THE EXPLOSION OF ARIANE 5

ANDRÉ SANTOS[1], MANUEL MOREIRA[2], RICARDO FERREIRA[3]

October 2018

CONTENTS

LIST OF FIGURES

ABSTRACT

On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million. A board of inquiry investigated the causes of the explosion and in two weeks issued a report. It turned out that the cause of the failure was a software error in the inertial reference system. Specifically a 64 bit floating point number relating

to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than 32,767, the largest integer storeable in a 16 bit signed integer, and thus the conversion failed.

[1] *up, Faculdade de Engenharia, Universidade do Porto*
[2] *up, Faculdade de Engenharia, Universidade do Porto*
[3] *up200305418, Faculdade de Engenharia, Universidade do Porto*

Figure 1: The Ariane 5 rocket, before ignition [1]

1 INTRODUCTION

The European Space Agency worked for 10 years to produce a rocket capable of launch, into orbit, two satellites weighing more than three tones each. These satellites would give Europe a big advantage and control over the commercial space agency.

On June 4, 1996, everything was ready at Kourou, French Guiana. The rocket, known has Ariane 5, was launched and just forty seconds after lift-off, exploded. The destroyed rocket and its cargo were valued in $500 million and the entire operation costs $7 billion dollar's.

Thirty seven seconds after ignition, Ariane 5 completely losses guidance and altitude information, due to specification and design error of the inertial reference software system. At 4km of altitude the self-destruction mechanism was triggered, because aerodynamic forces were ripping off parts of the rocket. Consequence of an abrupt course correction, that was not needed, compensating for a wrong turn that had not taken place. Steering was controlled by the on-board computer, which mistakenly thought the rocket needed a course change because of numbers coming from the inertial guidance system. The number was too big and caused an overflow.

When the guidance system shut down, it passed control to an identical, redundant unit, which was running the same software, and the same error occurred again, resulting in the explosion of Ariane 5.

Figure 2: The Ariane 5 explosion after a few seconds of ignition [1]

2 THE FAILURE

The rocket steering was controlled by a system, the Inertial Software System (SRI), that uses gyroscopes and accelerometers and sends data to the onboard computer. This software was continuously reading and calculating velocities and angles to correct the course of the rocket, sending information to the steering system.

In order to improve reliability there were a considerable redundancy at equipment level. Two SRIs operating in parallel, with identical hardware and software. If the active SRI fails, it immediately switches to the other one. The same was true for other Flight Control Units.

After thirty seven seconds of flight, the Inertial Software System sent an abrupt course correction to the steering system. Though, this correction was not needed, the Inertial Software System thought the rocket had mad a wrong turn that had not taken place.

This abrupt turn, created an unexpected pressure on some components of the engine and the rocket started to disintegrate, which triggered the self-destruct mechanism.

Everything was triggered by a conversion of a 64-bit number into a 16-bit number which caused an overflow and the shut down of the Inertial Software System. When this happened, the control was passed to the backup software, but this one also tried the same conversion, they were running the same software, and a chain of events occurred ending in the self destruction of Ariane 5.

Professor J. L. Lions, in charge of the analysis of the problem, in it's report [2] stated that the failure was:

"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."

Figure 3: The On Board computers were the faulty software was running [1]

2.1 How it happens

The On-Board Computer (OBC) commands the nozzles on base of data transmitted by the active Inertial Reference System (SRI). When the OBC asked flight information to the SRI, the data received did not contain proper information, but showed a diagnostic bit pattern of the computer running the SRI. This data was interpreted has flight data.

The OBC was not expecting bad data since a fall back system was in place, but both SRI's have failed due to a software exception.

The 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 were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.

The curious part is that this part of the software only computes meaningful results before lift-off, as soon has the launcher starts, the function serves no purpose. But, the function continues to calculate data for 40 seconds after lift-off. This time sequence was a requirement of the previous rocket, Ariane 4, and not a requirement of Ariane 5.

This happened only by a wrong design implementation and not well specified requirements.

2.2 An overflow in programming and how to prevent it

An overflow in programming [3] occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of digits, either larger than the maximum or lower than the minimum representable value.

An overflow can lead to unintended behaviour and compromise a program's reliability.

Like said on the previous section, on Ariane 5 the software was trying to convert a 64-bit floating point number to a 16-bit signed integer.

This type of conversions on Ariane 5 were not foreseen, but simple approaches could be used to avoid this software exception. Furthermore, overflow can always be avoid.

Some of the methods are:

- Allocating variables with data types large enough to contain all values that may be computed or stored in them
- Using tests or software routines to handling exceptions and do different processing to mitigate the overflow.
- Using flags in the code to propagate the error and void the calculation at the end

2.3 The Programming language used

The software was written in Ada, a programming language extended from Pascal[4].

Ada was named after Ada Lovelace[a], considered the first computer programmer and is a language well suited for embedded systems, real time applications, numerical and financial systems and object-oriented programming.

The language has built-in support for design by contract. This means that software designers can define conditions and specifications (contracts) related with the functions and it's results, giving an extra layer of reliability to the code. This does not replace unit testing, integration testing and system testing, rather complements it with internal self tests.

Features of Ada include: strong typing, modularity mechanisms (packages), run-time checking, parallel processing (tasks, synchronous message passing, protected objects, and nondeterministic select statements), exception handling, and generics.

