

# Software Quality Economics for Defect-Detection Techniques Using Failure Prediction \*

Stefan Wagner and Tilman Seifert  
Technische Universität München  
Boltzmannstr. 3  
85748 Garching, Germany  
{wagnerst|seifert}@in.tum.de

## ABSTRACT

Defect-detection techniques, like reviews or tests, are still the prevalent method to assure the quality of software. However, the economics behind those techniques are not fully understood. It is not always obvious when and for how long to use which technique. A cost model for defect-detection techniques is proposed that uses a reliability model and expert opinion for cost estimations and predictions. It is detailed enough to allow fine-grained estimates but also can be used with different defect-detection techniques not only testing. An application of the model is shown using partly data from an industrial project.

## Categories and Subject Descriptors

D.2.9 [Software Engineering]: Management; D.2.8 [Software Engineering]: Metrics; D.2.5 [Software Engineering]: Testing and Debugging

## General Terms

Economics, Reliability

## Keywords

Software quality costs, reliability model

## 1. INTRODUCTION

Quality assurance is a costly part of software development. Some estimates [13] relate up to 50% of the development costs to testing. Defect-detection techniques are the mostly used method to achieve quality in software. Therefore, we need to understand the relations between costs and benefits regarding those techniques, especially when aiming to compare them. Based on the understanding of those relationships and the characteristics of different techniques,

\*This research was supported in part by the *Deutsche Forschungsgemeinschaft (DFG)* within the project *InTime*.

we can optimise the usage of the techniques. Hence, for a cost-effective use we need a cost model of defect-detection techniques.

In an earlier paper [18] we concentrated on the usage of efficiency metrics for the determination of the costs. This paper extends this idea by incorporating software reliability models to predict the operational failure behaviour of the software. For this we use existing reliability models to predict the number of external failures of the software and hence the external costs. The other costs are determined based on expert estimations and direct measurements.

### 1.1 Problem

The underlying problem is the question of how to use the existing resources for quality assurance in the most cost-effective way. Because this cannot be solved in a single step we concentrate at first on how defect-detection techniques can be evaluated and compared on an economical basis.

### 1.2 Contribution

This paper proposes a structured approach to the evaluation of a defect-detection technique based on expert opinion and predicted failure behaviour. It allows a more precise cost analysis than similar approaches and has the specific advantage over other approaches based on reliability models that it can be used not only for testing but also for other techniques and considering different efficiencies.

### 1.3 Outline

We start in Sec. 2 with the introduction of the types of costs that we consider in our approach and describe the cost model in Sec. 3. The model is applied to project data in Sec. 4 and related work is discussed in Sec. 5. Finally, we finish with conclusions and future work in Sec. 6.

## 2. TYPES OF COSTS

An area that is under research in various domains are the costs of quality. We understand them as the costs that are associated with preventing, finding, and correcting defective work. These costs are divided into *conformance* and *non-conformance* costs, also called *control costs* and *failure of control costs*. We can break down the costs further to the distinction between prevention, appraisal, and failure costs which gives the model the name *PAF* model. The basic model was derived from the manufacturing industry but has been used repeatedly for software quality as well [8, 9, 16]. A graphical overview is given in Fig. 1.

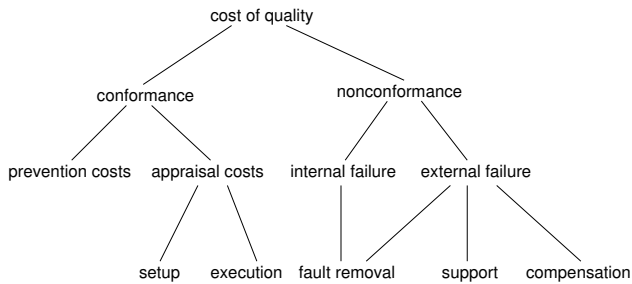


Figure 1: Cost types

## 2.1 Conformance Costs

The conformance costs comprise all costs that need to be spent to build the software in a way that it conforms to its quality requirements. This can be further broken down to *prevention* and *appraisal* costs. Prevention costs are for example developer training, tool costs, or quality audits, i.e. costs for means to prevent the injection of faults. The appraisal costs are caused by performance of various types of tests and reviews.

We added in Fig. 1 further detail to the PAF model by introducing the main types of concrete costs that are important for defect-detection techniques in our current cost model. Note that there are more types that could be included, for example, maintenance costs. The appraisal costs were detailed to the *setup* and *execution* costs. The former constituting all initial costs for buying test tools, configuring the test environment, and so on. The latter means all the costs that are connected to actual test executions or review meetings, mainly personnel costs.

