L-O. Damm, L. Lundberg and C. Wohlin, "Determining the Improvement Potential of a Software Development Organization through Fault Analysis: A Method and a Case Study", Proceedings of European Software Process Improvement Conference in Lecture Notes in Computer Science Series, Editor: Torgeir Dingsöyr, Springer Verlag, Heidelberg, Germany, 2004.

Determining the Improvement Potential of a Software Development Organization through Fault Analysis: A Method and a Case Study

Lars-Ola Damm[1, 2], Lars Lundberg[2], Claes Wohlin[2]

[1] Ericsson AB, Ölandsgatan 1,
Box 518, SE-371 23, Karlskrona,
Lars-Ola.Damm@ericsson.com
[2] School of Engineering, Blekinge Institute of Technology,
Box 520, SE-372 25 Ronneby,
{Lars-Ola.Damm, Lars.Lundberg, Claes.Wohlin}@bth.se

Abstract. Successful software process improvement depends on the ability to analyze past projects and determine which parts of the process that could become more efficient. One typical data source is the faults that are reported during product development. From an industrial need, this paper provides a solution based on a measure called faults-slip-through, i.e. the measure tells which faults that should have been found in earlier phases. From the measure, the improvement potential of different parts of the development process is estimated by calculating the cost of the faults that slipped through the phase where they *should* have been found. The usefulness of the method was demonstrated by applying it on two completed development projects at Ericsson AB. The results show that the implementation phase had the largest improvement potential since it caused the largest faults-slip-through cost to later phases, i.e. 81 and 84 percent of the total improvement potential in the two studied projects.

1 Introduction

For many modern software development organizations it is of crucial importance to reduce development cost and time-to-market while still maintaining a high level of product quality. Such organizations seek specialized processes that could give them rapid improvements. However, they often overlook existing routinely collected data that can be used for process analysis [4]. One such data source is fault reports since avoidable rework accounts for a significant percentage of the total development time, i.e. between 20-80 percent depending on the maturity of the development process [10]. In fact, related research states that fault analysis is the most promising approach to software process improvement [5].

A software development department at Ericsson AB develops component-based software for the mobile network. In order to stay competitive, they run a continuous process improvement program where they regularly need to decide where to focus the current improvement efforts. However, as considered a common reality in industry, the department is already aware of potential improvement areas; the challenge is to

prioritize the areas to know where to focus the improvement work [12]. Without such a decision support, it is common that improvements are not implemented because organizations find them difficult to prioritize [12]. Further, if a suggested improvement can be supported with data, it becomes easier to convince people to make changes more quickly [5]. As stated above, fault statistics is one useful information source in software process improvement; therefore, the general research question of this paper is:

> How can fault statistics be used for determining the improvement potential of different phases/activities in the software development process in an organization?

A commonly used approach for fault analysis is classification of faults from their causes, e.g. root cause analysis [9]. Although root cause analysis can provide valuable information about what types of faults the process is not good at preventing/removing, the technique is cost intensive and therefore not easy to apply on larger populations of faults. Further, the classification procedure should not require advanced skills that normally only professional analysts have; ordinary developers must be able to perform the classification [1]. Finally, and most importantly, root cause analysis does not quantify what the improvement potential of different phases is.

In order to satisfy the above given requirements, this paper introduces a method in which a central part is a 'faults-slip-through' measure. That is, the faults are classified according to whether they slipped through the phase where they should have been found. This approach to fault classification has been used before [1,6]; the main difference with the approach here is that it also calculates the improvement potential by relating the result of the fault classification with the average cost of finding and repairing the faults. From the classified faults, the method measures the improvement potential by multiplying the faults-slip-through distribution with the average benefit of finding a fault earlier. In order to verify the applicability of the method, the paper also provides an empirical case study where the method is applied on the faults reported in two finished development projects at Ericsson AB.

The paper is outlined as follows. Section 2 describes the proposed method for how to determine the improvement potential of an organization. Section 3 demonstrates the applicability of the method through an empirical case study. Section 4 discusses the validity and implications of the results and Section 5 concludes the work.

