Using Fault Slippage Measurement for Monitoring Software Process Quality during Development

Lars-Ola Damm[1], Lars Lundberg
School of Engineering,
Blekinge Institute of Technology
Box 520, SE-372 25 Ronneby, Sweden
{lars-ola.damm, lars.lundberg}@bth.se

ABSTRACT

In a competitive environment where time-to-market is crucial for success, software development companies initiate process improvement programs that can shorten the development time. They especially seek improvements in the verification activities since rework commonly constitutes a significant part of the development cost. However, the success of process improvement initiatives is dependent on early and observable results since a lack of feedback on the effect of improvements is a common cause of failure. This paper investigates how to monitor the verification process as input to decisions such as improvement actions. The suggested approach was applied on three industrial software products at Ericsson and the results determined that the approach can be used for quantitative monitoring of process quality and as decision support to do rapid improvement actions.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – *Process metrics.*

General Terms

Management, Measurement, Verification.

Keywords

In-process quality, process compliance, test metrics, software process improvement, fault slippage measurement.

1. INTRODUCTION

Significant research efforts have been invested in how to define processes that produce quality products in an efficient way. While this remains very important, implementing identified process improvements has emerged as a large challenge. In fact, the failure rate of process improvement implementation is reported to be about 70 percent [16]. Therefore, practitioners request more guidance on how, not just what to improve [15][18]. According to related research, a lack of attention to feedback in the process is a major cause of the failure of several improvement paradigms [11]. Further, the success of a process improvement initiative is dependent on early and observable results [14]. One important process related improvement area is to reduce the amount of avoidable rework during development due to introduced faults in products. The reason for this is because avoidable rework is commonly a large part of a development project [19][21]. Central to keeping the amount of avoidable rework low is to ensure that verification responsibilities are distributed in an efficient way. That is, larger systems commonly have several verification levels with different verification responsibilities, e.g. in the telecom industry it could be component, product, solution and network level testing. In such systems, process quality during development is not primarily about the number of faults that are found at different test levels. Process quality is to fulfill responsibilities as defined so that the faults that should have been found at a certain verification level do not slip through to subsequent levels. These responsibilities are commonly stated as verification strategies, i.e. they define the long-term goal of what is the most efficient way of working including what should be verified at which test level. The primary consequence of not working according to such a verification strategy is higher rework costs because more faults slip through to later test levels where they are extra expensive because they tend to halt the test process until the fault has been fixed. For example, a functional fault that should have been covered at an early test level can when found during a later performance test make it impossible to continue the performance tests until the functional fault has been fixed.

A department at Ericsson had put a lot of efforts on defining a verification strategy for the development projects to follow, i.e. what should be covered at each verification level. A previous study, i.e. [6], however showed that the adherence to this test strategy could be improved significantly. The department conducted regular assessments to identify improvement actions to take. To be able to cope with improvement work as a part of the constant customer pressure, organizational measurement goals, e.g. through balanced scorecards [13], were also set to increase visibility and thereby stress the importance of the improvement work. However, the improvement progress was slow because the assessments were only made as post-analysis upon project completion, i.e. there was a lack of early progress feedback. This was not enough because the projects were commonly rather long, i.e. 12-18 months. Thereby, the department wanted early and quantitative indications on the process quality that can be used for goal tracking and that can provide input to improvement actions.

[1] Is also an employee at Ericsson, lars-ola.damm@ericsson.com

1.1 Related Work

There is a vast amount of related work in the area of process assessment and improvement, e.g. audit-centered frameworks such as SPICE [7] and CMM[17]. Similar frameworks tailored for test processes have also been developed, e.g. TPI and TMM [21]. Such approaches are however not suitable when requiring frequent and quantified assessments. The most common approach to monitor quality is through the faults that are found, e.g. [3]. However, since the number of found faults is dependent on the amount of tests performed, there is not a correlation between the number of faults found and quality [19]. Further, reported faults state that something has been tested but not if that was what was supposed to be tested according to the test strategy.

