Cost of Software Design Diversity-A Case Study

Karama Kanoun

10-th IEEE International Conference on Software Reliability Engineering (ISSRE-99), Boca Raton, FL, USA, Nov. 1999

Cost of Software Design Diversity – A Case Study

Karama Kanoun

LAAS-CNRS — 7 Avenue du Colonel Roche
31077 Toulouse Cedex 4 — France
e-mail: kanoun@laas.fr

Abstract

This paper analyzes data related to working hours that have been recorded over seven years, during the development of a real-life software system. The software under consideration is composed of two diverse (dissimilar) units, called variants. The second variant is used for self-checking. The results of the two variants are compared: in case of agreement, the outputs of the principal variant are transmitted and in case of discrepancy an error is reported. For each development phase, we evaluate the cost overhead induced by the development of the second variant with respect to the cost of the principal variant, considered as a reference. The results show that the cost overhead varies from 25 % to 134 % according to the development phase.

Keywords:

Design diversity, error detection, data collection, development effort, cost overhead

1. Introduction

Design diversity is used to check the dynamic behavior of the software during execution. It consists in producing two or more units — referred to as variants — aimed at delivering the same services through separate design and realizations. It is used either only for error detection (comparison of the variant results) or for supporting fault tolerance in the N-Version programming approach [2] and N-self-checking programming approach [9, 11].

Since a long time ago, it has been shown — both experimentally and through modeling — that design diversity is most likely to improve system reliability (see e. g, [1], [5], [7] and [12, pp. 51-85] for experimental work, and [4], [10] for modeling). Also, it has been observed that design diversity i) aids the development of software variants, thanks to back-to-back testing [8] and ii) helps in detecting certain errors that are unlikely to be detected with other methods [12, pp. 29-49]. Moreover, it has been perceived in real experiments that design diversity does not double or triplicate the cost. Despite these positive considerations, a common argument against design diversity is the supplementary effort needed to develop and validate the additional variant(s). Indeed, the usual question is: Is it better to devote the extra effort to develop an additional variant or to the verification-and-validation of one "good" variant? An interesting discussion arguing in favor of diversity can be found in [6].

In this paper, we process data related to working hours recorded during the development of a real-life software system. The software under consideration is composed of two-self-checking diverse variants. Our objective is to evaluate the cost overhead induced by the development of the second variant with respect to the cost of the principal variant, considered as a reference. The results show that this overhead varies from 25 % to 134 % according to the development phase. The global overhead for all phases, excluding the effort spent for requirement specifications and system test, is 64 %. It lies between 42 % and 71 % if we exclude only the requirement specifications phase and assume that the system test overhead is respectively 0 % and 100 %. These results confirm those published in previous work. However, their main added value is that they have been observed in a real industrial development environment. To our knowledge, it is the only evaluation performed on a real-life system.

The paper is organized as follows. Section two presents briefly the software characteristics and outlines the organization of the development process. Section three gives an overview of the available databases while Section four analyses the data sets. Section five concludes the paper.

2. The software development characteristics

The considered software is composed of two diverse variants. Their results are compared. In case of agreement, the outputs of the principal variant (denoted PAL) are transmitted. In case of a discrepancy an error is reported. The secondary variant (denoted SEL), is thus used for self-checking.

Functional specifications are decomposed into elementary functions that are specified in a non-ambiguous graphical language (in-house formal language), in the form of specification sheets. These sheets are simulated automatically to check for data exchange consistency, types of variables, etc. In addition, a small part of the functionality cannot be specified in the formal language and is provided in natural language in the form of informal specifications.

193 specification sheets have been delivered to the software development teams: 113 are common to PAL and SEL, 48 are specific to PAL and 32 specific to SEL. The development teams have thus to work on 161 specification sheets for PAL and 145 specification sheets for SEL to derive the source and then the executable codes. The PAL coding language is Assembler. To make the variants as dissimilar as possible, half of SEL is written in Assembler and the other half is in a high level language. The resulting software has about 40,000 Non-Commented Lines of Code for PAL and 30,000 Non-Commented Lines of Code for SEL.

Comparison for error detection is performed at 14 checkpoints: simultaneously in both variants in 8 checkpoints (each checkpoint is specified once and is reproduced in each variant), and separately in 6 other checkpoints in only one variant (every checkpoint is specified within the associated variant).

The development process is incremental. Three departments are involved in producing the considered software: the Specification Department, the Software Department and the System Department. The Specification Department is in charge of producing the functional specifications from the system requirement specifications. Starting from functional specifications, the Software Department is in charge of the following activities: i) specification analysis, ii) high level and detailed design, iii) coding (including unit test) and iv) integration and general tests. The System Department is in charge of system tests based on i) functional tests, ii) tests with complete simulators and iii) field test.

