The Bug That Destroyed a Rocket

Mordechai Ben-Ari
Department of Science Teaching
Weizmann Institute of Science
Rehovot 76100 Israel
<moti.ben-ari@weizmann.ac.il>

Author's Note

In the 2000 December issue of *inroads*, Michael Williams suggested that the failure of the Ariane 5 rocket launch could be used as a case study in teaching programming concepts. Here is an article I wrote several years ago in which I present the story of the Ariane 5 in terms used to teach introductory computer science.

The morning of the 4th of June 1996 was partially cloudy at Kourou in Guyana as the European Space Agency (ESA) prepared for the first launch of the French-built Ariane 5 rocket. The rocket lifted off at 09:34. Just 37 seconds later, the rocket veered on its side and began to break up. The range safety mechanism identified the impending catastrophe and initiated explosive charges that blew up the rocket to prevent further damages and possible casualties. An investigation by the ESA determined that the accident was caused by a software 'bug . This is the story of that bug.

Why should this interest you?

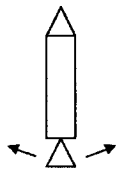
Students and other users of personal computers have become extremely tolerant of software failures. You simply utter an unprintable expletive and press `ctrl-alt-del`. The failure of the Ariane launch cost hundreds of millions of dollars and delayed the ESA space program by a year, to say nothing of the well-publicized embarrassment! What is interesting about the case of the Ariane is that ultimately the bug was not caused by a mistake alone, but by a sequences of failures in the *development process*. A more disciplined use of methods that we call software engineering should have caught the mistake; in this case, several shortcuts were taken that were not justified.

In our high-school course Foundations of Computer Science, we attempt to instill in the students the principles of a formal development process: requirements specification, design before code and thoughtful testing of the program. Many students rebel at these tasks, because they do seem onerous in the context of a program to compute the average of a sequence of numbers! As I tell the story of the Ariane bug, I will relate each step to the development process that we teach the students. Hopefully, students can learn to appreciate that we are not teaching these principles out of the wickedness of our hearts.

Sorry - a bit of physics

A short description of the physics involved will make the explanation of the bug more intelligible. Students whose health would be damaged by reading physics can skip this section.

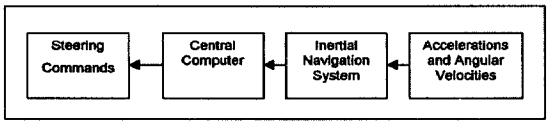
Newton s third law explains how a rocket is launched. The backwards force of the jet stream from the nozzle is balanced by a forward force on the body of the rocket. However, the rocket, which is just a long narrow tube, is very unstable. The rocket must be steered by changing the angle of the nozzle, because at low speeds the fins are not effective.



The accuracy requirements are so great that you can t control a rocket with a joystick; instead a computer must be used to make corrections dozens of times a second. The question is: How can the computer know where the rocket is so that it can tell it where to go? Does it look out the window?

The answer is that the rocket uses an inertial navigation system (INS). Behind this formidable name lies a simple idea which was used by sailors long before modern navigational equipment was developed. The idea is simply this: Measure your speed at short fixed intervals and compute the distance travelled by multiplying the speed by the duration of the interval. Suppose, for example, that you need to sail 50 km. and then turn right, and suppose that you measure your speed every five minutes for 25 minutes, resulting in readings of 12, 11, 9, 12 and 10 km/hour. Then you have gone (12+11+9+12+10)*(5/60)=54/12=4.5 km. and you still have a long way to go before turning.

A real INS is quite complex because it has to consider three dimensions of translation which are computed using linear accelerations, and three dimensions of rotation which are computed using angular velocities. Once the calculations have been performed, the position is passed to the main computer of the rocket which computes the steering commands for the nozzle that will bring the rocket back on course.



So what happened?

The sequence of events that led to the destruction of the Ariane 5 was as follows:

1. The INS attempted to convert a 64-bit number to a 16-bit number without checking that 16 bits was sufficient to hold the value. This caused a runtime error.
2. The runtime error caused the INS software to stop execution.
3. There was a backup computer executing the INS software, but (of course) it too encountered the same problem and terminated.
4. The INS hardware (of both computers) sent a report of the error to the main computer.
5. The main computer erroneously interpreted the report as extreme, but legal, data, and commanded the nozzle to fully deflect to one side.
6. The rocket turned at a sharp angle and was subjected to forces that it was not designed to withstand. It began to break up and was destroyed.

It s a sad story, isn t it?

Why did it happen?

"Hell is paved with good intentions. [Samuel Johnson] In this section, boldface indicates concepts that we teach our students.

We tell our students to carefully **document programs** so that they can be **re-used**. The designers of the Ariane 5 decided to re-use the INS of the Ariane 4. But even re-use requires careful thought and verification. From this initial decision, the drama unfolds as inevitably as in a Greek tragedy:

1. The bug was in a calibration computation that ran both before the launch and during the initial stages of the launch. However, in the Ariane 5, there was no longer any reason to run the computation during the launch. The **algorithmic problem** had changed, but the software was not modified accordingly.
2. The trajectory of the Ariane 5 differed from that of the Ariane 4 (it involved larger horizontal velocities). That is, the **input specification** in the requirements of the computation changed, but again, the computation was not modified.
3. The conversion of the 64-bit number to a 16-bit number is, in effect, a call to a function. When you call a function, you must ensure that the **precondition** holds. One way to check a precondition is with an `if`-statement. Alternatively, you can justify the precondition; for example, you can assume that the value of x^2+y^2 is non-negative and call the square root function without further ado. To **improve efficiency of the program**, several checks were omitted on physical grounds, but the justifications were not validated upon re-use.
4. Terminating the execution of a program upon detection of an error is not an acceptable form of **error handling** in a computer system. The designers of the Ariane wrongly assumed that an error would only be caused by a malfunction of the hardware. If this were true, terminating one computer so that the backup computer could take over would have been a reasonable decision. Modern programming languages such as Ada, C++, Eiffel and Java include exception handling for this purpose, but students who program in Pascal or C have no equivalent mechanism so this aspect of software is not emphasized in our course.
5. Under the assumption that the INS had been validated for the Ariane 4, no further validation was performed for the Ariane 5. In retrospect, even a single test with a **representative input** would have uncovered the problem.
6. The misunderstood message in the communications between the two computers a failure to match the **output specification** of one program to the **input specification** of another.
7. Finally, why was the bug not caught during **testing**? The reason is that you cannot debug the system by inserting breakpoints while the rocket is being launched! There are techniques for validating and testing 'disposable software that is run once and then discarded, but they are time-consuming and expensive, and were not carried out. We teach our students to **check algorithms and programs** using pencil-and-paper techniques before trying them out on the computer.

What can we learn from the Ariane 5 failure?

The work of software engineers is radically different from the type of work done by a student programmer. They spend more time specifying, designing and testing than they do 'writing code . Above all, they have to formally communicate with other software engineers and with specialists in other disciplines, such as finance, medicine and engineering. Students should be encouraged to practice software development skills as early as possible. I hope that the story of the Ariane 5 will help motivate them to do so.

Source

The ESA is to be commended for publishing the results of its investigation on the internet!
http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html

Reprint