Another common process assessment approach is to do fault profiling by classifying faults into categories related to activities in the process, e.g. Orthogonal Defect Classification (ODC) and the HP defect classification scheme [4][10]. Fault profiling techniques such as ODC trigger classification can provide quantification on fault distributions in relation to sub-processes but does not provide values that can be used as goal measures. Another common technique that also can be considered in process quality assessments is code coverage analysis [21], which can indicate how well-tested a piece of software is. However, code coverage analysis does not state if the responsibilities stated in the test strategy were covered. That is, an efficient test process does not test as much as possible, it only test the parts that according to the test strategy should be covered.

1.2 Research Question and Outline

The hypothesis of this paper was that an existing measurement concept that already had been widely implemented at Ericsson could be used to monitor the quality of the verification process. The concept is based on a measure called 'Faults-Slip-Through' (FST) [5], [6]. The idea with the measure is to count the faults that according to the test strategy should have been found in an earlier phase [6]. Thus, FST determines the quality of the process in form of compliance to the test strategy. The measure has been successfully introduced in the fault reporting process and has been used in post-analysis to assess process quality. However, its possible usefulness during development had not been determined. That is, pilot attempts to use FST during development were made at the studied department, but did not succeed due to a lack of proper method support, e.g. measuring FST during development is rather trivial but interpretation of the obtained data is not. Thus, the research question of this paper is as follows:

How can FST be used for assessing and improving the process quality during development?

This paper utilizes FST measurement in a way that it can provide information about early and comparable trends on the process quality (described in Section 2). Further, it provides guidance on how to interpret the trends. That is, applying the method on three products (described in sections 3 and 4) determined that the FST data can soon after faults start to be reported assess the quality of the verification process and provide input to improvement actions that can improve the quality of the process.

2. METHOD

This section describes the method used in the case study reported in this paper. Section 2.1 describes the basic measurement method and Section 2.2 describes how this method was applied to address the research question stated in the introduction (see Section 1.2).

2.1 Overview of FST Measurement

As stated in the introduction, FST is the basic measure used by the method suggested in this paper. FST is similar to phase containment measurements where faults should be found in the same phase as they were introduced [12]. The essence of such approaches is to analyze when faults are inserted and found and from that determine which faults are in-phase and out-of-phase [12]. The time between when the fault was inserted and found is commonly referred to as 'fault latency'.

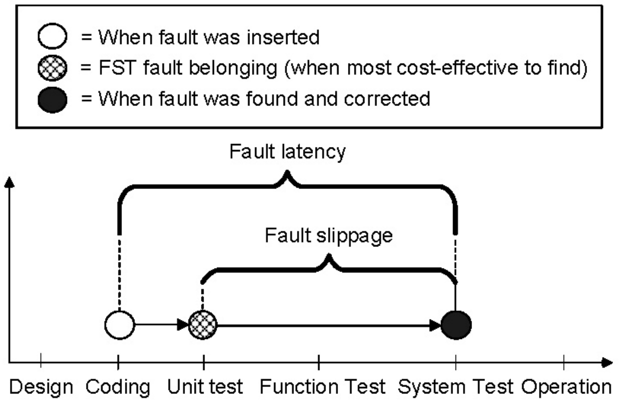

○ = When fault was inserted
⊗ = FST fault belonging (when most cost-effective to find)
● = When fault was found and corrected
Fault latency
Fault slippage
Design
Coding
Unit test
Function Test
System Test
Operation

Figure 1. Fault latency versus FST

Since most faults are inserted during early development phases, e.g. requirements elicitation, design, and implementation, the fault latency measure is not good at evaluating the quality of the verification process. Further, if for example a test activity is improved, it is not possible to use measurements based on when faults were inserted to evaluate the result since only later phases are affected by the improvement. Instead, the FST concept was, for the context of this study, considered more appropriate because the primary purpose of measuring FST is to make sure that the test process finds the right faults in the right phase, i.e. in most cases early. Figure 1 visualizes the difference between fault latency and FST. When measuring FST, the norm for what is considered 'right' is the test strategy of the organization. That is, if the test strategy states that certain types of tests are to be performed at certain levels, the FST measure determines to what extent the test process adheres to this test strategy. This means that all faults that are found later than when supposed are considered slips [6]. One of the most common ways that Ericsson uses FST data is for evaluating the degree of FST to a verification phase, i.e. as described in Equation 1. Details about how to define and measure FST are described in [6].