Figure 1 gives the organization of the development process. Functional specifications and system tests concern PAL and SEL at the same time. Within the Software Department, two distinct teams work in parallel and separately on PAL and SEL. Each team is provided with the common functional

specifications, its own specifications in the form of specification sheets, and informal specifications. The two teams are supposed not to communicate. However both participate in meetings organized by the Specification Department to respond to questions from the Software Department. Hence the same information related to specifications and to the hardware computers are provided to both.

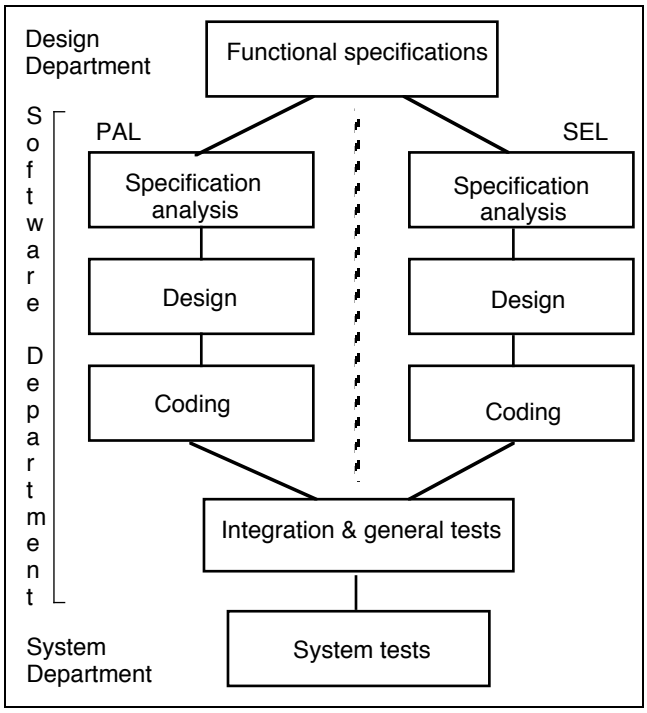

Figure 1: Organization of the development process

Within each team, any person can participate to coding and testing. The sole rule is that a person cannot test a part he or she has coded. Tests for the common specification sheets are specified and designed only once, but performed separately to PAL and SEL by each team.

Specification analysis consists in identifying those functions that are completely specified in specification sheets for which coding is automatic. The remaining ones need complete design and coding activities.

High level specifications consist in structuring the software while detailed design decomposes the functions into abstract machines.

Coding of the abstract machines is performed automatically or manually, depending on whether or not the function is entirely specified in the form of specification sheets. Unit tests are classical tests

defined from the specification sheets when the specifications are given in the form of specification sheets.

Integration, general tests and system tests take advantage of the existence of two variants (back to back testing for example). However, the test strategy does not only relay on such tests as additional particular tests are performed for each variant.

3. Information available in the databases

Information concerning the working hours within the Specification and the System departments have been recorded during seven years. They are available in the following form.

- 10,000 hours have been devoted to functional specifications within the Specification Department: 2,000 hours have been specifically dedicated to SEL, while the remaining 8,000 hours have been devoted to PAL and to the common specifications.
- The System Department spent 10,200 hours in system test. The target system is tested as a black box; the working hours related to PAL and SEL are not distinguished.

Within the Software Department, 24,375 hours have been devoted to PAL and SEL. The corresponding working hours have been recorded in two databases, Database 1 and Database 2. Database 1covers the first four years while Database 2 concerns the last three years of the study. Both databases give information about the development phases already mentioned: specification analysis, high level design, detailed design, coding (including unit tests), integration and general tests. In addition, they give the time spent in documentation. The main difference between Database 1 and Database 2 is that Database 1 discriminates between the working hours devoted to PAL and SEL while Database 2 does not make any distinction. Moreover, Database 2 introduces two new headings; maintenance and analysis. The fifth year was a transition period, all the work performed during this year has been attributed to maintenance.

Table 1 gives the percentage of working hours recorded in Database 2 with respect to working hours in Database 1 and Database 2. It shows that about two thirds of the working hours in the Software Department over the considered period have been recorded in Database 1. Figure 2 details the number of working hours per year, as derived from Database 1 and Database 2, without distinction between PAL and SEL.

