T U M

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

Efficiency Analysis of Defect-Detection Techniques

Stefan Wagner

TUM-I0413
September 04

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-09-I0413-0/1.-FI
Alle Rechte vorbehalten
Nachdruck auch auszugsweise verboten

©2004

Druck: Institut für Informatik der
Technischen Universität München

Efficiency Analysis of Defect-Detection Techniques*

Stefan Wagner
Institut für Informatik
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany

Abstract

Various effectiveness and efficiency metrics have been proposed for defect-detection techniques and quality assurance. This report aims at introducing and comparing the most common metrics that include the effort for the techniques. These metrics are based on code coverage and fault count. Furthermore two new metrics are introduced that use the failure intensity as a more reliability-oriented measure.

The latter three metrics for determining efficiency are applied in a field study with the German software and system house ESG. Defect and test data from a three-year project is used to analyse the efficiency of the used techniques during four releases. The analysis showed that the efficiency increased at first but decreased in later releases. A correlation between the different metrics cannot be shown. Therefore only counting faults is not sufficient for measuring efficiency with respect to reliability.

*This research was supported by the DFG within the priority program *SoftSpez* (SPP 1064) under project name *InTime*.

1 Introduction

The various flavors of testing are an often underestimated part of the development of software systems. They often constitute 50% of the total development costs [24]. Hence, there is a significant advantage for cost reductions.

To be able to optimise the application of testing, or more generally defect-detection techniques, a means to measure the efficiency is needed. This measure could take various quality attributes into account, such as usability or maintainability. However, we concentrate on the effects on reliability in this paper as this is arguably the most important factor for the user.

Many studies have been conducted about the effectiveness and efficiency of defect-detection techniques, especially inspections have been studied extensively. Most of these studies measure the *defect removal efficiency* as the average time spent to find a fault and the *defect removal effectiveness* as the number of removed faults proportional to the number of all faults. However, how can the number of all faults in a program be determined?

Moreover it is not crucial how many faults are removed but which faults are removed. In [6] the results from a study about the mean time to failure (MTTF) implications of faults are described. The figures show that in the observed software 33 percent of all faults lead to a MTTF greater than 5,000 years and only two percent of the faults lead to an MTTF of less than 50 years. Also in [2] it is suggested that "about 90 percent of the downtime comes from, at most, 10 percent of the defects".

For the understanding of the remainder of the paper it is necessary to define important terms. The distinction between faults and failures is fundamental for software reliability. *Failures* are a perceived deviation of the output values from the expected values whereas *faults* are the cause of failures in code or other documents. Both are also referred to as *defects.*

Therefore failures are the interesting notion for the user of a software system. Hence, the definition of *software reliability* is the probability of *failure*-free functioning of a software component for a specified period in a specified environment.

A common metric for reliability is *failure intensity.* It measures the mean number of failures that occur during a time interval, e.g. an execution hour. The reason for using failure intensity instead of reliability directly is that it is often easier to count the number of failures in a time interval than measuring exact timing information for each failure. Moreover failure intensity can be a very intuitive measure for reliability when it is associated with specific types of interaction with the system, for example the number of failures for 1000 queries to a database system.

The factors governing reliability partially depend on time and are in some sense probabilistic in nature as the distribution of inputs for example. Therefore software reliability can be modelled as a random process. A *software reliability growth model* expresses the failure behaviour of a software in relation to time, implying the dependence on the factors above. Various such models have been proposed, e.g. [22]. They can be used to *estimate* the current level of reliability and to *predict* the future development of reliability. This allows us to analyse different interesting metrics, e.g. the failure intensity at any point in time.

Reliability growth models depend commonly on the consideration of the operational profile of a software. This is the analysis of the usage of the system by the users described by a probability for each operation. The details of this approach were developed by Musa and can be found for example in [22, 21]. When testing is guided by the operational profile, sometimes called *representative* or *operational testing*, then the failure distribution during testing reflects the failure rate during operation. Therefore the operational profile has to be considered for reliability growth models.

There are also various synonyms and notions for the term *defect-detection technique*. We understand it as the generic term for dynamic testing and static reading. Dynamic testing requires the software to be executed during testing, e.g. functional testing or structural testing. Static reading is done on documents, e.g. code review or inspection.

Contribution. The contribution of this paper is two-fold. It gives a comprehensive overview of the possibilities to analyse the efficiency of defect-detection techniques including two new metrics that aim to overcome the deficiencies of the older measures. Furthermore the main traditional efficiency metric and the two new measures are compared on the basis of a field study with real project data.

Outline. The remainder of the report is organised as follows. The different approaches to the analysis of the efficiency of defect-detection techniques are described in Section 2. The efficiency based on coverage measures is introduced in Section 2.1 and based on the number of faults in Section 2.2. The two new methods using local failure data or a reliability growth model can be found in Section 2.3 and Section 2.4. The application of the methods in a field study is described in Section 3. We start with the goals and hypotheses for the field study in Section 3.1, develop a study plan in Section 3.2, and describe the study procedures in Section 3.3. The results of the study are presented in Section 3.4. In Section 4 the approaches for efficiency measurement are in compared in general and using the experience from the field study. Related work is discussed in Section 5 and final conclusions and directions for further research are given in Section 6.

2 Metrics

This section presents four approaches for measuring the efficiency of defect-detection techniques. Firstly, the commonly used metrics based on coverage and fault counts are described. Secondly, two new metrics based on a time-related notion of reliability are proposed. *Effectiveness* is the level of improvement that a defect-detection technique has without taking the effort into account. We subsume all setup time, testing, code reading, defect localisation, and defect removal under effort. In case these influences are incorporated, the *efficiency* of the technique is analysed. In the following we will use the abbreviation T_n for the used defect-detection technique at time n.

2.1 Coverage Efficiency

A very simple but also very disputed method to analyse the efficiency of tests is to measure the coverage of the code. There are numerous types of coverage but we only explain some of the most common in the following.

- *Line or statement coverage.* The simplest form of coverage analyses what fraction of the code has been executed.
- *Branch or decision coverage.* The fraction of the branches of the code that have been executed.
- *Multicondition or predicate coverage.* This type of coverage requires not only each branch to be executed but also each logical operand in a condition to take every possible value in every combination.
- *Path coverage.* One of the strongest coverage types is path coverage. This requires to test every possible path through the program. The number of paths is typically too large to test in practice.

The coverage that a defect-detection technique achieves is easy to measure because there are already existing tools, some of them are even available as open source, that can measure up to condition coverage. This is typically done by instrumenting the code (source or binary) at run-time or separately and counting the execution of statements, decisions, etc.

The coverage metric itself is very important for testing as it is the only measure that helps by quantifying how much and what parts of the code has been tested. There is a relationship between coverage and fault detection [26, 8, 10], although it is not fully understood. Therefore it is one important aspect of a testing technique how much coverage it can achieve in a certain amount of time.

The approach depicted in Figure 1 is straight-forward. For each defect-detection technique one kind of coverage and its required effort is measured and divided to analyse the efficiency. This yields a value for each defect-detection technique that is analysed and on this basis different techniques can be compared.

