Systematic vs. Operational Testing: The Necessity for Different Failure Models[*]

Michael Grottke
Lehrstuhl für Statistik und Ökonometrie
Universität Erlangen-Nürnberg
Lange Gasse 20
90403 Nürnberg

Klaudia Dussa-Zieger
imbus AG
Kleinseebacher Str. 9
91096 Möhrendorf

Contact: `Michael.Grottke@wiso.uni-erlangen.de`

Abstract

Software Reliability Growth Models assume that the application under test (AUT) is tested according to its operational profile. However, examining the test practices at different industrial test laboratories this is often not the case. Instead, a systematic approach to testing is taken. The differences in systematic versus operational testing, in particular their effect on the failure count model, is the topic of this paper. After a detailed discussion of the advantages and disadvantages of systematic and operational testing, the hypergeometric failure model by Rivers and Vouk is presented. In our work problems in the original formulation of the model are resolved and the derivation of the continuous form is more stringent. A different approach to the interpretation of testing efficiency is taken. The model is applied to data collected during a test project at imbus AG and the results of the failure prediction are presented. The paper concludes with an outlook on further research activities.

1 Systematic vs. operational testing

One possible way of testing software is using it according to its operational profile. This means that the operations - major system tasks of short duration with processing substantially different from other operations [9] - of the application under test (AUT) have to be determined together with their execution probabilities. Based on these probabilities the number of test cases is allocated to the different program functions. The testers should execute the specified test runs in random order, i.e. the different parts of the software should not be tested in strict succession. This method has two important advantages:

1. Since the operations that will be used frequently by the customers are tested more thoroughly, more failures will occur in these parts of the software during testing. Therefore, the faults detected and removed are those that contribute most to the hazard rate of the program (i.e. the probability of its instantaneous failure) perceived by the users [8].

2. If usage during testing does not differ from usage after release of the software, the reliability of the software product observed by the testers can directly be used as an estimate of the reliability the customers would perceive if testing was stopped right away [3]. This means that in addition to the detection of faults, operational testing also serves for the assessment of the current reliability of the piece of software.

Critics of operational testing feel that it puts too much effort in assessing reliability and that it therefore is not efficient enough with respect to improving it. For them testing is to be regarded as a destructive process with the goal of finding inputs that cause the software to fail and of identifying the bugs that are responsible for them. In general, a tester is the better, the more software faults he detects[**]. (Of course, the faults found should include those that would matter most to customers in terms of impact and frequency.) Many systematic or directed testing methods have been proposed for improving testing efficiency. “Partition testing”, for example, tries to reduce redundancy in the test cases by grouping those inputs for which computation

[*]This work was supported by the European Community in the framework of the specific programme for research, technological development and demonstration on a user friendly society (1998-2002), the “IST Programme”, Contract No. IST-1999-55017. The authors are solely responsible for this paper. It does not represent the opinion of the Community, and the Community is not responsible for any use that might be made of data appearing herein.
This article was published in: Proc. 5th Conference on Quality Engineering in Software Technology, Nuremberg, 2001, pp. 59 - 68.
[**]The terms “detecting a fault”, “identifying a bug” and “observing a failure” are used synonymously in this paper.

is expected to be equivalent. If this clustering was done perfectly, executing one unit drawn randomly out of each partition would be enough for covering the entire program code. In most cases, the partitions are derived from the requirements specification of the software product. Moreover, the test designers make use of their knowledge about typical errors of programmers and of their intuition for creating the clusters to be tested. E.g., the boundaries of equivalence classes of input variables may be treated as additional partitions, because faulty predicates in the code might cause a computation which is different from the computation for the other values in the equivalence class [11]. The critics of operational testing think that their systematic methodology is superior for a number of reasons:

1. Testing according to the operational profile means accepting a certain redundancy in the test cases. Even if no function is executed with the same input values for a second time, repeated runs may well result in similar computations, i.e. the execution of the same statements or the same paths of the code. The probability of the occurrence of a new, unknown failure can be expected to diminish as redundancy increases. Therefore, the efficiency of operational testing decreases as testing proceeds, because more and more parts of the software code have already been covered before [7]. Testing then may help to assess the current reliability and gain confidence in a certain quality of the software product, but it contributes little to its improvement.

2. The random order of executed test cases further reduces the efficiency of testing, because it requires more navigation between different parts of the program. Sometimes before one operation can be executed additional operations have to be run to ascertain that preconditions are met. The clever ordering of the set of operations to be tested can avoid many of such preparing actions to be necessary, because preconditions may be fulfilled with the help of other test cases that have to be executed anyway.

3. It is often argued that testing according to an operational profile means neglecting critical operations that have a low occurrence probability. With bad luck these operations might not be tested at all. Researchers who defend operational testing therefore propose that rare, critical new operations should intentionally be given more weight than their expected relative frequencies [9]. While this overcomes the problem stated, it is at the same time a partial retreat from the operational testing paradigm.

4. While for some types of software systems (like systems in the telecommunications industry) operational profiles can easily be obtained by logging the usage information, for many other types their determination requires a significant amount of effort. It is no coincidence that the methodology of operational testing was developed at AT&T [8].

5. For new applications or new functions historical information is not available at all. In these cases, the derivation of the operational profile can at best be guesswork. Moreover, the way in which software is used is not necessarily static, but it may change over time. A flawed operational profile will misguide the allocation of testing effort and cause reliability estimates differing from the quality actually perceived by the users.

