A Model for Software Rework Reduction through a Combination of Anomaly Metrics

Lars-Ola Damm, Lars Lundberg, and Claes Wohlin

Abstract. Analysis of anomalies reported during testing of a project can tell a lot about how well the processes and products work. Still, organizations rarely use anomaly reports for more than progress tracking although projects commonly spend a significant part of the development time on finding and correcting faults. This paper presents an anomaly metrics model that organizations can use for identifying improvements in the development process, i.e. to reduce the cost and lead-time spent on rework-related activities and to improve the quality of the delivered product. The model is the result of a four-year research project performed at Ericsson.

Keywords. Software Metrics, Process Improvement, Defects, Faults, Testing Strategies

1. INTRODUCTION

Despite the increasing demands to produce software in a faster and more efficient way, software development projects commonly spend about 40-50 percent of their development effort on rework that could have been avoided or at least been fixed less expensively (Boehm et al. 2000; Boehm and Basili 2001). Additionally, in a study, Boehm et al. (2000) found that among several approaches applied for reducing the efforts needed for software development, initiatives for reducing the amount of rework gave the highest returns. The major reason for this is that it is cheaper to find and remove faults earlier (Boehm 1981; Shull et al. 2002). In fact, Westland (2004) in one case study observed that faults become exponentially more costly with each phase they remain unresolved. It might seem easy to reduce the amount of rework just by putting more focus on early test activities such as peer reviews. However, peer reviews, analysis tools, and testing catch different types of faults at different stages in the development cycle (Boehm et al. 2000). Further, increased control over the way software is produced does not necessarily solve the problem since even the best processes in the world can be misapplied (Voas 1997). Thus, it is in most cases not obvious how to reduce the amount of rework.

A common way to guide improvement work aimed at rework reduction is by analyzing the problems reported during testing and operation. Grady even claims that problem reports are the most important information source for process improvement decisions (Grady 1992). Case study reports in the literature show some evidence of improvements occurring through the deployment of such metrics programs, e.g. (Daskalantonakis 1992; Butcher et al. 2002). At the same time several reports indicate a high failure rate of metrics programs, e.g. two out of three metrics programs do not last beyond the second year (McQuaild and Dekkers 2004). Further, in one survey, fewer than ten percent of the industry classified metrics programs as positive (Daskalantonakis 1992). One reason for a lack of successful implementation of software metrics is due to erroneous usage, e.g. because simple metrics such as plain fault counts are used in isolation without considering other product or process characteristics (Fenton and Neil 1999), e.g. the number of faults is not directly proportional to the cost of rework (Mashiko and Basili 1997). On the other hand, when companies initiate larger metrics programs, they tend to collect so many metrics that the data collection process becomes too costly and they do not know which data are relevant to use and not (Basili et al. 2002). Thus, there is still a lack of guidance for industry regarding which metrics to use in practice. More

Lars-Ola Damm is with Ericsson AB, Sweden, e-mail lars-ola.damm@ericsson.com
Lars Lundberg and Claes Wohlin is with the School of Engineering, Blekinge Institute of Technology, Sweden, e-mail: {llu|cwo}@bth.se

specifically, concepts such as Orthogonal Defect Classification (ODC) (Chillarege et al. 1992) and the HP Defect Categorization Scheme (Grady 1992) describe how to measure and use a set of fault metrics but it is not clear for practitioners how to apply them. In particular, it is in our experience not obvious how different metrics interrelate and how to combine different metrics for adequate decision support. Especially process metrics require an underlying model of how they interrelate, this is usually missing (Pfleeger 1997).

One way to get support for determining which metrics to apply is to use frameworks such as the Goal Question Metrics paradigm (GQM) (Basili 1994). By using GQM, an organization can ensure that only metrics tied to important organizational goals are used. However, GQM does not state which metrics should address which questions and goals and how generated metrics interrelate (El Emam 1993). Therefore, GQM need to be complemented with a bottom-up approach (Fuggetta et al. 1998). Finally, research in the area of fault metrics is commonly conducted as archival analysis. The result of this is that the daily project work is not fully exposed to the metrics. Therefore, it is not clear if practitioners really can use the results that the researchers advocate.