$$\%\,\text{FST (to phase X)} = \frac{\text{SF}^1\ (\text{X})}{\text{TF}^2\ (\text{X})} \qquad \textbf{(1)}$$

^{1}SF=No. faults found in phase X that slipped from earlier phases

^{2}TF= Total no. faults found in phase X

2.2 In-Process FST Measurement

As described in the previous section, the approach suggested in this paper utilized the FST measure defined in the formula in Equation 1. However, to create trend lines from the FST data, the registration date of each fault report needs to be added to the formula of each measurement point to be included in the trend graph. This is most easily done in a spread sheet tool such as MS Excel. Another alternative is to use a web based solution which is more expensive to develop but makes it possible to get real-time updated results, i.e. assuming that it is able to fetch the measurement data from the fault database. The appropriate frequency of measurement points, e.g. weekly or monthly, depends on the length of the monitored projects. In our experience, at least 10 measurement points should be included to be able to see the trends clearly (distributed with the same time-interval between each point). The result of plotting graphs from such measurement points can be seen in Figure 2 in Section 4.

There are two necessary prerequisites to apply this method:

1. The fault reports should include a field stating which phase each fault should have been found in, so that the statistics can be fetched from the fault database regularly. If this for any reason is not possible, an alternative is to have regular follow-up meetings, e.g. weekly, where the FST value of each reported fault is determined. Nevertheless, the more automated the measurement collection is, the better since manual overhead causes reported metrics not to be used [9].
2. If regular follow-up of the FST trends are going to be useful, goals that the trends can be compared against should be set. In our experience, the preferred input for such a goal is a baseline value obtained from a previously finished project. The baseline value should in that case be obtained from the previous projects of the same product since the percent FST in a project depends on the product complexity and how the verification strategy is defined [6]. Based on this value, the goal should be set after what differences are expected in the project to study, e.g. if a planned process improvement aims at reducing the FST to a certain degree, the goal value should be specified accordingly. It is also possible to set a goal without the baseline value. However, it will then be hard to know what an appropriate goal is. That is, since there is also a cost of reducing the degree of FST, it is important to set reasonable goals and then monitor them, i.e. the optimal FST level is in relation to minimized lead-time often not zero.

With these overall prerequisites met, it is then possible to monitor the process quality during the verification stages of projects. To make sure that the method becomes a part of continuous improvement work, we suggest that it is incorporated into established measurement processes such as QIP [1] or Six Sigma [2]. This also implies that in order to implement a measurement approach such as the one described in this paper, the organization need to be mature enough to be able to handle such measurement processes, e.g. in CMM, such measurements are part of level four and five on the maturity ladder [17].

3. Case Study Setting

The case study was conducted at a software development department at Ericsson, which develops some software products on its main site and one partly at an offshore location. The projects develop software to be included in new releases of existing products that are in full operation as parts of operators' mobile networks. A typical project such as the one studied in this paper lasts about 1-1.5 year and has on average about 50 participants. The projects are performed according to an in-house developed incremental development process. Besides inspections of documents during design, the products are verified in four steps: Basic Test, Integration Test, Function Test, and System Test. According to the test strategy of the organization, the faults that belong to different phases are in the case study in this paper divided as follows (the complete strategy is not included due to space limitations).

Basic Test (BT): Faults found during unit tests of a component.

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

Function Test (FT): Faults found when testing the features of the system, e.g. faults in user interfaces and protocols.

System Test (ST): Faults found when integrating with external systems and when testing non-functional requirements.

The faults that were reported and used in this paper originated from FT and ST, i.e. faults found earlier than FT were not reported in a written form that could be post-analyzed. Further, during the analysis, some of the reported faults were excluded because they turned out not to be real faults. Additionally, requirements faults were not reported in the fault reporting system. Instead, they were handled separately as change requests.

