Software Quality Economics for Combining Defect-Detection Techniques*

Stefan Wagner

Institut für Informatik
Technische Universität München
Boltzmannstr. 3, D-85748 Garching, Germany

Abstract. Software defect-detection techniques such as tests or reviews are still the prevalent means to assure the quality of a program. Although these techniques constitute a significant part of the overall development costs, their economics and interplay is still not fully understood. This paper proposes a model of software quality economics for defect-detection techniques with an emphasis on the combination of diverse techniques. This is important because in real development projects a technique is never used in isolation but in combination with others. This, however, makes the evaluation of a technique very difficult because of the influences of early used techniques on the ones that are later used. We base our combination economics on a model for software quality economics of single techniques and extend it by a model for the diversity of defect-detection techniques. This allows (1) to estimate the effects of a combination and (2) to remove such influences when evaluating a single technique.

1 Introduction

Quality assurance is a costly part of software development. Some estimates [18] relate up to 50% of the development costs to testing. Jones [9] still assigns 30 – 40% to quality assurance and defect removal. Defect-detection techniques are the mostly used means 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, we can optimise the usage. Hence, for a cost-effective use we need an economic model of defect-detection techniques.

Of special importance is here the combination of diverse defect-detection techniques. It is a well-known heuristics that the usage of different techniques yields better results in fault-finding than using just one. Littlewood et al. were even able to proof that in their model of the combination of diverse defect-detection techniques [17]. However, to model the effect of the combination especially on costs and benefits is still difficult because later techniques cannot find faults

* This research was supported in part by the DFG under project name *InTime*.

that were found earlier. Therefore, the problem is how to know which faults a specific technique would have found when used in isolation.

We base our model for the combination of defect-detection techniques on a model for the economics of single techniques [22, 24]. This model uses direct measurement, expert opinion and reliability models. The expert opinion is the basis for the estimation of the revenues by finding faults and the reliability model yields the future costs of failures at the client site. The combination is based on the above mentioned model for diverse defect-detection techniques [17]. It introduces a technique-specific difficulty function that is used to adjust the costs and benefits.

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 and their combination can be evaluated and compared on an economical basis.

Contribution. This paper proposes a structured approach to the evaluation of defect-detection techniques based on expert opinion and predicted failure behaviour. It allows a more precise cost analysis than similar approaches and has the unique ability to adjust to the combination of diverse techniques.

Outline. We first describe the types of costs that are used in our economics model in Sec. 2. Than the model for the usage of a single defect-detection technique is described in Sec. 3. This is extended in Sec. 4 by the introduction of diversity parameters to adjust the model. We describe an example in Sec. 5 and give related work in Sec. 6. We close with final conclusions and future work in Sec. 7.

2 Types of Costs

An area that is under research in various domains is 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 [11, 12, 21]. A graphical overview with some extensions that are described below is given in Fig. 1.

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

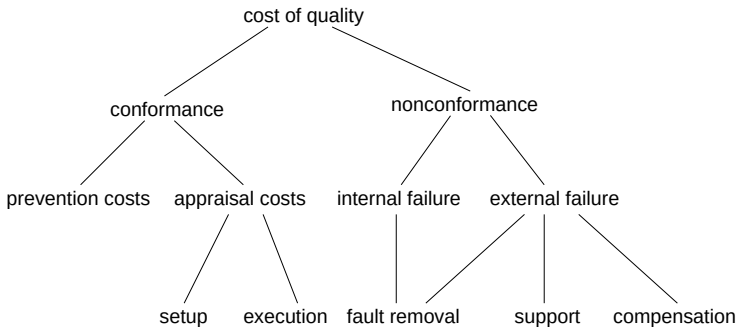


Fig. 1. Cost types

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 using 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 economics model. Note that there are more types that could be included, for example, preventive 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 failures 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 [9]. 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.

3 Single Economics

The following model for a single defect-detection technique is based on [22, 24]. The main metric 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 [21].

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 during 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 where n is the number of periods the technique was used.

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

3.2 Prediction Using a Reliability Model

The second part of the costs come from the occurrence of failures after the quality assurance ended and the software was delivered to the user. For this we need a means to predict the total number of residual faults or even better the failure behaviour of the software during operation. We want incorporate this 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 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

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

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.

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

where m 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.

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

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 directly 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.

$$NPVCF = r - c_{direct} - c_{fut}. \tag{5}$$

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. 2.