As mentioned above for analysing efficiency, we have to take the time that was invested in testing into account as well as the achieved coverage. This leads us to the following definition of *coverage efficiency*:

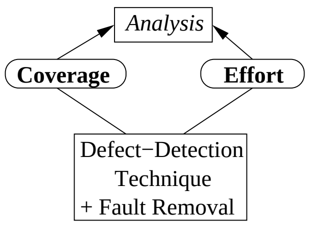

Figure 1: The approach for measuring a defect-detection technique using the coverage approach

$$\eta_C(L, T_n) = \frac{c(T_n)}{t(T_n)} \tag{1}$$

where L is the type of coverage, $c(T_n)$ is the coverage, and $t(T_n)$ is the time in person hours spent for testing and fault removal. The approach is also depicted in Figure 1.

Problematic is the transfer to static defect-detection techniques, i.e. techniques that do not execute the software. An instrumentation of the code cannot measure any coverage in that case. Therefore it is unclear what coverage a static analysis provides.

The general threat to the validity of studies using the coverage efficiency is that coverage is a poor measure for the effects of a defect-detection technique on reliability. It ignores how many defects as well as what kind of defects are found. Will a software be reliable if its test cases cover 100% of its statements? The studies presented in [8, 10] show that there is a relation between coverage and probability of fault detection. However, the results typically show an increase in effectiveness (efficiency is not considered directly) mostly between 90% and 100% coverage and that coverage alone is not a reliable indicator of the effectiveness.

2.2 Fault Count Efficiency

The typically used approach to analyse the effectiveness and efficiency of defect-detection techniques is to count the faults that were detected. The intuition is that a technique is more efficient if it detects more faults in a given amount of time than another technique. The measurement can often be made as a by-product of the documentation of inspections and test-runs.

To analyse efficiency the effort spent for the technique has to be taken into account as well as the number of faults recorded as can be seen in Figure 2. We also include the effort for the fault removal because it allows a better comparison with the following methods and it is also an attribute of the technique. For example the fault removal is easier after an inspection than after functional tests and we consider this as part of the efficiency.

This metric measures intuitively what effort was needed to find and fix a fault in the software.

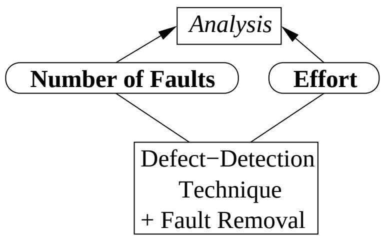

Figure 2: The approach for measuring a defect-detection technique using the fault count approach

This leads us to the following formula for *fault count efficiency*:

$$\eta_F(T_n) = \frac{m(T_n)}{t(T_n)} \tag{2}$$

where $m(T_n)$ is the number of faults found and $t(T_n)$ is the time spent for the technique and fault removal in person hours, person days etc.

To use fault efficiency in studies implies a threat to construct validity because the fault count is a problematic measure for the reliability effects of a defect-detection technique. It does not take the fault-exposure potential and criticality of faults into account. Furthermore the notion of a fault itself is not free of problems. It is discussed in [7] that the idea of a fault is not precise and has proven difficult to define formally because faults have no unique characterisation. Above the fault problem the history could play a role in this measure in case several defect-detection techniques are applied sequentially because the earlier applications typically reveal more faults as the later.

2.3 Local Failure Intensity Efficiency

The generally proposed approach for measuring reliability is to document the time between failures during operational testing or operation of the software system. This time-between-failure (TBF) data is used to analyse the failure behaviour, i.e. the reliability. A simpler measure commonly used is the failure intensity. For this we only need to count the failures and record the time interval in which the failures occurred. Given this data we can calculate the failure intensity in failures per hour for example.

Based on this measure we propose a new metric to analyse the effect of a defect-detection technique. The idea is that we gather the failure data of the software systems before and after the application of a testing method either by operational testing or operation. The two data sets can be further analysed to get failure intensity measures. The comparison should normally result in a failure intensity decrease. The efficiency can be determined by dividing the effect of the technique by the time spent on it. This approach is shown in Figure 3.

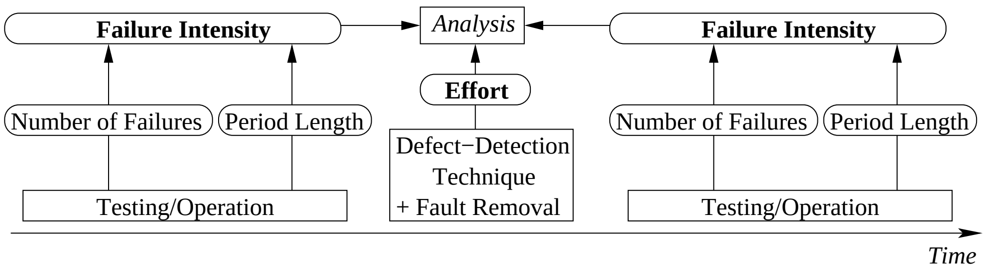

Figure 3: The approach for measuring a defect-detection technique using the local failure intensity approach

The failure intensity is commonly denoted by λ. Therefore the mean number of failures during a time interval for the operational testing and operation O can be calculated by the following formula.

$$\lambda(O) = \frac{f(O)}{l(O)}. \tag{3}$$

where $f(O)$ denotes the number of failures revealed by O and $l(O)$ the length of the test or operation interval. Having established this measure we can derive the formula for the technique efficiency η_L based on the local failure intensity.

$$\eta_L(T_n) = \frac{\lambda(O_b) - \lambda(O_a)}{\lambda(O_b) \cdot t(T_n)}. \tag{4}$$

where O_b is the operational testing *before* T_n and O_a correspondingly the operational testing *after* T_n, and $t(T_n)$ is the time spent for the technique in person hours. We divide in the formula by $\lambda(O_a)$ because we want to concentrate on the relative efficiency.

Variation. It is most feasible to have separate operational tests in a controlled environment of experiments. However, the application on field data without such separate tests is possible with a few restrictions. If we assumed that there are regularly functional test that are at least oriented on the operational profile this data could be used for determining the failure intensities. It allows (1) only to measure defect-detection techniques that have operational tests before and after it and (2) no analysis of certain operational tests themselves. The tests that cannot be analysed are the first and the last test in the project and each operational test that was done directly after or before any other technique. For example the effect on the failure intensity of an operational test that is done before an inspection (incl. fault removal) cannot be determined because the next determinable failure intensity contains the effects of both the operational test and the inspection. To speak in terms of the formulae established above, we use the T_n that represent operational testing as substitute for the missing O.

This efficiency measure has one threat in common with the fault efficiency. In case the defect-detection techniques are applied in a sequential manner they could interfere and hence the history plays an unwanted role. This effect is minimised by the usage of the relative efficiency because the efficiency depends on the current level of reliability. To totally avoid it, the operational testing has to be decoupled from the defect-detection technique in an experimental environment.

