Pengfei Li (pl189)
Project, ECE 590 & CS320 Software Reliability.

Ariane 5 incident analysis

1. Introduction and background

The Ariane 5 launch vehicle is a European-developed heavy-lift launch vehicle, the latest model in the Ariane series, for launching payloads such as artificial satellites into geosynchronous transfer orbit (GTO) or low Earth orbit (LEO). After ten consecutive years of Ariane's inability to profit directly from the Ariane 4 launcher, they developed the Ariane 5 launcher and launched it for the first time on June 4, 1996. But the launch was not a success. Thirty-seven seconds after launch, the rocket deviated from its original trajectory and spun ninety degrees in the air. The rocket structure could not withstand the pressure of the sudden turn at high speed, and the rocket disintegrated in the air. It triggered the activation of the self-destruction device and exploded in the air. The first flight of this project, which took ten years and cost over 700 million dollars, ended in failure.

Immediately after the Ariane 5 explosion, the European Space Agency organized a board of inquiry to investigate the disaster, using flight data, optical observations (infrared cameras, film), examination of recovered materials, and a review of software codes, and the results were quickly available. To quote from the official report summary: " *This loss of information was due to specification and design errors in the software of the inertial reference system. The extensive reviews and tests carried out during the Ariane 5 development programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system, which could have detected the potential failure.*" [1]. In this analysis we will elaborate on this issue from both the cause and prevention of the accident.

2. Cause analysis of the incident

During the investigation of the accident, investigators found that although the change in the hydraulic pressure of the main engine nozzle actuator gradually increased at 22 seconds after launch, this was a serious anomaly. But this was not the direct cause of the ninety-degree change in the rocket's flight path and eventual self-destruction. The direct cause of the rocket's disintegration was the forced conversion of the horizontal bias value from 64 to 16 bits signed integer in a system that determines the rocket's flight attitude. This meant that a multi-billion number was crammed into a variable with a capacity of only 65535. This problem did not show up in the first twenty seconds of the rocket's takeoff, but as the rocket's acceleration gradually increased, the variable overflowed. The program, unable to make reasonable storage, threw out the exception and filled the variable with the wrong number for the horizontal bias. However, due to no exception handling function in the Ada code, the exception leads SRI component to the error state, which is only used for debugging instead of these official launch procedures [2]. This action caused the rocket's system to make an incorrect judgment, which led to the tragedy.

In addition, it was mentioned in the accident report that the program that produced the error in calculating the horizontal deviation belonged to the Ariane 4, which had a significantly different trajectory compared to the Ariane 5 and therefore needed to calculate the horizontal deviation and horizontal speed. However, since the speed of Ariane 5 is more than five times higher than that of Ariane 4, and the values calculated by the system after takeoff have no meaning to control the flight of Ariane 5, it is a

redundant part. However, due to funding issues, many of the programs used for Ariane 4 were reused in Ariane 5, and the Ariane 5 team never use the real Ariane 5 data to run and test this SRI code before the launch process. This was one of the causes of the accident.

SRI's inappropriate exception handling mechanism also contributed to this failure. the exception handling mechanism of Ariane 5 states:" *In the event of any exception, the system specification stated the failure should be indicated on the data bus, the failure context should be stored in an EEPROM memory, and finally the SRI processor should be shut down.*" [2]. The reason for this handling was that Ariane was more focused on hardware problems than on software bugs. however, as prof. K.S. Trivedi points out, the number of industrial accidents caused by software problems is increasing dramatically, from 25 percent in 1985 to 40 percent in 2005 [4][5]. Therefore, software bugs are not negligible.

In this incident, we can analyze it from the perspective of fault, error, and failure.

Fault: A 64-bit variable is forced to be converted to a 16-bit signed integer

Error: An overflow is generated during the cast, an exception is thrown, and there is no handler function to handle the exception. The state of the rocket's internal systems deviated from the designer's expectations.

Failure: The rocket made an incorrect attitude adjustment, disintegrated, and eventually self-destructed. The launch failed.

The Fault is the bug of the rocket program system, and some bugs in this system would lead to the error, which is the 64 bits number casted to the 16 bits signed number. However, this error may not eventually cause the failure, the explosion, if the system just ignores the useless number or handles this error. However, this error occurs under improper handling, making the whole system produce results that are not in line with expectations. And it eventually leads to a failure, i.e., the whole system no longer complies with the desired service as in specifications.

Because this fault is just due to the type of cast error, and we can easily reproducible it by giving it another 64 bits large number, it is easily isolated and that manifests consistently under a well-defined set of conditions. Therefore, it is the Bohrbug. This kind of bug is a very common one, accounting for 79% of the bugs generated by mission-critical software systems [3] and should have been easily avoided. However, this incident happened anyway. Some of them are caused by human factors. In the face of increasing financial pressure and the urgent need for new products to bring in revenue for the company, it was caused by a lack of review and analysis of the old code, which led to a decrease in program reliability. According to the course, the Failure Severity of this fault can be CRITICAL, because this fault affects critical functionality or critical data, which brings a lot of economic loss and time loss.