## 2.2 Nonconformance Costs

The *nonconformance* costs come into play when the software does not conform to the quality requirements. These costs are divided into *internal failure* costs and *external failure* costs. The former contains costs caused by failures that occurred during development, the latter describes costs that result from failures at the client.

On the nonconformance side, we have *fault removal* costs that can be attributed to the internal failure costs as well as the external failure costs. This is because if we found a fault and want to remove it, it would always result in costs no matter whether caused in an internal or external failure. Actually, there does not have to be a failure at all. Considering code inspections, faults are found and removed that have never caused a failure during testing. It is also a good example that the removal costs can be quite different regarding different techniques. When a test identifies a failure, there needs to be considerable effort spent to find the corresponding fault. During an inspection, faults are found directly. Fault removal costs also contain the costs for necessary re-testing and re-inspections.

External failure also cause *support* costs. These are all costs connected to customer care, especially the effort from service workers identifying the problem. Finally, *compensation* costs could be part of the external failure costs, if the failure caused some kind of damage at the customer site. We might also include loss of sales because of bad reputation in the external failure costs but do not especially look at it in

this paper because it is out of scope.

## 2.3 Period

A further concept we need to introduce is that of a *period*. It denotes a specific time interval in the life cycle. It does not have to correspond to a classical waterfall phase and can be used with an iterative process as well. The point is only to be able to (1) estimate the fault removal costs depending on the point in time and (2) have a basis for the calculation of the present net values. The first is important because the additional work increases the later the fault is removed. For example if a design fault is detected early only the design documents need to be changed, later also the code and test cases have to be adapted. This is also shown, for example, by Jones [6]. The latter is important because to be able to compare costs the cash flows need to be analysed in dependence of the point in time when they occur. We depict a typical change in nonconformance costs during time in Fig. 2.

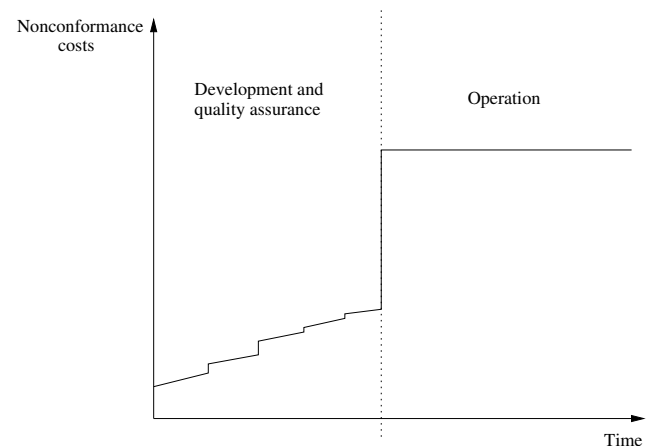


Figure 2: Change of nonconformance costs per failure over time

## 3. COST MODEL

The basis for the analysis of the quality costs is the net present value of the cash flows (NPVCF), i.e. the revenues and costs during the life cycle of the software that are related to quality. This metric is sufficient to judge the earning power and also to compare different defect-detection techniques. For further analyses other metrics, such as return on investment (ROI) or SQPI can be used [16].

### 3.1 Directly Measurable Costs

In this first part of the quality costs, we want to classify those costs that can be directly measured or at least be estimated accurately during the usage of the defect-detection technique. For a specific defect-detection technique we have fixed initial investments called *setup costs* ( $c_{setup}$ ) that contain for example tool or workstation costs. We assume them to emerge at the start of development and therefore have already the net present value. Furthermore, we have the dynamic part of the appraisal costs, e.g. personnel for tests and inspections, that we call *execution costs* ( $c_{exec}$ ). Furthermore, the *fault removal costs* for faults that were

found during the defect-detection technique under consideration can be measured directly. We call this  $c_{remv}(p)$  which denotes the fault removal costs at the period  $p$ .

With this information we can determine the *direct costs* ( $c_{direct}$ ) of the defect-detection technique by adding up the execution and fault removal costs for all periods where we have direct measurements and the setup costs. The periods that we look at here are typically the periods in which the defect-detection technique was used. The costs now need to be discounted to the net present value using the discount factor  $D$  that represents the average cost of capital. This results in the following equation.

$$c_{direct} = c_{setup} + \sum_{p=0}^n \frac{c_{exec}(p) + c_{remv}(p)}{(1+D)^p}. \quad (1)$$

### 3.2 Prediction Using a Reliability Model

One of the main deficiencies of the predecessor model in [18] is that it has no means to predict the total number of residual faults or even better the failure behaviour of the software during operation. We want to fill this gap by using software reliability models.

