This article has been accepted for publication in a future issue of a journal/proceedings, but has not been fully edited. Content may change prior to final publication. Citation information: Belli, F., Jedrzejowicz, P., An Approach to the Reliability Optimization of Software with Redundancy, IEEE Tranactions on Software Engineering, Vol. 17, 3, pp.310-312

Concise Papers

An Approach to the Reliability Optimization of Software with Redundancy

Fevzi Belli and Piotr Jedrzejowicz

Abstract—**An approach to the optimization of software reliability is proposed. The emphasis is put on the software redundancy to achieve fault tolerance, i.e., the results of the optimization process are applied to determine the optimal structure of software to be developed. Two optimization models are formulated covering, respectively, modified recovery block scheme and multiversion programming approaches. Both cases are illustrated by simple examples.**

Index Terms—**Multiversion programming, recovery block scheme, software redundancy, software reliability optimization.**

I. Introduction

As the mathematical background of reliability theory as well as subsequent reliability models have become more and more sophisticated, the quality of reliability behavior predictions has substantially improved [6]. It has been only natural to extend the range of application from reliability prediction to reliability optimization.

Reliability optimization appeared as a branch of general reliability theory in the late 1960's, as surveyed in [11]. Since that time numerous reliability optimization models and procedures have been proposed. All these models attempt to provide an answer to the question:

> "How should the system be designed and/or operated in order to obtain the required reliability level and at the same time minimize the amount of resources needed?"

Alternatively, maximum reliability is pursued while available resources are given. It should be observed that in spite of tremendous efforts, the field of reliability optimization has not produced any universal and practicable results until now. Existing models/procedures can be applied only under special circumstances or can be considered only as very elementary simplification of reality. The main reasons behind this situation include: the complexity of the reliability optimization problems, difficulties in identification of dependencies between use of resources and component reliabilities, the numerical complexity of the emerging problems, and lack of necessary information on components' reliability properties.

Approximately 15 years after the foundation of the general reliability theory the first effort at software reliability modelling appeared [8]. As it was observed in [10] software reliability (just as in case of hardware, or human reliabilities) has two different meanings. The first is the collection of all the techniques which can be used to design and test software so that it is relatively error free. The second meaning is the probabilistic definition: software reliability is the probability that a given software system does not fail on a random input from the set of all possible inputs. The reliability degree of a system (concerning both its interpretations) can be increased considerably through fault tolerance techniques.

By now, the predictive power of the software reliability models seems to be quite satisfactory provided that testing is well controlled as a statistical experiment and test space is reasonably congruent with use space (see the reports by M. L. Shooman, G. Richeson [9], and J. P. Musa [4]). Yet this power stems from the purely statistical nature of the software failure occurrence and detection and not from an understanding of how particular components' failures contribute (or not) to the failure of the whole software system.

Manuscript received October 18, 1988; revised November 1, 1990.

F. Belli is with the Department of Electrical and Electronics Engineering, University of Paderborn, D-4790 Paderborn, Germany.

P. Jedrzejowicz is with the Merchant Marine Academy, Gdynia, Poland.

IEEE Log Number 9041666.

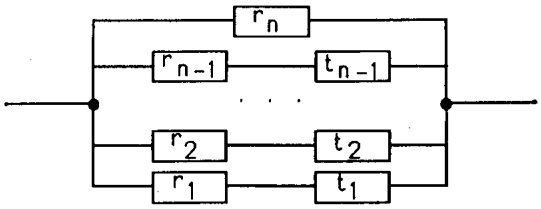

Fig. 1. Module of a software system.

Based on the recently published models (see the reviews [2], [3]) the following conclusions with respect to the present state of software reliability modelling can be formulated.

- The field is already well established and the available models enable sufficiently good predictions during the integration test phase or during field deployment. These predictions may cover complex software systems as well as parts thereof (modules, segments, programs).
- The existing models take advantage of the probability theory tools and concepts. Most of them (macro models) do not however aim at explaining the causes and effects of the software failures in terms of modern software engineering.