Each test level verifies a varying number of deliveries from the design department depending on the number of feature increments and the number of required bug-fix deliveries. Further, to save lead-time, the verification levels of the increments are performed partly in parallel, e.g. FT is not completed when ST starts. This introduces a risk for more fault slippages but if ST knows what in the delivery is tested and not in FT, the ability to start ST early on the parts of a feature that have been function tested saves more lead-time than what the additional cost of FST is worth. This is a major reason why the optimal FST goal rarely is zero.

4. CASE STUDY RESULTS

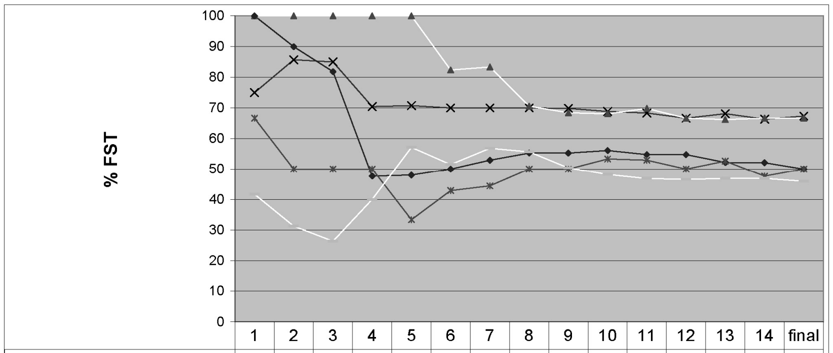
This section describes the results from applying the suggested method on projects from three products at Ericsson, i.e. two projects developed onsite (referred to as 'G' and 'S') and one partly developed at an offshore location (referred to as 'E'). As described in Section 3, fault data could be retrieved from the test phases Function Test (FT) and System Test (ST). However, in the partly offshore-developed project, data was only available from ST because FT was managed at the offshore location.

Section 4.1 describes the FST trends of the studied projects. After that, Section 4.2 analyzes observed common patterns of the data, i.e. by relating to when the faults were found. As mentioned in Section 2.2, the time scale in the trend graphs was for comparison reasons normalized to get the same amount of measurement points, i.e. 15. The result of this is that the time between each measurement point became between about 1.5-3 weeks.

4.1 Application of the Method on the Three Case Study Projects

Figure 2 presents the overall FST distributions over time with the percent FST to each test level at each measurement point included as a data table below. The graph provides two major findings:

a) The degree of FST during the first weeks of the projects varied a lot. However, a common trend was that except for ST of project E, the first three measurement points were higher than the final result. The identified reason for this is that the beginning of a test phase tends to find most of the stopping faults that prevent planned test activities to start.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	final
× Project G -- % FST to FT	75	86	85	70	71	70	70	70	70	69	68	67	68	66	67
▲ Project G -- % FST to ST	100	100	100	100	100	82	83	71	68	68	70	67	66	67	67
◆ Project S -- % FST to FT	100	90	82	48	48	50	53	55	55	56	55	55	52	52	50
✱ Project S -- % FST to ST	67	50	50	50	33	43	44	50	50	53	53	50	53	48	50
— Project E -- % FST to ST	42	31	26	40	57	52	57	56	50	48	47	47	47	47	46

Time

Figure 2. FST Trends for all Projects

b) In all studied projects, the percent FST after the first half of the monitoring period were similar to the final results, i.e. the largest difference is project E with about 10% difference. There are one positive and one negative implication of this trend. The positive implication is that it is early possible to see if the target value is likely to be reached or not. The negative implication is that the process quality does not improve during the project despite correction deliveries that should have higher quality than the first less tested ones. This confirms that it is hard to test quality into software [19]. When investigating the verification activities performed at different measurement points, it was possible to relate the curve changes to certain events during verification. In the studied projects, the following findings were made:

Project G -- FT: The project had some configuration problems in the beginning followed by quite many protocol faults belonging to BT (Basic Test). Since BT is only supposed to cover the fundamental protocol flow, it is likely that most of these faults were stopping and thereby caused re-deliveries and thereby prolonged the verification time. Further, already after a third of the time there were clear indications that the project was not likely to reach the goal of this test level (which was 50 percent).