2 Method

2.1 Estimation of Improvement Potential

The purpose of this paper is to demonstrate how to determine the improvement potential of a development process from historical fault data. This section describes the selected method for how to achieve this through the following three steps:

(1) Determine which faults that could have been avoided or at least found earlier
(2) Determine the average cost of finding faults in different phases.

(3) Determine the improvement potential from the results in (1) and (2).

In this context, a fault is defined as an anomaly that causes a failure [7].

When using fault data as basis for determining the improvement potential of an organization's development process, the essential analysis to perform is whether the faults could have been avoided or at least have been found earlier. As previously mentioned, the introduced measure for determining this is called 'faults-slip-through', i.e. whether a fault slipped through the phase where it should have been found. The definition of it is similar to measures used in related studies, e.g. phase containment metrics where faults should be found in the same phase as they were introduced [6], and goodness measures where faults should be found in the earliest possible phase [1]. In practice, the only difference between the faults-slip-through definition and the other definitions is when a fault is introduced in a certain phase but it is not efficient to find in the same phase, e.g. a certain test technique might be required to simulate the behaviour of the function. Table 1 provides a fictitious example of faults-slip-through between arbitrarily chosen development phases. The columns represent in which phase the faults were found (PF) and the rows represent where the faults should have been found (Phase Belonging, PB). For example, 25 of the faults that were found in Function Test should have been found during implementation (e.g. through inspections or unit tests). Further, the rightmost column summarizes the amount of faults that belonged to each phase whereas the bottom row summarizes the amount of faults that were found in each phase. For example, 49 faults belonged to the implementation phase whereas most of the faults were found in Function Test (50).

Table 1. Fictious example of faults-slip-through data (nr. faults found, belonging /phase)

PF: / PB:	Design	Impl.	Function Test	System Test	Operation	**Total belonging/phase**
Design	1	1	10	5	1	**18**
Impl.		4	25	18	2	**49**
Function Test			15	5	4	**24**
System Test				13	2	**15**
Operation					0	**0**
Tot. found/phase	**1**	**5**	**50**	**41**	**9**	**106**

When having all the faults categorized according to the faults-slip-through measure, the next step is to estimate the cost of finding faults in different phases. Several studies have shown that the cost of finding and fixing faults increases more and more the longer they remain in a product [3, 10]. However, the cost-increase varies significantly depending on the maturity of the development process and on whether the faults are severe or not [10]. Therefore, the average fault cost in different phases needs to be determined explicitly in the environment where the improvement potential is to be determined (see a fictitious example in Table 2). This measure could either be obtained through the time reporting system or from expert judgments, e.g. a questionnaire where the developers/testers that were involved in the bug-fix give an estimate of the average cost.

Table 2. Fictious example of average fault cost/phase found

	Design	Implementation	Function test	System test	Operation
Average fault cost	1	2	10	25	50

Expert judgments are a fast and easy way to obtain the measures; however, in the long-term, fully accurate measures can only be obtained by having the cost of every fault stored with the fault report when repairing it. That is, when the actual cost is stored with each fault report, the average cost can be measured instead of just being subjectively estimated. Further, when obtaining these measures, it is important not just to include the cost of repairing the fault but also fault reporting and re-testing after the fault is corrected.

The third step (e.g. the improvement potential) is determined by calculating the difference between the cost of faults in relation to what the fault cost would have been if none of them would have had slipped through the phase where they were supposed to be found. Fig. 1 provides the formulas for making such a calculation and as presented in the table in the figure, the improvement potential can be calculated in a two-dimensional matrix. The equation in the figure provides the actual formula for calculating the improvement potential for each cell (IP_{xx}). $PF_{x\ total}$ and $PBx_{\ total}$ are calculated by summarizing the corresponding row/column. As illustrated rightmost in the figure, the average fault cost (as discussed in the previous paragraph), need to be determined for each phase before using it in the formula (IP_{xx}). In order to demonstrate how to use and interpret the matrix, Table 3 provides an example calculation by applying the formulas in Fig. 1 on the fictitious values in Table 1 and Table 2.

