

Ariane 5

Flight 501 Failure—A Case Study of Errors

Ken Robinson
School of Computer Science and Engineering
University of New South Wales

16th December 1996
Revised: 22rd March 2011

©Ken Robinson 1996...2011

mailto:k.robinson@unsw.edu.au

1 Launch of Ariane 5

Let us start with a *bang!*

Let's do that again *and this time watch very carefully!*

On the 4th June 1996 at 1233 GMT (UTC) the European Space Agency launched a new rocket, Ariane 5, on its maiden unmanned flight,

seen here on youtube <http://www.youtube.com/watch?v=kYUrqdUyEpI&NR=1>

Also cited at the end this presentation.

Although this was an unmanned flight and therefore there were no human casualties, there is no reason to expect that the outcome would have been any different if the flight had been manned.

In such an event all onflight crew and passengers would have been killed.

Remember as we proceed through this case that this is a project of the very experienced European Space Agency. The project cost was \$ 7 billion.

Part of the payload were four satellites, *Cluster*, that would engage in a scientific investigation. These satellites had taken many years to develop and cost around \$ 100 million.

1.1 A Report by James Gleick

In the report by James Gleick cited at the end of this article, James Gleick says:

It took the European Space Agency 10 years and \$7 billion to produce Ariane 5, a giant rocket capable of hurling a pair of three-ton satellites into orbit with each launch and intended to give Europe overwhelming supremacy in the commercial space business.

All it took to explode that rocket less than a minute into its maiden voyage last June, scattering fiery rubble across the mangrove swamps of French Guiana, was a small computer program trying to stuff a 64-bit number into a 16-bit space.

One bug, one crash. Of all the careless lines of code recorded in the annals of computer science, this one may stand as the most devastatingly efficient. From interviews with rocketry experts and an analysis prepared for the space agency, a clear path from an arithmetic error to total destruction emerges.

Is this a correct assessment?

Others have said:

if Formal Methods had been used the Ariane 5 disaster would not have happened.

As with many assertions made by various people on this accident, the above assessments are both true and false.

Although the latter as you will see is mostly false!

As you will see, this case is very complex and it is very easy to make facile analyses.

1.2 Flight 501 Failure

Pay careful attention to the sequence of events.

H0 = 093359	on 4th June 1996
H0+7.5s	Ignition of solid booster stages and normal lift-off
Up to H0+37s	Flight guidance and trajectory normal. At this moment the velocity of the launcher was Mach 0.7 (807 kph) and its altitude 3,500m. Sudden swivelling of both solid booster nozzles up to the limit, recorded by telemetry.
H0+37s to H0+39s	This caused the launcher to tilt sharply, giving rise to intense aerodynamic loads on the launcher structure resulting in breakage. Following loss of the launcher integrity, destruction of all launcher elements by the onboard neutralisation system.

Read the above very carefully.

1.3 Flight Control System of Ariane 5

- SRI: Inertial Reference System (duplicated, same hardware and software). Computes angles and velocities on the basis of information from a “strap-down” inertial platform, with laser gyros and accelerometers. The software is quite old and had been used in Ariane 4.
- OBC: On Board Computer (duplicated). Executes the flight program and controls the nozzles of the solid boosters and the Vulcain cryogenic engine, via servo-valves and hydraulic actuators.

1.4 The Chain of Events

- SRI-1 fails due to a software exception and ceases to function;
- OBC switches to SRI-2;
- SRI-2 fails due to a —the same— software exception.
- OBC cannot switch to SRI-1; reverts to taking data on bus.

- this might have been a satisfactory strategy, *but* the data on the bus is actually part of a diagnostic message written to the bus by SRI-2.
- *This data was interpreted as flight data and used to control the solid boosters.*

1.5 Analysis

At the time of the failure, the software in the two SRIs was doing a data conversion from 64-bit floating point to 16-bit integer.

The floating point number had a value greater than could be represented by a 16-bit signed integer; this resulted in an overflow software exception.

1.6 Recapitulation

Before lift off the rocket is not tied down. It's *attitude* (orientation in space) is controlled by the booster rockets.

The current attitude of the rocket is monitored by sensors and the information sent by telemetry to the SRI software and computers.

The SRI computers are duplicated with the same software.

1.7 But Wait!

Have you noticed something?

The above describes the situation *before* lift off.

Hadn't this rocket already lifted off?

Yes

So what's the point of the SRI at this stage?

Absolutely nothing!

To determine the vulnerability of unprotected code, an analysis had been performed on every operation which could give rise to an exception, including an Operand Error.

In particular, the conversion of floating point values to integers was analysed and operations involving seven variables were at risk of leading to an Operand Error.

This led to protection being added to four of the variables, evidence of which appears in the Ada code. *However, three of the variables were left unprotected. No reference to justification of this decision was found directly in the source code.*

A memorandum was found to say the code should be re-written and made more robust later.