6. When testing interactive or real time systems the relative frequencies not only of different inputs but also of input sequences have to be known or estimated for building an operational profile [5].

7. For program units an operational profile often cannot be determined at all, especially if a module is to be reused in different programs [5]. This means that operational testing cannot be used at the unit testing stage but only later in the software development process.

8. It is often difficult to define operations that are tasks of short duration and still do not depend on the results of other operations as preconditions. If this cannot be achieved, the necessary ordering reduces the randomness in the sequence of test cases and may result in correlations between the failures observed, which is not accounted for by most software reliability models.

Several studies comparing the effectiveness of operational and systematic testing have been undertaken. Duran and Ntafos [2] and Hamlet and Taylor [6] run simulations for evaluating the probability of at least one failure occurrence under the different testing regimes. They conclude that systematic testing may be less favorable than intuition suggests. However, its advantage increases if the strategy leads to more sampling in regions of the code where faults are likely to be located [6]. Identifying these domains rather than trying to attain extensive coverage seems to be a promising strategy. This is corroborated by a recent simulation study stating that gains in code coverage have little additional impact on defect coverage when accounting for the common influence of testing intensity [1]. An analytical approach [4] examining the effects of the testing regime on the delivered reliability shows that scenarios can be constructed in which operational testing is superior to systematic testing and vice versa. Which testing method should be chosen may even depend on the budget in terms of the number of test cases that can be executed. Moreover, a systematic strategy may find more faults and still yield a lower reliability if the faults detected had a low probability of activation. The bottom line is that whereas operational testing is implicitly guided by the operational profile to detect the faults with the highest hazard rates, systematic testing depends on the intuition and experience of the testers and of those who generate the test specification. If they succeed in isolating the regions in which faults with high operational hazard rates lurk, systematic testing is superior to operational testing.

Even though there is no definite and unconditional advantage of systematic testing, most practitioners seem to think so. In fact, only a small minority of industrial organizations employs operational testing [4]. For this minority there exist software reliability models in abundance [3, 10, 13, 18]. These models typically assume that the program hazard rate tends to decrease at a slower and slower rate. This reflects the improvement in the reliability of the AUT, because the probability that the software will fail in an interval of defined length gets lower. Therefore, these models are referred to as "Software Reliability Growth Models" (SRGMs). There are two main reasons why the typical behavior of SRGMs seems to be appropriate for fitting failure data collected during operational testing:

1. When the AUT is tested according to its operational profile the different parts of the AUT are not used consecutively in strict order. Instead, it can be expected that those operations with high occurrence probabilities are tested first - no matter to which functional area of the AUT they belong. Therefore, the faults with high individual hazard rates are among the first ones to be detected. As testing proceeds, the decreases in the program hazard rate become smaller, because the faults remaining tend to have lower individual activation probabilities than the faults removed.

2. Since repeated execution of operations with input values belonging to the same partition is possible under operational testing, redundancy in the test cases increases as testing proceeds: The more faults have already been corrected, the longer it takes until a part of the program code still containing an undetected bug is executed. As a consequence, the time between the discontinuous decreases in the program hazard rate increases on average.

It is important to realize that the assumptions of most SRGMs are linked to characteristics of operational testing. This means that their unreflected application to data collected during systematic testing can hardly yield trustworthy results. However, only few publications directly address the problem of assessing software quality and testing efficiency during systematic testing. Achievements made in this field are of high practical importance.

2 The hypergeometric model for systematic testing

The hypergeometric distribution was applied to the field of software validation as early 1970 by Mills in an unpublished paper. However, his setup was a capture-recapture sampling, either in the form of error seeding or in the form of a two-stage edit procedure [17]. Tohma et al. later introduced an extended model for n-stage debugging [19]. In both models, program faults are drawn from the same population at each stage, which implies that the parts of the AUT already covered before are possibly tested again.

Building on earlier work [15, 20], Rivers and Vouk [14, 16], derive a hypergeometric density for the number of failures experienced at each stage of testing when the constructs tested are removed from the population, i.e. when there is no retest of constructs covered before. The constructs under study may be program size metrics like statements, basic blocks, paths, c-uses, p-uses, etc. Rivers and Vouk also consider the number of test cases executed and even execution time or calendar time. The probability that the random variable ΔM_i - the number of faults detected at the i^{th} stage of testing - takes value Δm_i, given that m_{i-1} faults have been discovered at the first $(i-1)$ stages, is

$$P(\Delta M_i = \Delta m_i \mid m_{i-1}) = \frac{\binom{g_i(N-m_{i-1})}{\Delta m_i}\binom{[K-Q_{i-1}]-[g_i(N-m_{i-1})]}{\Delta Q_i - \Delta m_i}}{\binom{K-Q_{i-1}}{\Delta Q_i}} \tag{1}$$

for $\Delta m_i \in \mathbb{N}$ and $\Delta m_i \leq \Delta Q_i$.