Through a research project, a department at Ericsson wanted to reduce the amount of rework in their software development projects. The chosen research approach for the project was 'industry-as-laboratory' (Potts 1993), which in practice meant that the researcher during a few years was located full-time in the industrial environment. With this setup, case studies became the main vehicle for conducting the research. They were not just used for status assessments or post-mortem analysis but also to study the long-term effects of applying research concepts on real ongoing development projects. From these case studies, intermediate research results on the usage of different measurements have been published (Damm and Lundberg 2005; Damm and Lundberg 2006; Damm et al. 2006; Damm and Lundberg 2007), i.e. several experiences about which metrics approaches are good and not in different situations have been collected. However, how to best use combinations and dimensions of such metrics has not previously been fully evaluated.

Based on experiences from four years of case studies, this paper presents a model that provides a way to combine different classification dimensions so that they can address different problems and contexts. If for example using GQM as measurement framework and deciding that rework reduction is one of the organizational goals to measure, practitioners can use the suggested model instead of inventing their own metrics based on their own subjective beliefs regarding which rework reduction measures address the goal. A major reason why the model was developed was because we through the experiences from several products and projects learnt that different metrics are more or less informative in different situations, e.g. sometimes most faults are related to a certain process area and sometimes to a certain part of the product. Additionally, varying organizational goals affect which types of metrics are most important to focus on in each particular situation, e.g. cost versus quality goals.

The model is not intended to be a generic solution applicable in the exact same way in different contexts. It is important to realize that although the same base measurements can be used widely, the specific measurement indicators to use may vary depending on the current information need (Mc Garry et al. 2001). Thus, the purpose of the model is rather to serve as a starting point for organizations with similar challenges and improvement goals. This gives organizations the possibility to take the model and adapt to their context and needs instead of starting from scratch, e.g. by defining goals, questions and metrics based on the GQM approach mentioned above. Instead, the model can provide support when goals and questions defined for example through GQM match the objectives of the model, i.e. serve as a measurement pattern (Lindvall et al. 2005).

The choices regarding what to include in the model and not have been determined through multiple case studies, i.e. continuous research conducted over a longer time (Section 6 summarizes the intermediate research results that the model evolved from). Additionally, to demonstrate the practical usefulness of the model, the paper includes a case study application where the model is applied on the anomalies reported in a project at Ericsson.

This paper is outlined as follows. Section 2 introduces concepts and related work to the suggested model. Section 3 describes the context of the research environment in which the results of this paper were developed. The proposed model for supporting metrics-based rework reduction is provided in Section 4 and after that, Section 5 presents the case study application of the model. Section 6 summarizes how the model evolved to include the current contents, i.e. why some metrics were included in the model and some not. Section 7 discusses potential validity threats to the results presented in this paper and finally, Section 8 concludes the work.

2. Concepts and Related Work

This section provides an overview of related work and concepts included in the model described in Section 4. The first sub-section defines terms commonly used in this paper. After that, an overview of different fault metrics is described. Section 2.3 introduces some measurement frameworks related to the model and finally, Section 2.4 provides an overview of the measurement concept that has a central role in the model.

2.1. Definitions of Terms Used in the Paper

A *fault* is according to the IEEE standard "a manifestation of an error in software. A fault, if encountered, may cause a *failure*" (IEEE 1983). This paper uses the terms fault and failure in accordance with this definition. The term *anomaly* is in the paper also used for reported issues that might be faults. That is, in accordance with the IEEE standard definition, an anomaly is "any condition that deviates from expectations based on requirement specifications, design documents, user documents, standards, etc. or from someone's perceptions or experiences" (IEEE 1993). The term *test strategy* is frequently used in this paper and is in this context defined as the distribution of test responsibilities in the organization, i.e. which faults shall be found where. The test strategy is not about how to achieve the responsibilities since that is a part of the test process. Further, we define a *measurement* as the process of assigning a value to an attribute and a *metric* states how we measure something (Mendonca and Basili 2000). Finally, the terms *pre-release* and *post-release* are used for distinguishing faults reported before and after the product is put into operation. However, it should be noted that in the context of this paper, all post-release faults are not necessarily found by customers. That is, the faults are shipped with the product but might as well have been found and removed internally within the organization, e.g. during maintenance work.

