Fault tolerance via diversity against design faults: design principles and reliability assessment

Bev Littlewood **Lorenzo Strigini**
Centre for Software Reliability
City University
Northampton Square,
London EC1V OHB, U.K.
+44 20 7477 8420 +44 20 7477 8245
b.littlewood@csr.city.ac.uk l.strigini@csr.city.ac.uk

Design faults account for a large part of failures in mature software-based products. It is accepted that software will contain faults, and the unreliability due to these faults cannot be bounded with sufficient confidence for many applications.

With mature, high-quality development processes, investing additional resources into avoiding faults yields unknown and presumably diminishing results. Fault tolerance offers an alternative: employing some resources in making a system more robust against residual faults, instead of trying to eliminate them. Fault tolerance adds code intended to detect and/or correct errors produced by other code, at execution time. It depends on "diversity": the added code should not succumb to the same errors, during the same executions, as the code that is to be protected.

Some degree of fault tolerance - defensive programming - is commonly applied and accepted practice. Some industries with safety concerns (civil aviation, railways, nuclear) have applied diversity in a more extensive form: e.g., "multiple-version programming" and forms of "functional diversity". These uses, however, have attracted much controversy. The important questions include: is diversity cost-effective for increasing reliability or safety, when compared with other means for improvement with comparable costs? Can diversity produce higher reliability or safety than other methods would? Does it make it easier to certify the achieved reliability or safety before a system is delivered for operation? How should a diverse development be managed to be effective in improving reliability or safety?

Research results indicate that (as usual in software engineering) these question can only be answered with reference to each specific application context and that diversity is no “silver bullet”. But diversity is an attractive option, made more interesting by current trends like the preference for COTS items, and it is important for practitioners to go beyond the summary opinions and misunderstandings that surround it.

This tutorial is designed for people involved in system design, acceptance or certification, especially in companies with high dependability requirements or plans to improve on current levels to move into more demanding markets. It is also appropriate for researchers in software engineering wishing to obtain an up-to-date view of knowledge in this area.

This tutorial describes:

- the motivations behind the use of software fault tolerance, and thus the circumstances in which it should be considered as a possible choice;
- what design schemes one may adopt, and which issues a designer needs to be aware of, for effective application. We present both examples of industrial use and explanations of the important design choices and trade-offs. In this part, we cover the widely published solutions of N-version programming and recovery blocks, but also describe the various options available to a designer, and interesting specific solutions adopted in the railway and aviation industry, and scheme for applications to safety systems. We discuss the factors that may decide the scheme to be adopted and the design of adjudication between conflicting results;
- "what one should really believe" about the effectiveness of software fault tolerance in improving reliability, beyond the controversy and the misunderstandings surrounding it. We give a picture, assembled from more than 10 years of research, of what evidence has really been produced for and against software diversity. We explain the weaknesses of the extreme opinions voiced for and against software fault tolerance, and discuss the criteria that should affect practical decisions about using it, about how to improve its effectiveness by appropriate decisions in developing alternate versions of software components, and about its value for system acceptance.

To appear in Proc. ICSE 2000, the 22nd International Conference on Software Engineering