L-O Damm, L. Lundberg and C. Wohlin, "Faults-slip-through - A Concept for Measuring the Efficiency of the Test Process", Wiley Journal of Software: Process Improvement and Practice, Volume 11, No. 1, pp. 47-59, 2006.

Faults-slip-through – A Concept for Measuring the Efficiency of the Test Process

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}@bth.se

SUMMARY

In market-driven development where time-to-market is of crucial importance, software development companies seek improvements that can decrease the lead-time and improve the delivery precision. One way to achieve this is by analyzing the test process since rework commonly accounts for more than half of the development time. A large reason for high rework costs is fault slippages from earlier phases where they are cheaper to find and remove. As input to improvements, this paper introduces a measure that can quantify this relationship. That is, a measure called faults-slip-through, which determines which faults that would have been more cost-effective to find in an earlier phase. The method presented in this paper also determines the excessive costs of the faults that slipped through phases, i.e. the improvement potential. The method was validated through practical application in two software development projects at the telecom company Ericsson AB. The results of the case study determined that the implementation phase had the largest improvement potential in the two studied projects since it caused a large faults-slip-through to later phases, i.e. 85 and 86 percent of the total improvement potential of each project.

KEYWORDS: fault metrics; software process improvement; faults-slip-through; early fault detection; fault latency

1. Introduction

Reducing development costs and time-to-market while still maintaining a high level of product quality is essential for many modern software development organizations. These 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 (Cook et al. 1998). 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 (Shull et al. 2002). In fact, related research states that fault analysis is the most promising approach to software process improvement (Grady 1992).

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 (Wohlwend and Rosenbaum 1993). Without such a decision support, it is common that improvements are not implemented because organizations find them difficult to prioritize (Wohlwend and Rosenbaum 1993). Further, if a suggested improvement can be supported with data, it becomes easier to convince people to make changes more quickly (Grady 1992). 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 assessing a test process and quantifying the improvement potential of it in a software development organization?

Classification of faults from their causes, e.g. root cause analysis, is a commonly used approach for fault analysis (Leszak et al. 2000). 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. Additionally, root cause analysis does not measure what the total cost of the slipped faults is. A test-oriented approach for fault analysis is fault trigger classification, which categorizes faults after which test activity that

triggered them (Chillarege and Prasad 2002). This technique can be used for identifying how good a phase is at finding the faults it should find. However, as for root cause analysis, the technique cannot quantify the cost of faults that should have been found earlier.

To be able to address the stated research question, this paper instead 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 (Basili and Green 1994, Hevner 1997). However, the method presented in this paper has two main differences. First, faults-slip-through focuses on when it is cost-effective to find each fault, it does not consider fault insertion phase. Second, the presented method also calculates the improvement potential by relating the result of the fault classification with the average cost of finding and repairing the faults. That is, 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 validate 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 work presented in this paper is based on previous work on faults-slip-through based improvement work (Damm et al. 2004). The paper is outlined as follows. Section 2 presents related work to the proposed method in this paper. Then, Section 3 describes the proposed method for how to determine the improvement potential of an organization. Section 4 demonstrates the applicability of the method through an empirical case study. Section 5 discusses the validity and implications of the results and Section 6 concludes the work.

2. Related Work

The Capability Maturity Model (CMM) is one of the most widely used models for software process improvement (Paulk et al. 1995). The essence of this model and its tailored variant SW-CMM (SW-CMM) is for organizations to strive for achieving higher maturity levels and thereby become better at software development. However, such blueprint models have been criticized because they are focused on making an organization more structured and work according to what is defined to be state of the art in the model. That is, although state of the art models can provide some guidance to areas of improvement, there are no generally applicable solutions (Glass 2004, Mathiassen et al. 2002).

The opposite of applying a model-based approach is the bottom-up approach where improvements are identified and implemented locally in a problem-based fashion (Jakobsen 1998). Originating from the widely recognized concept Total Quality Management (Deming 2000), a typical bottom-up approach is the Quality Improvement Paradigm (QIP). In QIP, a six-step improvement cycle guides an organization through continuous improvements based on QIP (Basili and Green 1994). From defined problem based improvements, QIP sets measurable goals to follow up after the implementation. However, QIP is more of a generic framework for which steps to include in an improvement cycle, it does not state exactly how to perform them. That is, in practice, the method in this paper could instead be included as a part of QIP, i.e. by including faults-slip-through as an important measure to follow up with respect to fault cost reduction.