2.4 Failure Intensity Model Efficiency

The second metric that we propose does also use the failure intensity as metric for reliability. However, it relies on software reliability growth models that were developed to estimate and predict the reliability of a software system based on its time-between-failure (TBF) data. General descriptions of reliability models for software can be found in [22, 19, 21]. This approach is the most complicated because we have to use a reliability model first to determine the reliability. This is fundamentally different from the local method because it takes the complete failure behaviour into account.

At first either TBF data or failure counts during time intervals are collected during operational testing. This data is the input to one or more software reliability growth models. This can be supported by a software tool like CASRE [25]. This allows to choose the best fitting model and determine the failure intensities based on that model. We take the failure intensity $\lambda(T_n)$ at the end of the application of a testing method. The difference of the values at the application and the application of the last technique can again be used to calculate the *failure intensity model efficiency.*

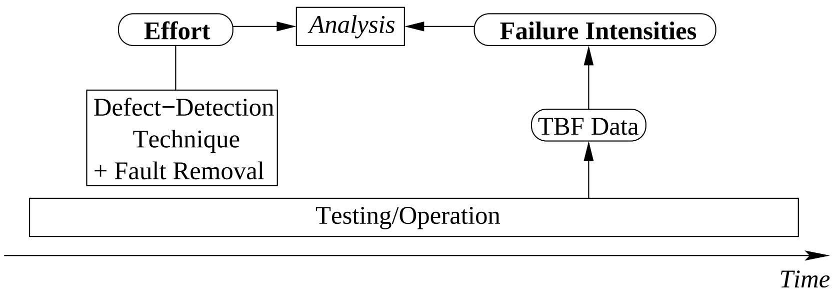

Figure 4: The approach for measuring a defect-detection technique using the model failure intensity method

$$\eta_M(T_n) = \frac{\lambda(T_{n-1}) - \lambda(T_n)}{\lambda(T_{n-1}) \cdot t(T_n)}. \tag{5}$$

where $t(T_n)$ is the time spent for the technique in person hours. We consider the relative efficiency by dividing by $\lambda(T_{n-1})$.

For the failure intensity model efficiency the history interaction is an inevitable threat to validity because reliability growth models build on the sequential application of defect-detection techniques. Furthermore it is a threat that an unsuitable model for the data could be chosen that does not reflect the actual failure behaviour of the software.

3 Field Study

The field study used to evaluate the different approaches to efficiency analysis of defect-detection techniques uses data from a project of the German software and systems company ESG. It was a project in the military sector lasting three years and resulting in a military information system. It consists of software as well as hardware but we will concentrate on the software part in the following.

3.1 Goals and Hypotheses

Building on [18] a GQM-like approach is used to describe the concept of the study to analyse the defect-detection techniques. In short, the goal is to compare functional tests among themselves considering several releases of the software. The comparison has its focus on the improvement of reliability that is measured by fault counts and change of failure intensities. The usage of more than one metric for efficiency originates from the superordinate goal of comparing the metrics. Furthermore the aim is to use realistic data from an industrial project.

Analyse **functional testing**
for the purpose of **comparison**
with respect to their **fault and failure intensity efficiency**
from the point of view of the **researcher**
in the context of **an industrial project**.

From these goals we can derive a hypotheses based on the metrics presented in Section 2. We will use three of them, the *fault*, *local failure intensity*, and *failure intensity model efficiency*, as efficiency measures of the defect-detection techniques. The hypothesis is generic and can be tested with each efficiency metric. However, we will not formally test the hypotheses as the main aim is to evaluate the efficiency metrics themselves. Hence we only use descriptive techniques in the following.

The hypotheses is concerned with the functional tests and the differences during releases. One would expect that the efficiency in later releases is lower than in earlier releases because there are probably more faults in earlier stages of the development and it should get harder and harder to improve reliability.

Hypothesis 1 *The efficiency decreases with each release.*

To simplify the statistical analysis such hypotheses typically are converted to *null hypotheses* stating that there is no difference and the original hypotheses become *alternative hypotheses.* The null hypothesis for our study is as follows.

Hypothesis 2 *There is no difference in the efficiency of functional tests of different release.*

The procedure that is necessary to analyse these hypotheses is described in the following based on the industrial project.

3.2 Study Plan

Following from the hypothesis developed above we consider one independent variable in our study. It is the number of the release on which the team was working on. The dependent variables are determined by the approaches we chose to measure the efficiency of the defect-detection techniques. Hence, we have the dependent variables counts of faults recorded, counts of failures recorded and elapsed time, and effort spent. The testers were the same during the whole project. Therefore the subjects vary only slightly during the study and are not randomly assigned because it is an industrial project that is analysed. This restricts the generalisation of the results.

The defect-detection technique that is analysed in this study is functional testing, also called black box testing. It uses only the specification to derive test cases. The functional tests were divided in week-long test periods. Only integration and system testing is considered here because unit testing was not documented.

The development in general followed mainly the iterative and incremental process model *Rational Unified Process (RUP)* [15]. That implies that test methods were used increasingly during the project's lifetime. Increments were delivered to the user after the quality assurance and failure reports were fed back. The analysis considers four releases of the software. The first release was after technique 24, the second after 47, the third after 64, and finally the fourth after technique 101.

The object under study is the software part of the system. Its faults and the failures that occurred during development and operation are used. We assume based on the defect database that faults are removed soon enough so that no failure resulting from a fault found by an earlier defect-detection technique occurs.

Approximately 15 developers worked on the software development and 1 to 5 were involved in the application of the defect-detection techniques. They are experienced testers but were not involved in RUP projects prior to this one.

3.3 Study Procedures

The study uses the defect database that was maintained during the project. The tool used for this purpose was *Rational ClearQuest* from IBM[1]. The data used from this tool are the defect number, the time of occurrence and whether it is a duplicate. The latter allows us to distinguish between faults and failures.

The data we have is based only on calendar time not *execution time*, i.e. the time that the system was actually executing. Therefore, we have to assume that there was a constant computer utilization during the project. In this case the execution time is proportional to the calendar time [21]. This is important for the applicability of reliability models. Furthermore it must be noted that a general assumption for the data is also that all failures are due to defects and not user errors.

As recommended in [21] two models are used for the analysis and the one that fits best is chosen. The models recommended are the *basic Musa* [20] and the *logarithmic Poisson* [23].

These models are also supported by the tool *CASRE (Computer Aided Software Reliability Estimation* [25]. CASRE is a software reliability estimation tool that runs under Microsoft Windows and was designed primarily for researchers. It allows the user to read in TBF data, display the data graphically, choose and apply various reliability models and estimate the parameters automatically. We used the tool for the following analysis.

The field study contains several threats to the validity of the results. Obviously we have to take all threats identified in Section 2 for the different approaches into account. Most importantly this is the history of the defect-detection techniques. In the considered project the techniques were applied sequentially and therefore the efficiency does interact. The earlier applied techniques can detect defects more easily because there are more faults in the system. In some cases the sample size can also be a threat because with only a few failures the calculation of a failure intensity can be questioned. A general threat in this kind of field study where the researcher is merely an observer is that the documentation – in this case of failures, failure times, and effort – can be bad or even wrong. In this case particularly the effort documentation is coarse-grained. Furthermore the application of defect-detection techniques changes over time as, for example, the experience of the testers increases. The efficiency can also be affected by the personnel or equipment.

