C. Wohlin, "Some New Aspects on Software Reliability Model Comparison",
Proceedings Software Engineering '88, pp. 38-42, Liverpool, United Kingdom, 1988.

SOME NEW ASPECTS ON SOFTWARE RELIABILITY MODEL COMPARISONS

C. Wohlin

Lund Institute of Technology, Sweden

INTRODUCTION

Confidence in the quality, discussed in Kitchenham (1) and (2), of a given software system has become an important issue, as systems become more and more complex.

It is still a long way to go until a complete and comprehensive software quality methodology exists, including measurements and models, but a lot of studies are going on. Different methods and models, which describe the characteristics of the software, have evolved during the last two decades.

One of the most important and studied quality aspects is the reliability of the software, for definition see IEEE Standard 729 (3). Software reliability is discussed thoroughly in Musa et al (4). Some general information about reliability and faults may be appropriate at this point. An error is made by a human being and results in a fault in the product. The manifestation of a fault, which means a departure from what the software is supposed to do, is referred to as a failure, IEEE Standard 982 (5).

The difference between reliability and fault content should be observed. A product may have a number of faults, but if they are located in paths that are seldom executed, the product is considered to be reliable. It should be noted that faults considered by reliability models are those that contribute to the reliability under present conditions, and not necessarily to the total fault content.

During development it is possible to estimate the fault content from complexity metrics and reliability models. The latter estimation is possible since, in many of the models, one of the parameters is the number of faults. The models applicable during development are highly dependent on the environment and techniques used, since the behaviour of the product is dependent on, for example, the testing strategies applied. This means that the estimate during development is only an indication of possible problems during operation; this limitation is due to the problem of imitating the operation phase. During operation, however, it is possible to use reliability models to estimate the reliability and the fault content. The fault content that is estimated at this time of the life cycle is the one that the user faces, which means that these are the faults that really contribute to the total life cycle cost. Fault content estimations from complexity metrics and software reliability models are discussed in Lennselius et al (6).

Techniques and tools for keeping track of the fault content and the reliability are needed, as long as fault free software can not be guaranteed. The customer, who buys software systems, needs to know if the product fulfills the quality constraints put on it, in this case on the reliability and the residual fault content at delivery. The tools available for this are mainly software reliability models. Various models have been proposed to characterize software reliability and its dependence on a number of factors related to the product or the software process, some of these are presented in Jelinski and Moranda (7), Goel and Okumoto (8), Schneidewind (9), Littlewood and Verral (10), and Musa and Okumoto (11).

In this paper we will only be concerned with time-dependent models and not those who only give a static value. Most models are, however, time-dependent and they are also the most commonly used type of models. The models are used not only for prediction of reliability and number of faults, but also in, for example, quality constraint models and models for optimal field entry. A quality constraint model for the release of the product based on reliability models is discussed in Wohlin and Vrana (12). Models for optimal field entry are discussed in Křtěn and Levy (13) and Wohlin and Körner (14).

COMPARE SOFTWARE RELIABILITY MODELS

The techniques for achieving and demonstrating high reliability are available, through the various reliability models. But how do we use them and do we really need this diversity of models?

To answer this question we have to compare models and study how they relate to each other. First of all we have to study the models and select those that are best suited for our environment, techniques and applications according to the assumptions made by the models. Having done this we are probably left with either a number of models or none at all. If we are left with none, we have to take an approach similar to the one presented in Wohlin (15), i.e. to develop a model which is tailored to the environment and the techniques used. Let us suppose that we still have at least two possible models.

An easy (but time-consuming and expensive) method is to use multi-modelling, i.e. to use all models and then choose the best one at the end. The criterion for choosing is hard to decide. Should we take the worst case, or the model which makes the management happy, or what? Even if a couple of models give the same result, can we be sure that it is the right result? There is a possibility that the result is the same because the models are similar to each other, and this result might not be accurate. The author suggests another approach, which we can call the pre-evaluation approach. We wish to stress the need to compare and evaluate the software reliability models before we use them in a given environment and organization, Goel (16). Our suggestion is that, instead of spending money buying expensive program packages for various models and then running them, effort should be put into investigation and classification. Having understood the models, tools can be selected for some of these models which are best suited to the individual application. Some classifications already exist (16), but they are mostly concerned with classifying the models according to the approach taken in developing them. This information is of little interest to the user of the models. The user is, in most cases, not especially interested in the mathematical background but rather in the usefulness of the results.