3. How to avoid in the future

a) Fault Avoidance

The root cause of this accident was the indiscriminate reuse of the Ariane 4 code. To some extent, this reflects a lack of understanding of the original code. If the developers had a deeper understanding of the requirements of Ariane 5 and had a good "documented design" and "Disciplined requirements management" habit, this accident could have been avoided.

Second, if there is a software that can automate programming (similar to Verilog auto generator and Swift/Kotlin auto generator for layout code), the tedious and error-prone process of type cast can be avoided. Leave these problems to the computer, and the engineer only needs to review and analyze the code for reasonableness and reliability.

This helps avoid mistakes that humans often make, but computers are less likely to make.
Eventually, after writing the code, we can run a simulation of the project using an automated test program to find some possible faults. as this accident shows, Ariane 5 was not simulated with real-life data before the launch, otherwise there is a high probability of finding faults caused by SRI system at five times the horizontal speed of Ariane 4.

b) Fault Tolerance
Such faults may be avoided in the future by increasing fault tolerance. One important way to improve fault tolerance is to increase the diversity of designs. Use multiple versions of the software system and execute different versions of the software system sequentially. For example, we can use Recovery Block [6] in the future Ariane 5 software system to add some redundancy to ensure that an alternate can succeed when the primary module fails.

c) Fault Removal
Testing is always a good way to find the bugs and remove them. Regression test suites can be used to test the performance of the Ariane 5 software system. It is necessary because in the different running conditions and external environments, a software system works today doesn't mean it will always work in the future. This is exemplified in the case of the Ariane 5 accident. Because the environment outside the system changed (the flight attitude and horizontal speed changed dramatically from Ariane 4 to Ariane 5), the system that worked well before created faults in the new environment. Therefore, regression testing is a very important way to find faults in the development of new products.
After the faults are found, we need to use well-designed test cases to find faults that can occur under extreme conditions, such as high-speed flight conditions approaching the approved speed.
After repeatedly testing this way, the product will have good reliability.

d) Analysis of some official advice
We can also summarize and conclude the official recommended modifications [1], so that we can learn some ideas from them. We can divide these solutions into the following categories:

- ADD MORE TESTING in the Fault Removal part:

R2: Prepare a test facility including as much real equipment as technically feasible, inject realistic input data, and perform complete, closed-loop, system testing. Complete simulations must take place before any mission. A high test coverage must be obtained.
R5: Review all flight software (including embedded software)
R10: Include trajectory data in specifications and test requirements.
R11: Review the test coverage of existing equipment and extend it where it is deemed necessary.
R13: Set up a team that will prepare the procedure for qualifying software, propose stringent rules for confirming such qualification, and ascertain that specification, verification and testing of software are of consistently high quality in the Ariane 5 program.

- DISCIPLINED REQUIREMENTS MANAGEMENT for Fault Avoidance

R1: Switch off the alignment function of the inertial reference system immediately after lift-off.
R3: Do not allow any sensor, such as the inertial reference system, to stop sending best effort data.

- For Fault Tolerance

R7: Provide more data to the telemetry upon failure of any component, so that recovering equipment will be less essential.
We can see that most of the official solutions can also be divided into the three areas: fault avoidance, fault tolerance and fault removal. It shows that these three approaches have a great role in helping us to design reliable systems.

4. Similar faults in history

Mars Climate Orbiter [7] is a U.S. instrument used to study the climate, environment, and surface state of Mars. The machine then lost contact with Earth as it was preparing to enter its expected orbit. After the investigation, NASA found that the cause of the accident was due to the use of different units when the software of the two companies was docked. This fault caused the power of the thruster ignition to deviate from the actual, resulting in the flight machine not following the projected orbit and thus losing contact with the Earth.
Similar to the fault in the analysis of this film, this fault was also of the BOH type, and NASA reported in a follow-up report that it was the result of a lack of pre-takeoff testing. Thus, it seems that testing is an essential part of improving software reliability.

5. Conclusion

In this analysis, we have analyzed and summarized the problem in terms of the background of the accident, the causes and possible solutions. In addition to this we also classify and summarize the official recommended solution measures, reflecting the need to deal with faults at different stages of the faults. These solutions and measures can be used to improve the reliability of our software systems. We also need to pay attention to the faults in our future work, so that our software can be more robust and reliable.

Reference:

[1] Prof. J. L. LIONS, Ariane5 report, 19 July 1996
[2] Bishr Tabbaa, A Modern Icarus — the crash and burn of Ariane 5 Flight 501, 03 June 2018
[3] G. Carrozza, D. Cotroneo, R. Natella, R. Pietrantuono, S. Russo, Analysis and Prediction of Mandelbugs in an Industrial Software System, ICST, 2012
[4] Jim Gray, Why do computers stop and what can be done about it? 1985
[5] Jim Gray, A census of tandem system availability between 1985 and 1990
[6] Brian Randell, et al., Reliability Issues in Computing System Design, ACM Computing Surveys, 1978
[7] From Wikipedia, Mars Climate Orbiter, https://en.wikipedia.org/wiki/Mars_Climate_Orbiter