The external validity, i.e. the generalisation of the results, is limited. Firstly, only one instance of a project is considered. Therefore the results cannot be representative. Furthermore the subjects and objects did not vary significantly during the project. Hence the experience of the tester, the complexity of the problem, or the quality of the software developers can have a strong influence on the results.

[1]http://www.ibm.com/software/awdtools/clearquest/

3.4 Results

First we want to look at the effort spent for each technique. Unfortunately this data is vague because there was no detailed documentation during the project and is based on questioning the testers. The curve describing the effort can be seen in Figure 5. Generally during the first part of development only one to two people worked on testing. This number was only increased before releases to the customer. These are the peaks around technique 20 and after 40. The high peak at technique 60 is a review with nine person weeks that is not further considered in the following.

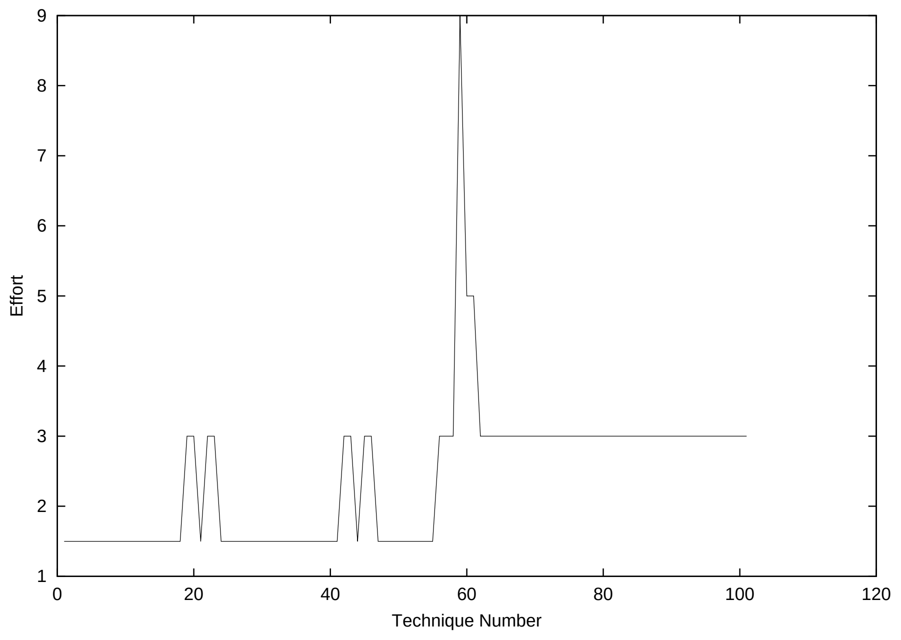

Figure 5: The effort spent for the defect-detection techniques during the project. Effort in person weeks.

All results for the different metrics can be found in the appendix. Several statistics of these values for the methods are summarised in Table 1. Hypothesis 2 is concerned with the differences of the efficiencies for the different releases. We have several box-plots for each method in the following to analyse the hypothesis.

Fault Count Efficiency. The results for the fault count efficiency are depicted in Figure 6. It can be seen that the data is not particularly regular. We have several peaks but also several low values what makes the curve rather jagged.

In the boxplot in Figure 7 can be seen that the data has several outliers that have much higher values than the mean. If we split the data according to the releases of the software these outliers become less. Furthermore it

App.	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
FC	0.330	1.330	4.670	6.867	8.000	35.330
LFI	-18.67	-0.67	-0.06	-1.02	0.23	0.64
FIM	0.00000	0.01000	0.01000	0.01424	0.02000	0.05000

Table 1: Summary of the efficiency values for the different approaches over all techniques

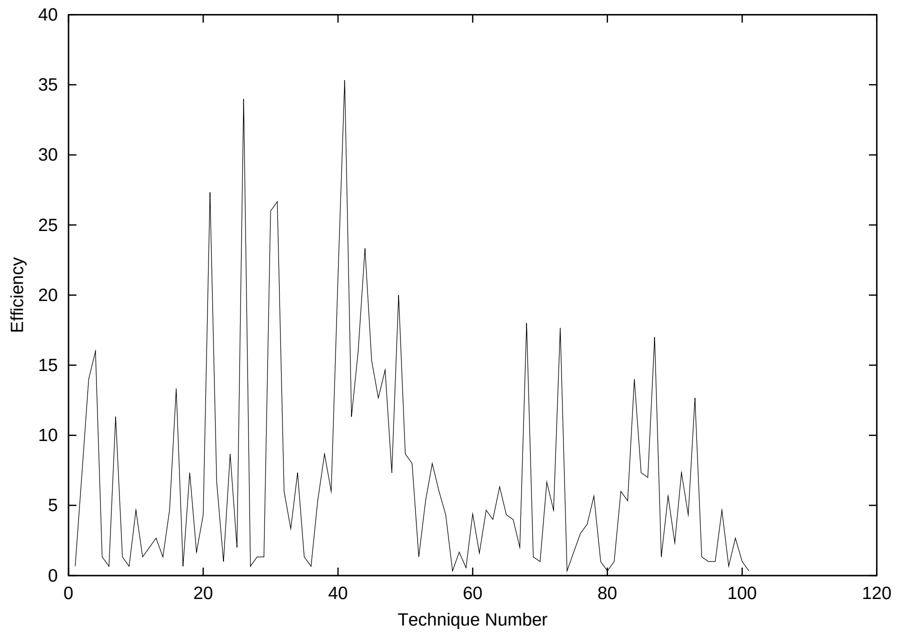

Figure 6: The fault efficiency of the defect-detection techniques during the project. Efficiency in faults per person week.

can be seen that we do not have a steady decrease in efficiency but that the efficiency increased for the second release and only afterwards decreased. This contradicts hypothesis 2.

The mean value of 6.867 that we calculated is lower than typical published values. For example in [11] the value for inspections and unit tests is 1.9 person hours per fault which are 15.8 faults per person week. However, the value from this project stems from integration and system testing. These are likely to find fewer faults than unit testing because they are applied later in the development.