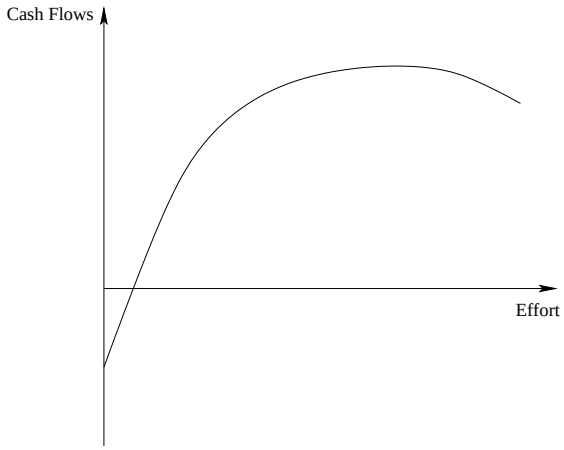


Fig. 2. Typical characteristic cash flows for a technique

The shape of this curve is also justified by the so-called S-curve of software testing [10, 5] that shows that the search for defects becomes less effective with more amount of time spent from a certain point in time on.

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. For the cash flows the usage of the internal rate of return allows a comparison because it is measured in percentages not monetary values. 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. Another problem is that the curve will change if other techniques are used before the technique under consideration.

The hypothesis behind all this is that each defect-detection technique has a characteristic mix of the underlying metrics involved in the cash flow calculation in relation to the effort spent resulting in such characteristic curves. A single metric is not able to capture the efficiency of a technique completely. Therefore, we use the following metrics:

- Severity of the failures of found faults
- MTTF of the found faults
- Effort to find a defect
- Tool and environment costs
- Complexity of fault localisation

All these metrics can be expressed in economic terms which hence constitutes the common unit for these measures and the only possibility to combine them.

4 Combined Economics

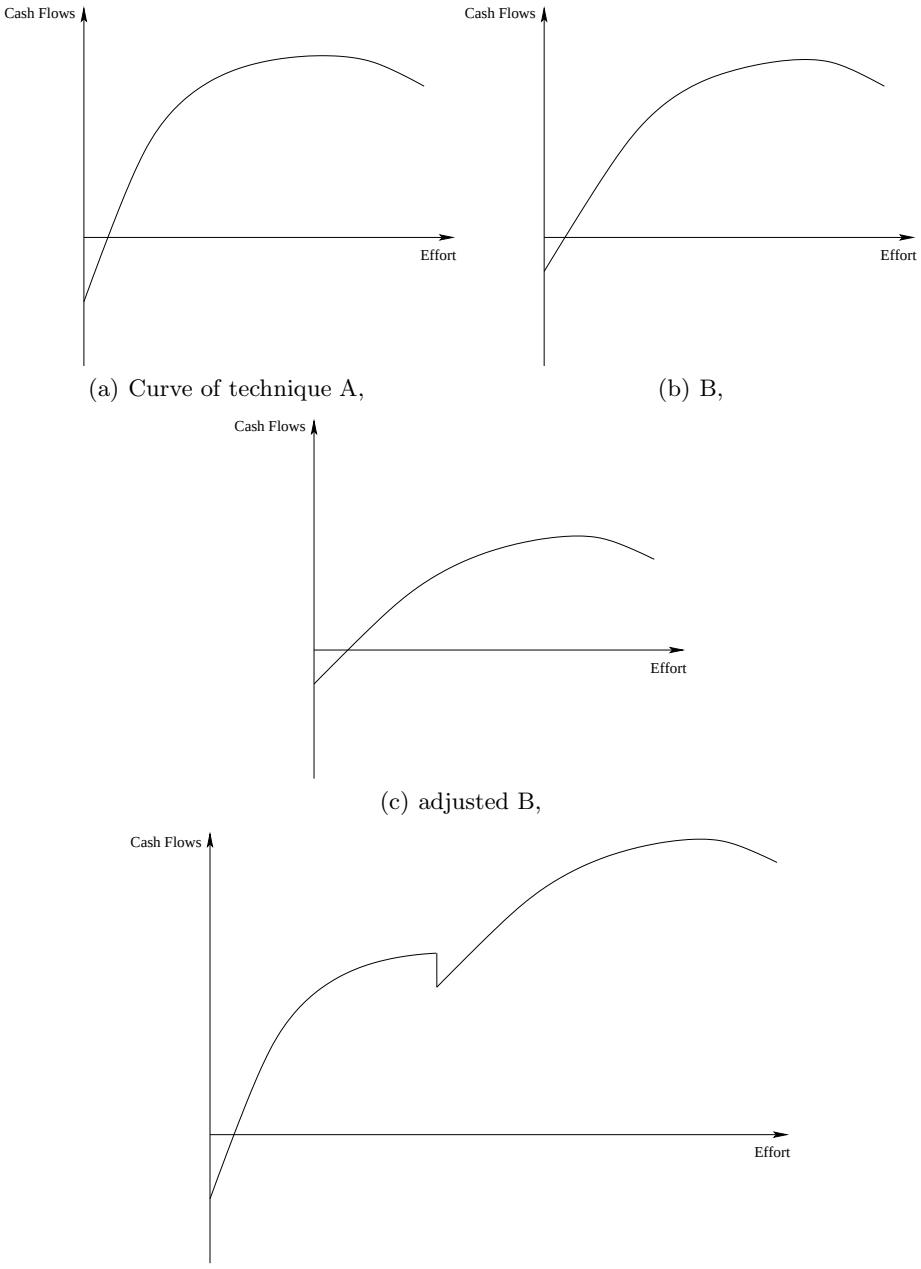
An inherent problem with the evaluation of defect-detection techniques is always that they are typically applied sequentially and later applied techniques always have more difficulties finding faults than the earlier ones. In our model it means that we have lower revenues r but also lower fault removal costs c_{remv} in techniques that are later used.