In this equation, N stands for the number of faults inherent in the software at the beginning of testing, and K is the number of constructs that are planned to be covered until the end of testing. Q_{i-1} constructs already have been covered at the first $(i-1)$ stages of testing, and ΔQ_i is the number of constructs covered at the i^{th} stage. $g_i := g(\frac{Q_i}{K})$ is a factor accounting for the "visibility" of the $N - m_{i-1}$ faults still remaining in the code. On the one hand, it is influenced by the structure of the code and by the constructs chosen: For example, the execution of several different paths may lead to detection of the same fault, whereas other faults may only be activated if a path is taken more often than just once. On the other hand, g_i may also account for fluctuations in the testers' effectiveness, e.g. due to learning. Therefore, g_i is referred to as "testing efficiency". According to the equation, the probability of finding Δm_i faults is the product of the number of ways in which these faults can be chosen out of $g_i(N - m_{i-1})$ remaining visible faults and the number of ways in which the other $\Delta Q_i - \Delta m_i$ constructs covered can be chosen out of the $[K - Q_{i-1}] - [g_i(N - m_{i-1})]$ constructs that neither have been covered before nor are faulty; this value is divided by the number of ways in which the ΔQ_i constructs can be picked out of the $K - Q_{i-1}$ uncovered statements.

The expected number of faults discovered at stage i given that m_{i-1} faults had been detected before is [12]

$$E(\Delta M_i \mid m_{i-1}) = g_i(N - m_{i-1})\frac{\Delta Q_i}{K - Q_{i-1}} = g_i(N - m_{i-1})\frac{\Delta q_i}{1 - q_{i-1}}, \tag{2}$$

where $\Delta q_i = \frac{\Delta Q_i}{K}$ is the fraction of constructs covered at stage i and $q_{i-1} = \frac{Q_{i-1}}{K}$ is the fraction of constructs covered before that stage. The unconditional expected value of ΔM_i is given by

$$E(\Delta M_i) = g_i\left[N - E(M_{i-1})\right]\frac{\Delta q_i}{1 - q_{i-1}}. \tag{3}$$

There seem to be two shortcomings of Rivers' and Vouk's formulation of the problem. First of all, the probability mass of the hypergeometric distribution (1) is distributed among the integer values from zero to ΔQ_i. It is not possible to find more faults than the number of constructs covered. While this may be no problem if the constructs measured are p-uses or some other metric dividing the program in a lot of small subdomains, it is unrealistic and restricting if the constructs studied are highly aggregated like test cases. Let us assume that each metric for which $\Delta m_i > \Delta Q_i$ seems possible can theoretically be replaced by another, more precise metric with $K^* \gg K$ being the number of constructs in the program. If for each stage its relative contribution to overall coverage according to the 'new' metric was the same as its relative contribution according to the 'old' metric, then $\frac{\Delta Q_i^*}{K^*} = \frac{\Delta Q_i}{K} = \Delta q_i \; \forall i$ and - as a consequence - $\frac{Q_i^*}{K^*} = \frac{Q_i}{K} = q_i \; \forall i$. (This comes close to stating that the different constructs of the 'old' metric are equally sized.) Substituting K^* for K and Q_i^* for Q_i in (1) clearly yields an altered probability distribution. However, its expected value is

$$\begin{aligned} E(\Delta M_i) &= g_i\left[N - E(M_{i-1})\right]\frac{\Delta Q_i^*}{K^* - Q_{i-1}^*} = g_i\left[N - E(M_{i-1})\right]\frac{\frac{\Delta Q_i^*}{K^*}}{1 - \frac{Q_{i-1}^*}{K^*}} = \\ &= g_i\left[N - E(M_{i-1})\right]\frac{\Delta q_i}{1 - q_{i-1}}, \end{aligned} \tag{4}$$

just the same as equation (3).

A second restriction of (1) is that the testing efficiency g_i has to be chosen such that $g_i(N - m_{i-1})$ is an integer value. Therefore, g_i cannot be assigned any desired continuous function of test coverage. We therefore propose to replace $g_i(N - m_{i-1})$ in (1) by a discrete random variable X_i with probability function $f(x_i) = 0 \; \forall \; x_i \notin \mathbb{N}$ and $E(X_i) = g_i(N - m_{i-1}) \; \forall i$. The resulting equation

$$P(\Delta M_i = \Delta m_i \mid m_{i-1}) = \frac{\begin{pmatrix} X_i \\ \Delta m_i \end{pmatrix}\begin{pmatrix} [K^* - Q_{i-1}^*] - X_i \\ \Delta Q_i^* - \Delta m_i \end{pmatrix}}{\begin{pmatrix} K^* - Q_{i-1}^* \\ \Delta Q_i^* \end{pmatrix}} \tag{5}$$

still has expected value (4).

Equation (4) can be used as a starting-point for deriving the mean value function $\mu(q)$, i.e. the expected cumulative number of faults found when a coverage of $100 \cdot q$ per cent is achieved, because $E(\Delta M_i) = \Delta E(M_i) = \Delta \mu_i$ [14].

Since all Q_is as well as all Q_i^*s have to take integer values, (3) and (4) are only defined for Δq_i and q_{i-1} being a multiple of K^{-1} or K^{*-1}, respectively. However, for $K^* \gg K$ or $K^* \to \infty$ the change in coverage attained at any stage may approach zero, because $\frac{1}{K^*} \to 0$. At the limit, the coverage measure q can take any value between zero and one.

It is therefore possible to transform the difference equation (4) into a differential equation by considering $\Delta q_i \to dq_i \to 0$ and - as a consequence - $q_i \to q_{i-1}$ and $\mu_i \to \mu_{i-1}$. For any coverage value $q \in (0,1)$,

$$d\mu(q) = g(q)(N - \mu(q))\frac{dq}{1 - q}. \tag{6}$$

From this equation a continuous approximation of the mean value function can be obtained [14]:

$$\mu(q) = N - (N - \mu_{min})\exp\left(-\int\limits_{q_{min}}^{q}\frac{g(\zeta)}{1 - \zeta}d\zeta\right). \tag{7}$$

q_{min} and μ_{min} denote the minimum meaningful coverage and the expected number of failures observed for this coverage, respectively, and are introduced for stabilization of the model. Rivers considers different functional forms of testing efficiency $g(q)$ and derives the mean value function for each of these special cases [14].