Local Failure Intensity Efficiency. As this is not a controlled experiment but a field study, the local failure intensity method has to be adapted as described in Section 2.3. We are not able to apply independent operational testing but use the functional tests from the project for the fail-

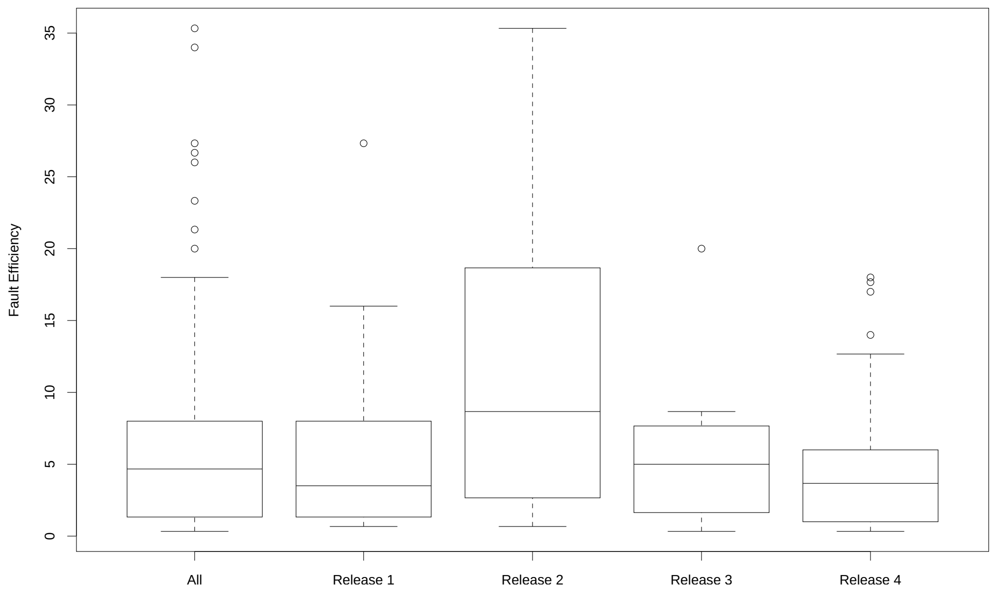

Figure 7: Boxplots for the fault efficiency of the techniques. The left boxplot shows all efficiency measures, the others only for functional tests and specific releases

ure intensity measurement. Therefore we take the failure intensity during the last technique and during the next technique to measure a technique.

Considering the local failure intensities leads to the interesting results depicted in Figure 8. It shows that this efficiency metric predominantly yields negative values. This is counterintuitive for this kind of metric. As the reliability increased in general one would suspect to see this reflected in the efficiency.

The reason for the predominantly negative values is probably that the effects were not already visible during the next week. Furthermore there could be violations of the assumption that the functional tests follow the operational profile.

The corresponding boxplot is depicted in Figure 9. It can easily be seen that considering all values as well as separated for the releases there are strong outliers. This suggests that this approach is not suitable for the field study. We see that for the second release the efficiency is slightly decreasing and then increases for the third release, whereas it is decreasing again in the fourth release.

Failure Intensity Model Efficiency. The efficiency analysed based on a reliability growth model yields the most intuitive data. As can be seen in Figure 10 the efficiency is decreasing during the testing. This is in accordance with the intuition that its getting harder and harder to improve reliability during the test process.

The *Musa-Okumota* model was used for the determination of the failure

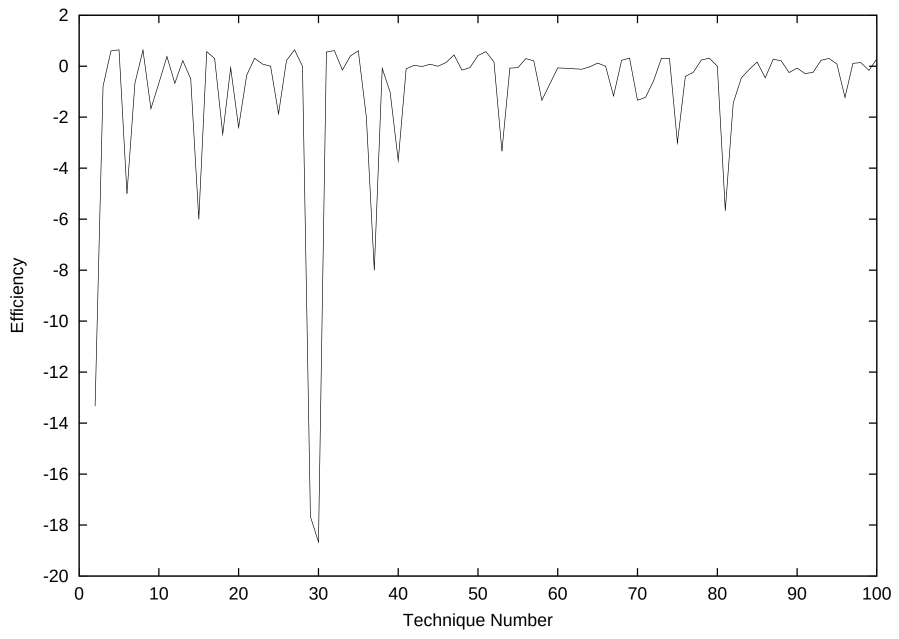

Figure 8: The local failure intensity efficiency of the defect-detection techniques during the project. Efficiency in 1 per person week.

intensity values. We used the *KS distance* calculated by *CASRE* as the similarity measure. This recommended the *Musa-Okumota* model. Furthermore because of constraints of the tool only the last 1000 failures are taken into account.

This approach yields the best boxplots (Figure 11). There are only few outliers and the values are close to the mean. The observation concerning the efficiency in dependency of the releases is contradictory to the fault count method. The efficiency increases from the second to the third release and decreases in the fourth release. This can be a result of the limitation that only the last 1000 failures could be analysed. The change in efficiency contradicts hypothesis 2.

A correlation between the values of the three different approaches can not be found. As an example we show the fault count against the failure intensity model method in Figure 12 because they have the best possibility to have correlation considering the boxplots from above. However, there is no correlation visible.

4 Comparison

This section gives a detailed comparison of the presented approaches to the efficiency of defect-detection techniques from Section 2. We start by summarising the advantages and disadvantages of each approach.

The *coverage efficiency* is easy to apply because coverage is simple to

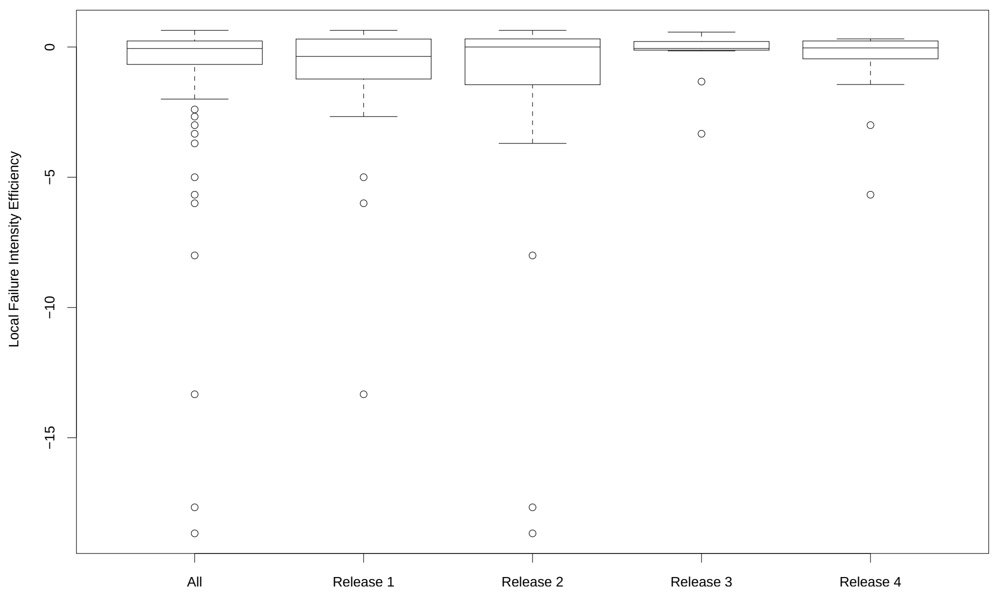

Figure 9: Boxplots for the local failure intensity efficiency of the techniques. The left boxplot shows all efficiency measures, the others only for functional tests and specific releases.

measure using a variety of tools supporting that task. The main disadvantage is that code coverage is a problematic metric for the effects of a defect-detection technique. The relationship between the fraction of code covered by a test suite and the effect on reliability is not clear. Furthermore structural tests can be optimised on a coverage criterion using the knowledge of the structure of the code. This makes a comparison futile. Moreover it is not possible to use this approach for reading techniques. The coverage is not measurable in this case. Finally it must be noted that there are various coverage criteria that can be used for this approach and Weyuker's hypothesis [5, 31, 9] states that the adequacy of a coverage criterion can only be intuitively defined. That means that it can not be objectively determined which coverage criterion should be used.

