T U M

I N S T I T U T F Ü R I N F O R M A T I K

A Software Reliability Model Based on a Geometric Sequence of Failure Rates

Stefan Wagner and Helmut Fischer

TUM-I0520
Dezember 05

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-12-I0520-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

A Software Reliability Model Based on a Geometric Sequence of Failure Rates*

Stefan Wagner
Institut für Informatik
Technische Universität München
Boltzmannstr. 3
D-85748 Garching b. München

Helmut Fischer
Siemens AG
COM E QPP PSO
Hofmannstr. 51
D-81379 München

Abstract

Software reliability models are an important tool in quality management and release planning. There is a large number of different models that often have strengths in different areas. This paper proposes a model that is based on a geometric sequence of the failure rates of faults. This property of the failure process was observed in practice at Siemens among others and led to the development of the Fischer-Wagner model. It is described in detail and evaluated using standard criteria. Most importantly the model is able to perform equally well as other models in terms of its predictive validity.

*This research was partially supported by the DFG in the project *InTime*.

1 Introduction

Software reliability engineering is an established area of software engineering research and practice that is concerned with the improvement and measurement of reliability. For the analysis typically statistical software reliability models are used. They model the failure process of the software and use other software metrics or failure data as a basis for parameter estimation. The models are able to (1) estimate the current reliability and (2) to predict future failure behaviour.

There already is a number of established models of which Miller classified several important ones as exponential order statistic (EOS) models in [5]. He divided the models on the highest level in deterministic and doubly stochastic EOS models meaning that the failure rates either have a deterministic relationship or are again randomly distributed. For the deterministic models, he gave several interesting special cases. The well-known Jelinsky-Moranda model [3] has *constant rates*, for example. He also described that *geometric rates* are possible and were observed by Nagel [9, 8].

This geometric sequence between failure rates of faults was also found in projects of the communication networks department of the Siemens AG. Several older projects were analysed and this relationship seemed to fit well on the data. Therefore, a software reliability model based on a geometric series of failure rates is proposed.

Problem. The problem software reliability engineering is still facing is that we need accurate models for different environments and projects. Detailed models with a geometric sequence of failure rates have not been proposed so far.

Contribution. We describe a detailed and practical software reliability model that was motivated out of practical experience and contains a geometric sequence of failure rates which was also suggested by theoretical results. A detailed comparison shows that this model has a constantly good performance over several projects although other models can perform better in specific projects.

Outline. We first describe different important aspects of the model in Sec. 2. In Sec. 3 the model is evaluated using several defined criteria, most importantly its predictive validity in comparison with established models. Based on the model a notion of test efficiency is developed in Sec. 4. We give final conclusions in Sec. 5. Related work is cited where appropriate.

2 Model Description

The core of the Fischer-Wagner model is a geometric sequence for the failure rates of the faults. This section describes this assumption and

others in more detail, introduces the main equations, the time component of the model and gives an example of how the parameters of the model can be estimated.

2.1 Assumptions

The main theory of this model is the idea to order the faults that are present in the software based on their failure rates. The term failure rate may describe in this context the probability that an existing fault will result in an erroneous behaviour of the system during a defined time slot or while executing an average operation. The ordering implies that the fault with the highest probability of triggering a failure comes first, then the fault with the second highest probability and so on. The probabilities are then put on a logarithmic scale because this results in a uniform distribution of the points on the x-axis. This is because there is the underlying assumption that there are a many faults with low failure rates and only a small number of faults with high failure rates.

As mentioned above the logarithmic scale brings the data points in approximately the same distance from each other. Therefore, this distance is approximated by a constant factor between the probabilities. Then we can use the following geometric sequence for the calculation of the failure rates.

$$p_n = p_1 \cdot d^{(n-1)}, \tag{1}$$

where p_n is the failure rate of the n-th fault, p_1 the failure rate of the first fault, and d is a project-specific parameter. It is assumed that d is an indicator for the complexity of a system (maybe expressed by the number of possible different branches in a program). In past projects of Siemens d was calculated to be between 0.92 and 0.96. d is multiplied and not added because the distance is only constant on a logarithmic scale.

The failure occurrence of a fault is assumed to be geometrically distributed. Therefore, the probability that a specific fault a occurred by time t is the following.

$$P(T_a \leq t) = F_a(t) = 1 - (1 - p_a)^t. \tag{2}$$

We denote with T_a the random variable of the failure time of the fault a.

In summary the model can be described as the sum of an infinite number of geometrically distributed random variables with different parameters which in turn are described by a geometric sequence.

2.2 Equations

The two equations that are typically used to describe a software reliability model are the mean number of failures $\mu(t)$ and the failure intensity $\lambda(t)$.

The mean value function needs to consider the expected value over the indicator functions of the faults.

$$\begin{aligned} \mu(t) &= E(N(t)) \\ &= E\left(\sum_{i=1}^{\infty} I_{[0,t]}(X_i)\right) \\ &= \sum_{i=1}^{\infty} E(I_{[0,t]}(X_i)) \\ &= \sum_{i=1}^{\infty} P(X_i \leq t) \\ &= \sum_{i=1}^{\infty} 1 - (1-p_i)^t \end{aligned} \tag{3}$$

This gives us a a typical distribution as depicted in Fig. 1. Note that the distribution is actually discrete which is not explicitly shown because of the high values used on the x-axis.