In this paper we propose to use a predictive power of the existing software reliability models in order to arrive at the optimal structure for software systems with redundancy. The approach involves application of the existing reliability models to obtain reliability predictions for components from which complex, fault-tolerant software structures are to be constructed, evaluation of the amount of resources needed to develop and implement each component, and calculations of the optimal software structure. It is only the last stage of this process which is discussed in our paper. Two optimization models are formulated covering, respectively, modified recovery block-scheme and multiversion programming approaches.

II. Reliability Optimization Model for the Modified Recovery Block-Scheme Approach

It is assumed that the software system is constructed with modules connected in series. Each module can be considered as an extension of the recovery block-scheme program [7]. The problem at hand is to find optimal redundancy level for each module within the software system. The concept of module is shown in Fig. 1.

Each module consists of n parallel paths; a path is understood as a sequence of instructions. Each path, excluding the nth one, consists of two parts. The first part is the sequence of instructions which performs the module functions. The second part is the testing segment. Its role is to test the results of computations of the first part and to accept (or reject) them. The testing segment is also subject to failures. Its failure means that either the performance of the first part is erroneously accepted or rejected. If the result of computations of the ith path is, however, rejected then the initial state is recovered and the $(i + 1)$st path is activated and performed. The nth path has no testing segment. Notwithstanding eventual errors, the next module is run after this path has been performed.

If a result is accepted by a testing segment, the next module is activated and run. It is assumed that all software failures within the

TABLE I
RELIABILITIES AND COSTS FOR THE EXAMPLE SYSTEM

j	r_{ji}	$t_{ji}(i \neq n_j)$	$c_{ji}(i \neq n_j)$	$c_{ji}(i = n_j)$
1	0.6	0.7	10	6
2	0.8	0.5	12	8

system are s-independent. This is justified by the hope that independently developed versions will display different failure behavior and thus will, eventually, fail on different inputs.

It is also assumed that with each part of every path in a module there is associated an amount of resources needed to design, deploy, and operate it. The total resource consumption of the system is the sum of the resource requirements of all modules.

It is furthermore assumed that the information on all module/path/segment reliabilities is available. Resource requirements for all modules/paths/segments are also known.

Based on the above the software reliability optimization model is formulated. The following notation is introduced.

m	Number of modules.
$j = 1, \cdots, m$	Index for modules in the system.
n_j	Number of paths in the jth module.
$i = 1, \cdots, n_j$	Index for paths in a module.
r_{ji}	Reliability of the ith path in the jth module (refers to the functioning segment).
t_{ji}	Reliability of the testing segment at the ith path in the jth module (probability that the error is detected if occurred).
c_{ji}	Amount of resources needed for the ith path in the jth module.
R_j	jth module reliability.
R	System reliability.
C	Amount of available resources.

The module reliability is now defined as

$$R_j = \sum_{i=1}^{n_j} P_{ji} r_{ji} \tag{1}$$

where p_{ji} denotes the probability that the ith path in the jth module will be executed and is defined recursively:

$$\begin{aligned} p_{j1} &= 1 \\ p_{j2} &= (1 - r_{j1}) t_{j1} \\ p_{j3} &= (1 - r_{j1}) t_{j1} (1 - r_{j2}) t_{j2} \\ &\cdots \\ p_{ji+1} &= (1 - r_{j1}) t_{j1} (1 - r_{ji}) t_{ji}. \end{aligned} \tag{2}$$

Hence, the system reliability is given by

$$R = \prod_{j=1}^{m} \sum_{i=1}^{n_j} p_{ji} r_{ji} \tag{3}$$

and the software optimization problem is

$$\max_{n_j} \prod_{j=1}^{m} \sum_{i=1}^{n_j} p_{ji} r_{ji} \tag{4}$$

subject to

$$\sum_{j=1}^{m} \sum_{i=1}^{n_j} c_{ji} \leq C \tag{5}$$

As an example we consider a system consisting of two modules. Component reliabilities and their costs are given in Table I. For the sake of simplicity, it has been assumed that reliabilities and costs of function segments within a given module, except for the nth one, are identical.