The main problem of the *fault count efficiency* is the usage of faults as underlying metric. As argued in Section 2.2 it has several disadvantages. For example it is not obvious how a fault can be localised in code. Furthermore the removal of a fault does not necessarily lead to an improvement of reliability. This is also reflected in the very heterogeneous results that this approach yields. There can be phases in which a large number of small faults are removed leading to a high efficiency of the defect-detection technique followed by phases where only a few but large-scale faults are removed leading to a low efficiency. Hence the metric is not well balanced considering the failure-potential and severity of faults.

The problems connected with faults can be overcome using the *local failure intensity efficiency*. It measures only failures that occurred during

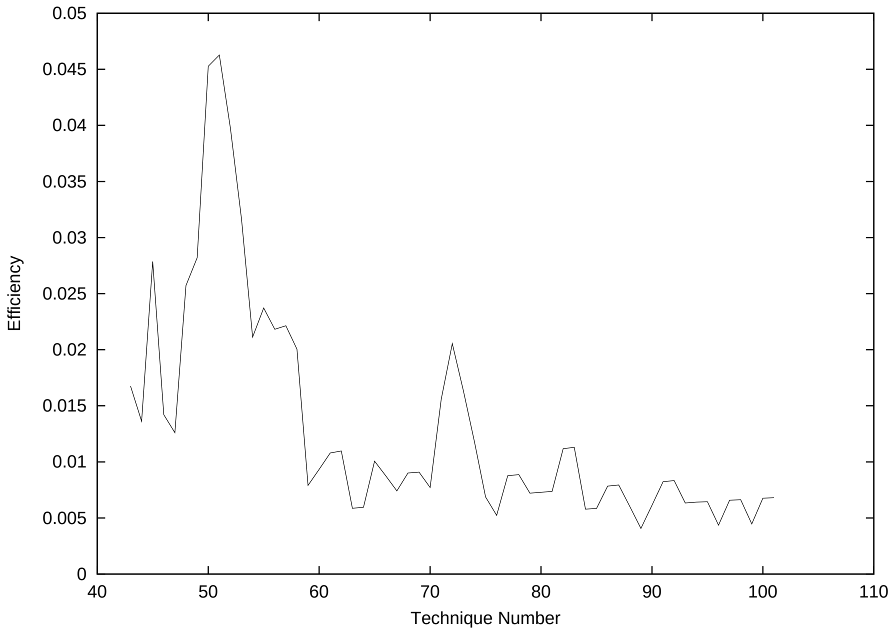

Figure 10: The failure intensity model efficiency of the defect-detection techniques during the project. Efficiency in 1 per person week.

operational testing. To use the failure intensity before and after the defect-detection technique to measure its efficiency allows to only measure direct effects of the technique. Although this is an advantage it can be also seen as a disadvantage. The fault removal that was caused by the defect-detection technique could have effects that are not directly visible during the next operational testing. Furthermore bad repairs, i.e. fault removal that injects new faults, can adulterate the results of the metric significantly. Another problem is that in this case it often produces negative values and strong outliers. This is counter-intuitive because efficiency should be positive as long as the reliability increases. However for studies where a totally independent operational testing can take place before and after each defect-detection technique this approach might yield good results.

Finally the *failure intensity model efficiency* has the problem that it involves a complex calculation because a software reliability growth model is used to determine the failure intensities. This also introduces impreciseness because we rely on statistically determined data. The complex calculation also implies the usage of tools. Therefore the constraints of the tool can also influence the results. However the overall results are better than directly using the failure data as in the local failure intensity efficiency because the whole failure behaviour of the software system is taken into account. This leads to intuitively plausible efficiency results.

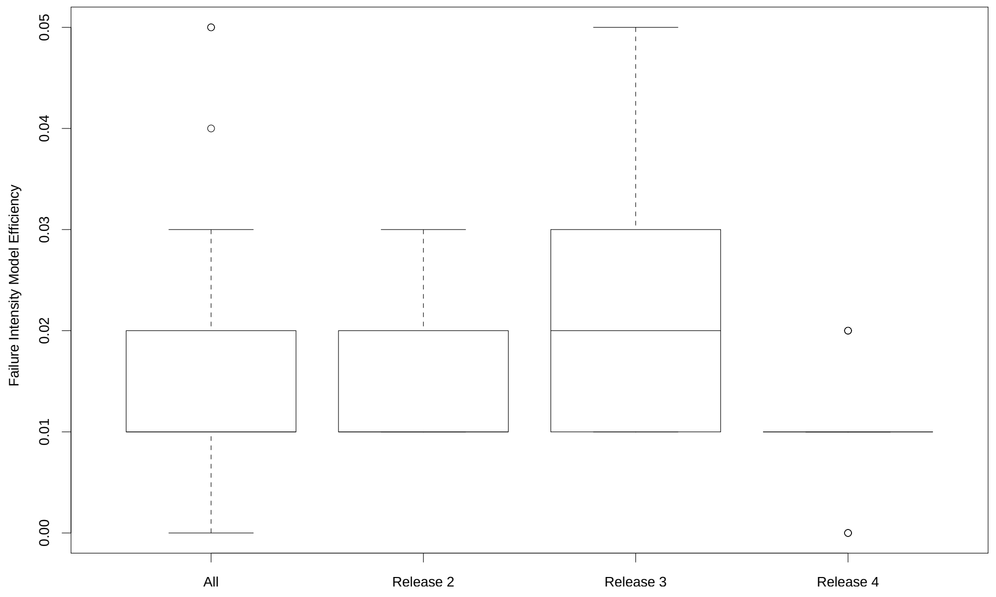

Figure 11: Boxplots for the failure intensity model efficiency of the techniques. The left boxplot shows all efficiency measures, the others only for functional tests and specific releases.

5 Related Work

Musa describes in [21] the test efficiency as the relative rate of reduction of failure intensity with respect to execution time. It is also a relative measure and should be used with the average over time periods. The drawback of this method is that the execution time is used in the formula instead of the more general time spent for a technique. Furthermore the approach is not well-elaborated.