Assuming a constant testing efficiency (TE), i.e. setting $g(q) \equiv \alpha$ yields

$$\mu(q) = N - (N - \mu_{min}) \left(\frac{1-q}{1-q_{min}} \right)^{\alpha}, \tag{8}$$

whereas using the linear TE $g(q) = \alpha(1-q)$ results in

$$\mu(q) = N - (N - \mu_{min}) \exp\left(-\alpha(q - q_{min})\right). \tag{9}$$

In the case of the unimodal TE $g(q) = \left(\alpha\beta q^{\beta-1}\right)(1-q)$, the following mean value function is obtained:

$$\mu(q) = N - (N - \mu_{min}) \exp\left(-\alpha(q^{\beta} - q_{min}^{\beta})\right). \tag{10}$$

For a real data set of coverage rates and the respective cumulative numbers of errors found, the parameter values of (8) to (10) - N, α and β - can be estimated by searching for those values that minimize the sum of squared errors (SSE) of the model.

The different functional forms for $g(q)$ do not only allow us to choose from a variety of models the one which attains the best fit for the data measured in a testing project. The estimated function $g(q)$ itself may be subject of discussion, because it reveals information about the testing process. In section 3, we will participate at the debate about what conclusions to draw from the estimated testing efficiency models.

In order to get a quick impression of the shape of the testing efficiency, an empirical estimate of $g(q)$ can be calculated at the coverage rates attained after each stage of testing, q_i. The formula for $\hat{g}(q_i)$ is derived from equation (4) by substituting the number of faults actually found before and at the i^{th} stage of testing, m_{i-1} and Δm_i, for their respective expected value:

$$\hat{g}(q_i) = \frac{\left(\frac{\Delta m_i}{N - m_{i-1}}\right)}{\left(\frac{\Delta q_i}{1 - q_{i-1}}\right)}. \tag{11}$$

If N is not known - which should practically always be the case -, it has to be estimated, for example by fitting one of the mean value functions (8) to (10) to the data set.

3 Application of the model

The models (8) to (10) have been applied to data collected during a testing project at imbus AG. The AUT was a commercial hardware configuration and driver software. For data logging, a test plan management tool keeping a tree representation of the test specification was used. For each test case, the time intervals in which the AUT was being used according to its specification were collected together with the times of failure occurrences. The way in which this information is stored facilitates the analysis from different angles, e.g. in the time domain, with respect to overall test case coverage, separately for different chapters of the test specification, etc.

Figure 1 shows the cumulative number of failures observed for different test case coverage levels q_i. The data have been grouped in steps of 2 per cent of the number of test cases planned. At the end of the first test cycle, about 79 per cent of these test cases had been executed, and 66 failures had occurred.

The three models (8) to (10) were fitted to the data; the values of the first data point of the grouped data, (0.02; 3), were used as q_{min} and μ_{min}, respectively. The graphical results are also depicted in figure 1. For each of the models, the estimated values of N, α and - where it applies - β as well as the SSE and the estimated number of failures observed at full coverage are documented in table 1.

Model fitted	$\hat{N}$	$\hat{\alpha}$	$\hat{\beta}$	SSE	$\hat{\mu}(1.0)$
Constant TE model	69.13	1.476	-	236.53	69.13
Linear TE model	131.92	0.806	-	222.15	73.54
Unimodal TE model	165.27	0.593	0.948	221.39	74.46

Table 1: *Results of fitting the different models to the data set*

The fitted functions are quite similar, although the SSE for the linear TE model and the unimodal TE model are smaller than the SSE for the constant TE model. Since there are three parameters in the unimodal TE