We have to compare the information we get from the models. If one model gives us all the information we need, why use the others? A thorough study of the available models, the information obtained from them and its accuracy has to be conducted. We have to catalogue the models and find out where they overlap. Once this is done we can use the results as guidelines on which model(s) to choose depending on the environment, applications and information needed. The model has first of all to be realistic with respect to the product's behaviour, which is highly influenced by the environment both during development and operation. The behaviour of the product is influenced by, for example, the testing strategies during development and the use of the product during operation. When the model is considered to be realistic and applicable, then the next step is to consider another important aspect when comparing models - the accuracy of estimates obtained from the models, Abdel-Ghaly et al (17).

The way to compare models and how this can be put in practice is explained in some more details below. The arguments above are supposed to motivate the need for comparisons and the structuring of the area, in order to make the research results of reliability models available to the industry developing software systems.

CRITERIA FOR COMPARISONS

An attempt, and a very good one, to systematize the area of how to compare different models is presented in Iannino et al (18). The criteria discussed there will shortly be commented and they will also be complemented in some parts. Then two new criteria for comparisons will be suggested. A structure of the seven criteria into three separate parts is presented. The following five criteria are presented in (18):

1. Predictive validity
"Predictive validity is the capability of the model to predict future failure behaviour during either test or operational phases from present and past failure behaviour in respective phase." This is discussed in detail in (17).

2. Capability
"Capability refers to the ability of the model to estimate with satisfactory accuracy quantities needed by software managers, engineers, and users in planning and managing software development projects or controlling change in operational software systems." It can be added here that one ought to consider distributions and variances as well as mean values, since they make it possible to calculate confidence intervals and consequently to prepare for different outcomes.

3. Quality of assumptions
"The assumptions should be plausible from experience, actual data etc."

4. Applicability
"A model should be judged on its degree of applicability across different software products, different development environments, different operational environments, and different life cycle phases."

5. Simplicity
"A model should be simple in three aspects; simple and inexpensive to collect data, simple in concept, and readily implemented as a program." The second aspect ought to be complemented. Simplicity is a quality of the model itself and not easily comparable between different models. The models should be compared mathematically: distributions, mean values and variances should be considered, taking both time-dependent and asymptotic values, in order to identify similarities and differences between models. This type of comparisons between models ought to be incorporated in the criterion simplicity, which means that a more appropriate name would be "simplicity and resemblance".

These five criteria need to be complemented with two new aspects on software reliability model comparisons.

6. Predictive quantities
The quantities predicted from a model is an important issue. This criterion is related to capability, but refers to the possibility to predict the quantities at all, where capability refers to the ability to estimate with satisfactorily accuracy. The reason for introducing this new criterion is to have a criterion which refers directly to the model, where capability refers to the comparison of models with failure data. This means that models can be compared with each other as to which quantities they can supply the user with. It is, for example, no use applying a more sophisticated model than is necessary. And on the other hand, if we need a lot of quantities, then we can use more than one model. This criterion makes sure that the models used complement each other.

7. Coverage
Coverage is the degree to which the model cover the stochastic variations of the failure data. This criterion is evaluated on past and present failure data. The evaluation of this criterion can be divided into a number of steps. First of all, the failure data is used to estimate the parameters of the different software reliability models. The second step is to determine the confidence intervals for the models, based on their distributions and approximations of distributions. The confidence intervals of the different models are then plotted in a diagram together with the observed failure data. The final step is to see how the confidence intervals cover the failure data. If we have chosen a 90% confidence interval, then 9 out of 10 of observed failure data should fall inside the interval. Otherwise the model either under-estimates or over-estimates the stochastic variations of the failure process, i.e. if the statistical uncertainty in the estimate of the parameters is neglected. For each model we can calculate a hit-rate, the relative number of data points inside the confidence interval, which should be compared with the degree of the calculated confidence interval. The model which has its hit-rate closest to the degree of the confidence interval is said to be the best. An exact match can not be expected, but the ability of the models to cover the variations can be judged by this criterion. The reason to evaluate this criterion is that, if the coverage is high at present it will probably be high in the future too. That is the confidence interval of the model can be used to plan for different outcomes of the failure process. We can plan, for example, how to apply our resources in order to reach target dates of the project even for the worst (probable) case.