Fault-oriented measures are within process assessment and improvement used for different purposes. That is, the purpose of fault analysis is to improve the process to make the products more fault tolerant, reduce the amount of introduced faults, or to prevent the faults from being introduced in the first place (Tian 2001). Based on a survey, it is concluded that the application domain and the current process affect which approach to prefer, but overall, it is most efficient to combine the three techniques (Tian 2001). The approach presented in this paper focuses on fault reduction and fault prevention.

A widely spread fault measurement approach is 'Six sigma', which is centered on a measure that determines the number of faults in relation to product size (Biehl 2004). Further, as earlier mentioned, the measure applied in this paper is similar to measures used in related work, e.g. phase containment metrics where faults should be found in the same phase as they were introduced (Hevner 1997), and goodness measures where faults should be found in the earliest possible phase (Berling and Thelin 2003). In contrast to the faults-slip-through measure, these measures are strongly related to the notion of fault latency, i.e. for how long time does a fault remain in a product. The implication of this is that since most faults are inserted during analysis, design and coding, the measures primarily provide feedback on earlier phases instead of the test process. Therefore, they are not suitable for improvements aimed at the test process.

Finally, related work on calculating the improvement potential from faults has been done before, i.e. by calculating the time needed in each phase when faults were found when supposed to in comparison to when they were not (Tanaka et al. 1995). 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. Method

3.1. 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:

Table 1. Fictitious 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**

(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 (IEEE 1988). The following three sub-sections describe how to perform each of the three steps.

3.1.1. Faults-slip-through measurement

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 (Hevner 1997), and goodness measures where faults should be found in the earliest possible phase (Basili and Green 1994). The main difference between the faults-slip-through measure and the other measures 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. Then it is not a faults-slip-through. Figure 1 further illustrates this difference.

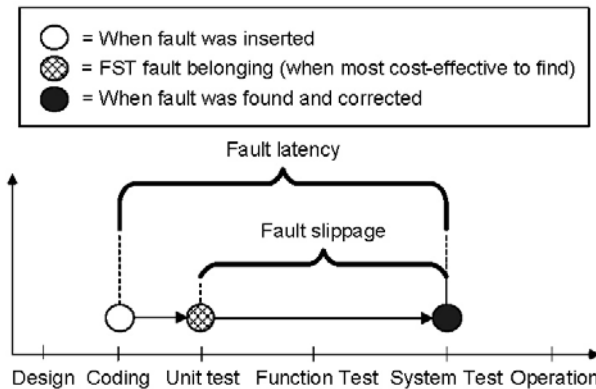

Figure 1. Fault latency versus faults-slip-through

A consequence of how faults-slip-through is measured is that a definition must be created to support the measure, i.e. a definition that specifies which faults that should be found in which phase. To be able to specify this, the organization must first determine what should be tested in which phase. Therefore, this can be seen as test strategy work. Thus, experienced developers, testers and managers should be involved in the creation of the definition. The results of the case study in Section 4.2 further exemplify how to create such a definition. 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).

3.1.2. Average Fault Cost

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 (Boehm 1983, Shull et al. 2002). However, the cost-increase varies significantly depending on the maturity of the development process and on whether the faults are severe or not (Shull et al. 2002). 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. Fictitious example of average fault cost/phase found

	Design	Implementation	FT*	ST*	Operation
Average fault cost	1	2	10	25	50

*FT=Function Test, ST=System Test

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. Note that the individual fault cost varies depending on the type of fault. Therefore, estimation of an average fault cost requires a relatively large amount of faults to be reliable.

3.1.3. Improvement Potential

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. Figure 2 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 (IPxy). PFx 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 (IPxy). In order to demonstrate how to use and interpret the matrix, Table 3 provides an example calculation by applying the formulas in Figure 2 on the fictitious values in Table 1 and Table 2. 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 45 hours (1 hour/fault times 49 faults minus 4). 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.