2.2. Fault Metrics

In software development, reported faults are typically fixed and then forgotten (Card 1998). However, as indicated in the introduction, more mature organizations have applied several approaches for collecting and using fault metrics. Fault metrics are in practice mostly used on a project level for example for tracking of problem reports, e.g. 75 percent in one survey (Fredericks and Basili 1998). Product metrics such as static/structural behavior are also quite common. However, such measures have become overrated since they are poor quality assessors (Voas 1997). Further, the advocated metrics are commonly either irrelevant in scope, i.e. not scalable to larger programs, or irrelevant in content, i.e. of little practical interest (Fenton and Neil 1999). Typical characteristics of good metrics are that they are informative, cost-effective, simple to understand, and objective (Daskalantonakis 1992). However, since not everything can be objectively measured, this does not mean that practitioners should not use subjective metrics. In fact, choosing only objective metrics may be worse (El Emam 1993).

It is possible to classify faults in many different ways such as by timing, location, cause, severity, and cost. However, the traditional and most widely used approach for using fault metrics in improvement work is plain collection of all reported corrected faults and then to perform a post-mortem Root Cause Analysis (RCA) on them. That is, to analyze the reason why every fault was injected (Card 1998; Leszak et al. 2000). The major drawback with RCA is that it is time-consuming to perform, i.e. most projects have too many faults to even consider analyzing them all (Card 1998). In one study, it for example took on average 19 minutes to analyze each fault (Leszak et al. 2000). Additionally, RCA tends to reveal several different causes that might be hard to prioritize, i.e. the key to successful process improvement is to identify a few areas of improvement and focus on those (Humphrey 2002). Nevertheless, RCA should in our experience still be considered as a complement to classification schemes, e.g. on a smaller sub-set of faults that already have been selected as a focus area to improve (to determine required actions to take to improve the area).

2.3. Measurement Frameworks

Some of the more dominant examples of fault metrics frameworks are Orthogonal Defect Classification (ODC), the HP classification scheme, and the formal standard provided by IEEE (Grady 1992; IEEE 1993). Of these frameworks, ODC is strongly related to the work in this paper and at least in the research community, ODC is probably the most well-known approach for fault classification. ODC includes a set of fault metrics but has two primary types of classification approaches for obtaining process feedback, i.e. fault type and fault trigger classification. Fault type classification can provide feedback on the development process whereas fault trigger classification can provide feedback on the test process (Chillarege 1992). As described in Section 6, ODC type classification did in the context of this research not work well enough to be included in the model. However, a tailored variant of ODC trigger classification was included.

A fault trigger is a test activity that makes a fault surface and an ODC trigger scheme divides faults into test related categories such as coverage, variation, and workload (Butcher 2002). Commonly, ODC fault triggers are used for determining which what types of faults different phases find, i.e. to evaluate the test process. As motivated in Section 6, a tailored variant of the ODC scheme was developed for the studied organization. The case study presented in Section 5 describes an example usage of the tailored scheme and details about the scheme are published in (Damm and Lundberg 2005).

2.4. Faults-Slip-Through (FST) Measurement

Faults-slip-through (FST) is a measurement concept that has a major role in the model described in Section 4. This section introduces the parts of the measure that are important to understand when applied in the model. A detailed measurement description is provided in (Damm et al. 2006). The primary purpose of measuring FST is to make sure that the right faults are found in the right phase, i.e. in most cases early. The norm for what is considered 'right' should be defined in 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 which extent the applied test process adheres to this strategy. This means that all faults that are found later than when the test strategy stated are considered slips (Damm et al. 2006). Several metrics can be obtained when conducting FST analysis on the test phases of a project, i.e. by analyzing which phase each fault belongs to. However, Equations (1) and (2) describe the two metrics considered most useful in the model suggested in this paper, i.e. Phase Input Quality (PIQ) and Phase Output Quality (POQ).

$$\text{PIQ (\% FST to phase X)} = \frac{SF^{1}}{PF^{2}} \quad (1)$$

$$\text{POQ (\% FST from phase X)} = \frac{TS^{3}}{TF^{4}} \quad (2)$$

[1] SF (Should have Found) = Number of faults found in phase X that slipped from earlier phases.
[2] PF (Phase Found) = Total number of faults found in phase X.
[3] TS (Total Slippage) = Total number of faults slipping from phase X (no matter when found).
[4] TF (Total Found) = Total number of faults found in all phases.