Specification analysis	18.3 %
High level design	4.1 %
Detailed design	16.4 %
Coding	18.2 %
Integration	9.8 %
General tests	46.3 %
Documentation	24.8 %
Maintenance	100 %
Analysis	100 %
Overall	**34.2 %**

Table 1: Percentage of working hours recorded in Database 2 with respect to working hours in Database 1 and Database 2

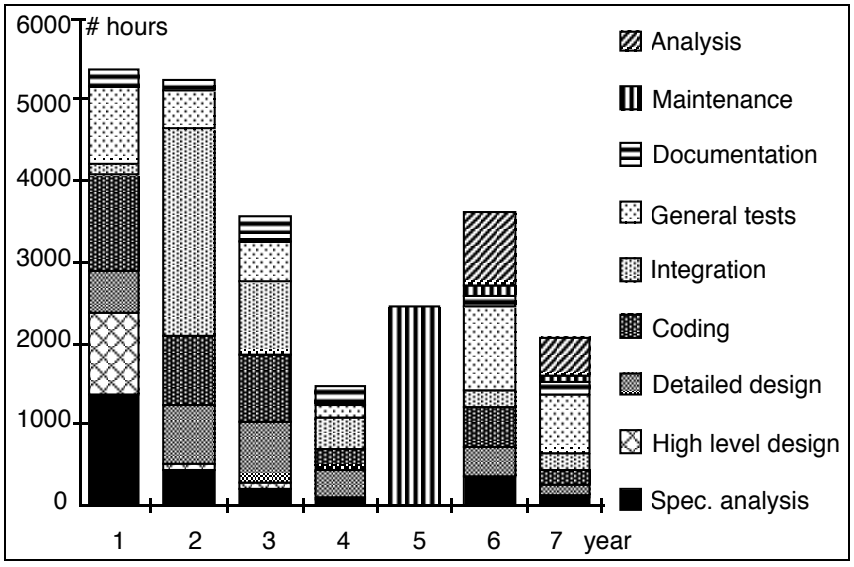

Figure 2: Working hours per year from Database 1 and Database 2

The partition between PAL and SEL, evaluated from Database 1 for the first four years, is given in Figure 3. It can be seen that SEL needed almost as much effort as PAL during the first years in the Software Department.

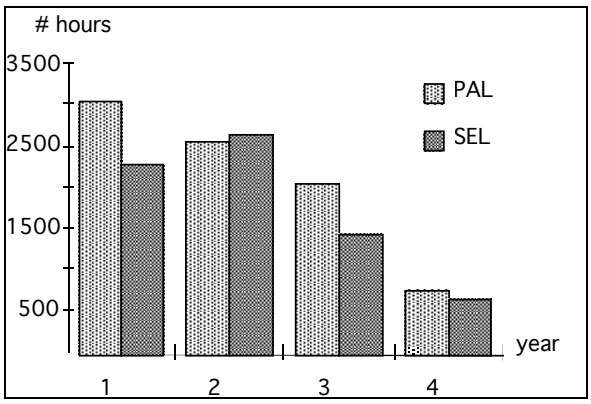

Figure 3: Partition of working hours among PAL and SEL during the first four years, from Database 1

4. Data analysis

Due to the difference in the nature of the information provided by the various departments as well as by Database 1 and Database 2 within the Software Department, we first consider the data set from Database 1 alone, then we consider the whole data set.

4.1- Analysis of the data set issued from Database 1

Figures 4 and 5 give the percentage of time dedicated to the various phases respectively for PAL and SEL. It can be seen that for SEL more than 50% of the time was allotted to coding and integration. This situation may have been induced by the fact that SEL is in two languages: coding, unit tests and integration were thus more time consuming.

Table 2 gives the percentage of time dedicated to each variant: the time devoted respectively to PAL and SEL was not uniform among the all phases. It varies from 33% to 60%. For example, only 35% of specification analysis was devoted to SEL while almost 60% of the coding effort was dedicated to SEL. Note that for integration and general tests, the effort was the same for PAL and SEL as the work was performed globally. Despite these differences, globally, for the whole period, 54.3% and 45.7% of the time where devoted respectively to PAL and SEL.