As mentioned before most studies in the area are focused on fault count efficiency. A well-known example for this is [12] where it is called *defect removal efficiency.*

Several of the most important and partly quite old experiments are briefly described in [18] that all concentrate on the number of faults or failures revealed, respectively. More recently [28] evaluated the fault count efficiency of six techniques empirically and [4] looked at the cost-effectiveness of inspections. These studies typically come to the conclusion that inspection is the most effective and efficient defect-detection technique followed by functional test and finally structural test. A comprehensive overview of experiments that have been conducted about testing methods and their results are given in [13].

[3] builds a detailed model for the efficiency of inspections for simulation without looking at reliability measures.

Another recent example is [29] where several experiments are summarized comparing different reading techniques for inspections. Efficiency is

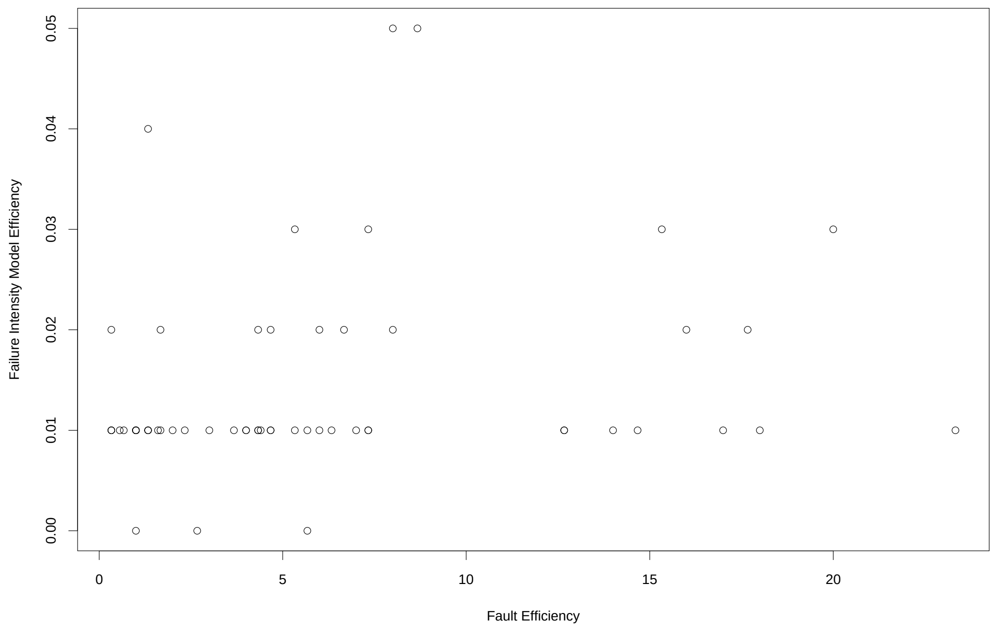

Figure 12: Relationship of the fault efficiencies and the failure intensity model methods

measured here in faults found per hour.

The same techniques are evaluated in [30] not only in terms of fault efficiency but also looking at the effectiveness at revealing the most critical faults.

A quite different approach was taken in [27]. Inspection effectiveness in measured in terms of *defect escapes*, i.e. the number of defects that can escape in the next phase. Still, the time between failures is not considered.

In [16] the combination of inspection and testing techniques was discussed but not analysed in terms of efficiency.

6 Conclusion and Further Research

From the field study it became evident that it is important for the local failure intensity efficiency that the test data used comes from independent testing that is based on the operational profile. Otherwise the values are distorted.

The failure intensity model method is comparably elaborate and contains several levels where statistical methods are used. This can introduce significant blurring in the results.

However, the main conclusion lies in the comparison of the new metrics with the fault count efficiency. There is not only no correlation between these measures but also the results are contradictory. The failure intensity model efficiency and the fault count efficiency show contrary trends.

This allows the conclusion that only counting faults is not sufficient for analysing the efficiency of a defect-detection technique in terms of reliability improvement.

One option that is interesting for further research is the combination of defect-detection techniques. Intuitively, different techniques will find different faults and hence, specific combinations of techniques might be more efficient than others. These interrelationships were theoretically expressed in a model in [17]. This model might be refined based on further experimentation.

Furthermore, an aspect that is often ignored is the criticality or severity of a failure. A defect-detection technique that is very efficient should detect more severe faults than others. We tested a method based on expert opinion in [14] and got promising results that ask for further research.

Another direction is the extension of the fault count method by estimates of fault exposure potential and severity of the fault. This implies all advantages of the fault count method, i.e. easy measurement, but takes also the real reliability and safety implications into account. What is needed are experts that are able to estimate these fault characteristics.

In [13] several interesting research issues were brought up. From the current experience we consider two of them especially interesting for our future research. Firstly, the analysis which defect-detection technique reveals which type of faults and failures must be intensified to be able to evaluate the interplay of different techniques. Secondly, is the search for specific program characteristics that influence the effectiveness and efficiency of defect-detection techniques a promising area.

Moreover, what is missing in this current analysis of the efficiency of defect-detection techniques is the cost factor. The effort considered so far only included the personnel. However, the costs of tools, environment simulation, and so on are particularly important for testing and should be taken into account.

Finally it is challenging to apply these approaches to various emerging areas, for example agile methods such as XP [1]. Especially the practice of pair programming is proposed as a substitute for inspections. Therefore it is interesting to compare the number and types of faults that are revealed by pair programming in contrast to inspections. Another interesting area is model-based testing. This emerging technique can be compared to traditional testing techniques based on the developed metrics.

Acknowledgments

I would like to thank Albert Kainzinger, Dieter Prochaska, and Jürgen Zeißler from ESG for providing the project data for this research and for the helpful discussions on the topic. Furthermore I am grateful to Manfred Broy, David Cruz, and Alexander Pretschner who commented on earlier versions of this paper.

References

[1] K. Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, 1999.

[2] B. Boehm and V.R. Basili. Software Defect Reduction Top 10 List. *IEEE Computer*, 34(1):135–137, 2001.

[3] L. Briand, K. El Emam, O. Laitenberger, and T. Fussbroich. Using simulation to build inspection efficiency benchmarks for development projects. In *Proc. 20th International Conference on Software Engineering (ICSE'98)*, pages 340–349. IEEE Computer Society, 1998.

[4] L. Briand, B. Freimut, and F. Vollei. Assessing the Cost-Effectiveness of Inspections by Combining Project Data and Expert Opinion. Technical Report 070.99/E, Fraunhofer IESE, 1999.

[5] A. Endres and D. Rombach. *A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories.* Pearson, 2003.

[6] N. Fenton, S.L. Pfleeger, and R.L. Glass. Science and Substance: A Challenge to Software Engineers. *IEEE Software*, 11(4), 1994.

[7] P. Frankl, D. Hamlet, B. Littlewood, and L. Strigini. Evaluating Testing Methods by Delivered Reliability. *Transactions on Software Engineering*, 24(8):586–602, 1998.

[8] P. Frankl and O. Iakounenko. Further Empirical Studies of Test Effectiveness. In *Proc. 6th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 153–162. ACM Press, 1998.