	PF_1	PF_2	PB_{total}
PB_1	IP_{11}	IP_{12}	$PB_{1\ total}$
PB_2	IP_{21}	IP_{22}	$PB_{2\ total}$
$PF_{\ total}$	$PF_{1\ total}$	$PF_{2\ total}$	$(PB/PF)_{total}$

AvFC
PB_1 AvFC
PB_2 AvFC

PF = Phase found, PB =Phase belonging, AvFC=Average Fault Cost
IP = Improvement potential

IP_{xx} = (Nr faults bel. (PBx) * PB_1 AvFC) – (Nr faults bel. (PBx) * PBxAvFC)

Fig. 1. Matrix formula for calculation of improvement potential

In Table 3, the most interesting cells are those in the rightmost column that summarizes the total cost of faults in relation to fault belonging and the bottom row that summarizes the total unnecessary cost of faults in relation to phase found. For example, the largest improvement potential is in the implementation phase, i.e. the phase triggered 710 hours of unnecessary costs in later phases due to a large faults-slip-through from it. Note that taking an action that removes the faults-slip-through from the implementation phase to later phases will increase the fault cost of the implementation phase, i.e. up to 49 hours (1 hour/fault times 49 faults that according to Table 2 belonged to the implementation phase). Further, System Test is the phase that suffered from the largest excessive costs due to faults slipped through (609 hours). However, when interpreting such excessive costs, one must be aware of that some sort of investment is required in order to get rid of them, e.g. by adding code inspections.

Thus, the potential gain is probably not as large as 609 hours. Therefore, the primary usage of the values is to serve as input to an expected Return On Investment (ROI) calculation when prioritizing possible improvement actions.

PF: PB	Design	Impl.	Function Test	System Test	Operation	**Total PB/phase**
Design	1*1-1*1 = 0	1*2-1*1 = 1h	10*10-10*1 = 90h	5*25-5*1 = 120	1*50-1*1 = 49	**260h**
Impl.		4*2-4*2 = 0	25*10-25*2 = 200h	18*25-18*2 = 414h	2*50-2*2 = 96h	**710h**
Function Test			15*10-15*10 = 0	5*25-5*10 = 75h	4*50-4*10 = 160h	**235h**
System Test				13*25-13*25 = 0	2*50-2*25 = 50h	**50h**
Operation					0	**0h**
Total potential/ PF		**1h**	**290h**	**609h**	**355h**	**1255h**

Table 3. Example of calculation of improvement potential (hours)

When measured in percent, the improvement potential for a certain phase equals the improvement potential in hours divided with the total improvement potential (e.g. in the example provided in Table 3, the improvement potential of System Test = 609/1255=49%). In the case study reported below, the measurements are provided in percent (due to confidentiality reasons).

In related work, calculations on the improvement potential from faults have been applied by calculating the time needed in each phase when faults were found when supposed to in comparison to when they were not [11]. Although the results of using such an approach are useful for estimating the effect of implementing a certain improvement, they require measurements on the additional effort required for removing the faults earlier. Such measurements require decisions on what improvements to make and estimates of what they cost and therefore they cannot be used as input when deciding in which phases to focus the improvements and what the real potential is.

3 Results from Applying the Method

The applicability of the described method was evaluated by using it on the faults reported in two projects at a department at Ericsson AB. The projects developed new functionality to be included in new releases of the two different products. Hence, previous versions of the products were already in full operation at customer sites. Further, the projects used the same processes and tools and the products developed in the projects were developed on the same platform (i.e. the platform provides a component-based architecture and a number of platform components that are used by both

products). The products were developed mainly in C++ except for a Java-based graphical user interface that constitutes minor parts of each product. Apart from the platform components, each product consists of about 10-30 components and each component consists of about 5-30 classes. The reason for studying more than one project was to be able to strengthen the validity of the results, i.e. two projects that were developed in the same environment and according to the same development process should provide similar results (except for eventual known events in the projects that affected the results). Further, two projects were chosen since the selected projects were the only recently finished projects and because earlier finished projects were not developed according to the same process. Thereby, it was the two selected projects that could be considered as representative for the organization.

