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., Towards Reliability Optimization of Fault-tolerant Software, Journal of Information Processing and Cybernetics - EIK, Vol. 88(9), pp.431-442

Towards Reliability Optimization of Fault-tolerant Software

By *Fevzi Belli* and *Piotr Jedrzejowicz*

Abstract: The paper surveys and classifies probabilistic models for the reliability prediction of computer programs. The controversial discussion about the pros and cons of the software reliability modelling will be sketched. An approach to the optimization of the software reliability achieved by the resources applied (as to amount and quality of programming efforts, development and operation environment, and redundancy to achieve fault-tolerance) is given. The results will be applied to determine the structure of software to be developed. The approach will be illustrated by simple examples.

1. Introduction

When reliability theory was born some 35 years ago (see for example [4]) it was intended to describe analytically and formally the dependency between component failures and a system failure with respect to, in general, hardware systems (mechanical, electrical etc.). The main application of the theory has now become the prediction making as to the prospective reliability of the system in question. As the mathematical background of the reliability theory as well as subsequent reliability models has been getting more and more sophisticated and complex, the quality of reliability behaviour predictions has substantially improved.

It has thus been only natural to extend the range of application from reliability prediction to reliability optimization. Reliability optimization as the branch of general reliability theory appeared in the late sixties (see [28]). Since that time numerous reliability optimization models and procedures have been proposed (see for example [8]). 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, the maximum reliability level is pursued while the available resources are given.

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

Approximately 15 years after the foundation of the general reliability theory the first effort at software reliability modeling appeared. The basic problem in the software area is that the complexity of the tasks which software must perform has grown faster than the technology for designing, testing and managing software development. As *L. Shooman* observes [23], software costs are primarily labor intensive, rather

than technologically dependent and man-hours spent on software development are roughly proportional to the size of the program. Thus, as software complexity has increased over the years the man-hours for a typical project have increased, as have labor costs due to inflation. The net result is that an increasingly larger share of computer system development costs are due to software.

As it was observed in [24] software reliability (just as in the 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. This approach will be called "perfectionistic". The second meaning is the probabilistic definition: software reliability is the probability that a given software system operates for a certain period without software error, on the machine for which it was designed, given that it is used within design limits. This approach can be viewed as the "probabilitistic approach". The reliability degree of a system (concerning both its interpretations) can be increased considerably through adopting the fault-tolerance techniques.

Since the late sixties software reliability modelling and prediction has attained a considerable degree of refinement and sophistication. Numerous existing models can be used to predict the reliability behaviour of the software systems [7]. Still, no generally accepted software reliability optimization models have apparently appeared. Moreover, the opinions about the term "software reliability" are very controversial. Thus, representatives of the perfectionistic approach reject even the use of the term "software reliability" in sense of the probabilistic approach.

In this paper the available software reliability models are reviewed and analysed from the point of view of reliability optimization. Furthermore, the constraints, conditions and limitations of eventual software reliability optimization models are investigated. The software model developed for these purposes includes structural redundancy to achieve fault-tolerance and can be seen as an extended form of the recovery block scheme [19; 1].

2. Software reliability models

A reliability model that is to be useful from the point of view of the systems designer and/or operator should possess at least two features: a good quality of the reliability prediction and the transparency of the model structure. We suggest that users are not interested solely in the quality of predictions which can be obtained from software reliability models but also in their explanatory power. This means that a satisfactory model should explain and respect main structural and functional interrelations governing system behaviour. As the traditional reliability theory shows, to obtain this kind of model quality requires a lot of modelling effort and considerable amount of theoretical research-work.

Looking at the present state of art of the software reliability modelling one should easily observe that a vast amount of models exist, enabling reliability predictions which do not take into consideration any structural nor functional properties of the software systems. These are known as *macro models*.

Macro models treat a software system as if it were a "black box" and the effect of an error in one portion of the program is treated the same as in other portions of the program. Most often the macro models are based on the assumption that the rate of failure is proportional to the number of errors present in the software. Thus the software error model, critical for the whole macro approach, tends to be relatively simple

and generally dependent on the error data collected empirically during development and testing.