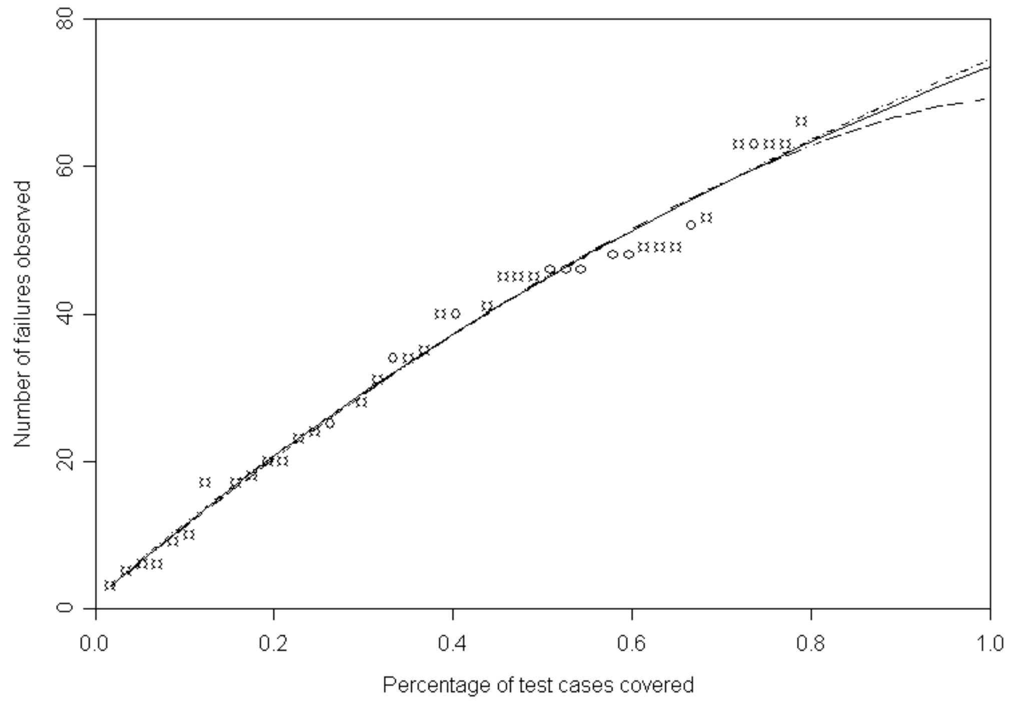

Figure 1: *Grouped data (o), fitted constant (- -), linear (–) and unimodal (·-) TE model*

model, it is not amazing that it is the model yielding the best fit. The estimated number of failures yet to be observed also does not differ much for the three models. While the constant TE model expects about three additional failures, according to the other models there are about eight more failures to be experienced until full coverage has been attained. Interestingly, the models assuming that the testers are doing worse at the end of the project (because the testing efficiency decreases to zero at a coverage level of 1.0) predict a higher number of faults detected before the end of testing. However, there is a notable discrepancy in the models with respect to the estimated number of faults present in the software at the end of testing. Since $\hat{\mu}(1.0) = \hat{N}$ is a property of the constant TE model, it always predicts that all faults inherent in the software when testing starts will have been found at full coverage - regardless of the estimated constant testing efficiency $\hat{\alpha}$! For $\hat{\alpha} = 1$ the mean value function is a linear function of test coverage. The better the testing efficiency $\hat{\alpha} > 1$, the more the slope of $\hat{\mu}(q)$ will decrease when q gets larger, because exorbitantly many faults have already been removed earlier. The worse the testing efficiency $\hat{\alpha} < 1$, the more the slope of $\hat{\mu}(q)$ will rise at the end of testing, i.e. more and more faults will be found per test case. Especially for $\hat{\alpha}$ values far from one, this effect of a constant testing efficiency is clearly counterintuitive. In our case, with $\hat{\alpha} = 1.476$, the saturation in the mean value function is still moderate.

As for the other two models, both $\exp\left(-\hat{\alpha}(1.0 - q_{min})\right)$ and $\exp\left(-\hat{\alpha}(1.0^{\hat{\beta}} - q_{min}^{\hat{\beta}})\right)$ can only reach zero for $\hat{\alpha} \to \infty$. For practically each project the models predict that a number of faults will never be detected. Even though the number of failures to be experienced according to the linear TE model and the unimodal TE model is quite similar to the result of the constant TE model for our data set, the two models suggest that at the end of testing about 58 or 91 faults will remain in the software, respectively.

Why the fitting of these two models obtains high estimates for N and low estimates for α - i.e. a large number of inherent faults at the beginning of testing, a lot of which are never detected due to the low level of testing efficiency - can be realized by comparing diagrams of the empirical and the estimated testing efficiency for each of the three models (figures 2 to 4).

Keep in mind that - unless N has actually been observed (e.g. because a post-mortem analysis is carried out after the software has been used for a long time) - the number of faults in equation (11) has to be estimated. Therefore, each model can influence the empirical testing efficiency. If $\hat{N}$ is large relatively to the number of failures already observed, then the factor $(\hat{N} - M_{i-1})^{-1}$ does not show much of a change from the beginning of testing to the end of testing. In this case, the dampening effect of the (as coverage increases with Δq_i remaining constant) ever-increasing denominator in equation (11) is not or only partially offset by the diminishing values of $\hat{N} - M_{i-1}$. This means that not enough account is taken of the fact that not only the

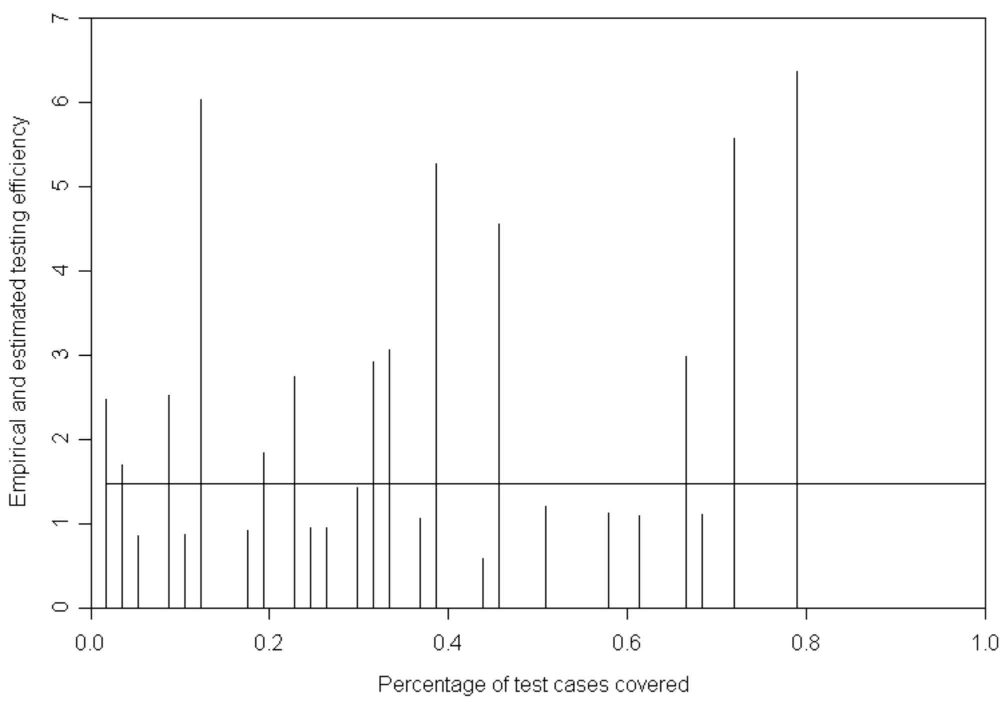

Figure 2: *Empirical (|) and estimated (—) testing efficiency for the constant TE model*