It is important to note that the decision to protect certain variables but not others was taken jointly by project partners at several contractual levels.

1.8 Rocket Destroyed by Useless Code

The error occurred in a piece of code that was meaningless after lift-off. The code was used only to stabilise the rocket during the count-down period.

The code runs for 50 seconds; approximately 40s after lift-off. The bug occurred in Ariane 5 after 36s. The period of 50s was based on the time needed for the ground equipment to resume full control of the launcher in the event of a hold.

1.9 Differences with Ariane 4

In Ariane 4, flights using the same type of inertial reference system there had been no such failure because the trajectory during the first 40 seconds of flight was such that the particular variable related to horizontal velocity cannot reach, with an adequate operational margin, a value beyond the limit present in the software.

Ariane 5 has a higher initial acceleration and a trajectory which leads to a build-up of horizontal velocity which is five times more rapid than for Ariane 4. The higher horizontal velocity of Ariane 5 generated —within the 40-second time frame— the excessive value which caused the inertial system computers to cease operation.

1.10 Fatal Fault Recovery Strategy

It was the decision to cease the processor operation which finally proved fatal.

The reason behind this drastic action lies in the culture within the Ariane programme of only addressing random hardware failures. From this point of view, exception —or error— handling mechanisms are designed for a random hardware failure, which —quite rationally— can be handled by a backup system.

Software doesn't have random failure modes similar to hardware.

2 Investigation of the Project

2.1 Testing and Specification

This piece of software was not re-tested prior to the Ariane 5 launch.

The SRI specification (which is supposed to be a requirements document for the SRI) does not contain the trajectory data as a functional requirement.

When the project test philosophy was defined, the importance of having the SRIs in the loop was recognised. At a later stage of the programme (in 1992), this decision was changed. It was decided not to have the actual SRIs in the loop for the following reasons: (only two selected)

1. The SRIs should be considered to be *fully qualified* at equipment level.
2. The simulation of failure modes is not possible with real equipment, *only with a model*. So much for testing!

2.2 Software versus Hardware

The preceding comments display a failure of (non-software) engineers to understand the nature of software.

Software is being treated like a hardware component, but software failure modes are quite different to hardware.

The engineers that controlled the Ariane 5 project were not software engineers, but those engineers were making decisions involving software.

2.3 Official Report: Cause of Failure

From the Enquiry Board's report:

The failure of the Ariane 501 was caused by the complete loss of guidance and attitude 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.

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.

On the basis of its analyses and conclusions, the Board makes the following recommendations.

1. Switch off the alignment function of the inertial reference system immediately after lift-off. More generally, no software function should run during flight unless it is needed.
2. 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.
3. Do not allow any sensor, such as the inertial reference system, to stop sending best effort data.
4. Organise, for each item of equipment incorporating software, a specific software qualification review. The Industrial Architect shall take part in these reviews and report on complete system testing performed with the equipment. All restrictions on use of the equipment shall be made explicit for the Review Board. Make all critical software a Configuration Controlled Item (CCI).
5. Review all flight software (including embedded software), and in particular:
 - (a) Identify all implicit assumptions made by the code and its justification documents on the values of quantities provided by the equipment. Check these assumptions against the restrictions on use of the equipment.
 - (b) Verify the range of values taken by any internal or communication variables in the software.
 - (c) Solutions to potential problems in the on-board computer software, paying particular attention to on-board computer switch over, shall be proposed by the project team and reviewed by a group of external experts, who shall report to the on-board computer Qualification Board.
6. Wherever technically feasible, consider confining exceptions to tasks and devise backup capabilities.
7. Provide more data to the telemetry upon failure of any component, so that recovering equipment will be less essential.

8. Reconsider the definition of critical components, taking failures of software origin into account (particularly single point failures).
9. Include external (to the project) participants when reviewing specifications, code and justification documents. Make sure that these reviews consider the substance of arguments, rather than check that verifications have been made.
10. Include trajectory data in specifications and test requirements.
11. Review the test coverage of existing equipment and extend it where it is deemed necessary.
12. Give the justification documents the same attention as code. Improve the technique for keeping code and its justifications consistent.
13. 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 a consistently high quality in the Ariane 5 programme. Including external RAMS experts is to be considered.
14. A more transparent organisation of the cooperation among the partners in the Ariane 5 programme must be considered. Close engineering cooperation, with clear cut authority and responsibility, is needed to achieve system coherence, with simple and clear interfaces between partners.

3 Conclusions and Questions

Prima facie, this was a software failure, but was it?

How many other faults can you see, just from the preceding account?

What are the differences between hardware and software failures?

How should you deal with possible software failures?

What lessons might you learn from this accident?

Formal Methods

This case is a favourite amongst the *formal methods* community.