It is obvious that the usefulness and accuracy of these models depend then mainly on the quality of test data. On the other hand the best data are collected once the system is deployed and has begun operation. This seems, of course, a bit too late to undertake any significant structural and/or functional changes, which also means that it is usually too late to attempt any software reliability optimization! Nevertheless, the majority of well-known software reliability models belong to the macro class. System behaviour with respect to failure occurrence, debugging and respective reliability growth is described by means of more or less complex stochastic distributions, the parameters of which are to be estimated by the user. The characteristics of some of these models are shown in Table 1 (see also e.g. [7; 10]).

Predictive power of the macro models seems to be quite satisfactory, providing a sufficient testing procedure is guaranteed (see the reports by *M. L. Shooman*, *G. Richeson* [26] and *J. P. Musa* [17]). Yet this power stems from the purely statistical

Table 1
Some macro models

1. Times between failures
   Models characteristics: In this class the process under study is the time between failures. It is usually assumed that the time between two successive failures follows a distribution whose parameters depend on the number of faults remaining in the program during this interval. Estimates of the distribution parameters are obtained from the empirical values of times between failures. Some authors treat the failure times as realizations of a stochastic process using appropriate time-series model.
   Example models: Jelinski-Moranda De-Eutrophication Model [9], *Littlewood-Verall* [14], *Schick-Wolverton* [22], *Goel-Okumoto* [5].

2. Fault count
   Models characteristics: The process under study is the number of faults or failures in specified time intervals. The failure counts are assumed to follow a stochastic process with a time dependent failure rate. Its parameters can be estimated from the empirical values of failure counts (or failure times).
   Example models: *Goel-Okumoto* [5], *Goel* [6], *Musa* [17], *Shooman* [24].

3. Fault seeding
   Models characteristics: In this class a known number of faults is "seeded" in a program which is assumed to have an unknown number of indigenous faults. The program is tested and the observed number of seeded and indigenous faults are counted. From these, an estimate of the fault content of the program prior to seeding is obtained.
   Example model: *Lipow* [11].

4. Input domain
   Models characteristics: In this class a set of test cases from an input distribution assumed to be representative of the operational usage of the program is generated. An estimate of program reliability is obtained from the failures observed during physical or symbolic execution of the test cases sampled from the input domain.
   Example model: *Ramamorthy-Bastani* [18].

nature of the software failure occuring and its detection and not from the understanding of how particular software component failures contribute (or not) to the failure of the whole software system.

It is argued [23] that the reason for clear simplification and generalization inherent to the macro models is that it is difficult to make a probabilistic model of a program structure (i.e. a *micro model*) and even more difficult to obtain data which are required to apply such a model. Thus, the micro models as a class have not been adequately exploited. They have, however the potential to link reliability with a model of the

Table 2
Some micro models

1. Markov/Semi-Markov reliability cost/reliability prediction
   Models characteristics: A system is considered in which switching takes place between subsystems according to a homogeneous continuous time finite state Markov process. It is assumed that failures occur in the subsystems according to Poisson processes and that failures may occur in the transitions between subunits according to the discrete time independent distributions. Failure processes are assumed to be s-independent. The overall reliability is measured in terms of time to next event and it is estimated from the individual subsystems reliabilities and the transition probabilities. Asymptotic results are given. In the semi-Markov version a randon variable representing the "cost" of the failure together with its distribution function is associated with each failure.
   Example model: *B. Littlewood* [12].

2. Reliability prediction
   Models characteristics: It will be assumed that the software has been written in structured or modular form so that decomposition into subsystems is simple and this decomposition can be related to several modules or other functional structures within the software. The failure rate of the whole system is estimated from the modules execution times, relative frequencies of modules executions and probabilities of failure along each module. Parameters of the model include:
   f_j: frequencies with which the jth module is run,
   t_j: running time of the jth module
   q_j: probability of error along the jth module.
   Failure rate is estimated as

$$Z_0 = \frac{\sum_{j=1}^{i} f_j q_j}{\sum_{j=1}^{i} f_j (1 - q_j/2)\ t_j}$$

   Example model: *M. L. Shooman* [25].