The reported faults originated from the test phases performed by the test unit at the department, i.e. faults found earlier were not reported in a written form that could be post-analyzed. Further, during the analysis, some faults were excluded either because they were rejected or because they did not affect the operability of the products, e.g. opinion about function, not reproducible faults, and documentation faults.

3.1 Faults-Slip-Through

Fig. 2 and Fig. 3 present the average percent faults-slip-through in relation to percent faults found and development phase from two finished projects at the department. The faults-slip-through measure was not introduced until after the project completions, and hence all the fault reports in the projects studied needed to be classified according to the method described in Section 2.1 in retrospect. The time required for performing the classification was on average two minutes/fault. Actually, several faults could be classified a lot faster but some of them took a significantly longer time since these fault reports lacked information about the causes of the faults. In those cases, the person that repaired the fault needed to be consulted about the cause. Additionally, in order to obtain a consensus on what faults should be considered as faults-slip-through and not, a workshop with key representatives from different areas at the department was held. The output from the workshop was a checklist specifying which faults that should be considered to belong to which phase. When assigning faults to different phases, the possible phases to select among were the following:

System Design (SD): Faults in the design of the external interfaces of the product.

Implementation (Imp): Faults found when implementing the components, e.g. low-level design and coding faults as well as faults found during inspections and unit tests.

Integration Test (IT): Faults found during primary component integration tests, e.g. installation faults and basic component interaction faults.

Function Test (FT): Faults found when testing the features of the system.

System Test (ST): Includes integration with external systems and testing of non-functional requirements.

Field Test + 6 months (FiT+6): During this period, the product is tested in a real environment (e.g. installed into a mobile network), either at an internal test site or

together with a customer. During the first six months, most issues should be resolved and the product then becomes accepted for full operation.

Field Test 7-12 months (FiT_7-12): Same as FiT+6; however, after 6 months of field tests, live usage of the product has normally begun.

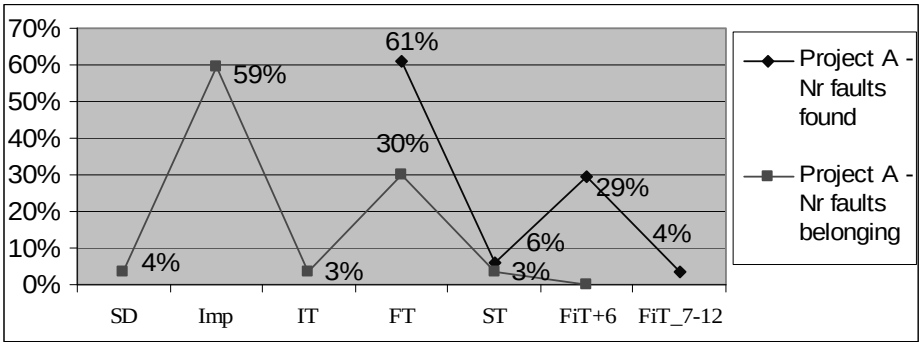

Fig. 2. Percent faults-slip-through in relation to percent faults found and development phase (Project A)

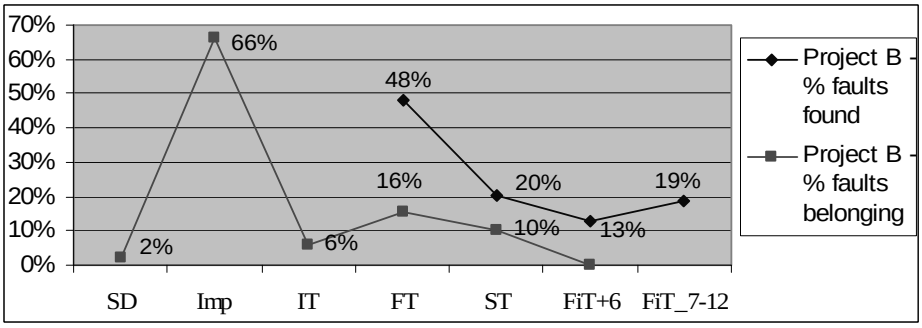

Fig. 3. Percent faults-slip-through in relation to percent faults found and development phase (Project B)

As can be seen in the figures, several faults belonged to the implementation phase (59, 66%). Further, in project A (Fig. 3), many faults were found in FiT+6 (29%). The primary reason for this was that the field tests started before ST was completed, i.e. the phases were overlapping which resulted in that ST continued to find faults during FiT+6. These ST faults could for practical reasons only be classified as FiT+6 faults.