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 fault slippage to System Test can be decreased by = 609/1255=49%). In the case study reported in the next section, the measurements are provided in percent (due to confidentiality reasons).

	PF_1	PF_2	PB_{total}
PB_1	IP_{11}	IP_{21}	$PB_{1\ total}$
PB_2	IP_{12}	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_{xy} = (No. faults bel. (PFxPBy) * PF_x AvFC) – (No. faults bel. (PFxPBy) * PByAvFC)

Figure 2. Matrix formula for calculation of improvement potential

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)

4. Results from Applying the Method

This section describes the case study application of the previously defined described method. First, Section 4.1 provides an overview of the case study environment. After that, sections 4.2-4.4 describes the result of applying the three steps of the method.

4.1. Case Study Setting

The applicability of the described method was evaluated by using it on the faults reported in two projects at a department at Ericsson AB. To be able to understand and interpret the results reported from these two projects, the contextual setting of the case study needs to be well described (Kitchenham et al. 2002).

The projects developed functionality to be included in new releases of two different telecom products. Hence, previous versions of the products were already in full operation at customer sites. The products are launched as software services operating in mobile networks. 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 process used for developing the products was based on an incremental approach including the traditional development phases: analysis, design, implementation, and test. Details about the phases relevant to the case study are described in Section 4.2. The participants in the projects had different experience and skill levels. However, most of the participants had several years of experience in software development within the application domain including experience with the tools and processes used by the organization. In fact, at most 10 percent of them had less than one year of practical experience. Further, most of the participants had at least a bachelor degree in computer science or the like.

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 new development process should provide similar results (except for 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. Finally, requirements faults were not reported in the fault reporting system. Instead, they were handled separately as change requests.

4.2. Faults-Slip-Through

Figure 3 and Figure 4 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 3.1 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. In the future, this overhead work could be avoided by making sure that the faults are classified on the fly instead. Section 5.1 describes how this was achieved at the studied department. 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 definition of which faults that should belong to which phase. Appendix A presents the obtained definition for each phase. However, note that the two last phases below (FiT+6, FiT_7-12) were not included in the definition since at that stage all faults were considered as faults-slip-through. When assigning faults to different phases, the possible phases to select among were the following:

Implementation (Imp): Faults found when implementing the components, e.g. coding faults found during code 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.

Regarding the possible phases to select between, it could have been possible to include earlier phases such as requirements analysis and system design. However, since the studied department wanted to focus on feedback on the test phases when using this measure, earlier phases were excluded.

As can be seen in the figures below, several faults belonged to the implementation phase in the two projects, i.e. 63 and 68 percent respectively. Further, in project A (Figure 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.

The figures below illustrate the faults-slip-through distributions of the projects in a good way. However, sometimes it is more feasible to present the measure as the total amount of faults-slip-through to a certain phase, e.g. at the studied department the measure was to be used as goal values in balanced scorecards (Kaplan and Norton 1996). That is, in this case only a few key goal measures were requested and therefore, a multi-valued graph was not feasible. Table 4 describes how this was applied at Ericsson AB. For example, in the table, 69, and 80 percent faults-slip-through to FT was calculated as a sum of all faults that should have been found in earlier previous phases divided with the number of faults found in FT.

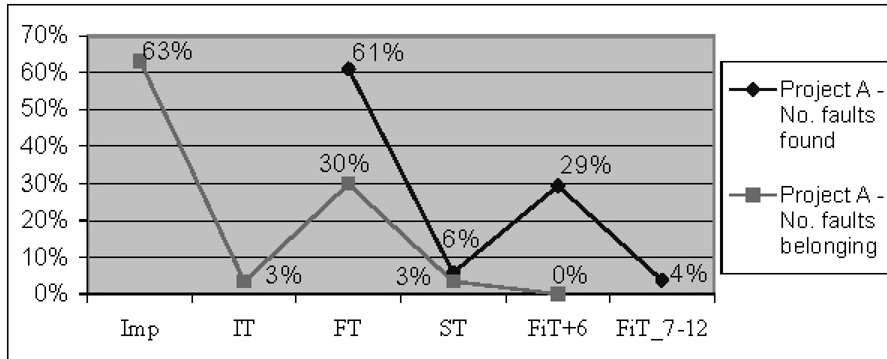

Figure 3. Percent faults-slip-through in relation to percent faults found and development phase (Project A)