3. Statistical prediction of programming errors
   Models characteristics: Multiple linear regression techniques were used to analyse software error data with variables related to software structure, software complexity and programmers capability. Length of the program was shown to be the best single predictor of errors per program. The number of program interfaces per 100 lines of source code was the best single predictor or error rate.
   Example model: *W. Motley, W. D. Brooks* [16].

software structure. Furthermore, if this dependence is known it can help to choose the rational (optimal) structure.

The existing micro models usually assume that the program structure can be represented as a set of paths or program modules (segments). This approach is shown in Table 2 (see also [2; 7; 10; 24; 29]).

Based on the above review the following conclusions with respect to the present state of the art in the field 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.
- The existing models take good advantage of the existing probability theory tools and concepts. They do not however aim at explaining the causes and effects of the software failures in terms of modern software engineering.
- The role of software reliability modelling and reliability predictions should not be limited to guiding the remainder of the development cycle but should also cover the phase of the very early design stages when the future system shape and characteristics are conceived and formulated.

It can be concluded that further research effort will bring some changes in the basic form of the model to incorporate a new understanding of the underlaying principles.

Criteria for the evaluation of the software reliability models for the purposes of reliability optimization will be considered in the next section.

3. Software reliability optimization: general remarks

Assuming that reliability optimization would emerge as soon as reliability theory reaches a certain level of confidence in terms of both predictive and explanatory power, it should be noted that the software reliability optimization is not too far from becoming a well established field of research (not to mention its practicability which has yet to be proved).

Nevertheless, it is clear that producing software systems requires resources (time, highly qualified labor, computer hardware, redundancy to achieve fault-tolerance etc.). These resources can be used in a multitude of fashions producing software systems of different (or similar) qualities and characteristics. It is reasonable to assume that the software user expects his/her software to perform the required predefined functions with a satisfactory reliability and that he/she is interested in meeting those expectations with the minimal use of resources. Hence the rationale for software reliability optimization can not be questioned and the following optimization problems are of real practical interest:

I. Minimize total use of resources
Subject to the fact that the software system performs required functions with a required reliability

II. Maximize software system reliability
Subject to the fact that the software system performs required functions within available resources

Both alternative formulations are of a very general nature. They are however sufficient to raise the following question:

Is there any possibility to build software reliability optimization models encompassing formal descriptions of the dependency between use of resources (including various software structures, design procedures, fault-tolerance techniques, operation models

etc.) and the software reliability? If the answer is positive then there is an open field for research on models design and validation, solution finding algorithms etc. If the answer is negative it remains to concentrate on developing the collection of all analytical and heuristic techniques which can be used to design and test the software so that it is relatively error-free. Eventually both approaches can be combined and applied jointly.

To answer the question of the feasibility of the software reliability optimization one can learn some lessons from the hardware systems reliability optimization. In doing this, the crucial differences between software and hardware systems and their reliability must be strictly taken into account. Some of these differences are expressed below:

- Software reliability is growing during the system lifetime, provided that the software errors will be removed steadily and efficiently.
- Software components are not subject to wear and tear processes.
- Faultless software remains so practically forever.
- Software systems are getting old only morally, i.e. from the point of view of the user: software becomes rather "old fashioned" than erroneous because of physical altering processes.

Software life-cycle can be compared to the first stage of the life-cycle of the hardware system when the design and production faults were gradually removed. (The term "life-cycle" is concerned here with the different periods of failure rate. Thus, the first stage of the life-cycle will be characterized by the early failures.)

Software systems can not be represented generally and directly by the concept of reliability structure as the hardware ones can. The reliability optimization of the hardware systems is feasible because they can be constructed with alternative components and/or can have alternative reliability structures. Moreover the costs (in terms of the resources spent) of different structures and/or components is straightforward. Hence, the least resource-consuming combination of the alternative components/structures producing the required reliability is optimal.