If formal methods has been used the Ariane 5 disaster wouldn't have happened

Not true. It is clear that the conversion procedure should have been proved formally. But this would only have proved correctness subject to assumptions (preconditions).

Without addressing those preconditions, the failure would still have occurred.

4 The INRIA Report

INRIA is probably the most prestigious scientific research centre in France and a significant world research centre. INRIA conducted its own investigation into the Ariane 5 disaster.

An independent report from the French Research Institute INRIA came to somewhat different conclusions than those reached in the official report.

The authors of the INRIA report analyse the failure from the Systems Engineering viewpoint. This is an important analysis and highlights the many factors that clearly contributed to the final failure.

Comments from Jézéquel and Meyer

The following interesting comments come from <http://www.irisa.fr/pampa/EPEE/Ariane5.html>

Is this just incompetence?

No. Everything indicates that the software process was carefully organized and planned. The ESA's software people knew what they were doing and applied widely accepted industry practices.

Is it an outrageous software management problem?

No. Obviously something went wrong in the validation and verification process (otherwise there would not have a story to write), and the Inquiry Board makes a number of recommendations to improve the process, it is clear from its report that systematic documentation, validation and management procedures were in place.

The contention often made in the software engineering literature that most software problems are primarily management problems is not borne out here. The problem is technical. (Of course one can always argue that good management will spot technical problems early enough.)

Is it the programming language's fault?

Although one may criticize the Ada exception mechanism, it could have been used here to catch the exception. In fact, quoting the report:

Not all the conversions were protected because a maximum workload target of 80% had been set for the SRI computer. 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."

In other words the potential problem of failed arithmetic conversions was recognized. Unfortunately, the fatal exception was among the three that were not monitored, not the four that were.

Is it a design error?

Why was the exception not monitored? The analysis revealed that overflow (a horizontal bias not fitting in a 16-bit integer) could not occur. Was the analysis wrong? No! It was right – for the Ariane 4 trajectory. For Ariane 5, with other trajectory parameters, it does not hold any more.

Is it an implementation error?

Although one may criticize the removal of a protection to achieve more performance (the 80% workload target), it was justified by the theoretical analysis. Engineering is making compromises. If you have proved that a condition cannot happen, you are entitled not to check for it. If every program checked for all possible and impossible events, no useful instruction would ever get executed!

Is it a testing error?

Not really. Not surprisingly, the Inquiry Board's report recommends better testing procedures, and testing the whole system rather than parts of it (in the Ariane 5 case the SRI and the flight software were tested separately). But if one can test more one cannot test all. Testing, we all know, can show the presence of errors, not their absence. And the only fully "realistic" test is to launch; this is what happened, although the launch was not really intended as a \$500-million test of the software.

So what is it?

It is a reuse error. The SRI horizontal bias module was reused from a 10-year-old software, the software from Ariane 4.

But this is not the full story: It is a reuse specification error

The truly unacceptable part is the absence of any kind of precise specification associated with a reusable module.

The requirement that the horizontal bias should fit in 16 bits was in fact stated in an obscure part of a document. But in the code itself it was nowhere to be found!

5 Final Reflections

Without at all apologising for software failures —there are far too many— it is possible to blame software far too glibly.

While the actual failure at the time was caused by software, it should be clear that the factors that led to that failure were many, and betray a serious misunderstanding —even among software engineers— of the nature of software.

Superficial analysis is analogous to blaming the manufacturer of the O-ring for the *Challenger* disaster.

But we can let them get off too easy

This story contains some absolutely appalling design errors:

1. the code duplication;
2. the handling of the error message: *what were they thinking!*
 - Remember that this is one of the top rocket organisations in the world.
 - Remember the cost of the project: \$7 billion.
 - Remember the priceless payload the rocket was carrying: 4 satellites; many years in development, around \$100 million, unable to be repeated.

We could let them get off too easily.

6 References

You Tube: Unmanned European rocket explodes on first flight [http://www.youtube.com/watch?v=kYUrqdUyEpI&NR=1.\[1ex\]](http://www.youtube.com/watch?v=kYUrqdUyEpI&NR=1.[1ex])

A Bug and a Crash, report by James Gleick. <http://www.around.com/ariane.html>

Put it in the contract: The lessons of Ariane Jean-Marc Jézéquel, IRISA & Bertrand Meyer, ISE <http://www.irisa.fr/pampa/EPEE/Ariane5.html>

Wikipedia: http://en.wikipedia.org/wiki/Ariane_5#Launch_history

ARIANE 5 Flight 501 Failure, report by the Enquiry Board.

<http://www.di.unito.it/~damiani/ariane5rep.html>

The Ariane 5 Flight 501 Failure—A Case Study in Systems Engineering for Computer Systems (INRIA)

<http://www.cse.unsw.edu.au/se4921/PDF/the-ariane-flight-failure.pdf>