Choosing Effective Methods for Design Diversity
- How to Progress from Intuition to Science

Peter Popov\textsuperscript{1}, Lorenzo Strigini\textsuperscript{1}, and
Alexander Romanovsky\textsuperscript{2}

\textsuperscript{1}Centre for Software Reliability, City University, Northampton Square,
London EC1V 0HB, U.K.
{ptp,strigini}@csr.city.ac.uk

\textsuperscript{2}Centre for Software Reliability, University of Newcastle-upon-Tyne,
Newcastle upon Tyne NE1 7RU, U.K.
Alexander.Romanovsky@newcastle.ac.uk

Abstract. Design diversity is a popular defence against design faults in safety
critical systems. Design diversity is at times pursued by simply isolating the
development teams of the different versions, but it is presumably better to
"force" diversity, by appropriate prescriptions to the teams. There are many
ways of forcing diversity. Yet, managers who have to choose a cost-effective
combination of these have little guidance except their own intuition. We argue
the need for more scientifically based recommendations, and outline the
problems with producing them. We focus on what we think is the standard basis
for most recommendations: the belief that, in order to produce failure diversity
among versions, project decisions should aim at causing "diversity" among the
faults in the versions. We attempt to clarify what these beliefs mean, in which
cases they may be justified and how they can be checked or disproved
experimentally.

1 Introduction

This paper is a preliminary discussion of the main hitherto un-addressed questions in
achieving effective design diversity, i.e., producing diverse-redundant systems with
low probability of common-mode failures of the channels.

Developers of critical systems often employ diversity between redundant channels.
Redundancy protects the system against physical failures of the individual channels,
but leaves it vulnerable to design faults which, if repeated in them all, can cause
common-mode failures. So, in applications such as nuclear plant protection it is
customary to employ parallel, diverse channels. Each channel separately inputs and
processes plant data and can trigger a safe shut-down if it detects indications of unsafe
conditions. Two current trends are increasing the interest for design diversity:
increased reliance on off-the-shelf products, which may lack complete documentation

© Springer-Verlag Berlin Heidelberg 1999
of quality development procedures, and the practical disappearance of non-software based alternatives (e.g., non-smart sensors) for many functions.

For software, design diversity is sought by having two or more separate teams develop variants (often called versions) of a program. It is hoped that, if one version fails, the other[s], being internally different, will not fail at the same time: if they contain bugs, these will not cause failures in exactly the same circumstances in all versions. The versions must exhibit the same functional (externally visible) behaviour. The two or more versions are then run in a redundant configuration, so that failures in a subset of the versions may be masked or at least detected. More refined arrangements are possible, e.g. with some version only performing a monitoring or auditing function on others which have active control functions [1, 2]. Other benefits are also sought from implementing multiple versions, e.g., "back-to-back" testing provides a cheap, though imperfect, oracle for automated testing.

An important problem with design diversity (as with most other techniques for reducing or tolerating design faults) is that the reliability gain that it produces is difficult to evaluate. We know that one cannot assume diverse versions to fail independently, and all other techniques for assessing high levels of reliability are no less problematic for multiple-version than for ordinary software. For a summary of research results on this problem readers can refer to [3-5].

The other important question is how best to achieve effective diversity, i.e., a low probability that the versions will fail together. A project manager can indirectly control this by various decisions. To preserve diversity, the teams developing the versions are typically not allowed to exchange information about the development. Considering that people engaged in similar activities often make similar mistakes, they may also be given explicit directives for diversifying the internal structures of their products (e.g., using different algorithms). However, how do we know that these decisions will actually improve the delivered multi-version product?

The existing literature, and even standard documents, contain lists of such decisions that a design manager can apply to pursue diversity, which can be seen as "common-sense" advice (e.g., [6] gives developer-oriented advice, [7, 8] give customer-oriented requirements). For brevity, we shall call them "DSDs", for "diversity-seeking decisions". A complete list of plausible DSDs would span the whole development process, from team selection, to using different development environments, different tools and languages at every level of specification, design and coding, implementing each function with different algorithms, applying different V&V methods, etc. Some DSDs (like choice of algorithms) will be specific to an individual product.

But how can a project manager choose from such a "shopping list"? One may think that the more DSDs are applied, the better. But most of them have a cost: duplication of activities, added co-ordination effort, need for staff with specific skills. How many DSDs are enough for the desired level of assurance against design faults, or what is a cost-effective set of DSDs? There is currently no scientific answer to this question. We are not even sure that the advantages from various DSDs add up. We could think, for instance, that a DSD (say, specifying diverse algorithms for the various versions) produces benefits by giving a team a "scrambled" version of the problem seen by another team, so that they are not likely to make the same mistakes. However, perhaps