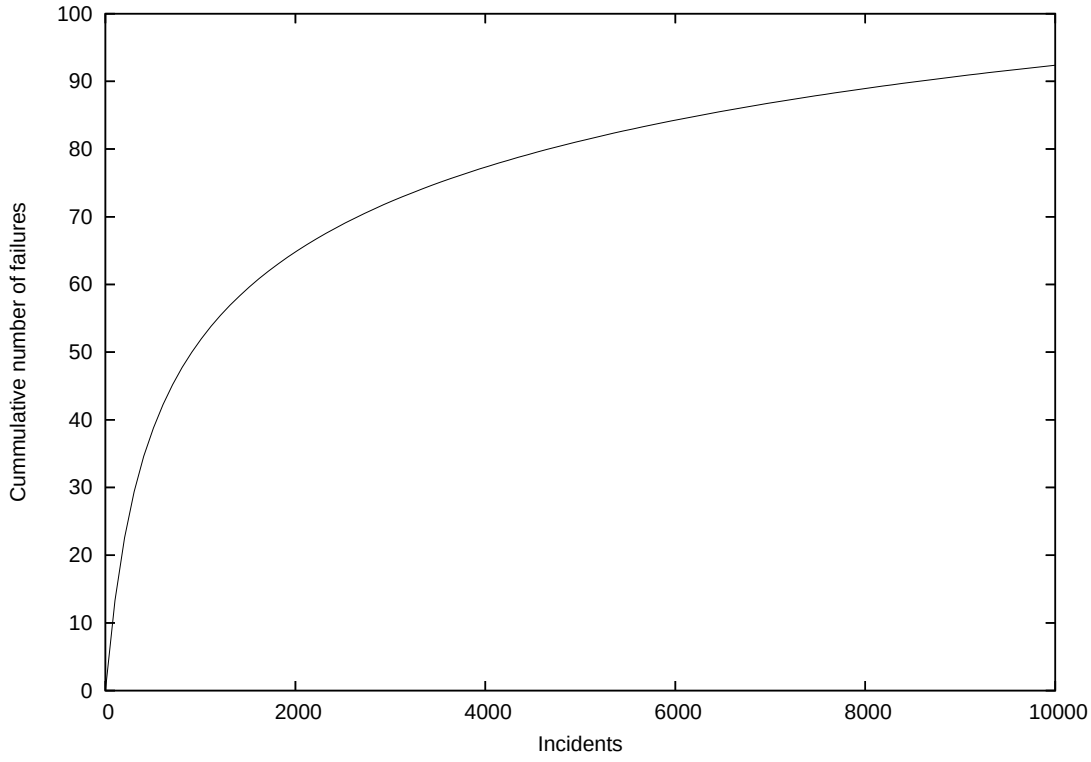

Figure 1: A typical distribution of the model

We cannot differentiate the mean value equation directly to get the failure intensity. However, we can use the probability density function (pdf) of the geometric distribution to derive this equation. The pdf of a single fault is

$$f(t) = p_a(1-p_a)^{t-1} \tag{4}$$

Therefore, to get the number of failures that occur at a certain point in time t, we have to sum up the pdf's of all the faults.

$$\lambda(t) = \sum_{a=1}^{\infty} p_a(1-p_a)^{t-1} \tag{5}$$

An interesting quantity is typically the time that is needed to reach a certain reliability level. Based on the failure intensity objective that is anticipated for the release, this can be derived using the equation for the failure intensity. Rearranging Eq. 4 gives:

$$t = \frac{\ln \lambda}{\sum_{a=1}^{\infty} p_a - p_a^2} + 1 \tag{6}$$

What we need, however, is the further needed time Δt to determine the necessary length of the test or field trial. We denote the failure intensity objective λ_F and use the following equation to determine Δt:

$$\Delta t = t_F - t = \frac{\ln \lambda_F - \ln \lambda}{\sum_{a=1}^{\infty} p_a - p_a^2} \tag{7}$$

Having this, the result needs to be converted in calendar time to be able to give a date for the end of the test or field trial.

2.3 Time Component

In the Fischer-Wagner model time is typically measured in incidents each representing a usage task of the system. These incidents are also converted into calendar time. For this it is necessary to introduce an explicit time component. This means there need to be explicit means to convert from one time format into another.

There are several possibilities to handle time in reliability models. The most preferable way is to use execution time directly but this is often not possible. Then a suitable substitute must be found. In case of testing this could be the number of test cases, in the field the number of clients and so forth. Fig. 2 shows the relationships between different possible time types.

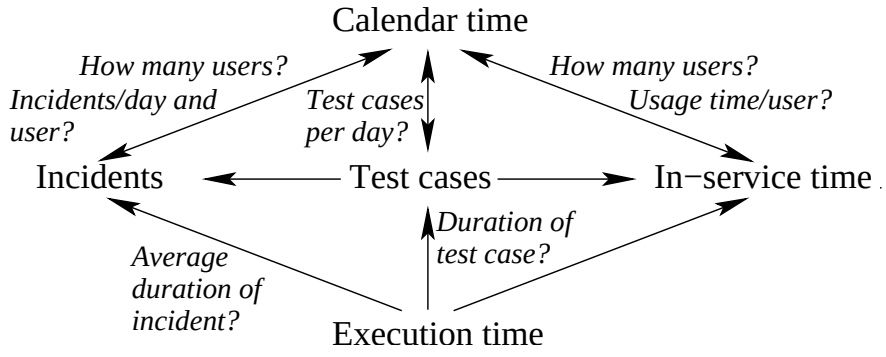

Figure 2: The relationships between different types of time possible

The first possibility is to use in-service time as a substitute. For this we need to know the number of users and the average usage time per user. Then the question is how does this relate to the test cases in system test.

Maybe the average duration of a test case can be used. Instead of the in-service time the number of incidents is a way to measure time. The main advantage of using incidents, apart from the fact that they are already in use at Siemens, is that in this way, we can get very intuitive metrics, e.g., average number of failures per incident. There are often some estimations of the number of incidents per client and data about the number of sold client licenses.

However, also the question of the relation to test cases is open here. A first cut would be to assume a test case is equal to an incident but probably a test case has more "time value" than one incident because it is typically directed testing which means that situations with a high probability of failure are preferred. In addition, a test case is usually unique in function or parameter set while the normal use of a product typically consists of similar actions. Then we do not follow the operational profile and this should be accounted for. A possibility for that is described in Sec. 4.

2.4 Parameter Estimation

There are two techniques for parameter determination currently in use. The first is prediction based on data from similar projects. This is useful for planing purposes before failure data is available.

However, estimations should also be made during test, field trial, and operation based on the sample data available so far. This is the typical approach most reliability models use and it is also statistically most advisable because it is sample data from the population we actually want to analyse. For this, techniques such as Maximum Likelihood estimation or Least Squares estimation are typically used to fit the model to the actual data.

Maximum Likelihood. The Maximum Likelihood method essentially uses a likelihood function that describes how probable it is that a certain number of failures occurred up to a certain time. This function is filled with sample data and then optimised to get the parameters with the maximum likelihood.

The problem with this is that the likelihood function of this model gets extremely complicated. We have essentially an infinite number of random variables that are geometrically distributed but all with different parameter p. Even if we constrain ourselves to a high number N of variables under considerations it still results in a sum of $\begin{pmatrix} N \\ x \end{pmatrix}$ different products. In detail that means we have to sum up every possible permutation in which x failures have occurred up to time t. The number of possibilities is $\begin{pmatrix} N \\ x \end{pmatrix}$. Each summand is a product of a permutation where different

faults resulted in failures.

$$
\begin{aligned}
L(p_1, d) = \quad & \textstyle\prod_{i=1}^{x} 1 - (1-p_i)^t \cdot \prod_{i=x+1}^{N} (1-p_i)^t + \\
& \textstyle\prod_{i=2}^{x+1} 1 - (1-p_i)^t \cdot \prod_{i=x+2}^{N} (1-p_i)^t \cdot (1-p_1)^t + \\
& \textstyle\prod_{i=3}^{x+2} 1 - (1-p_i)^t \cdot \prod_{i=x+3}^{N} (1-p_i)^t \cdot \prod_{i=1}^{2} (1-p_1)^t + \\
& \ldots,
\end{aligned}
\tag{8}
$$

where $p_i = p_1 d^{i-1}$.