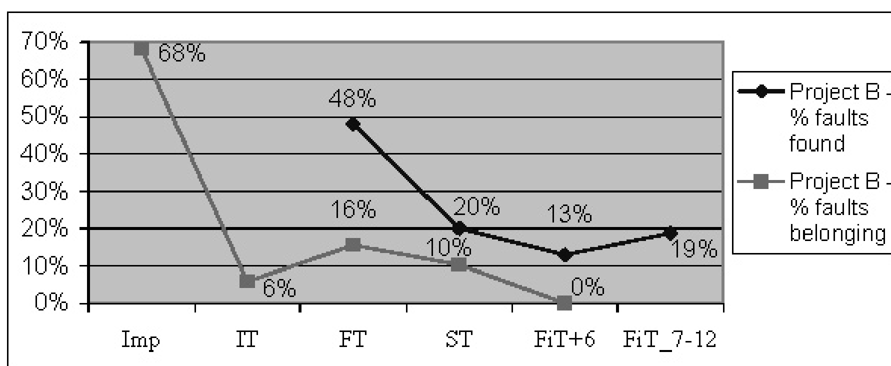

Figure 4. Percent faults-slip-through in relation to percent faults found and development phase (Project B)

Table 4. Percent Faults-slip-through (FST) to FT and ST

	Project A	Project B
FST to Function Test	69%	80%
FST to System Test	77%	77%

4.3. 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 5 presents the result of the estimations. For example, a fault costs 13 times more in System Test (ST) than in Implementation (Imp). In the table, the cost estimates only include the time required for reporting, fixing and re-testing each fault, which means that there might have been 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.

Table 5. Estimated average fault-cost/phase (relative cost)

Phase found	Imp	IT	FT	ST	FiT+6	FiT_7-12
Average cost/fault	1	2.5	8.2	13	17	17

4.4. Improvement Potential

Table 6 and Table 7 present the improvement potential of the two studied projects from the fault statistics provided in Sections 4.2 and 4.3, calculated according to the method provided in Section 3.1. As can be seen in both tables, faults-slip-through from Implementation comprised a significant proportion of the improvement potential (85, 86%); 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 percent 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 4.2). Finally, it should also be noted that the total improvement potential in relation to fault origin phase (rightmost columns) were similar for both projects, which strengthens the assumption 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 6. Percent improvement potential (Project A)

Phase found / Phase belonging	FT	ST	FiT+6	FiT_7-12	Total potential /origin phase
Imp	37	5.6	37	5.4	**85**
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 7. Percent improvement potential (Project B)

Phase found / Phase belonging	FT	ST	FiT+6	FiT_7-12	Total potential /origin phase
Imp	30	16	12	28	**86**
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**

5. Discussion

This section discusses the proposed method and the results of applying it. First, Section 5.1 presents a few lessons learned from defining and applying the central part of the method, i.e. the faults-slip-through measure. After that, Section 5.2 describes observed implications of the results. Finally, Section 5.3 discusses the validity threats to the results.

5.1. Faults-slip-through: Lessons learned

Applying faults-slip-through in an industrial environment gave several experiences regarding what works and not. First of all, creating the definition resulted in a few lessons learned:

- Start with defining the test strategy if it is not already clearly specified. That is, there is a direct mapping between what types of tests a phase should cover and which faults should be found when executing those tests.
- Specify the definition after the goal test strategy to reach within a few years time, not the current situation. This is very important to make the measure possible to improve against and to keep the definition stable. If the definition is changed between each project, the projects are not possible to compare against each other.
- Create the definition iteratively, i.e. develop a suggestion, test it on a set of faults, and then refine it iteratively until satisfactory. The reason for this is because it is during practical usage possible flaws are found, especially fault types that the involved people forgot to include in the definition.