[9] P.G. Frankl and E.J. Weyuker. A Formal Analysis of the Fault-Detecting Ability of Testing Methods. *IEEE Transactions on Software Engineering*, 19(3):202–213, 1993.

[10] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. 16th International Conference on Software Engineering (ICSE'94)*, pages 191–200. IEEE Computer Society Press, 1994.

[11] P. Jalote and M. Haragopal. Overcoming the NAH Syndrome for Inspection Deployment. In *Proc. 20th International Conference on Software Engineering (ICSE'98)*, pages 371–378. IEEE Computer Society Press, 1998.

[12] C. Jones. *Applied Software Measurement: Assuring Productivity and Quality.* McGraw-Hill, 1991.

[13] N. Juristo, A.M. Moreno, and S. Vegas. Reviewing 25 Years of Testing Technique Experiments. *Empirical Software Engineering*, 9:7–44, 2004.

[14] W. Khachabi. *Messung von Zuverlässigkeits- und Sicherheitsmethoden.* Diploma Thesis, Technische Universität München, 2004. In German.

[15] P. Kruchten. *The Rational Unified Process. An Introduction.* Addison-Wesley, 2nd edition, 2000.

[16] O. Laitenberger, B. Freimut, and M. Schlich. The Combination of Inspection and Testing Technologies. Technical Report 070.02/E, Fraunhofer IESE, 2002.

[17] B. Littlewood, P.T. Popov, L. Strigini, and N. Shryane. Modeling the Effects of Combining Diverse Software Fault Detection Techniques. *IEEE Transactions on Software Engineering*, 26(12):1157–1167, 2000.

[18] C.M. Lott and H.D. Rombach. Repeatable Software Engineering Experiments for Comparing Defect-Detection Techniques. *Empirical Software Engineering*, 1(3), 1996.

[19] M.R. Lyu, editor. *Handbook of Software Reliability Engineering.* IEEE Computer Society Press and McGraw-Hill, 1996.

[20] J.D. Musa. A theory of software reliability and its application. *Transactions on Software Engineering*, SE-1(3):312–327, 1975.

[21] J.D. Musa. *Software Reliability Engineering.* McGraw-Hill, 1999.

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

[23] 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.

[24] G.J. Myers. *The Art of Software Testing.* John Wiley & Sons, 1979.

[25] A.P. Nikora, M.R. Lyu, W.H. Farr, and T.M. Antczak. CASRE An Easy-to-Use Software Reliability Measurement Tool. Technical report, NASA Jet Propulsion Laboratory, 1993.

[26] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner. An Evaluation of Model-Based Testing and its Automation. Submitted to the 27th International Conference on Software Engineering (ICSE) 2005.

[27] T. Raz and A.T. Yaung. Inspection effectiveness in software development: a neural network approach. In *Proc. 1994 conference of the Centre for Advanced Studies on Collaborative research*, page 61. IBM Press, 1994.

[28] S.S. So, S.D. Cha, T.J. Shimeall, and Y.R. Kwon. An empirical evaluation of six methods to detect faults in software. *Software Testing, Verification, and Reliability*, 12, 2002.

[29] T. Thelin. Empirical evaluation of usage-based reading and fault content estimation for software inspections. *Empirical Software Engineering*, 8:309–313, 2003.

[30] T. Thelin, P. Runeson, C. Wohlin, T. Olsson, and C. Andersson. Evaluation of Usage-Based Reading — Conclusions after Three Experimens. *Empirical Software Engineering*, 9:77–110, 2004.

[31] E.J. Weyuker. The Evaluation of Program-based Software Test Adequacy Criteria. *Communications of the ACM*, 31(6):668–675, 1988.

Appendix

Tech.	FC	LFI	FIM
1	0.67	–	–
2	7.33	-13.33	–
3	14.00	-0.79	–
4	16.00	0.60	–
5	1.33	0.64	–
6	0.67	-5.00	–
7	11.33	-0.67	–
8	1.33	0.63	–
9	0.67	-1.67	–
10	4.67	-0.67	–
11	1.33	0.38	–
12	2.00	-0.67	–
13	2.67	0.22	–
14	1.33	-0.50	–
15	4.67	-6.00	–
16	13.33	0.57	–
17	0.67	0.30	–
18	7.33	-2.67	–
19	1.67	-0.06	–
20	4.33	-2.40	–
21	27.33	-0.36	–
22	6.67	0.31	–
23	1.00	0.08	–
24	8.67	0.00	–
25	2.00	-1.87	–
26	34.00	0.22	–
27	0.67	0.64	–
28	1.33	0.00	–
29	1.33	-17.67	–
30	26.00	-18.67	–
31	26.67	0.56	–
32	6.00	0.61	–
33	3.33	-0.15	–
34	7.33	0.40	–
35	1.33	0.61	–
36	0.67	-2.00	–
37	5.33	-8.00	–
38	8.67	-0.08	–
39	6.00	-1.03	–
40	21.33	-3.70	–
41	35.33	-0.10	–
42	11.33	0.03	–
43	16.00	-0.01	0.02
44	23.33	0.08	0.01
45	15.33	0.00	0.03
46	12.67	0.15	0.01
47	14.67	0.44	0.01
48	7.33	-0.15	0.03
49	20.00	-0.05	0.03
50	8.67	0.42	0.05
51	8.00	0.57	0.05
52	1.33	0.17	0.04
53	5.33	-3.33	0.03
54	8.00	-0.07	0.02
55	6.00	-0.06	0.02
56	4.33	0.30	0.02
57	0.33	0.21	0.02
58	1.67	-1.33	0.02
59	0.56	–	0.01
60	4.40	-0.07	0.01
61	1.60	–	0.01
62	4.67	-0.10	0.01
63	4.00	-0.12	0.01
64	6.33	-0.03	0.01
65	4.33	0.12	0.01
66	4.00	0.00	0.01
67	2.00	-1.17	0.01
68	18.00	0.23	0.01
69	1.33	0.31	0.01
70	1.00	-1.33	0.01
71	6.67	-1.22	0.02
72	4.67	-0.58	0.02
73	17.67	0.31	0.02
74	0.33	0.30	0.01
75	1.67	-3.00	0.01
76	3.00	-0.40	0.01
77	3.67	-0.23	0.01
78	5.67	0.24	0.01
79	1.00	0.31	0.01
80	0.33	0.00	0.01
81	1.00	-5.67	0.01
82	6.00	-1.44	0.01
83	5.33	-0.46	0.01
84	14.00	-0.13	0.01
85	7.33	0.16	0.01
86	7.00	-0.45	0.01
87	17.00	0.27	0.01
88	1.33	0.22	0.01
89	5.67	-0.25	0.00
90	2.33	-0.07	0.01
91	7.33	-0.29	0.01
92	4.33	-0.24	0.01
93	12.67	0.23	0.01
94	1.33	0.31	0.01
95	1.00	0.08	0.01
96	1.00	-1.22	0.00
97	4.67	0.11	0.01
98	0.67	0.14	0.01
99	2.67	-0.17	0.00
100	1.00	0.29	0.01
101	0.33	–	0.01