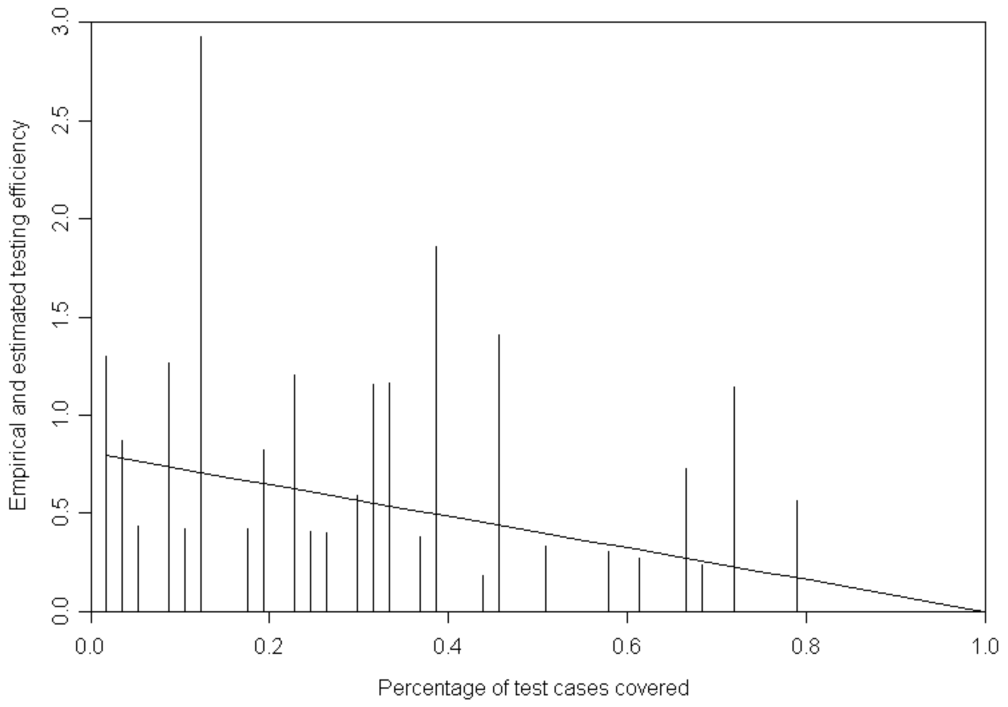

Figure 3: *Empirical (|) and estimated (—) testing efficiency for the linear TE model*

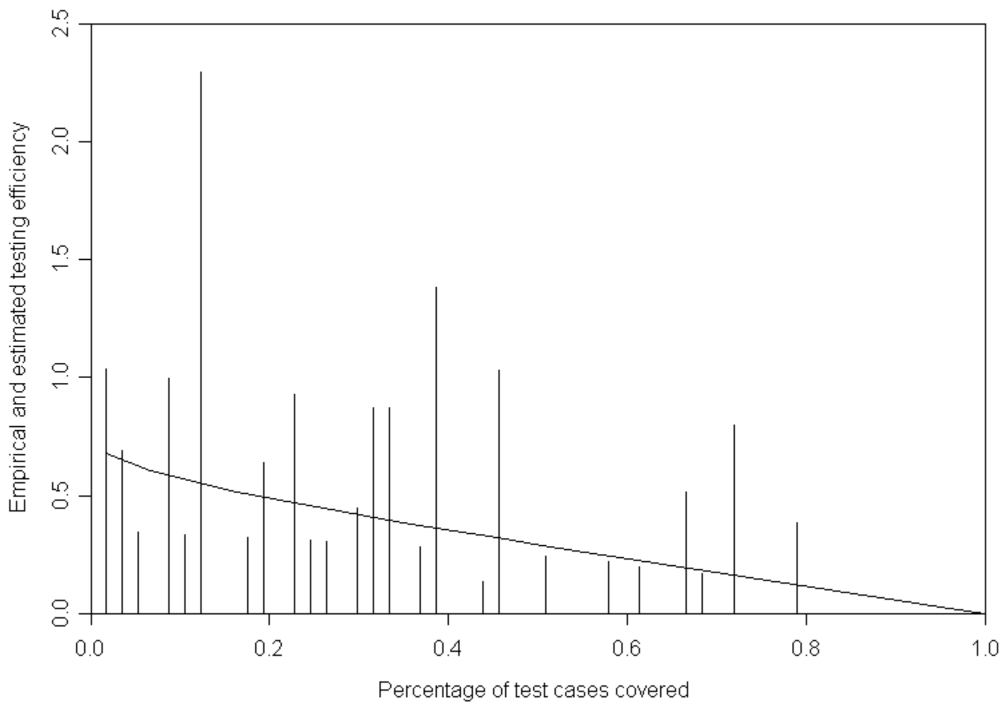

Figure 4: *Empirical (|) and estimated (—) testing efficiency for the unimodal TE model*

number of constructs already covered before increases, but the number of faults to be detected decreases at the same time. Whereas the first effect makes testing easier insofar as many of the faults remaining can be expected to be in the small region not yet covered (this could be dubbed an axiom of systematic testing), the second effect lets finding the same number of faults at a late stage of testing as at an early stage appear to be a bigger accomplishment, because of the reduced number of faults that possibly can be detected. The high estimated number of inherent faults for the linear TE model and the unimodal TE model - about twice as much as the number of faults to be discovered until the end of testing! - seems to be an artifact of the decreasing shape of the empirical testing efficiency obtained for large values of $\hat{N}$ - cf. figures 3 and 4.