Project G – ST: From an FST perspective, the start of this test level turned out to be catastrophic. In a post-analysis of the project, it was concluded that the major reasons for this were due a new complex database that caused some slippages and due to a new automated test tool in FT. That is, the new tool caused some test areas to be neglected because it caused too much focus to be put on the areas that the tool covered whereas some other responsibilities of FT were inadequately tested. If this data would have been available directly, it would have been possible to save time by taking corrective actions before the next delivery, i.e. through improved quality assurance in earlier phases. The project could already after a few weeks have known that the goal (49%) would not be met without improvements.

Project S – FT*:* This project started with very high FST levels but soon went down to a level below the set goal (57%). The reason for this is explained in Section 4.2. Another positive aspect of this project is that the declining curve shows that the process quality goes up at the end of the verification stage.

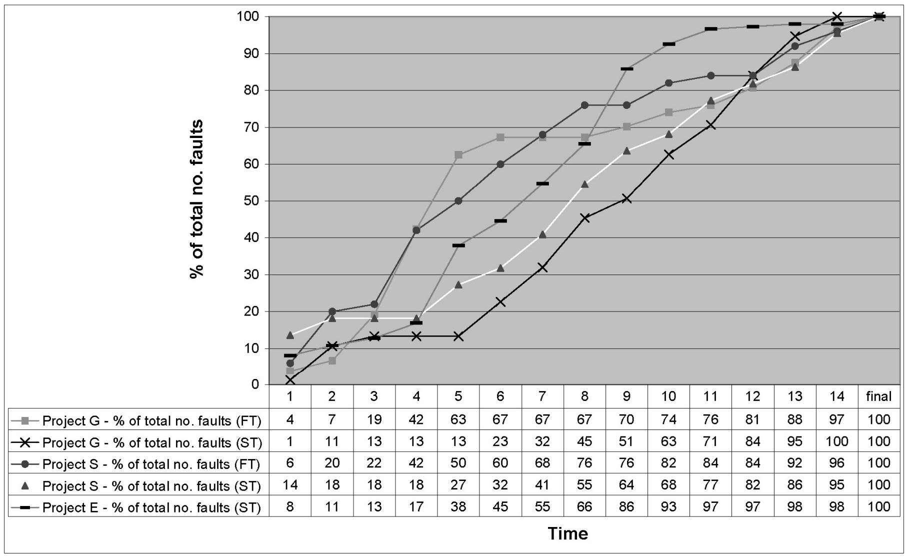
Project S – ST: As for FT of the same project, the FST level quickly goes down to a stable level, which indicates that there were no major surprises in the beginning. This project is different from the others in that it had a temporary low point (no. five). This indicates that the quality level differed between different deliveries. Finally, due to relatively good quality of the input all the way through, the goal of 50% FST was met.

Project E – ST: As can be seen in the figure, ST of project E started with a low FST level in relation to the final result. The reason for this was because a new fault-prone database was introduced and the initial database configuration tests were part of the ST level responsibility. At measurement points four and five, the FST level increased significantly because the functional tests that used the database were also fault prone despite this was a part of the FT responsibility. At the end, the curve flattens out but the ambitious goal of 29 percent could no longer be reached because of the slipping database faults.

4.2 Result Analysis

Applying the proposed method on the case study projects identified some patterns on how the FST trend changes during a verification level. But why are the FST levels so different in the beginning and why do they even out when half of the time has passed? To answer these questions, the FST data was compared against when the faults where found to see if that affected the FST distribution. The fault data is due to confidentiality reasons only presented as percent of the total number of faults.