3.2 Average Fault Cost

When estimating the average fault cost for different phases at the department, expert judgments were used since neither was the fault cost reported directly into the fault reporting system nor was the time reporting system feasible to use for the task. In practice, this means that the persons that were knowledgeable in each area estimated

the average fault cost. Table 4 presents the result of the estimations. For example, a fault costs 16 times more in System Test (ST) than in System Design (SD).

Table 4. Average fault-cost/phase at the department (in relative terms)

Phase found	SD	Imp	IT	FT	ST	FiT+6	FiT_7-12
Average cost/fault	1	1.2	3	9.8	16	20	20

The performed cost estimations only include the time required for reporting, fixing and re-testing each fault, which means that there might be additional costs such as the cost of performing extra bug-fix deliveries to the test department. Such a cost is hard to account for since the amount of deliveries required is not directly proportional to the amount of faults, i.e. it depends on the fault distributions over time and the nature of the faults. The reason why FiT_7-12 was estimated to have the same cost as FiT+6 was because the system was still expected to be in field tests although live usage in reality actually might already have started. Further, during the first 12 months after the field tests have started, few systems have been installed although the system becomes available for live usage already during this period. That is, the fault cost rises when more installed systems need to be patched, but, in reality, this does not take any effect until after FiT_7-12.

3.3 Improvement Potential

Table 5 and Table 6 present the improvement potential of the two studied projects from the fault statistics provided in Sections 3.1 and 3.2, calculated according to the method provided in Section 2.1. As can be seen in both tables, faults-slip-through from Implementation comprised a significant proportion of the improvement potential (81%, 84%); therefore, this is foremost where the department should focus their improvement efforts. Further, in project B, all the test phases had a significant improvement potential, e.g. FT could be performed at a 32% lower cost by avoiding the faults-slip-through to it. On the contrary, project A had more diverse fault distributions regarding phase found. The reason for this is mainly due to overlapping test phases (further discussed in Section 3.1). Finally, it should also be noted that the total improvement potential in relation to fault origin phase (rightmost columns) are similar for both projects, which strengthens the assumption in that the improvement potential is foremost process related, i.e. the faults-slip-through did not occur due to certain product problems or accidental events in the projects.

Table 5. Improvement potential (Project A)

Phase found / Phase belonging	FT	ST	FiT+6	FiT_7-12	Total potential /origin phase
SD	3.3%	0.0%	0.6%	0.0%	**3.9%**
Imp	33%	5.6%	37%	5.4%	**81%**
IT	2.2%	0.0%	0.5%	0.5%	**3.2%**
FT	0.0%	0.7%	10.1%	0.6%	**11%**

ST	0.0%	0.0%	0.8%	0.1%	**0.9%**
Total potential/test phase	**39%**	**6.3%**	**48%**	**6.6%**	**100%**

Table 6. Improvement potential (Project B)

Phase found **Phase belonging**	**FT**	**ST**	**FiT+6**	**FiT_7-12**	**Total potential /origin phase**
SD	0.6%	1.9%	0.0%	0.0%	**2.5%**
Imp	30%	14%	12%	28%	**84%**
IT	1.7%	2.5%	2.2%	0.0%	**6.4%**
FT	0.0%	1.6%	1.3%	2.0%	**4.9%**
ST	0.0%	0.0%	1.5%	0.8%	**2.3%**
Total potential/test phase	**32%**	**20%**	**17%**	**30%**	**100%**

4 Discussion

4.1 Validity Threats to the Results

When conducting an empirical industry study, the environment cannot be controlled to the same extent as in isolated research experiments. In order to be able to make a correct interpretation of the results presented in Section 3, one should be aware of threats to the validity of them. As presented below, the main validity threats to this case study concern conclusion, internal, and external validity [8]. Construct validity is not relevant in this context since the case study was conducted in an industrial setting.