It is important to note that not only the occurrence of failures is interesting to predict but also their severity. There are two basic possibilities to do this: (1) Either analyse the failure behaviour separately for each severity class or (2) estimate the fraction that each severity class constitutes of the whole number of failures. We use the second approach because it is easier to use and gives us more sample data for the reliability prediction.

We basically can use any existing reliability model that can yield the mean number of failures experienced up to time  $t$  (often denoted by  $\mu(t)$ ). The number of experienced failures in each period  $p$  is than

$$f(p) = \mu(t_p) - \mu(t_{p-1}), \quad (2)$$

where  $t_p$  is the time to the end of period  $p$  and  $t_{p-1}$  is the end of period  $p-1$ . This means that we subtract the cumulated number of failures at the last period from the cumulated number of failures at this period which leaves us with the number of failures experienced in this period. Note that many models use execution time whereas the periods are measured in calendar time. Hence, a conversion might be necessary.

Having the number of failures the fractions of severity and estimates of costs per severity class are needed. They are used to determine how many of the failures belong to which severity class. Then the number can be multiplied with the estimated costs per severity class.

This gives as the following equation for the future costs.

$$c_{fut} = \sum_{i=n}^u \frac{\sum_{s=1}^S f(i)P(s)c_{ext}(s)}{(1+D)^i}, \quad (3)$$

where  $n$  is the period in which we start the prediction,  $u$  is the upper limit of the prediction periods,  $S$  is the highest severity class,  $P(s)$  is the fraction or probability of severity class  $s$ , and  $c_{ext}(s)$  are the estimated external costs for a failure of severity class  $s$ .

### 3.3 Estimation Using Expert Opinion

Finally, we want to estimate the revenues that we have because of the usage of the defect-detection technique. These

are the external failure costs that we saved by finding and removing faults before releasing the software to the customer. Therefore we can use the same cost categories as above, i.e. fault removal, support, and compensation costs. The main difference is that we do not have an empirical basis to estimate the occurrence of the faults. Therefore, we use expert opinion to estimate the *mean time to failure (MTTF)* of each fault and the *severity* of the corresponding failure.

All the revenues have to be discounted to the present value. For this we need the discount factor  $D$  again that represents an average cost of capital. It is difficult to estimate the time until an external failure will occur and will result in further costs. This can be estimated based on the MTTF estimated earlier. All this assumes that we have a fixed granularity for the periods that is used also for the estimate of time periods for external failures. The severity is finally used again to estimate compensation costs.

Based on this the following equation can be used to calculate the net present value of the revenues.

$$r = \sum_{i=n}^u \frac{c_{remv}(i) + c_{sup}(i) + c_{comp}(i)}{(1+D)^i}, \quad (4)$$

where  $n$  is the period after the usage of the defect-detection technique,  $u$  is the upper limit of the estimation periods,  $c_{remv}$  are again the fault removal costs,  $c_{sup}$  the support costs, and  $c_{comp}$  possible compensation costs.

### 3.4 Quality Economics

By having established the metrics above, we can combine them in an equation for quality economics for defect-detection techniques. We have the direct measurable costs from the defect-detection technique usage, the predicted future costs from the failure behaviour of the software and we estimated the revenues based on MTTF and severity estimates for the found faults. Hence, we can give the equation for the net present value of the cash flows.

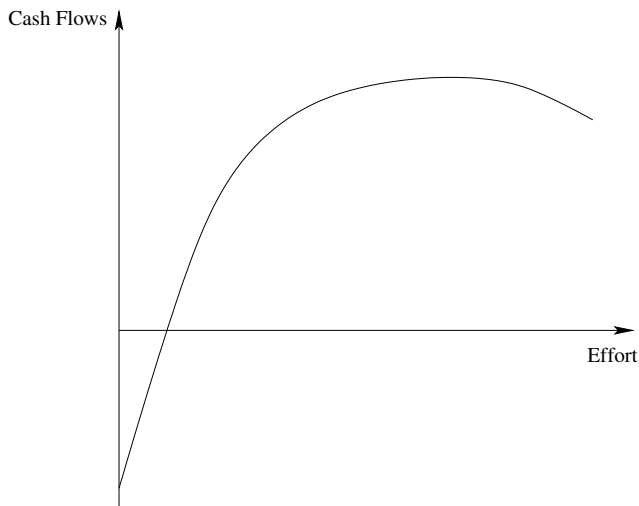
$$NPVCF = r - c_{direct} - c_{fut}. \quad (5)$$

This means that we subtract the directly measurable costs and the future costs from the revenues to get the net present value of the defect-detection technique which represents the benefit from the usage of the technique.