We are currently not able to find an efficient possibility to maximise this complex function.

Least Squares. For the Least Squares method typically an estimate of the failure intensity is used and the relative error to the estimated failure intensity from the model is minimised. We use the estimate of the mean number of failures for this because it is the original part of the model. Therefore, the square function to be minimised in our case can be written as follows.

$$
S(p_1, d) = \sum_{j=1}^{m} [\ln r_j - \ln \mu(t_j; p_1, d)]^2, \tag{9}
$$

where m is the number of measurement points, r_j is the measured value for the cumulated failures, and t_j is the time at measurement j.

This function is minimised using the simplex variant of Nelder and Mead [10].

We found this method to be usable for our purpose. We first looked at two different approaches: (1) to use only the last sample data point and (2) to use all available data points. The motivation for (1) was to reduce the costly computation but finally we found that (2) was feasible and yields a better fit as can be seen, for example, in Fig. 3 for an example application from Siemens.

3 Evaluation

We describe several criteria that are used to assess the Fischer-Wagner model.

3.1 Criteria

The criteria that we use for the evaluation of the Fischer-Wagner model are derived from [6]. We assess according to five criteria, four of which can mainly be applied theoretically, whereas one criterion is based on practical applications of the models on real data. The first criterion is the *capability* of the model. It describes whether the model is able to yield important quantities. The criterion *quality of assumptions* is used to assess the plausibility of the assumptions behind the model. In what cases the model can

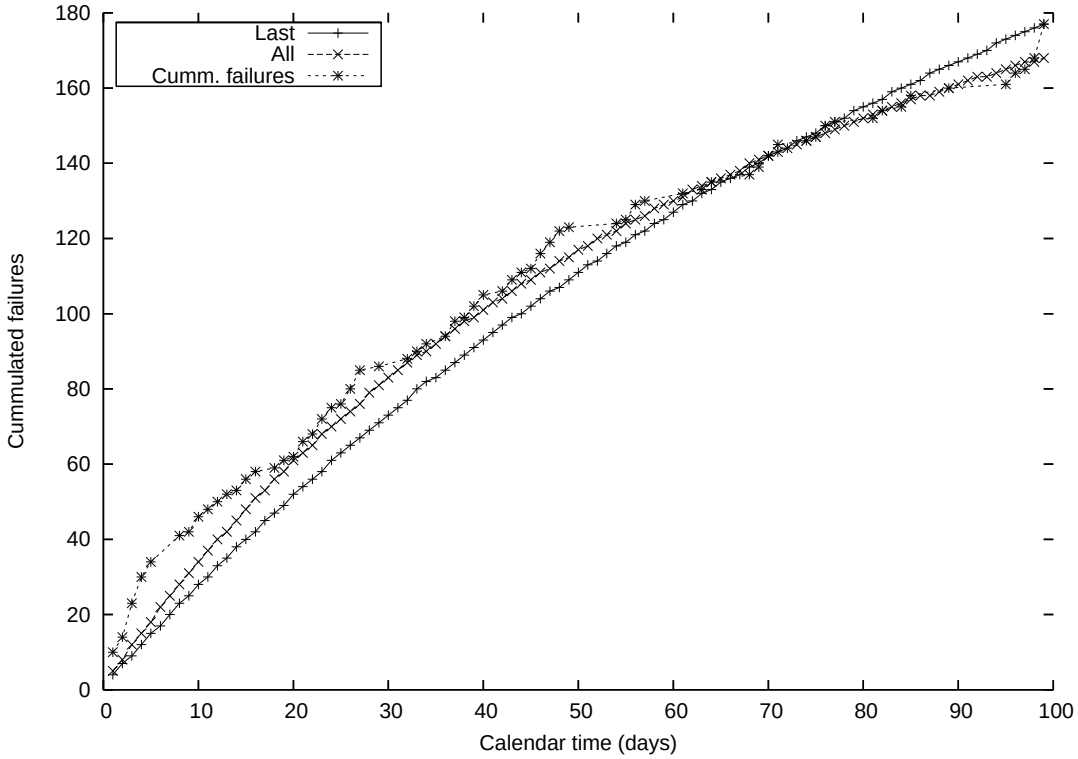

Figure 3: A comparison of the fit of the model when using only the last or all sample data points

be used is evaluated with the criterion *applicability*. Furthermore, *simplicity* is an important aspect for the understandability of the model. Finally, the *predictive validity* is assessed by applying the model to real failure data and comparing the deviation.

3.2 Capability

The main purpose of a reliability model is to aid managers and engineers in planning and managing software projects by estimating useful quantities about the software reliability and the reliability growth. Following [6] such quantities, in approximate order of importance, are:

1. current reliability
2. expected date of reaching a specified reliability
3. human and computer resource and cost requirements related to the achievement of the objective

Furthermore, it is a valuable part of a reliability model if it can predict quantities early in the development based on software metrics and/or historical project data.

The model yields the current reliability as current failure intensity and mean number of failures. It is also able to give predictions based on parameters from historical data. Furthermore, the expected date of reaching a specified reliability can be calculated. Human and computer resources are not explicitly incorporated. There is an explicit concept of time but it is not as sophisticated as, for example, in the Musa-Okumoto model [7].

3.3 Quality of Assumptions

As far as possible each assumption should be tested by real data. At least it should be possible to argument for the plausibility of the assumption based on theoretical knowledge and experience. Also the clarity and explicitness of the assumptions is important.

The main assumption in the Fischer-Wagner model is that the failure rates of the faults follow a geometric sequence. The intuition is that there are many faults with low failure rates and only a small number of faults with high failure rates. This is in accordance with software engineering experience and supported by [1]. Moreover, the geometric sequence as relationship between different faults was also found in a study of NASA and is documented in [9, 8].

Furthermore, an assumption is that the occurrence of a failure is geometrically distributed. The geometric distribution fits because it can describe independent events and we did originally not consider continuous time but discrete incidents.

Finally, the infinite number of faults makes sense when considering imperfect debugging, i.e., fault removal can introduce new faults or the old faults are not completely removed.

3.4 Applicability

It is important for a general reliability model to be applicable to software applications in different domains and of different size. Also different project environments or life cycle phases should be possible. There are four special situations identified in [6] that should be possible to handle.