Conclusion validity concerns whether it is possible to draw correct conclusions from the results, e.g. reliability of the results [8]. The threats to conclusion validity are as follows. First, when determining which phase each fault belonged to, several faults were assigned to the implementation phase. However, some of these faults might as well belong to System Design due to a lack of information on the causes of the faults. That is, from a described fault cause, it was possible to determine that the fault should have been found before testing started but determining whether a fault belonged to system design or to implementation was sometimes hard since the fault description did not state if the fault occurred due to a fault in a specification or in the implementation. Nevertheless, the practical effect of this uncertainty was small since the cost of finding faults in these two phases was estimated as almost the same (see Table 3). Second, in order to be able to draw conclusions from the results, the department must have a common view on which phase each fault should belong to. That is, managers and developers should together agree on which fault types that should be considered as faults-slip-through and not. At the studied department, this was managed by having workshops where checklists for how to estimate fault-slip-through were developed. However, continuous improvements and training are required in the

first projects in order to ensure that everyone have the same view on how to make the estimations. Further, regarding the average fault cost for different phases, the result was obtained through expert judgments and therefore, the estimations might not exactly reflect the reality. However, this was minimized by asking as many 'experts' as possible, i.e. although there might be deviations, the results were good enough to measure the improvement potential from and hence use as basis for decisions. However, in the future, direct fault cost measures should be included in the fault reports so that this uncertainty is removed. Finally, since the improvement potential is calculated from the faults-slip-through measure and the average fault cost measure, the accuracy of the improvement potential is only dependent on the accuracy of the other measures.

Internal validity concerns how well the study design allows the researchers to draw conclusions from causes and effects, i.e. causality. For example, there might be factors that affect the dependent variables (e.g. fault distributions) without the researchers knowing about it [8]. In the case study presented in Section 3, all faults were post-classified by one researcher which thereby minimized the risk for biased or inconsistent classifications. Another threat to internal validity is whether certain events that occurred during the studied projects affected the fault distribution, i.e. events that the researchers were not aware of. This was managed through workshops with project participants where possible threats to the validity of the results were put forward. Additionally, since two projects were measured, the likelihood of special events that affected the results without being noticed decreased.

External validity concerns whether the results are generalizable or not [8]. In this case study, the results are in overall not generalizable since they are only valid for the studied department having certain products, processes, and tools. However, since two projects were studied and gave similar fault distributions, the results are generalizable within the context of the department. Further, the results on average fault costs in different phases (see Table 4) acknowledge previous claims in that faults are significantly more expensive to find in later phases [3, 10]. Nevertheless, in this paper, the main concern regarding external validity is whether the method used for obtaining the results is generalizable or not. Since the method contains no context dependant information, there are no indications in that there should be any problems in applying the method in other contexts. Thus, the method can be replicated in other environments.

4.2 Implications of the Results

The primary implication of the results is that they provide important input when determining the Return On Investment (ROI) of process improvements. That is, since software process improvement is about reducing costs, the expected ROI needs to be known; otherwise, managers might not want to take the risk to allocate resources for handling upfront costs that normally follow with improvements. Additionally, the results can be used for making developers understand why they need to change their ways of working, i.e. a quantified improvement potential motivates the developers to cooperate [11].

Improvement actions regarding the issues resulting in the largest costs were implemented in subsequent projects, e.g. more quality assurance in earlier phases. Besides shortening the verification lead-time, the expected result of decreased faults-slip-through percentages was to improve the delivery precision since the software process becomes more reliable when many defects are removed in earlier phases [11]. The projects using the method will be studied as they progress.

An unanticipated implication of introducing the faults-slip-through measure was that it became a facilitator for discussing and agreeing on what to test when, e.g. improvement of test strategies. This implies that the measure could serve as a key driver for test process improvement.

5 Conclusions and Further Work

The main objective of this paper was to answer the following research question:

> How can fault statistics be used for determining the improvement potential of different phases/activities in the software development process in an organization?