### 3.5 Characteristics for a Technique

The future costs can have costs for different techniques mingled together in case not only one technique was used because it is not clear which technique is responsible for the failures. Therefore the revenues and the direct costs constitute the characteristic cash flows for a specific technique. Most interesting is the characteristic curve of the cash flows in relation to the effort spent for the technique. Following the intuition each technique starts with negative cash flows as we have initial investments. With further usage of the technique the cash flows become positive rapidly until it reaches an area of satisfaction where less and less faults are found. Finally, with still more effort only the costs rise but no revenues are generated because only few or no new faults are found. Hence, the sum of the cash flows decreases. An example curve following this intuition is depicted in Fig. 3.

The shape of this curve is also justified by the so-called S-curve of software testing [7, 4] that shows that the search



**Figure 3: Typical characteristic cash flows for a technique**

for defects becomes less effective with more amount of time spent.

The establishment of such characteristic curves for several techniques requires extensive measurement experiments. However, having such curves allows the early planning of quality assurance using optimisation techniques. For this it might be necessary to normalise different curves from different projects and techniques by dividing each axis by a size metric like lines of code (LOC) to be able to compare different curves. We also need to investigate on which factors the curves depend. It might be the case that the curves are different in different domains or depending on the experience of the test engineers.

## 4. APPLICATION

We use the above developed cost model for an example evaluation of black-box system testing using the data from a field study [17]. This study analysed the efficiency of defect-detection techniques in a project of a German software and systems company. It was a project in the military sector lasting three years and resulting in a military information system.

The development in general followed the iterative and incremental process model *Rational Unified Process (RUP)* [10]. We analyse a late phase of system testing before a release to the users and use the later defect reports as basis for the prediction of the future reliability. Also the number of found defects by the testing technique is taken directly from the project data. However, we have no real data about testing and failure costs from the project. Therefore, we use hypothetical data for the sake of the example.

### 4.1 Direct Costs

First we need to calculate the direct measurable costs of the functional tests that we want to analyse. As we use highly automated test execution, we have considerable setup costs of 50,000 currency units. The system test ran for four weeks, whereas three testers worked on it in the first two weeks and five test engineers in the second two weeks. We

Year	Failures
1	1.95
2	0.62
3	0.38
4	0.28
5	0.22
6	0.18
7	0.15
8	0.13
9	0.12
10	0.10

**Table 1: Predicted number of failures for each year of operation**

calculate with 500 currency units per person per day. Finally, 8, 9, 8, and 14 faults were found in each week, respectively. We use an average of 200 units per fault for the fault removal. Using Eq. 1 and a discount factor of 5% this gives us the following direct costs:

$$C_{direct} = 50,000 + \frac{44,800}{1.05^1} = 92,666.67 \quad (6)$$

### 4.2 Future Costs

The next costs we want to predict are the future costs based on the prediction of a reliability model. For this we first need to define the fractions of the different severity classes or how probable each class is and how much a failure of each class will cost. For the sake of the example we define the following with severity 1 being the lowest and 5 the highest.

1. 0.9, 500
2. 0.05, 500
3. 0.03, 20,000
4. 0.019, 500,000
5. 0.001, 1,000,000

We then used the software reliability tool SMERFS [3] to predict the number of failures in each year of operation. For this we used the last 150 time between failure data and performed a maximum likelihood method inside the tool to estimate the model parameters and determine also the model that best fits the data. We chose to use the Musa-Okumoto logarithmic Poisson model [12] because the similarity measure *KS distance* recommended the usage of this model. Furthermore, we assume a usage of over 2400 CPU-days per year. This gives us the number of failures occurring each year in Tab. 1.

Using Eq. 3 yields future costs of 39,885.76.