As described in Section 6.2, it was after a while concluded that PIQ/POQ analysis should not be performed without relating to the number of faults. More specifically, Fig. 1 illustrates the established relationship between the PIQ and POQ metrics in relation to the total number of faults, i.e. for PIQ in relation to the number of faults found in the phase and for POQ in relation to the total slippage from all phases. In the figure, the relationships many versus few faults and a low versus a high ratio are used. However, it is not explicitly defined what is considered few/many or low/high. The reason is that it is context dependent in relation to what is acceptable. In practice this can most easily be judged by comparing different measurements points to each other, e.g. if a project had 10 slipping faults from unit test and 100 slipping faults from function test, the latter can be considered many and the former can be considered few.

FIGURE 1 should be placed here

Fig. 1. Relationship between FST and number of faults found

From the obtained PIQ/POQ values and number of faults found, it is possible to determine which quadrant fits best for each phase. For example, if the FST-PIQ ratio is low but many faults were found (the lower right quadrant of Fig. 1A), the test strategy is probably not strict enough since it allowed many faults to be found, e.g. some test responsibilities should be moved to earlier phases. However, to get a more complete picture of the situation, the outputs of the two graphs should be combined. For example, the following combinations are in our experience common when comparing the slippage to and from a phase:

- (A: lower left, B: upper left): A high POQ ratio combined with few faults found indicates that the phase had a narrow responsibility in relation to other phases and thus found fewer faults. However, from a slippages point of view, the performance of the phase was not good.
- (A: upper right, B: upper right): In this situation, the phase struggled with low input quality and because of that probably had problems verifying what it should in time before delivering to the next phase. Thus, since the responsibilities of the test strategy are not fulfilled, the compliance to it was low. This then caused a high slippage to later phases.

3. RESEARCH CONTEXT

This section describes the context of the research results presented in this paper, i.e. the research method used and the industrial context of the research.

3.1. Research Approach

The research study presented in this thesis was conducted based on two research approaches, i.e. industry-as-laboratory (Potts 1993) leading to the suggested model and a case study application of the model.

Industry-As-Laboratory

The industry-as-laboratory approach was chosen because then the research becomes problem-focused, i.e. the detailed research questions come from a detailed understanding of the application environment (Potts 1993). Otherwise, it is common that what the researcher thinks is a significant problem is not and vice versa (Potts 1993). Further, the research results evolve through regular feedback cycles from practical application. This approach ensures that the researchers obtain immediate feedback on the applied research method to direct further research (Potts 1993).

To achieve a proper industry-as-laboratory setting, the researcher was during a few years located 100 percent of the time in the industrial environment where the research studies were performed. In this research environment, the researcher performed a number of case studies including both assessment studies and studies on the effects of applying changes to the processes used at the company. More specifically, multiple projects from several products at different Product Development Units (PDUs) have to different degrees been studied. Focus has however been put on four products belonging to one PDU where the researcher totally monitored about 20 projects during approximately four years time. Section 6 describes the intermediate results from the conducted research, which led to the model. That is, an evolutionary description of conclusions drawn from using different measurements. Through this research setup, the practical feasibility of several different measurement methods were evaluated. This was not only performed as a post-mortem analysis on data from finished projects as today is common in the research community. Additionally, measurements were included as a part of the development processes that the company uses. Thus, the measurements methods leading to the model evolved through regular feedback cycles from practical application.