Rivers argues that a constant testing efficiency means that there is not enough learning within a testing project. Testers are not able to or not allowed to run test cases departing from the test specification and therefore cannot use the experience made to improve their effectiveness [14]. We disagree with this interpretation. A constant testing efficiency, i.e. the same percentage of remaining faults detected for the same percentage of previously untested constructs covered can be seen as an indicator for a good test specification if the “constructs” used as a measure of completeness are test cases derived in a black-box fashion, e.g. according to the methodology of partition testing.

There is the risk that the person who specifies the test cases does a bad job in identifying the partitions of the program, i.e. test cases which he assumes to cover previously uncovered constructs of the code (like blocks or branches) mostly execute constructs that have already been tested before. Such test cases can be expected to detect less new faults than test cases covering more untested constructs. Obviously the potential redundancy in test cases increases as testing proceeds, because less and less constructs remain to be tested. Therefore, one may expect the testing efficiency to decrease in time. However, if there is only a slight decrease in the number of faults detected per test case this seems to indicate that only small regions of the subdomains covered by the test cases overlap. With respect to this the data of the project studied here suggest a good quality of the test specification.

The fault detection capability of the test specification, which is captured by Rivers’ and Vouk’s “testing efficiency”, can be separated from a different effect being a result of systematic testing guidelines. As noted earlier, one of the advantages of directed testing over operational testing is the possibility of a clever ordering of test cases. This may reduce the necessary testing effort, because one test case can be used to create or ascertain the pre-conditions of another one, and navigation between the different parts of the AUT can be reduced. The further testing has proceeded, the more test cases can draw on actions executed earlier. Furthermore, as the testers gain experience in using the AUT they become faster in following the test specification:

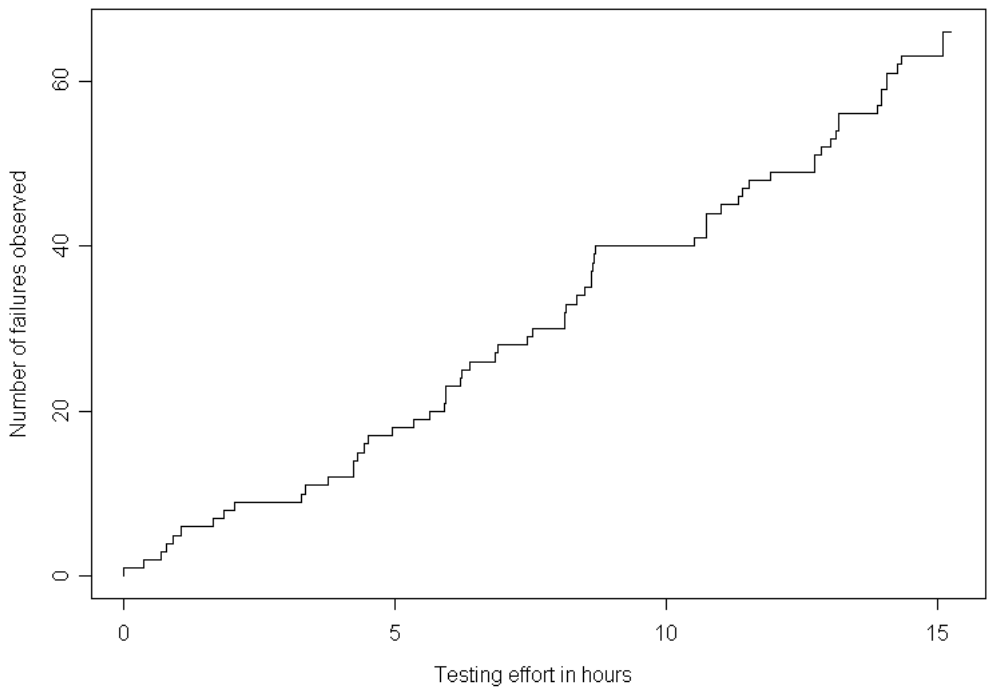

Figure 5: *Cumulative number of failures observed set in relation to the testing effort*

the testers *learn*.

Indeed, figure 5 depicting the cumulative number of failures observed as a function of testing effort (i.e. the time spent on execution of test cases) shows an approximately linear growth. The reduced fault detection capability is more than offset by the increased speed of the testers. Comparing the analysis in the coverage domain to the analysis in the time domain - rather than a pure analysis in the coverage domain as proposed by other researchers - seems to be an appropriate means for identifying the learning of testers during test set execution within one test cycle.