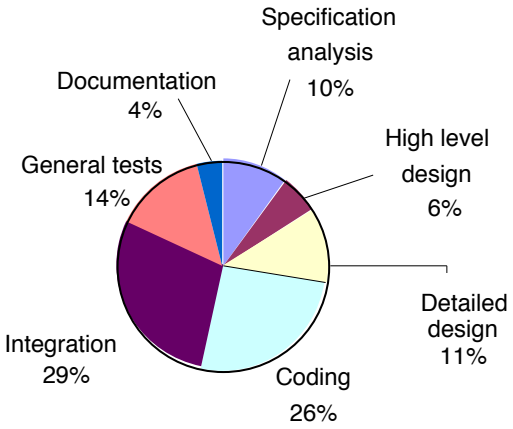

Figure 4: PAL: percentage of time devoted to the various phases

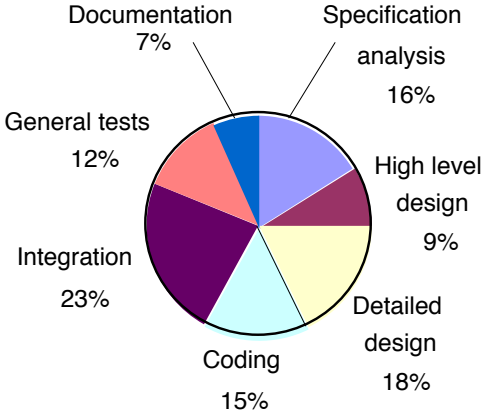

Figure 5: SEL: percentage of time devoted to the various phases

	PAL	SEC
Specification analysis	65	35
High level design	64.5	35.5
Detailed design	65.2	34.8
Coding	41.1	58.9
Integration	49	51
General tests	50.6	49.4
Documentation	67.4	32.6
Overall	54.3	45.7

Table 2: Percentage of working hours per phase during the first four years in the Specification Department

Specification analysis	54 %
High level design	55 %
Detailed design	53 %
Coding	143 %
Integration	104 %
General tests	98 %
Documentation	48 %
Overall	**84 %**

Table 3: Cost overhead per phase for the Specification Department, as evaluated from Database 1

It is worth noting that we are comparing the cost of development of a second variant with respect to the cost of development of the principal one (developed for self-checking). In particular, PAL includes 10 checkpoints added specifically for self-checking purpose. These checkpoints have certainly induced an extra cost for specification analysis, design and coding. On the other hand, they have surely helped during validation (from integration to analysis). Unfortunately, we are not able to estimate the cost induced by such additional specific features from the recorded data set. However, the influence of the corresponding time is reduced if we consider the two data sets Database 1 and Database 2. This is due to the fact that most of the work related to specification analysis, design and coding has been recorded in Database 1 while most of the work related to validation has been recorded in Database 2 (Cf. Table 1). As a result, even if the cost overheads evaluated in this paper do not correspond exactly to the exact extra cost induced by design diversity, they give a good order of magnitude of this overhead.

4.2- Analysis of the whole data set

To evaluate the cost overhead due to the additional variant for self-checking, during the development, we have to consider all development phases from functional specifications to field test. The data available from Database 2 and from the System Department should thus be partitioned according to PAL and SEL.

Concerning Database 2, the development teams felt that they spent almost the same time for PAL and SEL for all phases. Indeed, this figure is confirmed by the more accurate data collected in

Database 1: table 2 indicates that during the first four years, globally 45.7 % of the effort was dedicated to SEL. The working hours in Database 2 are thus equally partitioned among PAL and SEL.

Table 4 gives the cost overhead for the different phases within the Specification and Software departments (excluding requirement specifications and system test): it varies from 25 to 134 % according to the development phase with an overall percentatge of 64 %.

Functional specifications	25 %
Specification analysis	61 %
High level design	56 %
Detailed design	59 %
Coding	134 %
Integration	104 %
General tests	99 %
Documentation	59 %
Maintenance	100 %
Analysis	100 %
Overall	**64 %**

Table 4: Cost overhead per phase for the Specification Department, Database 1 and Database 2

Concerning the System Department, as we were not able to obtain detailed information, we made two extreme assumptions:

- an optimistic one, in which we assume that the second variant has not induced any time overhead,
- a pessimistic one, in which we assume that the time overhead for SEL test within the System Department is 100%.

The first assumption is not realistic. However, it allows evaluation of a low bound of the cost overhead. The second one is not likely either, because one should take into account the fact that the second variant has helped in several situations in detecting errors. As in the previous case, we use this assumption to evaluate an upper bound of the cost overhead.

Using these assumptions, the global overhead from functional specifications to system test, is between 42 and 71 %. This means that the cost increase factor is between 1.42 and 1.71. These results do not include the requirement specifications phase that is performed for the whole software system.

Taking into account the corresponding time will reduce the overall cost overhead. Unfortunately, the associated effort has not been recorded and all what we know is that it is a time consuming phase.

5. Conclusion

In this paper, we have presented some results related to the cost overhead induced by software design diversity for a real-life software system. We have analyzed the data sets extracted from two databases recorded within the Software Department in addition to the data provided by the Specification and the System departments.