As it was shown in Section 2 there exist two different approaches to the software reliability modelling. Models of the first class (macro models) disregard completely the resources used to construct and deploy a software system. Models of the second class (micro models) consider use of resources. Only the second class can be considered as a potential vehicle for software reliability optimization.

Unfortunately the models in question deal in fact not with software reliability but precisely with probabilistic modelling of software errors. The micro modelling approach may eventually lead to the real software reliability modelling. Hence, to attempt any software reliability optimization the reliability micro-model is needed, taking into account resources spent. Present software reliability theory approach does not provide such a model but seems close to be able to provide it.

From the preceding, the following requirements would have to be met to make the idea of the software reliability optimization practicable:

- Software reliability models should relate reliability of the components with the system reliability, as well as system structure (i.e. the way in which components are connected and how the redundancy is to be introduced into the system to achieve fault-tolerance) with system overall reliability,
- Each software component must be evaluated in terms of resources needed for its design, implementation and operation.

To meet these requirements the problem of software components definition must be solved. It is known that the idea of a hardware component is well defined but the

definition of software modules is largely unexplored. More work must be done on defining what we mean by a particular type of program module. Once the idea of the software component becomes accepted the evaluation of components' reliabilities would require an extensive amount of data gathering and analysis.

Looking at the present state of research work in the field of software reliability, it seems that both problems (i.e. defining software components and evaluating their reliabilities) are likely to be successfully solved. Then, in our opinion the answer for the basic question raised in this section is a positive one: There is a possibility to build a meaningful and practicable software reliability optimization model. To illustrate this possibility, the idea of a software reliability optimization model is introduced in the next section.

4. Software reliability optimization: example model with redundancy

We assume that the software system is constructed with modules connected in series. They can be operated, of course, concurrently, but the system function will be accomplished if all modules work without failure. Thus, our asumption that the modules are connected in series implicates no constraint of generality.

The concept of module is shown in Fig. 1. Each module consists of n parallel paths (path is understood as the sequence of instructions). Each path, excluding the nth one, consists of two parts. The first one includes the sequence of instructions which perform the module functions. The second one forms the testing segment. Its role is to test whether the instructions of the first part have been realized error-free. The testing segment is also prone to failures — as any piece of software is. Its failure means that the performance of the first part is evaluated as error-free but errors in fact have occurred and vice versa. If the error is, however, detected by the testing segment of the ith path then the $(i+1)^{\text{st}}$ path is activated and performed. The nth path has no testing segment. Notwithstanding eventual errors the next module is activated after this path has been performed.

If no error is detected by the testing segment of the ith path, the next module is started. Failures of each path of a module as well as failures of functioning and testing segments of modules are s-independent. It is assumed that with each part of every path there is associated an amount of resources needed to design, deploy and operate it. Total resource consumption of the module is the sum of the resource requirements for each path.

It is furthermore assumed that the information on all the modules/paths/segments reliabilities is available with respect to the given time horizon. Resources' requirements for all modules/paths/segments are known.

Based on the above assumptions we formulate following the software reliability optimization model. The following notation is introduced:

m number of modules

$j = 1, \dots, m$ index for modules in the system

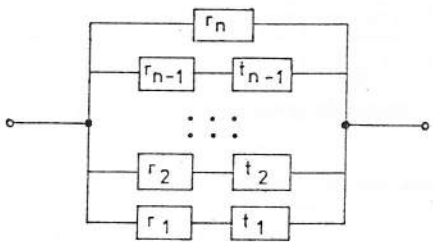

Fig. 1. Module of a software system

n_j	number of paths in the jth module
$i = 1, \ldots, n_j$	index for paths in the module
r_{ji}	reliability of the ith path in the jth module (refers to the functioning segment)
t_{ji}	reliability of the testing segment of the ith path in the jth module (probability that the error is detected if occurred)
c_{ji}	value of the resources needed for the ith path in the jth module
R_j	j^{th} module reliability
R	system reliability
C	value of the 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} is defined recursively:

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

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 reliability optimization problem is:

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

subject to

$$\sum_{j=1}^{m} \sum_{i=1}^{n_j} c_{ji} \leqq C\,.$$

As an example we consider the system consisting of 2 modules. Reliabilities and costs of its components are given in Table 3. It is assumed that the value of available resources is 50.