One problem arises when to calculate the hit-rate. Should all data points be considered, or only the one from the first time the model converge and forward? The arguments for the latter is that the model under observation is not considered to be realistic, until its parameters can be estimated from the failure data. The models need a number of data points in order to find a solution to the equations found by the maximum-likelihood method or the minimum-square method, i.e. if they can find

a solution at all. The choice of how to calculate the hit-rate is, however, not critical, the criterion can be evaluated in both cases.

An example of how two models can be compared under this criterion is presented in Wohlin (19). The two models compared are the Jelinski-Moranda de-eutrophication model (the J-M model), (7), and the Goel-Okumoto Non-Homogeneous Poisson process model (the G-O model), (8). The choice to compare these two models was based on three things:

- These two models are often referenced in the literature and consequently two of the most commonly used models.
- Especially, these two models have been applied to a number of Swedish software projects, as reported in Vrana and Wallander (20) and Kristiansen (21).
- The two models are formulated for the study of two different processes, but they can be compared analytically very well all the same. We will come back to the analytical comparison below, see also above under the extension to the simplicity criterion.

These seven criteria can be divided into three parts. A structure of the criteria for comparisons is proposed in table 1.

TABLE 1 - A structure of the criteria for comparisons

	Model*		
Compared with	Other models	Environment	Failure data
Criteria evaluated	- Simplicity - Predictive quantities	- Quality of assumptions - Applicability	- Predictive validity - Capability - Coverage

* The model to be evaluated.

The structuring of the criteria will make the problem of comparisons easier and hopefully lead to that the software industry adopt the techniques and tools developed, in order to achieve and demonstrate high reliability on their software products.

TWO PHASES FOR THE COMPARISONS

The problem of software reliability model comparisons can be divided into two phases separated in time. The first phase ought to be done once and for all and only be complemented as the research of software reliability modelling provides new results. While the second phase has to do with company and project dependent factors. The three parts, mentioned above, can now be placed in the two phases and we shall see what they can mean in the two different phases.

Phase 1

This phase ought to render in a software reliability handbook, which should contain the framework needed when to introduce software reliability modelling at a company. Most of the information, to be found in the handbook, is available today, but in separate books and articles. The perhaps most complete book today is (4).

Let us look at the three parts one by one.

Model - Other models. Models can be compared with each other without any knowledge of the environment they shall be used in or the failure data. It is quite obvious that it is possible to describe and list predictive quantities and simplicity, except perhaps for the suggested extension of simplicity. That is the problem of analytical comparison of models, which gives a measure of the resemblance between models. We will now return to the two models, shortly mentioned above, the J-M model and the G-O model, and see how they can be compared analytically. The time-dependent models can be placed in two different classes, based on the failure process studied. This is further discussed in (16).

'Times between failures' models. The general approach for this class of models is to assume that the time between failure number (i-1) and failure (i) follows a probability distribution, whose parameters depend on the number of faults remaining. One of the first and most commonly used models is the J-M model, (7).

'Failure count' models. This class of models assumes that the number of detected failures in an interval follows a stochastic process with a time-dependent discrete or continuous failure rate. A well known model from this class is the G-O model, (8).

The possibilities for comparing models within the classes are many, but the main problem is how to compare models when they study different failure processes. To compare models from different classes some common parameters have to be found. It is well known from probability theory and queueing theory, e.g. Kleinrock (22) and White et al (23), that there is a relationship between the distribution of times between two consecutive events and the distribution of numbers of events in an interval. Thus, if this relationship could be used to compare one model from the 'times between failures' and one from the 'failure count' class, then it would be possible to use these two models as reference models for their classes. The reference models would work as a bridge between the classes, i.e. by comparing all models within a class with the reference model in that class, the models will be compared with the models from the other class too. This bridge can be constructed between the time-dependent classes by comparing the two models mentioned above.

The relationship between the times between occurrences and the number of events in an interval can be used for deriving the distribution of the number of detected faults over time t for the J-M model; since this is known for the G-O model, it is possible to compare probability distributions, mean values, variances etc. for the two models.