The curves in Fig. 4 show an illustrative example to aid the intuition of the effects of a combination of techniques. Fig. 3(a) and Fig. 3(b) show the characteristic curves of the techniques A and B, respectively. The curve that describes B when A was used before is depicted in Fig. 3(c). The difference is only that the curve is pushed down depending on the diversity of A and B. The combination of both techniques in a single curve can look like Fig. 3(d). In this case we used A until the apex of the curve was reached and then stop with A and start B.

We define a function $comb(T, E, m)$ that introduces the effect of the combination of several techniques into the model. The function needs the technique under consideration T, the set of earlier applied techniques E, and the monetary part of the model m that is to be adjusted as parameters.

We base the economics for the combination of defect-detection techniques on a model from Littlewood et al. [17] which is in turn based on models of design diversity for fault-tolerant software [16, 4]. A main idea in this model is that there exists a difficulty function $\theta_A(i)$ that describes the difficulty of a technique A to find the fault i. In other words, it is the probability that a usage of the defect-detection technique finds this fault. It is important to note that this function is believed to vary from one fault to another. Based on this the ineffectiveness of a technique A is given in [17] as:

$$P(A \text{ fails to detect a randomly chosen fault}) = \sum_i p_i^* \cdot \theta_A(i) = E_{p^*}(\theta_A(i)), \quad (6)$$



(a) Curve of technique A, (b) B,

(c) adjusted B,

(d) and A and B combined.

Fig. 3. Effect of the combination on the characteristic curves

where p_i^* is the probability distribution of the faults to be present in a program and E_{p^*} denotes the mean obtained with respect to this probability distribution.

However, for our purposes it is more interesting to get the proportion of faults that are found by a technique:

$$E_{p^*}(1 - \theta_A(i)) = \phi_A. \tag{7}$$

Furthermore, we need the proportion of the faults that could be found by both techniques. We can give this based on [17] as

$$E_{p^*}((1 - \theta_A(i)) \cdot (1 - \theta_B(i))) = \phi_{AB}. \tag{8}$$

The result depends here on the covariance of the difficulty functions of the techniques. If we have a negative covariance, the techniques are strongly diverse and the effect will be small. A positive covariance makes the effect stronger.

Having established these equations, we can use them as rough approximation in our function *comb*, i.e. to adjust parts of the single economics model to account for combination.

$$comb(T, E, m) = \frac{m(\phi_T - \sum_{e \in E} \phi_{Te} + \sum_{e \in E} \sum_{f \in E} \phi_{ef})}{\phi_T}, \tag{9}$$

where $f > e$. The function adjusts a monetary value m from a technique T according to the combination with the numbered set of earlier used techniques E. We divide by ϕ_T to normalise m and then multiply by the difference between the proportion of faults that are found by T in isolation and the proportion of faults that are also found by earlier techniques.