The optimal system structure is shown in Fig. 2. Reliability of the system shown in Fig. 2 is 0.739 and its cost 48.

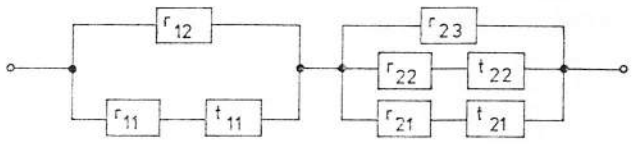

Fig. 2. Optimal structure of the example system

Table 3
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	.6	.7	10	6
2	.8	.5	12	8

5. Discussion and comparison with related work

The module and testing concept described in the last section is slightly different from the conventional concept of recovery blocks [19; 1]. First, in our concept a testing segment is considered to be constructed separately for each path. The test to be performed can then be a single absolute test (as it will be recommended in the recovery block concept and assumed in our module concept), but also a series of the tests $t_1, t_2, \ldots, t_{n-1}$. Further, instead of an absolute test a relative test can be performed. In the case of relative testing all the testing segments $(t_1, t_2, \ldots, t_{n-1})$ or a combination of them can be used as the test segment of each path of the module instead of one single test. Finally, for each path a voter would compare the results. The availability of a sufficient amount of time redundancy and appropriate reconfiguration techniques are of course assumed in these cases. These variations are shown in Fig. 3.

A second difference of our concept to the recovery block scheme stems from the characteristic properties of the relative testing. As in the case of n-version program-

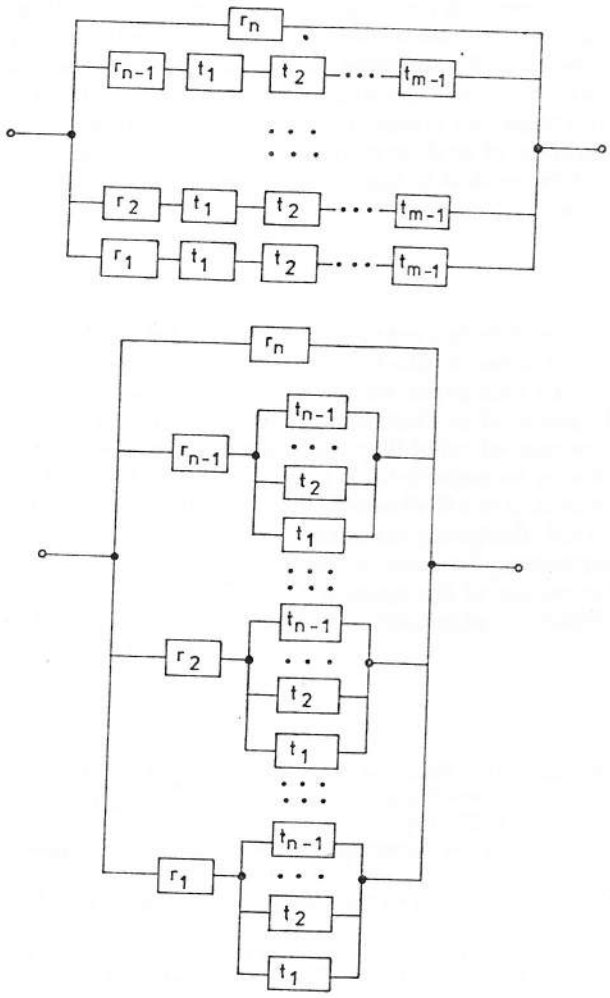

Fig. 3. Variations of the module concept

ming, all the paths can be executed simultaneously, and the final one relative test will be performed by the voter. Thus, the described concept can be viewed as an extension of the recovery block technique. The concept described here can be moreover refined through the refinement of the module structure. Instead of the assumption that the module function will be performed by solely one segment, a series of segments can be considered to carry out the module function. In this case "module" is to be substituted by "path" in the model described in Section 4. Appropriate modifications are then necessary and trivial in the formulas (1) to (5) of the last section.