### 4.3 Revenues

Finally, we determine the revenues that were generated by the functional tests. We evaluate each of the 39 found faults during the tests and estimate its MTTF and the severity of its impact. Based on this the expected period of the occurrence of the fault and subsequently the removal, support and possible compensation costs can be estimated. As the estimates for all the faults are too large for this paper we

Fault	MTTF	Severity	Exp. Period	$C_{remv}$	$C_{sup}$	$C_{comp}$
1	1,000	1	1	200	10,000	0
2	10	1	1	100	1,000	0
3	100	1	1	200	2,000	0
4	2,000	3	1	200	1,500	10,000
5	100	1	1	300	1,000	0
6	5,000	5	3	200	2,000	100,000
7	100	1	1	100	5,000	0
8	500	1	1	200	1,000	0

**Table 2: Matrix of the failure costs for the application**

only show the first eight faults that were found in the first week in Tab. 2. Note that the MTTF is given in CPU-days and we calculate with 2430 CPU-days/year.

Using the estimated costs from all weeks in Eq. 4 results in total revenues of 173,417.63.

#### 4.4 Quality Economics

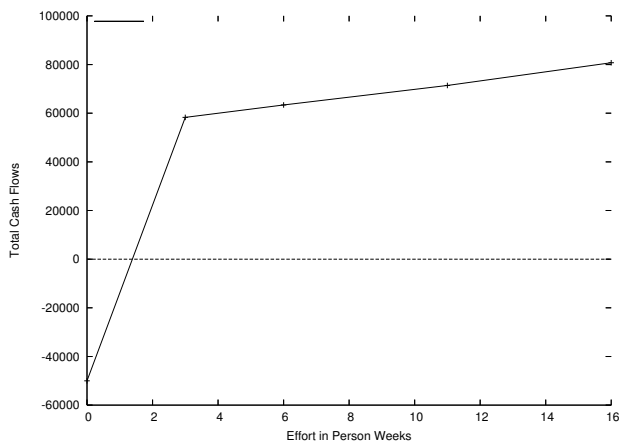
For a comprehensible evaluation of the quality costs and revenues, we finally calculate the net present values of the cash flows. Using Eq. 5 this gives the following:

$$\begin{aligned} NPVCF &= 173,417.63 - 92,666.67 - 39,885.76 \\ &= 40,865.20 \end{aligned} \quad (7)$$

Hence, the system tests yielded over 40,000 currency units by avoiding failures to occur in the field.

#### 4.5 Characteristics

We also want to analyse the characteristic curve of this technique by analysing it week by week and evaluating the change in cash flows. The result can be seen in Fig. 4.



**Figure 4: The characteristic cash flows for the testing technique**

As we only have five sample points including the start of the tests, the curve is not directly the same as the one in Fig. 3 but resembles the general appearance. Note that the curve still increases what indicates that we used the technique not necessarily to its optimum but we might have benefited from investing more effort.

## 5. RELATED WORK

Based on experience from the manufacturing area quality cost models have been developed explicitly for software [8, 16, 9] which stay rather abstract. If calculations are possible at all, only coarse-grained estimates such as defect densities are used by these models.

Humphrey presents in [5] his understanding of software quality economics. The defined cost metrics do not represent monetary values but only fractions of the total development time. Furthermore the effort for testing is classified as failure cost instead of appraisal cost.

Collofello and Woodfield propose in [2] a metric for the cost efficiency but do not incorporate fault MTTFs and severities explicitly. Furthermore the metric lacks consideration of the cost of capital and therefore the net present value is not used.

Pham describes in [14] various flavours of a software cost model for the purpose of deciding when to stop testing. It is a representative of models that are based on reliability models. The main problem with such models is that they are only able to analyse testing, not other defect-detection techniques and that the differences of different test techniques cannot be considered. However, they are highly useful as they allow a mathematical optimisation of the test time which is currently not possible with our model.

Based on the general model for software cost estimation COCOMO, the COQUALMO model was specifically developed for quality costs in [1]. This model is different in that it is aiming at estimating the costs beforehand and that it uses only coarse-grained categories of defect-detection techniques. In our work, we want to analyse the differences between techniques in more detail.

Holzmann describes in [4] his understanding of the economics of software verification. He describes some trends and hypotheses that are similar to ours and of which we can support most ideas although they need empirical justification at first.

## 6. CONCLUSIONS

We finally want to summarise the presented model and finish with directions for further research.

### 6.1 Summary

We present a model for quality economics of defect-detection techniques. It is based (1) on the experience from the manufacturing area that has been brought to the software domain and (2) on software reliability models to predict the future failure behaviour.

Moreover, our model improved and refined existing mod-

els by using expert opinion on the revenues caused by the faults that were found by the technique. We use the efficiency metrics MTTF and severity to improve and detail the cost estimates. An reliability model is used to analyse the future failure behaviour and therefore to predict such costs. These are brought together with directly measured costs of the technique to calculate the net present value of the cash flows which represents the benefit from the defect-detection technique.