Some say that the Ada language does not provide good exception handling and that was the origin of the problem. But that wasn't the true [5], neither the cause of the problem. Ada has everything to avoid error. The problem was on a poor requirements description and analyses and not using the full potential of the language.

The same conclusion was written on the report of the incident[2]:

"The failure of the Ariane 501 was caused by the complete loss of guidance and altitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system." "Not all the conversions were protected because... To determine the vulnerability of unprotected code,

an analysis was performed on every operation which could give rise to an ... operand error. This led to protection being added to four of seven variables... in the Ada code. However, three of the variables were left unprotected."

2.4 Lack of Software Engineering guidelines

ESA[b] did not prosecute the team of engineers that develop the SRI software, but the board that analysed the disaster found some ethical and methodologies issues.

In a project like this, engineering methods should have been used [6], but they were totally neglected. The software development should had follow the following topics:

- Design
- Documentation
- Testing
- Review

The team of engineers involved should also had a more ethical approach with the project by:

- Exposing risks openly to supervisors
- Ensuring that their products meet highest professional standards
- Exposing their knowledge to others

All the protocol was neglected which lead to erroneous software and bad engineering practices, which ended in bad software, that is:

- Hard to maintain
- Very expensive
- Without remote access
- Without error handling
- Without a proper test suit

Like other disciples, Software engineering [6], has at its disposal methods for proper creation of software to ensure software quality and flexibility. The Ariane 5 event was a warning to the engineering community and that this methodologies and techniques are not only academic. The Inquiry Board [2] mad a number of recommendations to improve the quality of software like:

"Give the justification documents the same attention as code. Improve the technique for keeping code and its justifications consistent."

"Review the test coverage of existing equipment and extend it where it is deemed necessary."

[a] https://en.wikipedia.org/wiki/Ada_Lovelace

"Wherever technically feasible, consider confining exceptions to tasks and devise backup capabilities."

"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 has to be obtained."

More recommendations can be found at the report.

3 LOSSES AND IMPACTS

The European Space Agency, ended up spending 7 billion dollar's and it's reputation.

Ariane 5 was part of the Ariane family of rockets, created to deliver satellites to the earth orbit. With this program ESA was trying to achieve commercial control on the area of satellites, but ended up loosing the rocket mechanism, revenue and market share. ESA lost the European faith on the program.

The project had some stack holders like European Space Agency, Centre Nationel de d'Etudes Spatial, governments, european taxpayers, european shareholders, and all of them lost money with this software bug, France alone lost 3 billion dollars.

After an extensive investigation, nobody was found liable and there was no one to blame, only faulty software. At the time of the event, pursuing the team of developers would have been even more disastrous and would ended up in loss of clients and loss of trust in the program.

This event attracted the attention of the public, politicians and the head of organizations to the high risk of bad practices in complex computational systems, which increased investment in research [7].

4 CONCLUSION

The defect on the Ariane 5 was the result of several factors, but the most significant was neglecting software engineering guidelines. The software didn't had proper exception handling neither a good test scenario. Furthermore, the part of the software that caused the bug was a requirement of the former Ariane 4 and of no use in Ariane 5. This shows that besides bad practices in development the team had also fail to specify the requirements for the project.

Despite all the financial losses, the Ariane 5 explosion, brought to attention of the public the high risk of bad practices in software de-

[a] *European Space Agency*

velopment, specially if we are talking about life critical systems. This increased the investment in research and now, design by contract[8] and other design techniques and guidelines are well documented and used in a large variety of projects.

This event was carefully investigated by an independent inquiry board which produced a report with valuable information about the failure, comments and suggestions, which were used in other programs of the European Space Agency.

REFERENCES

[1] How not to code blog. A space error: 370 million for an integer overflow, 2016. URL: https://hownot2code.com/2016/09/02/a-space-error-370-million-for-an-integer-overflow/.

[2] Prof. J. L. LIONS et al. ARIANE 5 flight 501 failure, 1996. URL: http://www-users.math.umn.edu/~arnold/disasters/ariane5rep.html.

[3] Wikipedia. Integer overflow, 2018. URL: https://en.wikipedia.org/wiki/Integer_overflow.

[4] Wikipedia. Ada (programming language), 2018. URL: https://en.wikipedia.org/wiki/Ada_(programming_language).

[5] Peter B. Ladkin. The ariane 5 accident: A programming problem?, 1998. URL: http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/Research/Rvs/Misc/Additional/Reports/ariane.html#ESA96a.

[6] Nato Science Committee. Software engineering techniques. pages 1–170, 1970.

[7] Jean-Marc Jézéquel and Bertrand Meyer. Design by contract: The lessons of ariane, 1997. URL: https://archive.eiffel.com/doc/manuals/technology/contract/ariane/.

[8] Wikipedia. Design by contract, 2018. URL: https://en.wikipedia.org/wiki/Design_by_contract#Relationship_to_software_testing.

[9] James Gleick. A bug and a crash, 1996. URL: https://hownot2code.com/2016/09/02/a-space-error-370-million-for-an-integer-overflow/.

[10] Abhishek Prakash. A floating point error that caused a damage worth half a billion,

2018. URL: https://hownot2code.com/2016/09/02/a-space-error-370-million-for-an-integer-overflow/.