1. software evolution
2. classification of severity of failures into different categories
3. ability to handle incomplete failure data with measurement uncertainties
4. operation of the same program on computers of different performance

All real applications of the Fischer-Wagner model have been in the telecommunications area. However, it was used for software of various sizes and complexities. Moreover, during the evaluation of the predictive validity we applied it also to other domains (see Sec. 3.6). The phase it is

used in is before and during the field trial. Software evolution is hence not explicitly incorporated. A classification of failures is possible but has not been used so far. Moreover, the performance of computers is not a strong issue in this domain.

3.5 Simplicity

A model should be simple enough to be able to use it in real project environments. This includes that it has to be simple to collect the necessary data, easy to understand the concepts and assumptions, and the model should be implementable in a tool.

The concepts are not difficult to understand but the model in total is rather complicated because it not only involves failures but also faults. Furthermore, for all these faults the failure is geometrically distributed but each with a different probability.

A main critic is also that the assumed infinite number of faults make the model difficult to handle. In practical applications of the model and when building a tool, an upper bound of the number of faults has to be introduced to be able to calculate model values. This actually introduces a third model parameter in some sense.

The two parameters, however, can be interpreted as physical quantities. p_1 is the failure probability of the most probable fault and d can be seen as a measure of system complexity.

3.6 Predictive Validity

The most important and "hardest" criterion for the evaluation of a reliability model is its predictive validity. A model has to be a faithful abstraction of the real failure process of the software and give valid estimations and predictions of the reliability. For this we follow again [6] and use the *number of failures approach*.

We assume that there have been q failures observed at the end of test time (or field trial time) t_q. We use the failure data up to $t_e (\leq t_q)$ to estimate the parameters of the mean number of failures $\mu(t)$. The substitution of the estimates of the parameters yields the estimate of the number of failures $\hat{\mu}(t_q)$. The estimate is compared with the actual number at q. This procedure is repeated with several t_es.

For a comparison we can plot the relative error $(\hat{\mu}(t_q) - q)/q$ against the normalised test time t_e/t_q. The error will approach 0 as t_e approaches t_q. If the points are positive, the model tends to overestimate and the other way round. Numbers closer to 0 imply a more accurate prediction and hence a better model.

We apply as comparison models four well-known models from the literature: Musa basic, Musa-Okumoto, Littlewood-Verall, and NHPP. All these models are implemented in the tool SMERFS [2] that was used to

calculate the necessary predictions. We describe each model in more detail in the following.

Musa basic. The Musa basic execution time model assumes that all faults are equally likely to occur, are independent of each other and are actually observed. The execution times between failures are modelled as piecewise exponentially distributed. The intensity function is proportional to the number of faults remaining in the program and the fault correction rate is proportional to the failure occurrence rate.

Musa-Okumoto. The Musa-Okumoto model, also called logarithmic Poisson execution time model, was first described in [7]. It also assumes that all faults are equally likely to occur and are independent of each other. The expected number of faults is a logarithmic function of time in this model and the failure intensity decreases exponentially with the expected failures experienced. Finally, the software will experience an infinite number of failures in infinite time.

Littlewood-Verall Bayesian. This model was proposed for the first time in [4]. The assumptions of the Littlewood-Verall Bayesian model are that successive times between failures are independent random variables each having an exponential distribution. the distribution for the i-th failure has a mean of $1/\lambda(i)$. The $\lambda(i)$s form a sequence of independent variables, each having a gamma distribution with the parameters α and $\phi(i)$. $\phi(i)$ has either the form: $\beta(0) + \beta(1) \cdot i$ (linear) or $\beta(0) + \beta(1) \cdot i^2$ (quadratic). We used the quadratic version of the model.

NHPP. Various models based on a non-homogeneous Poisson process are described in [11]. The particular model used also assumes that all faults are equally likely to occur and are independent of each other. The cumulative number of faults detected at any time follows a Poisson distribution with mean $m(t)$. That mean is such that the expected number of faults in any small time interval about t is proportional to the number of undetected faults at time t. The mean is assumed to be a bounded non-decreasing function with $m(t)$ approaching The expected total number of faults to be detected as the length of testing goes to infinity. It is possible to use NHPP on time-between-failure data as well as failure counts. We used the time-between-failure version in our evaluation.

We apply the reliability models to several different sets of data to compare the predictive validity. Three data sets are provided by *The Data & Analysis Center for Software* of the US-American Department of Defence, the other three projects were done at Siemens. The former are called *DACS* together with their system number, the latter were given the name *Siemens* and are consecutively numbered.

3.6.1 DACS 1

The first project data under consideration had the aim of developing a real time command and control application. The size of the software measured in delivered object code instructions is 21,700. 136 failures were observed during the system test. The system code of this project is 1. The data is based on execution time, therefore all models were easily applicable.