Because of the differences to the conventional recovery block scheme as explained above we can not apply the reliability models of recovery block as introduced elsewhere (e.g. [20; 21]). In our concept, emphasis is on the general software components or modules as well as their testing complements.

For the sake of simple explanation we build our optimization model according to the assumptions made at the beginning of the last section. Furthermore, in the example above we solved the optimization problem for a simple case to demonstrate the feasibility of our considerations. In this simple case we assume no diversification of instruction sequences within a module and therefore integer programming techniques effectively generate a solution. Diversification of module contents brings about a numerical complexity problem. The authors believe that in the discussed case the optimization problem becomes then NP-complete.

6. Conclusions

The idea of optimizing software reliability has yet to prove its feasibility and practicability. We 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 model presented in Section 4, shows that it is possible to formulate and solve some software related reliability optimization problems. It also shows 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. Moreover we are of the opinion that designing software systems with redundancy will become common software engineering practice. Although our results have rather general and preliminary character we are of the opinion that they are convincing in demonstrating that software reliability optimization is worth researching and experimenting.

References

[1] *Anderson, T., P. A. Lee*: Fault Tolerance. Prentice-Hall International, 1981.

[2] *Belli, F., P. Jędrzejowicz*: Ein Ansatz zur Zuverlässigkeits-Optimierung fehlertoleranter Software. Informationstechnik it **29** (1987) 2, 61–68.

[3] *Brooks, W. D., R. W. Motley*: Analysis of Discrete Software Reliability Models. Reports RADC-TR-80-84, 1984.

[4] *Carhart, R. R.*: A Survey of the Current Status of the Reliability Problem. Rand Corporation RM-113, 1953.

[5] *Goel, A. L., K. Okumoto*: A Time Dependent Error Detection Rate Model for Software Reliability and other Performance Measures. IEEE Trans. Reliability **R-28** (1979), 769–774.

[6] *Goel, A. L.*: Software Reliability Modelling and Estimation Techniques. Rep. RADC-TR-82-263, 1982.

[7] IIT Research Institute: Quantitative Software Models. Report; available from Data and Analysis Center, Rome Air Development Center, Rome, NY, 1979.