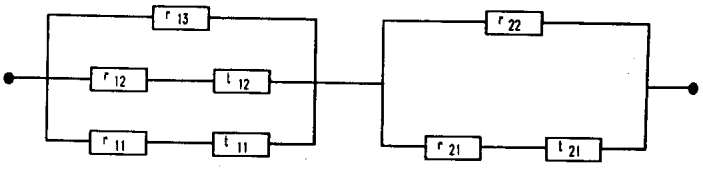

Fig. 2. Optimal structure of the example system.

TABLE II
FAILURE PROBABILITIES AND COSTS FOR THE EXAMPLE SYSTEM

i	e_{1i}	e_{2i}	C_{1i}	C_{2i}
1	0.08	0.10	10	12
2	0.05	0.15	12	7
3	0.12	0.16	8	6

It is assumed that the amount of available resources is 50. The optimal system structure is shown in Fig. 2. This is obtained by solving the nonlinear programming problem defined by (4) and (5) with the numerical values given in Table I.

The reliability of the system shown in Fig. 2 is 0.717 and its cost 46.

III. RELIABILITY OPTIMIZATION MODEL FOR THE MULTIVERSION PROGRAMMING APPROACH

Another reliability optimization model is based on the multiversion programming approach characterized by the following assumptions:

- each program consists of n parallel segments which may run concurrently;
- each segment consists of a different sequence of instructions;
- a voter selects the first of results that passes the test. If the set of similar errors outnumbers the set of good (similar) results at a decision point, then the decision algorithm will arrive at an erroneous decision result;
- a voter may itself be failed.

It is assumed that the software is constructed from the multiversion modules connected in series. Let the following symbols denote:

m	Number of modules.
$l = 1, \cdots, m$	Index for the modules in the system.
n_l	Number of segments in the lth multiversion program (module).
e_{li}	Probability that the ith segment in lth module is failed ($i = 1, \cdots, n_l$).
d_l	Probability that the voter in the lth module cannot select the result out of two or more correct ones.
c_{li}	Amount of resources needed for the ith segment in the lth module.
v_l	Amount of resources needed for the voter in the lth module.

The probability of failure for the multiversion (to be precise n_l-version module) is now given by (for details see [1])

$$F_l(n_l) = \sum_{i=1}^{n_l+1} f_{il} - \sum_{i=1}^{n_l+1} \sum_{j>i}^{n_l+1} (f_{li} \cap f_{lj}) + \cdots + (-1)^{n_l} \bigcap_{i=1}^{n_l+1} f_{li} \tag{6}$$

where

$$f_{lk} \equiv e_{lk}^{-1} \prod_{i=1}^{n_l} e_{lk'} \qquad k = 1, \cdots, n_l. \tag{7}$$

f_{lk} can be interpreted as the probability that out of n_l independent segments all are failed except the kth one.

$$f_{l_{n+1}} \equiv d_l \tag{8}$$

TABLE III
EXAMPLE FEASIBLE SOLUTIONS

Solution No.	System Structure: Segments used in module No. 1	System Structure: Segments used in module No. 2	System reliability	Amount of resources used
1.	1&2	2&3	0.9639	45
2.	2&3	2&3	0.9609	42
3.	1&3	2&3	0.9568	41
4.	2	1&2&3	0.9467	42
5.	1&3	1&3	0.9447	46
6.	1	1&2&3	0.9168	40
7.	1&2	1	0.8941	39
8.	3	1&2&3	0.8769	38
9.	1&2&3	2	0.8492	42
10.	1&2&3	3	0.8393	41

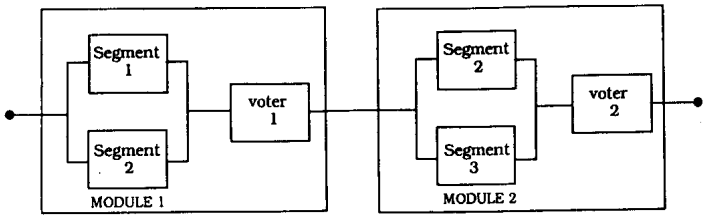

Fig. 3. Structure of the optimal system.