When the definition was created, the measure was also added to the fault reporting process, i.e. to make sure that it would be reported when correcting the faults so that a post-mortem analysis would not be needed after the next project. Experiences from doing this gave the following major lessons learned:

- The developers fixing the fault should specify the faults-slip-through measure in the fault reports. Testers can be good to use as support in some situations but in our experience, it is the developers that knows best in most cases. However, the later test phase a fault is found in, the more influence should the testers have since they know that area better. Further, the testers are always good to have as a point of validation to make sure that the value was reported correctly.

- Make sure to educate the developers and testers properly so that they have a common view on how to use it. Inevitably, there is a degree of subjectiveness in the definition but creating a common mindset can minimize this. Additionally, the first time the measure is applied in a project, someone that took part in creating the definition should validate the reported values to verify that everyone understood the measure correctly.

Make sure that the values are easy to follow up, i.e. automate the data analysis as much as possible, e.g. generate the faults-slip-through results on a webpage or make the data easy to generate into a spreadsheet.

5.2. Implications of the Results

The primary implication of the results is that they provide important input when determining the Return On Investment (ROI) of suggested 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 (Tanaka et al. 1995).

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 faults are removed in earlier phases (Tanaka et al. 1995). 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 driver for test process improvement. Finally, the definition turned out to be a good support for driving test process alignment work, i.e. by making different projects and products adhere to the same faults-slip-through definition. That is, it provides a way to specify how to work and then follow up if this way of working is achieved.

5.3. 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 4, 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 (Wohlin et al. 2003). 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 (Wohlin et al. 2003). The threats to conclusion validity are as follows. First, 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 (Wohlin et al. 2003). In the case study presented in Section 4, 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. That is, the obtained improvement

potentials of the two studied projects were very similar (85 and 86%), which indicates that the causes of the results were directly related to the projects' common denominator, i.e. that they used the same tests process. The final validity threat is about the possibility that a fault might actually be found before it should have been found. In the case study projects, such faults were considered not to affect the results because they were very rare, i.e. they constituted less than one percent of the total amount of faults. Therefore, these faults were classified as non-slips but not treated specially in any other way.

External validity concerns whether the results are generalizable or not (Wohlin et al. 2003). In the performed case study, the results are not fully generalizable since they are dependent on the studied department having certain products, processes, and tools. However, since two similar projects were studied and gave similar fault distributions, the results are at least valid within the context of the department. That is, the results are generalizable as long as the context is the same but the generalizability decreases more and more the less alike the considered context is. Further, the results on average fault costs in different phases (see Table 5) acknowledge previous claims in that faults are significantly more expensive to find in later phases (Boehm 1983, Shull et al. 2002). 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.

6. Conclusions

This paper presents and validates a method for measuring the efficiency of the software test process. The main objective of the paper was to answer the following research question:

How can fault statistics be used for assessing a test process and quantifying the improvement potential of it in a software development organization?

The method developed for answering the research question determines the improvement potential of a software development organization through 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 in the right phase.

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 39 and 32 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. 85 and 86 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 should be related to the underlying test activities in order to be able to focus more efforts on specific test activities that had a high faults-slip-through.

7. 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).

8. References

Basili V. 1994. Software Modeling and Measurement: The Goal Question Metric Paradigm. *Computer Science Technical Report Series*. CS-TR-2956 UMIACS-TR-92-96. Technical report. University of Maryland.

Berling T, Thelin T. 2003. An Industrial Case Study of the Verification and Validation Activities. *Proceedings of the Ninth International Software Metrics Symposium. IEEE*. 226-238

Bhandari I, Halliday M, Tarver E, Brown D, Chaar J, Chillarege R. 1993. A Case Study of Software Process Improvement During Development. IEEE Transactions on Software Engineering 19(12): 1157-1171.

Biehl R. 2004. Six Sigma for Software. *IEEE Software* 21(2): 68-71.

Boehm B. 1983. Software Engineering Economics. Prentice-Hall: NJ, U.S.A.