4 Conclusions

Further research and the analysis of several projects will have to be undertaken for confirming our interpretation of testing efficiency and learning. Comparing not only the changes in the testing efficiency like Rivers and Vouk [14, 16] but also in the testers' learning for consecutive releases of the same piece of software may yield interesting insights. Moreover, it remains to be examined whether the testing efficiencies for different AUTs developed by the same company (and therefore following the same processes) show similarities. Linking information on the software development process to failure models is the objective of the PETS project (Prediction of software Error rates based on Test and Software maturity results), which is funded by the European Community. Progress in the area of software failure models is especially needed for test data collected while following the methodology that most companies actually use: systematic testing.

References

[1] Briand, L. C.; Pfahl, D.: *Using Simulation for Assessing the Real Impact of Test-Coverage on Defect-Coverage*, IEEE Trans. Reliability 49 (2000), pp. 60 - 70

[2] Duran, J. W.; Ntafos, S. C.: *An Evaluation of Random Testing*, IEEE Trans. Software Eng. 10 (1984), pp. 438 - 444

[3] Farr, W.: *Software Reliability Modeling Survey*, in: Lyu, M. R. (ed.): *Handbook of Software Reliability Engineering*, New York, San Francisico, et al., 1996, pp. 71 - 117

[4] Frankl, P. G.; Hamlet, R. G.; Littlewood, B.; Strigini, L.: *Evaluating Testing Methods by Delivered Reliability*, IEEE Trans. Software Eng. 24 (1998), pp. 587 -601

[5] Hamlet, R. (D.): *Random Testing*, Department of Computer Science, Portland State University, URL = ftp://ftp.cs.pdx.edu/pub/faculty/hamlet/random.ps.Z (site visited 2001-05-31)

[6] Hamlet, D.; Taylor, R.: *Partition Testing Does Not Inspire Confidence*, IEEE Trans. Software Eng. 16 (1990), pp. 1402 - 1411

[7] Mitchell, B.; Zeil, S. J.: *A Reliability Model Combining Representative and Directed Testing*, Technical Report TR 95-18, Old Dominon University, 1995

[8] Musa, J. D.: *Operational Profiles in Software-Reliability Engineering*, IEEE Software, March 1993, pp. 14 - 32

[9] Musa, J. D.: *Software Reliability Engineering - More Reliable Software, Faster Development and Testing*, New York, St. Louis, et al., 1998

[10] Musa, J. D.; Iannino, A.; Okumoto, K.: *Software Reliability - Measurement, Prediction, Application*, New York, St. Louis, et al., 1987

[11] Myers, G. J.: *Methodisches Testen von Programmen*, 6th edition, München, Wien, et al., 1999

[12] Newbold, P.: *Statistics for Business and Economics*, 3rd edition, Englewood Cliffs, 1991

[13] Pham, H.: *Software Reliability*, New York, Berlin, et al., 2000

[14] Rivers, A. T.: *Modeling Software Reliability During Non-Operational Testing*, Ph.D. thesis, North Carolina State University, 1998, URL = http://renoir.csc.ncsu.edu/Faculty/Vouk/Papers/Rivers/Thesis/Rivers.Thesis.pdf.zip (site visited 2001-05-31)

[15] Rivers, A. T.; Vouk, M. A.: *An Empirical Evaluation of Testing Efficiency during Non-Operational Testing*, Proc. Fourth Software Engineering Research Forum, Boca Raton, 1995, pp. 111 - 120, URL = http://renoir.csc.ncsu.edu/Faculty/Vouk/Papers/SERF96.ps (site visited 2001-05-31)

[16] Rivers, A. T.; Vouk, M. A.: *Resource-Constrained Non-Operational Testing of Software*, Proc. Ninth International Symposium on Software Reliability Engineering, Paderborn, 1998, pp. 154 - 163

[17] Schick, G. J.; Wolverton, R. W.: *An Analysis of Competing Software Reliability Models*, IEEE Trans. Software Eng. 4 (1978), pp. 104 - 120

[18] Singpurwalla, N. D.; Wilson, S. P.: *Software Reliability Modeling*, International Statistical Review, Vol. 62 (1994), No. 3, pp. 289 - 317

[19] Tohma, Y.; Yamano, H.; Ohba, M.; Jacoby, R.: *The Estimation of Parameters of the Hypergeometric Distribution and Its Application to the Software Reliability Growth Model*, IEEE Trans. Software Eng. 17 (1991), pp. 483 - 489

[20] Vouk, M. A.: *Using Reliability Models During Testing With Non-Operational Profiles*, Computer Science Department, North Carolina State University, 1992, URL = http://renoir.csc.ncsu.edu/Faculty/Vouk/Papers/non_op_testing.ps (site visited 2001-05-31)

Biographical Note

Michael Grottke works as a research assistant at the Chair of Statistics and Econometrics at the University of Erlangen-Nuremberg. He is a member of the steering committee of the PETS project, for which he conducts research. In May 1999, Grottke received an M.A. from Wayne State University (Detroit), where he had spent two semesters studying economics on a scholarship from the German Academic Exchange Service (DAAD). Between 1996 and 2000, he studied business administration at the University of Erlangen-Nuremberg, statistics and information systems being his fields of concentration. He received a Dipl.-Kfm. degree in June 2000.

Klaudia Dussa-Zieger has graduated with a Dipl.-Inf. from the University of Erlangen-Nuremberg in 1989. In addition she holds a MS in computer science from the University of Maryland at College Park (1992), and a Ph.D. in computer science from the University of Erlangen-Nuremberg (1998). Since fall 1998 she has been working at imbus in the area of software test and software quality management. Currently she is the manager of the imbus office in Munich.