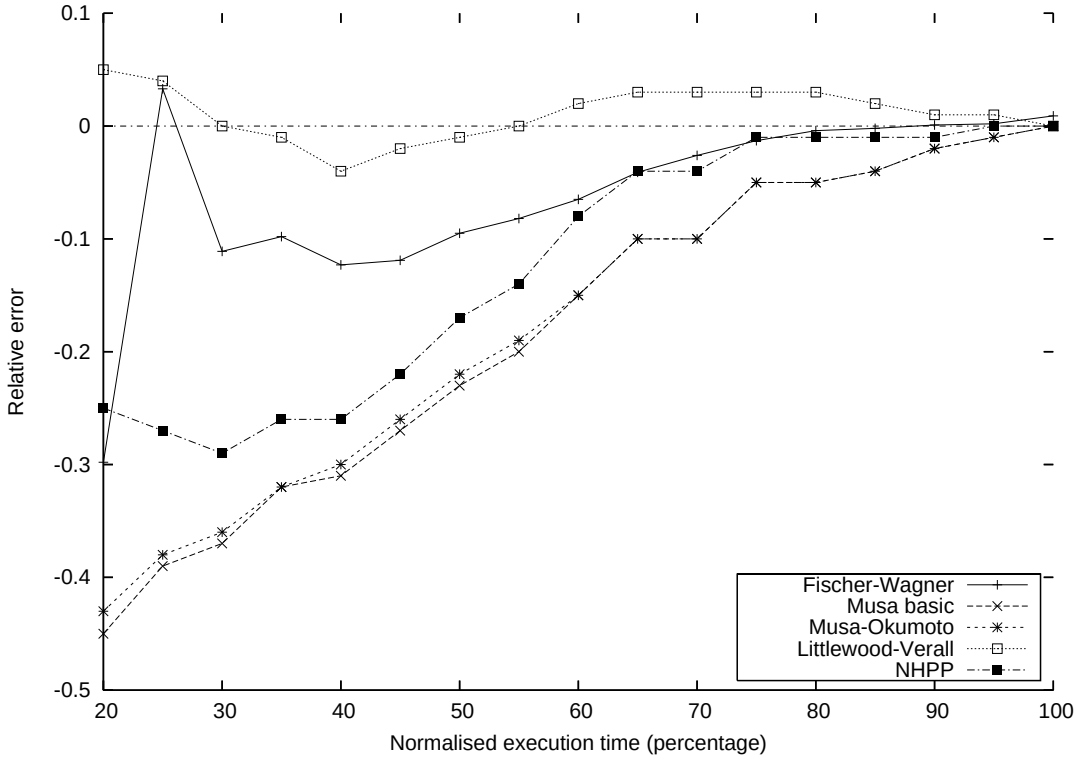

Figure 4: Relative error curves for the models based on the DACS 1 data set

The plot in Fig. 4 shows the relative error curves for all considered models. The predictions become more and more accurate as increasingly more sample data is available which was expected. The NHPP, Musa basic, and Musa-Okumoto models all tend to predict too little failures whereas the Littlewood-Verall model mostly overestimates the number of future failures. The latter model also yields the best predictions from early stages on. The Fischer-Wagner model has a quite similar predictive validity. From 80% of the time on the predictions are even more accurate than all the others although the predictions are in the beginning worse than the ones from the Littlewood-Verall model.

3.6.2 DACS 6

This data set comes from the subsystem test of a commercial subsystem with 5,700 delivered object code instructions. In total 73 failures occurred. We tried to use all models on this data as well but the models were not applicable at all stages. Especially the NHPP model was only able to make predictions for about half of the analysed parts of the data set. The results can be found in Fig. 5.

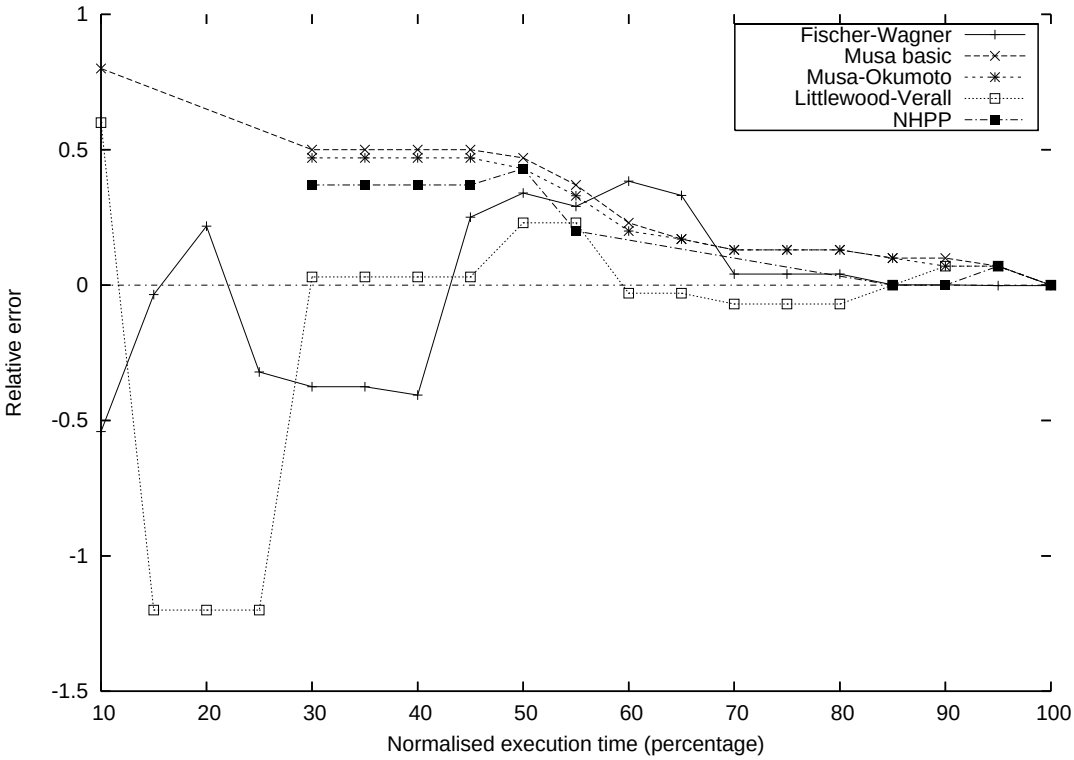

Figure 5: Relative error curves for the models based on the DACS 6 data set

It is obvious that the models behave strongly differently. Note that the Littlewood-Verall model gives the most accurate predication although it is the worst before 30% of the execution time is over. The Fischer-Wagner model is similar to the other models. Sometimes it is able to predict more accurately (between 45% and 55%, and after 70%), sometimes the predictions are worse (between 55% and 70%).

3.6.3 DACS 40

The DACS 40 test data describes the results of the system test of a military application with 180,000 delivered object code instructions. The Musa basic model was not applicable to this data. All the other models perform well

with relative errors not bigger than 0.25. The results are again illustrated in a plot in Fig. 6.