Chillarege R, Prasad K. 2002. Test and development process retrospective - a case study using ODC triggers. *Proceedings of the International Conference on Dependable Systems and Network 669-678. IEEE.*

Cook E, Votta L, Wolf L. 1998. Cost-Effective Analysis of In-Place Software Processes. IEEE Transactions on Software Engineering 24(8): 650-662.

Damm L-O, Lundberg L, Wohlin C. 2004. Determining the Improvement Potential of a Software Development Organization through Fault Analysis: A Method and a Case Study. *Proceedings of the 11th International Conference on Software Process Improvement* 138-149. Lecture Notes in Computer Science 3281. Springer-Verlag.

Deming E W. Out of the Crisis. *The MIT Press*: Cambridge, MA, USA. ISBN 0-262-54115-7.

Glass R. 2004. Some Heresy Regarding Software Engineering. *IEEE Software* 21(4): 104-107.

Grady R. 1992. Practical Software Metrics for Project Management and Process Improvement. Prentice Hall.

Hevner A. 1997. Phase Containment for Software Quality Improvement. Information and Software Technology 39: 867-877.

IEEE 1998. IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software. IEEE/ANSI Standard 982.2-1988.

Jakobsen A. 1998. Bottom up Process Improvement Tricks, *IEEE Software* 15(3): 64-68.

Kaplan S, Norton D. 1996. *The Balanced Scorecard*, Harvard Business School Press. Boston, MA, U.S.A.

Kitchenham B, Pfleeger S, Pickard L, Jones P, Hoaglin D, El Emam K, Rosenberg J. 2002. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering* 28(8): 721-734.

Leszak M, Perry D, Stoll D. 2000. A Case Study in Root Cause Defect Analysis, *Proceedings of the 22nd Int. Conference on Software Engineering* 428-437. *ACM Press*.

Mathiassen L, Pries-Heje J, Ngwenyama O. 2002. *Improving Software Organizations: From Principles to Practice*, Addison-Wesley, NJ, USA.

Paulk M., Weber C, Curtis B, Chrissis M. 1995. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison Wesley Longman, Inc, MA, USA.

Shull F, Basili V, Boehm B, Brown W, Costa P, Lindwall M, Port D, Rus I, Tesoriero R, Zelkowitz M. 2002. What We Have Learned About Fighting Defects. *Proceedings of the Eight IEEE Symposium on Software Metrics* 249-258.

SW-CMM. http://www.sei.cmu.edu/cmm/cmm.html. Last Accessed: 2005-02-04.

Tanaka T, Sakamoto K, Kusumoto S, Matsumoto K, Kikuno T. 1995. Improvement of Software Process by Process Description and Benefit Estimation. *Proceedings of the 17th International Conference on Software Engineering 123-132, ACM*.

Tian J. 2001. Quality Assurance Alternatives and Techniques: A Defect-Based Survey and Analysis. *Software Quality Professional* 3(3): 6-18. ASQ.

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

Wohlin C, Höst M, Henningsson K. 2003. Empirical Research Methods in Software Engineering. In Empirical Methods and Studies in Software Engineering: Experiences from ESERNET 7-23. Editors Reidar Conradi and Alf Inge Wang. Lecture Notes in Computer Science. Spinger-Verlag. Germany. LNCS 2765.

Appendix A

Basic Test:

- Basic stability problems that can be detected when testing components or sub-features in isolation (including component level load, performance, and endurance tests).
- Basic interface inconsistencies between components or within a sub-feature. That is, every component or sub-feature should work ‘stand alone’, including error cases.
- Faulty revisions in deliveries.
- Isolated faults in scripts and output text files including for example spelling faults and not understandable text.

Integration Test:

- Basic software installation/uninstallation faults
- Faults in the current system installations that should work when starting each subsequent test phase
- Faults found in main function flows (smoke tests).

Function Test:

- Every function should work ‘stand alone’ in a simulated environment (including human interfaces)
- The system should adhere to specified protocol standards.

System Test:

- Load and stability faults (including performance, robustness, availability, load balancing, and reboot related faults)
- Faults found only when connected to “real” external systems.
- Complex multi-function faults.
- Restore and backup faults.
- Other installation faults, for example hardware related, data migration etc.