Design by Contract: The Lessons of Ariane

by Jean-Marc Jézéquel, IRISA
and Bertrand Meyer, ISE

This article appeared in a slightly different form in Computer (IEEE), as part of the Object-Oriented department, in January of 1997 (vol. 30, no. 2, pages 129-130).

Reader reactions to the article published in IEEE's Computer magazine appear at the end of the article.

Keywords: Contracts, Ariane, Eiffel, reliable software, correctness, specification, reuse, reusability, Java, CORBA, IDL.

How not to test your software

Several earlier columns in *IEEE Computer* have emphasized the importance of Design by Contract™ for constructing reliable software. A $500-million software error provides a sobering reminder that this principle is not just a pleasant academic ideal.

On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff. Media reports indicated that the amount lost was half a billion dollars -- uninsured.

The CNES (French National Center for Space Studies) and the European Space Agency immediately appointed an international inquiry board, made of respected experts from major European countries, who produced their report in hardly more than a month. These agencies are to be commended for the speed and openness with which they handled the disaster. The committee's report is available on the Web in these two places:

- (English) http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html
- (French) http://www.cnes.fr/activites/vehicules/lanceurs/ariane5/1Qualification.htm

It is a remarkably short, simple, clear and forceful document.

Its conclusion: the explosion was the result of a software error -- possibly the costliest in history (at least in dollar terms, since earlier cases have caused loss of life).

Particularly vexing is the realization that the error came from a piece of the software that was *not* needed during the crash. It has to do with the Inertial Reference System, for which we will keep the acronym SRI used in the report, if only to avoid the unpleasant connotation that the reverse acronym could evoke for US readers. Before lift-off certain computations are performed to align the SRI. Normally they should be stopped at -9 seconds, but in the unlikely event of a hold in the countdown resetting the SRI could, at least in earlier versions of Ariane, take several hours; so the computation continues for 50 seconds after the start of flight mode -- well into the flight period. After takeoff, of course, this computation is useless; but in the Ariane 5 flight it caused an exception, which was not caught and -- boom.

The exception was due to a floating-point error: a conversion from a 64-bit integer to a 16-bit signed integer, which should only have been applied to a number less than 2^15, was erroneously applied to a greater number, representing the "horizontal bias" of the flight. There was no explicit exception handler to catch the exception, so it followed the usual fate of uncaught exceptions and crashed the entire software, hence the on-board computers, hence the mission.

This is the kind of trivial error that we are all familiar with (raise your hand if you have never done anything of the sort), although fortunately the consequences are usually less expensive. How in the world can it have remained undetected, and produced such a horrendous outcome?

Is this 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 be no 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. To engineer is to make 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 on 16 bits was in fact stated in an obscure part of a document. But in the code itself it was nowhere to be found!

From the principle of Design by Contract expounded by earlier columns, we know that *any* software element that has such a fundamental constraint should state it explicitly, as part of a mechanism present in the programming language, as in the Eiffel construct

```
convert (horizontal_bias: INTEGER): INTEGER is
        require
                horizontal_bias <= Maximum_bias
        do
                ...
        ensure
                ...
        end
```

where the precondition states clearly and precisely what the input must satisfy to be acceptable.

Does this mean that the crash would automatically have been avoided had the mission used a language and method supporting built-in assertions and Design by Contract? Although it is always risky to draw such after-the-fact conclusions, the answer is probably yes:

- Assertions (preconditions and postconditions in particular) can be automatically turned on during testing, through a simple compiler option. The error might have been caught then.
- Assertions can remain turned on during execution, triggering an exception if violated. Given the performance constraints on such a mission, however, this would probably not have been the case.
- But most importantly the assertions are a prime component of the software and its documentation ("short form", produced automatically by tools). In an environment such as that of Ariane where there is so much emphasis on quality control and thorough validation of everything, they would be the QA team's primary focus of attention. Any team worth its salt would have checked systematically that every call satisfies the precondition. That would have immediately revealed that the Ariane 5 calling software did not meet the expectation of the Ariane 4 routines that it called.

The lesson for every software developer

The Inquiry Board makes a number of recommendations with respect to improving the software process of the European Space Agency. Many are justified; some may be overkill; some may be very expensive to put in place. There is a more simple lesson to be learned from this unfortunate event:

Reuse without a contract is sheer folly!

From CORBA to C++ to Visual Basic to ActiveX to Java, the hype is on software components. The Ariane 5 blunder shows clearly that naïve hopes are doomed to produce results *far worse* than a traditional, reuse-less software process. To attempt to reuse software without Eiffel-like assertions is to invite failures of potentially disastrous consequences. The next time around, will it only be an empty payload, however expensive, or will it be human lives?

It is regrettable that this lesson has not been heeded by such recent designs as Java (which added insult to injury by *removing* the modest **assert** instruction of C!), IDL (the Interface Definition Language of CORBA, which is intended to foster large-scale reuse across networks, but fails to provide any semantic specification mechanism), Ada 95 and ActiveX.

For reuse to be effective, Design by Contract is a requirement. Without a precise specification attached to each reusable component -- precondition, postcondition, invariant -- no one can trust a supposedly reusable component.

Reader reactions

The February 1997 issue of *IEEE Computer* contained two letters from readers commenting on the article. Here are some extracts from these letters and from the response by one of the authors:

Tom Demarco, The Atlantic Systems Guild (co-author of *PeopleWare*):

Jean-Marc Jézéquel and Bertrand Meyer are precisely on-target in their assessment of the Ariane-5 failure. This was the kind of problem that a reasonable contracting mechanism almost certainly would have caught; the kind of problem that almost no other defense would have been likely to catch.

I believe that the use of Eiffel-like module contracts is the most important non-practice in software today.

Roy D. North, Falls Church, Va.:

Our designs must incorporate safety factors, and we must freeze the design before we produce the product (the software).

Bertrand Meyer's response:

What software managers must understand is that Design by Contract is not a pie-in-the-sky approach for special, expensive projects. It is a pragmatic set of techniques available from several commercial and public-domain Eiffel sources and applicable to any project, large or small.

It is not an exaggeration to say that applying Eiffel's assertion-based O-O development will completely change your view of software construction ... It puts the whole issue of errors, the unsung part of the software developer's saga, in a completely different light.

To learn more

An extensive discussion of Design by Contract and its consequences on the software development process is in the following book:

Object-Oriented Software Construction, 2nd edition

Learn more about the using Eiffel to develop mission-critical systems, read this book:

Object-Oriented Software Engineering with Eiffel

Also, ISE's Web pages contain an introduction to the concepts of Design by Contract.

For a contrarian perspective

For a different view of the issue (written in response to the *IEEE Computer* article) see Ken Garlington's paper. Although we disagree with Mr. Garlington's analysis, as expressed in Usenet discussions, we feel it is part of this site's duty to its readers to give them access to contrarian views, letting them make them make up their own minds, for the greater benefit of software quality.