Now the problem of finding the optimal numbers of segments in the software system constructed of multiversion programs can be stated as follows:

$$\max_{g_l} \prod_{l=1}^{m} (1 - F_l(|g_l|)) \tag{9}$$

subject to

$$\sum_{l=1}^{m} V_l \sum_{i=1}^{g_l} C_{li} \leq C \tag{10}$$

$$g_l \subseteq G_l, \qquad l = 1, \cdots, m \tag{11}$$

where

g_l Denotes a subset of the finite set G_l, $l = 1, \cdots, m$ of possible functional segments. Elements of this subset are enumerated from 1 to n_l.

$|g_l|$ Denotes cardinality of g_l.

As an example we again consider a system consisting of two modules. For the sake of simplicity it has been assumed that for each module the finite set of possible functional segments consists of three elements. The respective failure probabilities and costs are given in Table II.

It is also assumed that the probability of the voter failure is $d_l = 0.02$ and its cost $v_l = 5$ both for $l = 1, 2$. The amount of available resources is 46.

In Table III, ten feasible solutions to the example problem are presented. The optimal solution, obtained here by the exhaustive search guarantees overall system reliability at the level of 0.9639 and uses 45 resource units. The structure for the optimal software system is shown in Fig. 3.

The optimization problem defined by (9), (10), and (11) clearly has a combinatorial character and belongs to the NP-complete class. Hence, it is unlikely that there exists a polynomial time algorithm for solving it. Consequently, for the large scale problems, approximate algorithms will have to be employed.

IV. CONCLUSIONS

The idea of optimizing software reliability has yet to prove its feasibility and practicability. The authors believe that the extensive research effort in the field of software reliability will bring such a proof quite soon. In this paper we have outlined several problems which are still open. The models presented in Sections II and III show that it is possible to formulate and solve some software related reliability optimization problems. They further show that the concept of redundancy to achieve fault tolerance (basic for the traditional theory of reliability) can be used in the field of software reliability optimization.

The next step in the development of the reliability optimization models can lead toward directly incorporating correlated failure models as, for example, the one proposed in [5]. It must be however stressed that even within the present models the correlated failure behavior can be accounted for. It is rather the problem of employing proper software reliability prediction models reflecting the increased probabilities of failure due to the same input, than trying to make the optimization models' structure more complex.

Moreover, the authors are of the opinion that designing software systems with redundancies will become a common software engineering practice. Although our results have a rather preliminary character, we are of the opinion that they are convincing in demonstrating that software reliability optimization is worth researching and experimenting.

REFERENCES

[1] F. Belli and P. Jedrzejowicz, "Fault-tolerant programs and their reliability," *IEEE Trans. Rel.*, vol. 39, no. 2, pp. 184–192, 1990.

[2] S. Bittani, Ed., *Software Reliability Modelling and Identification* (Lecture Notes in Computer Science, No. 341). Berlin: Springer, 1987.

[3] IIT Research Institute, "Quantitative software models," Data and Analysis Center, Rome Air Development Center, Rome, NY, Tech. Rep., 1979.

[4] J. D. Musa, "A theory of software reliability and its applications," *IEEE Trans. Software Eng.*, vol. SE-1, no. 3, pp. 312–327, 1975.

[5] V. F. Nicola and A. Goyal, "Modeling of correlated failures and community error recovery in multiversion software," *IEEE Trans. Software Eng.*, vol. 16, no. 3, pp. 350–359, 1990.

[6] C. V. Ramamoorthy and F. B. Bastani, "Software reliability—Status and perspectives," *IEEE Trans. Software Eng.*, vol. SE-8, no. 4, July 1982.

[7] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, no. 2, pp. 220–231, 1975.

[8] M. L. Shooman, "Software reliability: A historical perspective," *IEEE Trans. Rel.*, vol. R-33, no. 1, pp. 48–55, 1984.

[9] M. L. Shooman and G. Richeson, "Reliability of shuttle mission control centre software," in *Proc. Annu. Rel. Maintainability Symp.*, 1983, pp. 125–135.

[10] M. L. Shooman, *Software Engineering: Design, Reliability, Management.* New York: McGraw-Hill, 1985.

[11] F. A. Tillman, C. L. Hwang and W. Kuo, "Optimization techniques for system reliability," *IEEE Trans. Rel.*, vol. R-26, no. 3, pp. 148–153, 1977.