This is interesting if there is existing data about a technique and we want to know the effect of combining it with another technique that is used before. However, we might also be interested in the other direction. If we had data about the combination of two techniques, we would be able to adjust it to find out how the techniques would have behaved in isolation. For this we define the reverse function:

$$\overline{comb}(T, E, m) = \frac{m \cdot \phi_T}{\phi_T - \sum_{e \in E} \phi_{Te} + \sum_{e \in E} \sum_{f \in E} \phi_{ef}}, \tag{10}$$

where $f > e$. In the economics model, two parts are actually affected by the combination. This is (1) the revenues r and (2) the fault removal costs c_{remv}. Given the function above, we can redefine them as:

$$r' = comb(T, E, r), \tag{11}$$

and

$$c'_{remv} = comb(T, E, c_{remv}). \tag{12}$$

These new definitions can be substituted for the old ones in the economics model for single techniques to account for the effect of combination. The definitions using the reverse function $\overline{comb}$ can be written accordingly. However,

what is needed for both functions are typical values for the difficulty functions θ to be able to estimate the model parameters ϕ_A and ϕ_{AB}. These may come from earlier experiments with the same techniques or must be estimated with the current data, e.g. information on the defect types that are found.

5 Example

There are currently no applications of the model to real world data. Therefore, we use the example from [17] to illustrate our approach of the combination of defect-detection techniques. We do not go into details of the single economics model. A detailed example can be found in [24].

The two techniques used are code checking and functional testing. They were applied in controlled laboratory experiments with computer science students. The students applied the techniques to a railway signaling system. In [17] estimates are given for all θ values for the eight seeded faults. We can use this to calculate the function *comb* to adjust for combination.

We do not have cost data for these techniques. For the sake of the example we choose two hypothetical representatives of each technique. We assume we have a usage of code checking with $c_{setup} = 1000$ monetary units and testing with $c_{setup} = 2000$. Which technique found which fault can be seen in Tab. 1.

Fault id	Checking	Testing
F11	x	-
F12	-	-
F13	-	x
F14	x	-
F15	x	-
F16	-	x
F17	-	x
F18	x	x

Table 1. Matrix for the faults and the two techniques

We consider only one period and therefore the discounting factor is not used. Because fault removal is easier after code checking than after testing, we estimate $c_{remv} = 200$ for checking and $c_{remv} = 500$ for testing. The execution costs (c_{exec}) are 1000 for checking and 800 for testing. We further estimate the revenues r to be 3000 for checking and 5000 for testing. The future costs c_{fut} are 2000 in both cases.

This gives us a net present value for each technique as $NPVCF_C = -1200$ and $NPVCF_T = -500$. Both are negative because the future costs are too high. This indicates that more quality assurance is needed. However, the characteristics for each technique are only c_{direct} and r. These are positive in both cases. We now adjust the characteristic values for the techniques if combined by checking the

code first and then test. That means we have to adjust the revenues and removal costs of the tests. Based on the estimates in [17] we can calculate values for ϕ_T and ϕ_{CT}. This gives us $\phi_T = .52$ and $\phi_{CT} = .28$.

With these parameters we can use the function *comb* to adjust r and c_{remv} because the data we have is from independent applications of the techniques:

$$\begin{aligned} r' &= comb(T, \{C\}, r) = \tfrac{5000 \cdot (.52 - .28)}{.52} = 2,293.05 \\ c'_{remv} &= comb(T, \{C\}, c_{remv}) = \tfrac{200 \cdot (.52 - .28)}{.52} = 91.72 \end{aligned} \tag{13}$$

Therefore, a combination in the proposed way would give us a net present value of $-1,798.67$ when incorporating the adjusted values. It is obvious that the unadjusted sum of the values would have resulted in a far too high net present value of 500.

6 Related Work

Based on experience from the manufacturing area quality cost models have been developed explicitly for software [11, 21, 12] 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 [7] 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 [3] 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 [19] 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 [2]. 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.

Boehm et al. present in [1] the iDAVE model that uses COCOMO II and COQUALMO. This model allows a thorough analysis of the ROI of dependability. The main difference is again the granularity. Only an average cost saving per found defect is considered.

Building on iDAVE, Huang and Boehm propose a value-based approach for determining how much quality assurance is enough in [6]. In some respect that

work is also more coarse-grained than our work because it considers only the defect levels from COQUALMO. However, it contains an interesting component that deals with time to market costs that are currently missing from our model.

A somehow similar model to COQUALMO in terms of the description of the defect introduction and removal process is described in [8]. However, it offers means to optimse the resource allocation. The only measure for defect-detection techniques used is defect removal efficiency.

Holzmann describes in [5] 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.