The result is that the number of failure occurrences over time t for the J-M model follows a binomial distribution, with exactly the same time-dependent mean value as the G-O model. This result is well known from queueing theory, the derivation is, for example, done in (23). This means that we should compare a Poissonian and a binomial distribution with the same mean value. We also observe that the variances for these distributions are well known and can be compared with each other. Thus we can link the two time-dependent classes. The results from the analytical comparison between these two models are together with the evaluation of the coverage criterion discussed in more detail in (19).

The link gives us the possibility to compare the models from the both classes and describe similarities and differences.

Model - Environment. In this phase we can only recommend in which environments, e.g. for different test strategies, the model is possible to use.

Model - Failure data. It is in this phase possible to recommend the type of failure data, calendar time, workdays, execution time etc., that the model is suitable for.

Phase 2

This phase has to be carried out at the companies developing software systems, since the customers will probably demand it. In the near future the customer may, in the specification, put a constraint on the reliability or the fault content. This will enforce the developer to introduce methods for achieving and demonstrating high reliability.

Each company will need the handbook from phase 1 and program packages for the suitable models. In the long run the problem of software reliability modelling at companies can be solved with an expert system, covering phase 2. This solution is today, perhaps, more a vision than a reality, but we can already say how the expert system ought to look like. It should: contain basic information about models (the information from phase 1), ask questions to the user about the environment and information wanted, and view available failure data. Based on this the system compare models for the seven criteria. For each criterion the models will be given a goodness figure and when all criteria have been evaluated the system will choose model(s) and obtain the information wanted through the right program package, by this way the developer will stay in control of the fault content and the reliability of the software product.

Let us look at the three parts for this phase too.

Model - Other models. This part is the easiest one during this phase. The only consideration which has to be done is a comparison between the models and the information wanted. That is we have to evaluate the criterion of predictive quantities.

Model - Environment. As stated above the behaviour of the software product is influenced by, for example, testing strategies during development and the use of the product during operation. The necessity to choose a specific model based on environment considerations, as applicability and quality of assumptions, is stressed in (16).

The way to do the comparison is to develop a process model of the actual life cycle phase. The technique to develop a process model is discussed in Huff et al (24). The process model will give us a possibility to evaluate the environment based criteria. If no software reliability model is found to be suitable for a specific environment, it is possible to take an approach similar to the one presented in (15), where an environment-adapted software reliability model is developed.

Model - Failure data. The perhaps most important criterion, predictive validity, is evaluated in this phase. This criterion is stressed in Mellor (25) and is thoroughly investigated in (17). Another comparison of models is presented in Schick and Wolverton (26). The importance of predictive validity does not mean that the other criteria should be left out. Both the capability and the coverage have to be evaluated carefully. The problem with these three criteria, especially predictive validity and capability, is that they can not really be evaluated until late in the project, when it may be too late to take advantage of it in the project going on. It is, however, important to build up a corporate-memory, and the evaluation can be of great use in new similar projects to come.

CONCLUSIONS

Tools and techniques for achieving and demonstrating high reliability are needed. One such is software reliability models. In order to make these usable in the industry, the area has to be structured. This calls for comparisons between models and a software reliability handbook based on the comparisons. The comparisons have to be done based on a number of criteria. In this paper we have adopted five criteria, suggested in (18), and complemented them on some points. We have also suggested two new criteria that have to be evaluated when comparing models.

The criteria have been structured into three separate parts of comparison to go through with a model; other models, environment and failure data. A way to go to get models into practical use has been proposed. Two phases have to be carried out; phase 1: the development of a general software reliability handbook, phase 2: the introduction of the handbook, or at least suitable models, at companies.

The structuring of the criteria for model comparisons and the suggestion of how the comparisons can be carried out is supposed to encourage and simplify the work that has to be done, i.e. the transformation of research results to practical use in the software industry.

ACKNOWLEDGEMENT

This project is supported by Telelogic AB and the Swedish Telecommunication Administration, Sweden.

Many thanks to Mr B. Lennselius and Dr U. Körner at the Department of Communication Systems, Lund Institute of Technology, Lund, Sweden, for constructive comments, which were helpful in improving the quality of the paper.