For each development phase, we have evaluated the cost overhead induced by the development of the second variant. The cost of "the reference" variant (the principal one in our study) is evaluated in the real environment in which two-self-checking variants are developed. The results show that this overhead is: 134% for Coding and unit tests together, about 100% for integration tests, general tests, maintenance and analysis, about 60% for specification analysis, design and documentation and 25% for functional specifications. As we do not have enough detail about system tests, we made an optimistic and a pessimistic assumption to evaluate the range of the overall cost overhead: it is between 42% and 71%, excluding the effort devoted to the requirement specifications.

Even though our results correspond to the cost of the second variant with respect to the cost of one variant developed for self-checking, they compare very well to those already observed in controlled experiments, in which the cost overhead is evaluated with respect to the cost of a non-fault-tolerant variant.

For example, in [1], the cost of a variant in a recovery block programming approach has been evaluated to 0.8 times the cost of a non-fault tolerant variant (i. e., the cost increase factor is 1.6). Another example can be found in [12, pp. 127-168], where the cost of a variant in an N-Version programming approach has been evaluated to 0.75 times the cost of a non-fault tolerant variant (i. e., the cost increase factor is 2.26 for three variants). Also, similar figures have been obtained through cost models developed in [9]. Likewise, it has been observed in some experiments that the cost of software design diversity with N variants is not N-times the cost of a single software variant (see e.g., [3], [12, pp. 9-21] and [12, pp. 51-84]). However, no specific figures were provided in these experiments.

References

[1] T. Anderson, P. A. Barrett, D. N. Halliwell, and M. R. Moulding, "Software Fault Tolerance: An Evaluation", *IEEE Trans. on Software Eng.*, Vol. SE-11 (12), pp. 1502-1510, 1985.

[2] A. Avizienis, "The N-version Approach to Fault-Tolerant Systems", *IEEE Trans. on Software Eng.*, Vol. 11 (12), pp. 1491-1501, 1985.

[3] P. G. Bishop, D. G. Esp, M. Barnes, P. Humphreys, G. Dahl, and J. Lahti, "PODS — A Project on Diverse Software", *IEEE Trans. on Software Eng.*, Vol. 12(9)pp. 929-940, 1986.

[4] J. B. Dugan and M. R. Lyu, "Dependability Modeling for Fault-Tolerant Software and Systems", in *Software Fault Tolerance*, M. Lyu, Ed., Wiley & Sons, 1995, pp. 109-138.

[5] D. E. Eckhardt, A. K. Caglayan, J. C. Knight, L. D. Lee, D. F. McAllister, M. A. Vouk, and J. P. J. Kelly, "An experimental Evaluation of Software Redundancy as a Strategy for Improving Reliability", *IEEE Trans. on Software Eng.*, Vol. 17 (7)pp. 692-702, 1991.

[6] L. Halton, "N-Version Design Versus Good Version", *IEEE Software*, Vol. Nov./Dec. pp. 71-76, 1997.

[7] J. P. J. Kelly, D. E. Eckhardt, M. A. Vouk, D. F. McAllister, and A. Caglayan, "A Large Scale Second Generation Experiment in Multi-version Software: Description and Early Results", Proc. *IEEE 18th Int. Symp. on Fault-Tolerant Computing (FTCS-18)*, Tokyo, Japan, 1988, pp. 9-14.

[8] J. P. J. Kelly, T. I. McVittie, and W. I. Yamamoto, "Implementing Design Diversity to Achieve Fault Tolerance", *IEEE Software*, Vol. July, pp. 61-71, 1991.

[9] J.-C. Laprie, J. Arlat, C. Béounes, and K. Kanoun, "Definition and Analysis of Hardware-and-Software Fault-tolerant Architectures", *IEEE Computer Magazine*, Vol. 23 (7), pp. 39-51, 1990.

[10] J.-C. Laprie, J. Arlat, C. Béounes, and K. Kanoun, "Architectural Issues in Software Fault Tolerance", in *Software Fault Tolerance*, M. Lyu, Ed., J. Wiley & Sons, Ltd, 1995, pp. 47-80.

[11] P. Traverse, "Dependability of Digital Computers on Boards Airplanes", in *Dependable Computing for Critical Applications*, *Dependable Computing and Fault-Tolerant Systems, Vol. 1*, A. Avizienis and J. C. Laprie, Eds., Springer-Verlag, 1987, pp. 133-152.

[12] U. Voges, "Software Diversity in Computerized Control Systems," in *Dependable Computing and Fault-Tolerant Systems*, vol. 2, A. Avizienis, H. Kopetz, and J.-C. Laprie, Eds. Wien, New York: Springer-Verlag, 1988.