 5 rocket ignited its engines and began speeding away from the coast of French Guiana. 37 seconds later, the rocket flipped 90 degrees in the wrong direction, and less than two seconds later, aerodynamic forces ripped the boosters apart from the main stage at a height of 4km. This caused the self-destruct mechanism to trigger, and the spacecraft was consumed in a gigantic fireball of liquid hydrogen.

The disastrous launch cost approximately $370m, led to a public inquiry, and through the destruction of the rocket's payload, delayed scientific research into workings of the Earth's magnetosphere for almost 4 years. The Ariane 5 launch is widely acknowledged as one of the most expensive software failures in history.

What went wrong?

The fault was quickly identified as a software bug in the rocket's Inertial Reference System. The rocket used this system to determine whether it was pointing up or down, which is formally known as the horizontal bias, or informally as a BH value. This value was represented by a 64-bit floating variable, which was perfectly adequate.

However, problems began to occur when the software attempted to stuff this 64-bit variable, which can represent billions of potential values, into a 16-bit integer, which can only represent 65,535 potential values. For the first few seconds of flight, the rocket's acceleration was low, so the conversion between these two values was successful. However, as the rocket's velocity increased, the 64-bit variable exceeded 65k, and became too large to fit in a 16-bit variable. It was at this point that the processor encountered an operand error, and populated the BH variable with a diagnostic value.

ANALYSIS OF THE FAILURE

In general terms, the Flight Control System of the Ariane 5 is of a standard design. The attitude of the launcher and its movements in space are measured by an Inertial Reference System (SRI). It has its own internal computer, in which angles and velocities are calculated on the basis of information from a "strap-down" inertial platform, with laser gyros and accelerometers. The data

from the SRI are transmitted through the databus to the On-Board Computer (OBC), which executes the flight program and controls the nozzles of the solid boosters and the Vulcain cryogenic engine, via servovalves and hydraulic actuators.

In order to improve reliability there is considerable redundancy at equipment level. There are two SRIs operating in parallel, with identical hardware and software. One SRI is active and one is in "hot" stand-by, and if the OBC detects that the active SRI has failed it immediately switches to the other one, provided that this unit is functioning properly. Likewise there are two OBCs, and a number of other units in the Flight Control System are also duplicated.

The design of the Ariane 5 SRI is practically the same as that of an SRI which is presently used on Ariane 4, particularly as regards the software.

Based on the extensive documentation and data on the Ariane 501 failure made available to the Board, the following chain of events, their inter-relations and causes have been established, starting with the destruction of the launcher and tracing back in time towards the primary cause.

- The launcher started to disintegrate at about H0 + 39 seconds because of high aerodynamic loads due to an angle of attack of more than 20 degrees that led to separation of the boosters from the main stage, in turn triggering the self-destruct system of the launcher.
- This angle of attack was caused by full nozzle deflections of the solid boosters and the Vulcain main engine.
- These nozzle deflections were commanded by the On-Board Computer (OBC) software on the basis of data transmitted by the active Inertial Reference System (SRI 2). Part of these data at that time did not contain proper flight data, but showed a diagnostic bit pattern of the computer of the SRI 2, which was interpreted as flight data.
- The reason why the active SRI 2 did not send correct attitude data was that the unit had declared a failure due to a software exception.
- The OBC could not switch to the back-up SRI 1 because that unit had already ceased to function during the previous data cycle (72 milliseconds period) for the same reason as SRI 2.
- The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.
- The error occurred in a part of the software that only performs alignment of the strap-down inertial platform. This software module computes meaningful results only before lift-off. As soon as the launcher lifts off, this function serves no purpose.
- The alignment function is operative for 50 seconds after starting of the Flight Mode of the SRIs which occurs at H0 - 3 seconds for Ariane 5. Consequently, when lift-off occurs, the function continues for approx. 40 seconds of flight. This time sequence is based on a requirement of Ariane 4 and is not required for Ariane 5.

- The Operand Error occurred due to an unexpected high value of an internal alignment function result called BH, Horizontal Bias, related to the horizontal velocity sensed by the platform. This value is calculated as an indicator for alignment precision over time.
- The value of BH was much higher than expected because the early part of the trajectory of Ariane 5 differs from that of Ariane 4 and results in considerably higher horizontal velocity values.

Improvement Suggestions (If you are the project manager, what would you do differently?)

The most important suggestion would be for the team to create a risk management plan. It seems like the ESA did not have a risk management plan in place even though there is a risk management in Arianespace launch agreement set in place for commercial space transactions. A risk management plan was needed to be set to address factors such as risk identification, risk assessment, risk control, and risk financing. These steps would have pointed out all the risks that could have occurred during the project development cycle. Risk management plan would have included first party risks that states that these types of risks are reciprocal waivers of liability, which limits the claims that might arise from a specific launch. These risks could include any property or personnel damage or injury and it also includes Launch Mission Failure. This means that any damage that occurs to the project team would be responsible for paying any liability or expenses related from that failure. If the team had created and followed a proper risk management plan they would have identified these risks and avoided them altogether. This would not only save them additional costs but would have made the launch successful (Hermida).

One of the suggestions I would recommend is having more project managers. There needs to be more control over the project because the testing phase was never followed through on. Additionally, there was no oversight on the requirements being followed and were missed completely. That is where project managers need to have frequent communication with the sub teams. The PM's need to create a schedule and make sure that deadlines are met and not overlooked. In this case, the engineers did not look at the designs and did not complete tasks they were assigned. If they conducted the proper testing on the software, the engineers would have caught the self-destruct activation mode. Hence, there needs to be a project manager for each team to follow through on the tasks, their deadlines and actual completion. This will ensure that all tasks were actually addressed (Fister Gale, 2013).

Q3: As software becomes more pervasive, risks to the public (due to faulty programs) become an increasingly significant concern. Develop a doomsday but realistic scenario in which the failure of a computer program could do great harm, either economic or human.

Answer:

There are literally dozens of real life circumstances to choose from. For example, software errors that have caused major telephone networks to fail, failures in avionics that have contributed to plane crashes, computer viruses (e.g., Michelangelo) that have caused significant economic losses and attacks on major e-commerce sites.