The answer to the research question constitutes a new method for determining the improvement potential of a software development organization. The method comprises the following three steps:

(1) Determine which faults that could have been avoided or at least found earlier, i.e. faults-slip-through.
(2) Determine the average cost of faults found in different phases.
(3) Determine the improvement potential from the measures in (1) and (2), i.e. measure the cost of not finding the faults as early as possible.

The practical applicability of the method was determined by applying it on two industrial software development projects. In the studied projects, potential improvements were foremost identified in the implementation phase, e.g. the implementation phase inserted, or did not capture faults present at least, too many faults that slipped through to later phases. For example, in the two studied projects, the Function Test phase could be improved by up to 32 and 38 percent respectively by decreasing the amount of faults that slipped through to it. Further, the implementation phase caused the largest faults-slip-through to later phases and thereby had the largest improvement potential, i.e. 81 and 84 percent in the two studied projects.

The measures obtained in this report provide a solid basis for where to focus improvement efforts. However, in further work, the method could be complemented with investigations on causes of why faults slipped though the phase where they should have been found. For example, the distribution of faults-slip-through could be related to different software modules in order to be able to focus more efforts on modules that have a high faults-slip-through. Further, when using the method described in this paper, it would be very interesting to be able to quantify what effect the provided measures have on the development lead-time and delivery precision, e.g. through fault prediction models. To clarify, it is quite obvious that an organization that has a high improvement potential plausibly also has a longer development lead-time than necessary. Further, when having a longer verification lead-time, the organi-

zation cannot be as certain about when the product will reach an adequate quality level and thereby is ready to be delivered. The challenge is to be able to quantify this relation.

6 Acknowledgements

This work was funded jointly by Ericsson AB and The Knowledge Foundation in Sweden under a research grant for the project "Blekinge - Engineering Software Qualities (BESQ)" (http://www.bth.se/besq).

References

1. Berling, T., Thelin, T., An Industrial Case Study of the Verification and Validation Activities, Proceedings of the Ninth International Software Metrics Symposium, IEEE, (2003) 226-238
2. Bhandari, I., Halliday, M., Tarver, E., Brown, D., Chaar, J., Chillarege, R., A Case Study of Software Process Improvement During Development, IEEE Transactions on Software Engineering, Vol. 19, Issue 12, Proquest (1993) 1157-1171
3. Boehm, B., Software Engineering Economics, Prentice-Hall (1981)
4. Cook, E., Votta, L., Wolf, L., Cost-Effective Analysis of In-Place Software Processes, IEEE Transactions on Software Engineering, Vol. 24, Issue 8, Proquest (1998) 650-662
5. Grady, R., Practical Software Metrics for Project Management and Process Improvement, Prentice Hall (1992)
6. Hevner, A. R., Phase Containment for Software Quality Improvement, Information and Software Technology 39 (1997) 867-877
7. IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE/ANSI Standard 982.2-1988.
8. Wohlin, C., Höst, M., Henningsson, K., Empirical Research Methods in Software Engineering, In Empirical Methods and Studies in Software Engineering: Experiences from ESERNET, pp. 7-23, editors Reidar Conradi and Alf Inge Wang, Lecture Notes in Computer Science, Spinger-Verlag, Germany, LNCS 2765
9. Leszak, M., Perry, D., Stoll, D., A Case Study in Root Cause Defect Analysis, Proceedings of the 22nd Int. Conference on Software Engineering, ACM Press, (2000) 428-437
10. Shull, F., Basili V., Boehm B., Brown W., Costa, P., Lindwall, M., Port, D., Rus, I., Tesoriero, R., Zelkowitz, M., What We Have Learned About Fighting Defects, Proceedings of the Eight IEEE Symposium on Software Metrics, IEEE (2002) 249-258
11. Tanaka T., Sakamoto K., Kusumoto, S., Matsumoto K., Kikuno T., Improvement of Software Process by Process Description and Benefit Estimation, Proceedings of the 17th International Conference on Software Engineering, ACM (1995) 123-132

12. Wohlwend H., Rosenbaum S., Software Improvements in an International Company, Proceedings of the 15th International Conference on Software Engineering, IEEE Comput. Soc. Press. (1993) 212-220