Special thanks to Dr B. Kitchenham, the City University, London who provided many valuable comments on a related paper, which gave rise to this paper.

REFERENCES

1. Kitchenham, B., and Walker, J., 1986, "The Meaning of Quality", in "Software Engineering '86", ed. D. Barnes and P. Brown, Peter Peregrinus Ltd, Stevenage, United Kingdom, 393-406.

2. Kitchenham, B., 1987, Software Engineering Journal, July, 105-113.

3. IEEE Standard 729, 1983, "Glossary of Software Engineering Terminology".

4. Musa, J., Iannino, A., and Okumoto, K., 1987, "Software Reliability: Measurement, Prediction, Application", McGraw-Hill Book Company, New York, USA.

5. IEEE Standard 982, 1986, "Measures for Reliable Software".

6. Lennselius, B., Wohlin, C., and Vrana, C., 1987, Microprocessors and Microsystems, 11:7, 365-375.

7. Jelinski, Z., and Moranda, P., 1972, Statistical Computer Performance Evaluation, 465-484.

8. Goel, A., and Okumoto, K., 1979, IEEE Transactions on Reliability, R-28:3, 206-211.

9. Schneidewind, N., 1975, "Analysis of Error Processes in Computer Software", Conference Proceedings "Int. Conf. on Reliable Software", 337-346.

10. Littlewood, B., and Verral, J., 1973, Applied Statistics, 22:3, 332-346.

11. Musa, J., and Okumoto, K., 1983, "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement", Conference Proceedings "7th Int. Conf. on Software Engineering", 230-237.

12. Wohlin, C., and, Vrana, C., 1986, "A Quality Constraint Model to be Used During the Test Phase of the Software Life Cycle", Conference Proceedings "6th Int. Conf. on Software Engineering for Telecommunication Swithing Systems", 136-141.

13. Křtěn, O., and Levy, D., 1980, "Software Modelling for Optimal Field Entry", Conference Proceedings "Annual Reliability and Maintainability Symposium", 410-414.

14. Wohlin, C., and Körner, U., 1988, "Software Faults: Spreading, Detection and Costs", Technical Report, Dept. of Communication Systems, Lund Institute of Technology, Lund, Sweden.

15. Wohlin, C., 1986, "Software Testing and Reliability for Telecommunication Systems", in "Software Engineering '86", ed. D. Barnes and P. Brown, Peter Peregrinus Ltd., Stevenage, United Kingdom, 27-42.

16. Goel, A., 1985, IEEE Transactions on Software Engineering, SE-11:12, 1411-1423.

17. Abdel-Ghaly, A., Chan, P., and Littlewood, B., 1986, IEEE Transactions on Software Engineering, SE-12:9, 954-967.

18. Iannino, A., Musa, J., Okumoto, K., and Littlewood, B., 1984, IEEE Transactions on Software Engineering, SE-10:6, 687-691.

19. Wohlin, C., 1986, "A Comparison Between Two Software Reliability Models: Jelinski-Moranda's De-Eutrophication Model and Goel-Okumoto's Non-Homogeneous Poisson Process Model", Technical Report no. 4, Dept. of Communication Systems, Lund Institute of Technology, Lund, Sweden.

20. Vrana, C., and Wallander, A., 1983, "Software Quality and Complexity - Different Aspects and Measurement Results", Conference Proceedings "5th Int. Conf. on Software Engineering for Telecommunication Switching Systems", 121-127.

21. Kristiansen, L., 1983, "Swedish Hardware/Software Reliability", Conference Proceedings "Annual Reliability and Maintainability Symposium", 297-301.

22. Kleinrock, L., 1975-76, "Queueing Systems Vol. 1 and 2", John Wiley and Sons, New York, USA.

23. White, J., Schmidt, J., and Bennett, G., 1975, "Analysis of Queueing Systems", Academic Press Inc., New York, USA.

24. Huff, K., Sroka, J., and Struble, D., 1986, Software Engineering Journal, January, 17-23.

25. Mellor, P., 1987, Information and Software Technology, 29:2, 81-98.

26. Schick, G., and Wolverton, R., 1978, IEEE Transactions on Software Engineering, SE-4:2, 104-120.