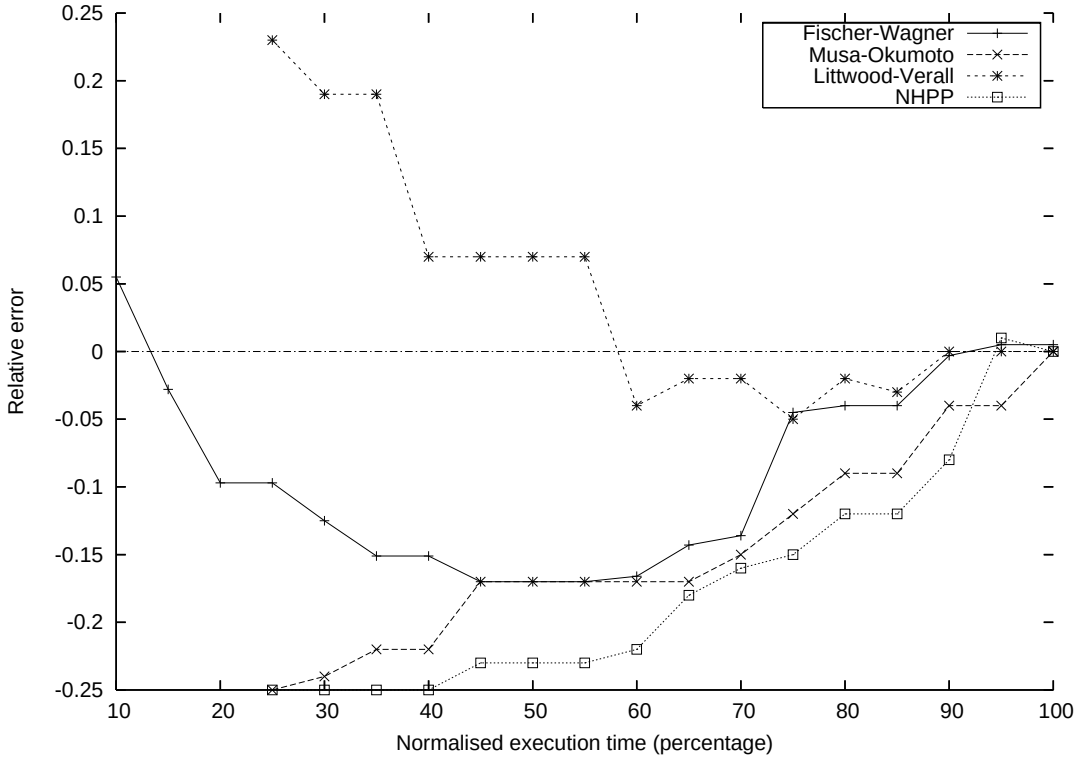

Figure 6: Relative error curves for the models based on the DACS 40 data set

For this data the Musa-Okumoto model is able to make the best predictions with an relative error of about 0.05 after 40% of the execution time. The Fischer-Wagner model is able to outperform the others in the early stages but is worse than the Musa-Okumoto model from 35% onwards. However, beginning at 75% the predictive validity is similar.

3.6.4 Siemens 1

This data comes from a large Siemens project that we call *Siemens 1* in the following. The software is used in a telecommunication equipment.

We only look at the field trial because this gives us a rather accurate approximation of the execution time which is the actually interesting measure regarding software. It is a good substitute because the usage is nearly constant during field trial. Based on the detailed dates of failure occurrence, we cumulated the data and converted it to time-between-failure (TBF) data. This was then used with the Fischer-Wagner, Musa-basic, Musa-Okumoto, and NHPP models. The results can be seen in Fig. 7. In this case we omit the Littlewood-Verall that made totally absurd predictions of over a thousand future failures.

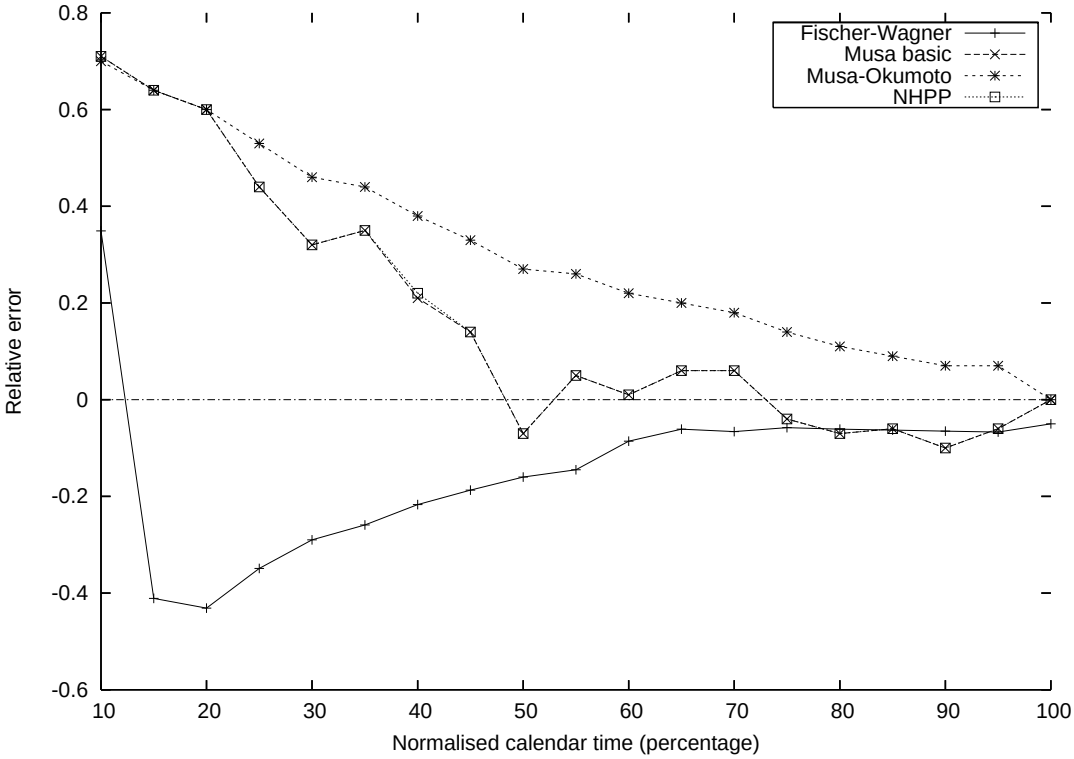

Figure 7: Relative error curves for the models based on the Siemens 1 data set

The Musa-basic and the NHPP models yield similar results all the time. They overestimate in the beginning and slightly underestimate in the end. The Musa-Okumoto model overestimates all the time, the Fischer-Wagner model underestimates. All models make again reasonably well predictions. The Fischer-Wagner model has a relative error below 0.2 from 45% on, the Musa basic and the NHPP models even from 40% on.

3.6.5 Siemens 2

Siemens 2 is a web application for which we only have a small number of field failures. This makes predictions more complicated as the sample size is smaller. However, it is interesting to analyse how the different models are able to cope with this. For this, we have plotted the results in Fig. 8.

Again not all models were applicable to this data set. The NHPP model only made predictions for a small number of data points, the Musa basic and the Musa-Okumoto models were usable mainly in the middle of the execution time. All models made comparably bad predictions as we expected because of the small sample size. Surprisingly, the Fischer-Wagner model did extremely well in the beginning but worsened in the middle until its prediction became accurate in the end again. Despite this bad performance in the middle of the execution time it is still the model