Figure 3 presents when the faults were found in each project in relation to the same measurement points as in Figure 2. The first important observation is that the faults were distributed rather evenly over time. However, the most apparent exception is ST of project G where the percent faults after a third of the project (point five) still is only 13 percent. Relating this to the FST curve explained why the FST level could decrease from 100 to about 70 percent so fast, i.e. too few faults were found in the beginning to have a reliable FST value. On the other hand, when a significant part of the faults were found early, the curve stabilized early as well, e.g. FT of project G. When cross-checking all projects, one can see that the FST trend reaches a stable level when about 20-30 percent of the faults have been found.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	final
■ Project G - % of total no. faults (FT)	4	7	19	42	63	67	67	67	70	74	76	81	88	97	100
✕ Project G - % of total no. faults (ST)	1	11	13	13	13	23	32	45	51	63	71	84	95	100	100
● Project S - % of total no. faults (FT)	6	20	22	42	50	60	68	76	76	82	84	84	92	96	100
▲ Project S - % of total no. faults (ST)	14	18	18	18	27	32	41	55	64	68	77	82	86	95	100
▬ Project E - % of total no. faults (ST)	8	11	13	17	38	45	55	66	86	93	97	97	98	98	100

Figure 3. Fault distribution Over Time (% of total) – All Projects

The major implication of this is that as long as only a minor part of the faults have been found, the FST values might still change a lot. This implication however causes a problem since the total number of faults is not known until the monitored verification level is finished. This is further discussed in the next section.

5. DISCUSSION

This section provides a discussion on the value, validity and generalizability of the results described in the previous section. The case study results showed that it is possible to get good indications of the average input quality already in the first half of a verification stage. Such data makes it possible to implement process corrections early. Further, by relating the FST status to parts that were verified at certain measurement points, causes of FST can also be revealed. However, when applying this in real-time, the relationship with percent faults found is not possible to do since that is not known until the verification level is completed. Nevertheless, relating a FST value to the percent faults found does not require an exact science. That is, today, managers make decisions about software quality using best guesses; it seems like this will always be the case and the best that researchers can do is to recognize this fact and do what they can to improve the guessing process [8]. Some possible ways to replace the percent of total number of faults relationship are to:

a) Estimate the final number of faults based on the outcome of previous projects and size relationships between the projects.
b) Replace the percent of total number of faults relationship with for example percent of planned total number of test cases.

Project managers commonly already estimate total number of faults, test cases, and test effort as a part of ordinary project planning (to be able to estimate delivery dates), so this should neither be perceived as hard or time-consuming to do.

Regarding the basic FST measurement used, a basic assumption made is that the defined test strategy equals an efficient process. Generic advice cannot be made about what represents an efficient test strategy, i.e. organization and product dependent aspects affect this. Nevertheless, clear responsibilities and few surprises in form of stopping faults and minimal redundant testing should make the defined test strategy rather efficient no matter how the responsibilities are distributed. It is also important to be aware that a perfect test strategy is not the one that finds the most faults but rather one that reflects the most efficient way to assure the quality to a level that makes the customers satisfied. Therefore, the optimal FST goal is as earlier mentioned often not zero either. Instead, the purpose of measuring FST is to use it as a process improvement and project management tool to balance quality against lead-time and cost.

The most common validity critique of FST measurements is that it is subjective, i.e. since the phases belonging of the faults are determined by the people in the projects [6]. However, our advice is to specify the test strategy well and regularly perform post validations of the reported measurements to increase the accuracy. Previous experiences with such validations discovered that although people do not always report correctly, the misclassifications tend to even each other out. Further, people taking part of a measured project should participate in the follow-up analysis of obtained data, i.e. because measures should never be decisive; they just serve as decision support.

Finally, regarding the generalizability of the results, the reported FST levels are only generalizable within the organization, i.e. because they are dependent on test strategies and product complexity. Further, the commonality of the FST Trends should have some degree of generalizability, e.g. that they tend to not change during the second half of projects. However, this is dependent on the similarity of the development process applied. Nevertheless, the method presented is fully replicable in other contexts as long as a fault reporting process with sufficient tool support to collect the measurements is in place.

6. CONCLUSIONS

This paper suggests and validates a measurement-based approach for monitoring the quality of the verification process during development. The paper shows how the approach can provide useful information during development, e.g. early indications of process quality, status of performance goals related to the process quality, and input to improvement actions. The proposed approach determines the input quality between internal verification levels such as unit and function test. Since the actual process quality is not known until a verification level is finished, i.e. when all faults are found, the approach creates trend data from different measurement points to assess the status and predict the final quality. This status information is of interest to check regularly because it is not obvious that the quality always should be as high as possible; it is a trade-off against cost and lead-time.