[8] *Jędrzejowicz, P., L. Rosicka*: Multicriterial Reliability Optimization Problem. Foundations of Control Engineering 8 (1983) 3/4, 165—173.
[9] *Jelinski, Z., P. B. Moranda*: Software Reliability Research. In: Statistical Computer Performance Evaluation; ed.: *W. Freiberg*; Academic Press, New York 1972; pp. 465—484.
[10] *Keiller, P. A., B. Littlewood, D. R. Miller*: Comparison of Software Reliability Predictions. In: Proceedings of 13th FTCS; 1983; pp. 128—134.
[11] *Lipow, M.*: Estimation of Software Package Residual Errors. TRW, Software Series Rep., TRW-SS-72-09, 1972.
[12] *Littlewood, B.*: A Reliability Model for Markov Structured Software. Appl. Statist. **24** (1975) 2, 172—177.
[13] *Littlewood, B., A. Sofer*: A Bayesian Modification of the Jelinski-Moranda Software Reliability Growth Model. Unpublished manuscript.
[14] *Littlewood, B., J. L. Verrall*: A Bayesian Reliability Growth Model for Computer Software. J. Roy. Statist. Soc. (Appl. Statist.) 22/1973, 332—346.
[15] *Littlewood, B., D. R. Miller*: A Conceptual Model of the Effect of Diverse Methodologies on Coincident Failures in Multi-Version Software. In: Fault-Tolerant Computing Systems; eds.: *F. Belli, W. Görke*; Springer-Verlag, 1987; pp. 263—272.
[16] *Motley, R. W., W. D. Brooks*: Statistical Prediction of Programming Errors. Report RABC-TR-77-175, 1977.
[17] *Musa, J. D.*: A Theory of Software Reliability and its Applications. IEEE Trans. Software Engrg. **SE-1** (1973) 3, 312—327.
[18] *Ramamoorthy, C. V., F. B. Bastani*: Software Reliability: Status and Perspectives. IEEE Trans. Software Engrg. **SE-8** (1982) 7, 359—371.
[19] *Randell, B.*: System Structure for Software Fault Tolerance. IEEE Trans. Software Engrg. **SE-1** (1975) 2, 220—231.
[20] *Scott, R. K., J. W. Gault, D. F. McAllister, J. Wiggs*: Experimental Validation of six Fault-Tolerant Software Reliability Models. In: Proceedings 14th FTCS; 1984; pp. 102—107.
[21] *Scott, R. K., J. W. Gault, P. F. McAllister*: Fault Tolerant Software Reliability Modelling. IEEE Trans. Software Engrg. **SE-13** (1987) 5, 582—592.
[22] *Schick, G. J., R. W. Wolverton*: An Analysis of Computing Software Reliability Models. IEEE Trans. Software Engrg. **SE-4** (1978) No. 2.
[23] *Shooman, M. L.*: Software Reliability: A Historical Perspective. IEEE Trans. Reliability **R-33** (1984) 1, 48—55.
[24] *Shooman, M. L.*: Software Engineering: Design, Reliability, Management. McGraw-Hill, New York 1983.
[25] *Shooman, M. L.*: Structural Models for Software Reliability Prediction. In: Proceedings 2nd IEEE International Conference on Software Engineering; 1976; pp. 268 to 280.
[26] *Shooman, M. L., G. Richeson*: Reliability of Shuttle Mission Control Centre Software. In: Proceedings of Annual Reliability and Maintainability Symposium; 1983; pp. 125 to 135.
[27] *Shooman, M. L., A. K. Trivedi*: A Many-State Markov Model for Computer Software Performance Parameters. IEEE Trans. Reliability **R-25** (1976) 2, 66—68.
[28] *Tillman, F. A., C. L. Hwang, W. Kuo*: Optimization Techniques for System Reliability. IEEE Trans. Reliability **R-26** (1977) 3, 148—153.
[29] Handbook of Software Engineering. Eds.: *C. R. Vick, C. V. Ramoorthy*. Van Nostrand, New York 1984.

Резюме

Дан обзор и классификация вероятностных моделей определения надежности вычислительных программ. Обрисована бурная дискуссия о праве на существование моделирования надежности вычислительных программ. Представлен возможный подход оптимизации надежности относительно стоимости (программирования, экс-

плуатационного и операционного оборудования) и относительно избыточности достижения нечувствительности к ошибкам. Результаты применены для определения оптимальной по стоимости программной структуры. Подход иллюстрирован простыми примерами.

Kurzfassung

Kriterien zur Klassifizierung und Bewertung wahrscheinlichkeitstheoretischer Methoden zur Bestimmung von Software-Zuverlässigkeit werden erstellt. Die heftige Diskussion über die Existenzberechtigung der Software-Zuverlässigkeitsmodellierung wird umrissen. Ein Ansatz zur Optimierung der Zuverlässigkeit im Hinblick auf Kosten (wie für Programmierung, Entwicklungs- und Einsatz-Umgebung) und auf Redundanz zur Erzielung der Fehlertoleranz wird vorgestellt. Die Ergebnisse werden zur Bestimmung einer kostenoptimalen Software-Struktur eingesetzt.

(Received: first version October 24, 1986,
present version February 4, 1988)

Authors' addresses:

Prof. Dr.-Ing. F. Belli
Hochschule Bremerhaven
Dept. of Systems Analysis
Bürgermeister-Smidt-Str. 20
2850 Bremerhaven
Federal Republic of Germany

Prof. Dr. P. Jędrzejowicz
Merchant Marine Academy
Dept. of Systems Analysis and Informatics
ul. Czerwonych Kosynierow 83
Gdynia
Poland