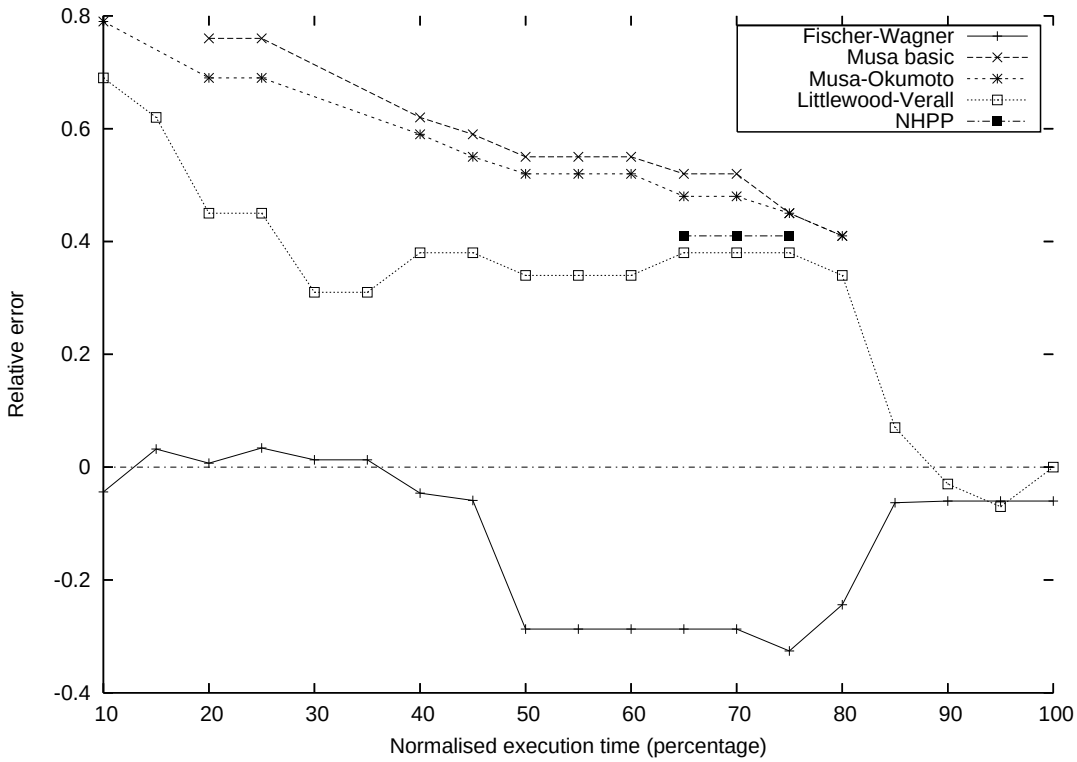

Figure 8: Relative error curves for the models based on the Siemens 2 data set

with the best predictive validity in this case. Only the Littlewood-Verall model comes close to these results. This might be an indication that the Fischer-Wagner model is good suited for smaller sample sizes.

3.6.6 Summary

For a better general comparison we combined the data into one plot which can be found in Fig. 9. This combination is straight-forward as we only considered relative time and relative errors. To avoid that strongly positive and strongly negative values combined give very small errors we use medians instead of average values. The plot shows that with regard to the analysed projects the Littlewood-Verall model gives very accurate predictions, also the NHPP and the Fischer-Wagner models are strong from early on.

However, for an accurate interpretation we have to note that the data of the Littlewood-Verall model for one of the Siemens projects was not incorporated into this comparison because its predictions were far off with a relative error of about 6. Therefore, the model has an extremely good predictive validity if it gives reasonable results but strongly weak predictions for some projects. A similar argument can be made for the NHPP model which made the weakest predictions for one of the DACS projects.

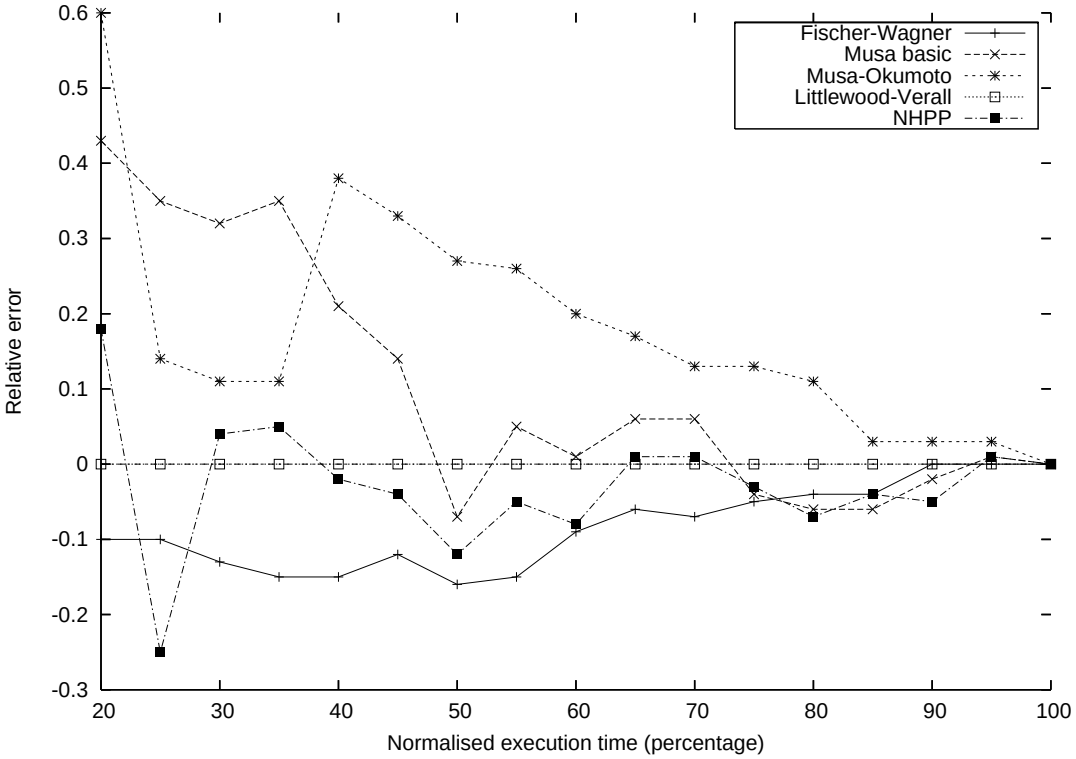

Figure 9: Median relative errors for the different models based on all analysed data sets

The Fischer-Wagner model cannot reach the validity of these models for particular projects but has a more constant performance over all projects.

4 Test Efficiency

One idea that comes out of being based on incidents as substitute of execution time is to measure the efficiency of the system test by the ratio of incidents per test case that were achieved. The underlying assumption is that system test is more efficient in revealing failures because special and border cases are analysed forcefully. Therefore each executed test case should be as good as several field incidents in finding a defect.

Formally, we define the ratio of incidents per test case as

$$\rho = \frac{i_{system}}{c_{system}}, \tag{10}$$

where i_{system} is the number of incidents that the system test represents and c_{system} is the actual and measured number of test cases that were executed during system test.

That this additional factor makes sense can be seen in Fig. 10 where we try to incorporate the system test data based on the assumption that one test case corresponds to one incident. Obviously, this gives extremely bad predictions during system test.