Kusumoto et al. describe in [14, 13] a metric for cost effectiveness mainly aimed at software reviews. They introduce the concept of virtual software test costs that denote the testing cost that have been needed if no reviews were done. This implies that we always want a certain level of quality.

In [15] several combination approaches for the techniques inspection and testing are discussed. These include special analyses and orthogonal defect classification to improve the interplay of the techniques. When analysing these special combination these results are very useful but cannot be generalised to an universal model of technique combination.

7 Conclusions

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

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 models 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.

However, the main contribution is the introduction of a possibility to adjust the model to account for the combination of diverse defect-detection techniques. For this we use the model of Littlewood et al. [17] to describe the diversity, i.e. the proportion of faults that are found by different techniques. We are not aware of a software quality economics model that provides means for that.

Future Work. A main 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 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.

We also 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 [20] 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 these tools are documented in [23]. Especially, the defect types were analysed in this study which could be used as a starting point for the estimation of the ϕ parameters.

Acknowledgments

We are grateful to Tilman Seifert for the joined work on the single economics.

References

1. B. Boehm, L. Huang, A. Jain, and R. Madachy. The ROI of Software Dependability: The iDAVE Model. *IEEE Software*, 21(3), 2004.
2. S. Chulani and B. Boehm. Modeling Software Defect Introduction and Removal: COQUALMO (COnstructive QUALity MOdel). Technical Report USC-CSE-99-510, University of Southern California, Center for Software Engineering, 1999.
3. J.S. Collofello and S.N. Woodfield. Evaluating the Effectiveness of Reliability-Assurance Techniques. *Journal of Systems and Software*, 9(3):191–195, 1989.
4. D.E. Eckhardt and L.D. Lee. A Theoretical Basis of Multiversion Software Subject to Coincident Errors. *IEEE Transactions on Software Engineering*, 11(12):1511–1517, 1985.
5. G.J. 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.

6. L. Huang and B. Boehm. Determining How Much Software Assurance is Enough? A Value-based Approach. In *Proc. Seventh Workshop on Economics-Driven Software Research (EDSER-7)*, 2005.
7. W.S. Humphrey. *A Discipline for Software Engineering*. The SEI Series in Software Engineering. Addison-Wesley, 1995.
8. P. Jalote and B. Vishal. Optimal Resource Allocation for the Quality Control Process. In *Proc. 14th International Symposium on Software Reliability Engineering (ISSRE'03)*, 2003.
9. C. Jones. *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, 1991.
10. S.H. Kan, J. Parrish, and D. Manlove. In-Process Metrics for Software Testing. *IBM Systems Journal*, 40(1):220–241, 2001.
11. S.T. Knox. Modeling the costs of software quality. *Digital Technical Journal*, 5(4):9–16, 1993.
12. H. Krasner. Using the Cost of Quality Approach for Software. *CrossTalk. The Journal of Defense Software Engineering*, 11(11), 1998.
13. S. Kusumoto. *Quantitative Evaluation of Software Reviews and Testing Processes*. PhD dissertation, Osaka University, 1993.
14. S. Kusumoto, K. Matasumoto, T. Kikuno, and K. Torii. A New Metric for Cost Effectiveness of Software Reviews. *IEICE Transactions on Information and Systems*, E75-D(5):674–680, 1992.
15. O. Laitenberger, B. Freimut, and M. Schlich. The Combination of Inspection and Testing Technologies. Technical Report 070.02/E, Fraunhofer IESE, 2002.
16. B. Littlewood and D.R. Miller. Conceptual Modeling of Coincident Failures in Multiversion Software. *IEEE Transactions on Software Engineering*, 15(12):1596–1614, 1989.
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. G.J. Myers. *The Art of Software Testing*. John Wiley & Sons, 1979.
19. H. Pham. *Software Reliability*. Springer, 2000.
20. 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)*. ACM Press, 2005.
21. S.A. Slaughter, D.E. Harter, and M.S. Krishnan. Evaluating the Cost of Software Quality. *Communications of the ACM*, 41(8):67–73, 1998.
22. S. Wagner. Towards Software Quality Economics for Defect-Detection Techniques. In *Proc. 29th Annual IEEE/NASA Software Engineering Workshop*, 2005.
23. 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.
24. S. Wagner and T. Seifert. Software Quality Economics for Defect-Detection Techniques Using Failure Prediction. In *Proc. 3rd Workshop on Software Quality (3-WoSQ)*. ACM Press, 2005.