To investigate the usefulness of the suggested approach, it was applied on five verification levels in three products at Ericsson. The case study results showed that it is possible to obtain good indications of the quality of the verification process already when 20-30 percent of the faults have been found. Further, when 50 percent of the faults have been found, the quality level remains more or less constant. Such information makes it possible to predict the expected final outcome early and if that outcome is not satisfactory, it is at such an early stage not to late to implement improvements to improve the quality. For example, in one of the studied products, the trend data showed that the first deliveries had lower quality than expected. If this data would have been available in real-time, it would have been possible to save time by taking corrective actions before consecutive deliveries, i.e. through improved quality assurance in earlier phases. Further, the case studies demonstrated several examples of how to use the trend data to identify improvement actions. Further work involves replicating the method on more projects to further validate common trend patterns and to determine what the relationship is between the trend data and related factors such as percent of test cases executed.

7. ACKNOWLEDGMENTS

This work was funded jointly by Ericsson 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

[1] Basili, V. 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, 1994.

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

[3] Cangussu, J. W., DeCarlo, R. A., Mathur, A. P., Monitoring the Software Test Process using Statistical Process Control: A Logarithmic Approach, *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT Int. Symposium on Foundations of Software Eng., 28*, 5 (2003), 158–167.

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

[5] Damm, L-O, Lundberg, L., Results from Introducing Component-Level Test Automation and Test-Driven Development, *In press: The Journal of Systems and Software*, Elsevier, Dec. 2005.

[6] Damm, L-O, Lundberg, L., Wohlin C., Faults-slip-through – A Concept for Measuring the Efficiency of the Test Process", To be published in: Wiley Software Process: Improvement and Practice, Special Issue, 2006.

[7] El Emam, K., Drouin, J-N., Melo, W., *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*, , IEEE, Los Alamitos, CA, 1998

[8] Fenton, N., A Critique of Software Defect Prediction Models, *IEEE Transactions on Software Eng., 25*, 5 (1999)

[9] Gopal, A., Krishnan, M.S., Mukhopadhyay, T., Goldenson, D.R., Measurement Programs in Software Development: Determinants of Success, *IEEE Transactions on Software Engineering, 28*, 2002, 863-875.

[10] Grady, R., *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, 1992.

[11] Gray, E., Smith, W., On the Limitations of Software Process Assessment and the Recognition of a Required Re-orientation for Global Process Improvement, *Software Quality Journal, 7*, 1, Kluwer, 1998.

[12] Hevner, A. R., Phase Containment for Software Quality Improvement*, Information and Software Technology Vol. 39,* 13 (1997), 867-877

[13] Kaplan R. S., Norton, D. P., *The Balanced Scorecard*, Harvard Business School Press, Boston, 1996.

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

[15] Niazi M., Wilson, D., Zowghi, D., A Maturity Model for the Implementation of Software Process Improvement: An Empirical Study, *The Journal of Systems and Software, 74*, 2 (2005), Elsevier, 155-172.

[16] Ngwenyama, O., Nielsen, P. A., Competing Values in Software Process Improvement: An Assumption Analysis of CMM from an Organizational Culture Perspective, *IEEE Transactions on Eng. Management, 50*, 1 (2003), 100-112.

[17] Paulk, M. C., Weber, C. V., Curtis, B., and Chrissis, M. B., The Capability Maturity Model: Guidelines for Improving the Software Process, Addison Wesley, 1995.

[18] Rainer, A., Hall, T., Key Success Factors for Implementing Software Process Improvement: A Maturity-Based Analysis, *The Journal of Systems and Software, 62*, 2 (2002), 71-84.

[19] Runeson, P., Holmstedt-Jönsson, M., and Scheja, F., Are Found Defects an Indicator of Software Correctness? An Investigation in a Controlled Case Study, *Proceedings of the 15th International Symposium on Software Reliability Engineering,* IEEE, 2004.

[20] 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

[21] Veenendaal E., *The Testing Practitioner*, UTN Publishers., 2002.