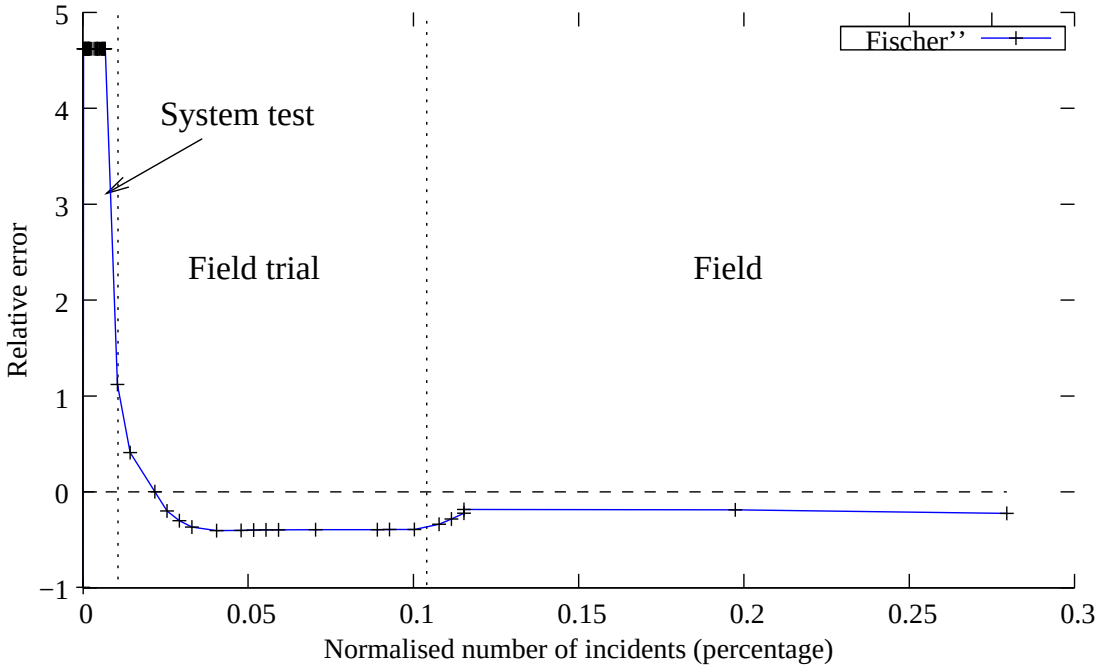

Figure 10: Relative error curve for the Fischer-Wagner model assuming one test corresponds to one incident for Siemens 2

A definitive value for ρ can only be determined after the system was released. Using field failure data, it can be estimated using a similar approach as for the other parameters (cf. Sec. 2.4). However, we now have to optimise for t as well. That means that we change the square function to the following.

$$S(p_1, d) = \sum_{j=1}^{m} [\ln r_j - \ln \sum_{a=1}^{\infty} 1 - (1 - p_a)^{i_{total}^{-\delta_j}}]^2, \qquad (11)$$

where m is the number of measurement points, r_j is the measured value for the cumulated failures, and δ_j is the distance of the current number to the total number of incidents before the new estimation. We need this distance to be able to handle more than one sample data point and still keep the distances and have only one parameter i_{total}. Note that this only works if the sample data comes from the field trial and field otherwise ρ would take effect.

Having this t_j we can determine ρ by interpreting t_j as the total number of incidents i_{total}. Then we have to subtract the number of incidents in the field i_{field} and in the field trial i_{ft} to get the incidents in the system

test. We divide this number by the number of test cases executed in the system test c_{system} to get ρ. The other way round means that we can calculate the total number of incidents with these figures.

$$i_{total} = i_{field} + i_{ft} + c_{system} \cdot \rho \tag{12}$$

Further investigation in this direction has to be done and the optimisation has to be implemented to analyse the improvement in the predictive validity during system test.

5 Conclusions

We conclude with a summary of our investigations and give some directions for future work.

Summary. We propose a software reliability model that is based on a geometric series of the failure rates of faults. This basis is suggested from the theory by Miller in [5] as well from practice from Nagel in [9, 8] and from the experience in Siemens projects.

We give the model a state-of-the-art parameter determination approach and also made a prototype implementation of it. Several data sets from DACS and Siemens were used to evaluate the predictive validity of the model in comparison to well-established models. We find that the Fischer-Wagner model often has a similar predictive validity as the comparison models and outperforms most of them. However, there is typically one of the models that performs better than ours.

Future Work. Another possibility would be to base the early estimation of the parameters on other system parameters. For example the parameter d of the model is supposed to represent the complexity of the system. Therefore, one or more complexity metrics of the software code could be used for the prediction. This needs extensive empirical analysis but could improve the estimation in the early phases significantly.

Furthermore, a time component that also take the uncertainty into account would be most accurate. The Musa basic and Musa-Okumoto models were given such components (see [6]). They model the usage as a random process and give estimates about the corresponding calendar time to an execution time.

The test efficiency approach has to be implemented and applied to several case study to judge its usefulness. Finally, we also plan to use the model in an economics models for software quality [12].

Acknowledgments

We are grateful to Christine Dietrich and Lothar Quoll for the help in gathering the needed data.

References

[1] E.N. Adams. Optimizing Preventive Service of Software Products. *IBM Journal of Research and Development*, 28(1):2–14, 1984.

[2] W.H. Farr and O.D. Smith. Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide. Technical Report NAVSWC TR-84-373, Naval Surface Weapons Center, 1993.

[3] Z. Jelinski and P.B. Moranda. Software Reliability Research. In W. Freiberger, editor, *Statistical Computer Performance Evaluation*. Academic Press, 1972.

[4] B. Littlewood and J.L. Verall. A Bayesian Reliability Growth Model for Computer Software. *Applied Statistics*, 22(3):332–346, 1973.

[5] D.R. Miller. Exponential Order Statistic Models of Software Reliability Growth. *IEEE. Trans. Software Eng.*, 12(1):12–24, 1986.

[6] J.D. Musa, A. Iannino, and K. Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, 1987.

[7] J.D. Musa and K. Okumoto. A Logarithmic Poisson Execution Time Model for Software Reliability Measurement. In *Proc. Seventh International Conference on Software Engineering (ICSE'84)*, pages 230–238, 1984.

[8] P.M. Nagel, F.W. Scholz, and J.A. Skrivan. Software reliability: Additional investigations into modeling with replicated experiments. NASA Contractor Rep. 172378, NASA Langley Res. Center, Jun. 1984.

[9] P.M. Nagel and J.A. Skrivan. Software reliability: Repetitive run experimentation and modeling. NASA Contractor Rep. 165836, NASA Langley Res. Center, Feb. 1982.

[10] J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[11] Hoang Pham. *Software Reliability*. Springer, 2000.

[12] Stefan Wagner and Tilman Seifert. Software quality economics for defect-detection techniques using failure prediction. *SIGSOFT Software Engineering Notes*, 30(4), 2005.