We also showed using data from an industrial project how the model can be used to analyse the costs and revenues of a technique. Very interesting is furthermore the characteristic cost curve of the test technique which already allows first evaluations of the optimal usage.

## 6.2 Future Work

Firstly, there are many ways in which the model can be extended. We want to use the theoretical model of combining diverse techniques from [11] to incorporate those effects in our cost model. This obviously aims mainly on the mechanisms of finding faults that could potentially be found by several techniques.

Another point for further research is the incorporation of all types of maintenance costs into the cost model. Especially for software the maintenance costs are important because software is in general changed easily. Therefore adaptive, preventive, and perfective maintenance need their place in the cost model as well. This means that these activities would be added to the cost side but also the resulting revenues when corresponding changes are simpler have to be included.

Furthermore, it is important to find a means to quantify and predict possible opportunity costs. That are mostly lost sales because of a image loss or annoyed customers. Also the time to market has to be considered.

On the basis of the characteristic curves for different techniques it should be possible to mathematically optimise the usage of different techniques at the planning stage of a project.

However, for all of this to yield solid information, we need to assess the impact of model uncertainties. This is important as the whole revenues are based only on expert opinion which are unreliable. Sensitivity analysis or Monte Carlo simulation are approaches we currently investigate for this.

Secondly, we want to detail the model to more specific types of defect-detection techniques, for example considering the details of inspections or automated testing.

Based on the evaluation of model-based testing in [15] a further study is planed that emphasises the cost aspects of the technique. In this study the developed quality cost model will be used for the evaluation.

A further application example will be to evaluate static analysis tools, also called bug finding tools, based on their costs using this cost model. First experiences with this tools are documented in [19].

## 7. REFERENCES

- [1] S. Chulani and B. Boehm. Modeling Software Defect Introduction and Removal: COQUALMO (COConstructive QUALity MOdel). Technical Report USC-CSE-99-510, University of Southern California, Center for Software Engineering, 1999.
- [2] J. Collofello and S. Woodfield. Evaluating the Effectiveness of Reliability-Assurance Techniques. *Journal of Systems and Software*, 9(3):191–195, 1989.
- [3] W. Farr and O. Smith. Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide. Technical Report NAVSWC TR-84-373, Naval Surface Weapons Center, 1993.
- [4] G. Holzmann. Economics of Software Verification. In *Proc. 2001 ACM SIGPLAN–SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE’01)*, pages 80–89. ACM Press, 2001.
- [5] W. Humphrey. *A Discipline for Software Engineering*. The SEI Series in Software Engineering. Addison-Wesley, 1995.
- [6] C. Jones. *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, 1991.
- [7] S. Kan, J. Parrish, and D. Manlove. In-Process Metrics for Software Testing. *IBM Systems Journal*, 40(1):220–241, 2001.
- [8] S. Knox. Modeling the costs of software quality. *Digital Technical Journal*, 5(4):9–16, 1993.
- [9] H. Krasner. Using the Cost of Quality Approach for Software. *CrossTalk. The Journal of Defense Software Engineering*, 11(11), 1998.
- [10] P. Kruchten. *The Rational Unified Process. An Introduction*. Addison-Wesley, 2nd edition, 2000.
- [11] B. Littlewood, P. 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.
- [12] J. 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.
- [13] G. Myers. *The Art of Software Testing*. John Wiley & Sons, 1979.
- [14] H. Pham. *Software Reliability*. Springer, 2000.
- [15] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner. One Evaluation of Model-Based Testing and its Automation. In *Proc. 27th International Conference on Software Engineering (ICSE’05)*, 2005.
- [16] S. Slaughter, D. Harter, and M. Krishnan. Evaluating the Cost of Software Quality. *Communications of the ACM*, 41(8):67–73, 1998.
- [17] S. Wagner. Efficiency Analysis of Defect-Detection Techniques. Technical Report TUMI-0413, Institut für Informatik, Technische Universität München, 2004.
- [18] S. Wagner. Towards Software Quality Economics for Defect-Detection Techniques. In *Proc. 29th Annual IEEE/NASA Software Engineering Workshop*, 2005.
- [19] S. Wagner, J. Jürjens, C. Koller, and P. Trischberger. Comparing Bug Finding Tools with Reviews and Tests. In *Proc. 17th International Conference on Testing of Communicating Systems (TESTCOM’05)*, volume 3502 of *LNCS*, pages 40–55. Springer, 2005.