Case Study Application

To demonstrate the practical usefulness of the model, the research study presented in this paper also includes a case study application of the model. Since all the metrics that were required for the model already were included in the anomaly reporting process, the required information could be extracted and analyzed from there. Although the measurements were conducted during the ongoing project, the analysis according to the model was conducted as a post-mortem study. To decrease the risk for incorrect classifications, all reported anomalies were validated before summarizing the result. When the validation of the anomaly reports did not contain sufficient information for the classifications, the involved project members were consulted to determine the correct classification. The anomalies were analyzed by importing them to a spreadsheet application, i.e. to a spreadsheet template that had automated support for anomaly imports and calculations. When including anomaly validation and spreadsheet calculations, on average about two minutes were spent on each anomaly.

3.2. Company Context

The part of Ericsson AB where the research has been conducted is a provider of software systems for mobile networks. The PDU hosting the research environment described in this paper develops a set of products at one location, and one product is also partly developed at an offshore development site. All studied projects developed functionality to be included in new releases of the existing products that already are in full operation at customer sites. The products are built on component-based architectures and are mainly developed in Java and C++.

The development process at the PDU is based on an incremental approach including the traditional development phases: analysis, design, implementation, and test. At the time of the study, each project lasted about 1-2 years. The test activities included code reviews and Unit Test (UT) before delivery to the test department, Function Test (FT) of the integrated components in a simulated environment, and finally System Test (ST) commonly conducted in a test lab running a complete mobile network with focus on system interfaces and non-functional requirements.

The reported anomalies originated from the test phases performed by the test department, i.e. faults found in code reviews and unit test were not reported. Analyzing those faults as well could have provided more information about types of mistakes made at early stages. However, this was for the context of the model and study not considered important since the identified improvements primarily would have affected early phases instead of the test phases. Further, requirements enhancements were not managed in the fault reporting system. Instead, they were handled

separately as change requests. The fault reporting system used at the company is an in-house built tool with similar capabilities as commercial alternatives such as Rational Clear Quest.

4. ANOMALY METRICS MODEL

The metrics model provided in this section was developed in the context described in the previous section and is based on the concepts introduced in Section 2.

Fig. 2 illustrates the metrics model and the text after the figure explains every sub-step in the model. As can be seen in the figure, the model is divided into 4 analysis levels where levels 1-3 identify groups of anomalies to focus on whereas level 4 aggregates the results from previous levels and based on an in-depth analysis of the focus areas, the most important improvements are identified. The analysis levels in the model are based on the findings described in detailed in Section 6. The model may be applied both as a part of a post-mortem analysis or continuously during projects. However, in our experience, the model should preferably be applied continuously during projects since it is important to obtain early feedback. This also requires good tool support for automated metrics collection, preferably in form of automated generation of statistics onto a webpage. The analysis can be performed by anyone that understands how the metrics are obtained. However, it is important that people that are familiar with the studied context make metrics validations and interpretations, i.e. metrics are good for guidance but not absolute recipes (Voas 1999).

FIGURE 2 should be placed here

Fig. 2. Anomaly Metrics Model

Analysis level 1: The basic analysis starts with sorting the reported anomalies after which actions were taken from them, e.g. whether they required a product fix or not. Each fault can only be classified into one category.

1a: Commonly, a majority of the reported anomalies belong to this category of real faults in the software or documentation delivered together with the software.

1b: A not reproducible anomaly is an observed failure during testing that cannot be reproduced by the developer that is assigned to fix it. Getting many such failures might be due to the existence of many intermittent faults in the product. This indicates a robustness problem that probably requires improvements to the product architecture. Insufficient debugging environments are other common reasons of not being able to reproduce the failures.

1c: This category of anomalies occurs when the requirements documentation is vague or incomplete. For example, when a tester and a developer interpret a requirement differently, the tester is likely to submit an anomaly report. In these cases, the anomaly report is defined as an opinion of function report which might also result in a correction. When an organization reports many anomalies of this type, it indicates that the requirements are not clear enough. Note that as mentioned in Section 3.2, requests for new features are at the studied PDU not reported as anomalies.

1d: Duplicate reports of anomalies are common in organizations with parallel projects developing enhancements in the same code base. Common reasons for having many duplicates are in our experience insufficient anomaly coordination between projects/releases or insufficient procedures for branch/merge of product deliverables. Another possible reason for duplicates can be faults that materialize as two or many failures.

Analysis level 2: The purpose of analysis level 2 is to determine if the found faults were injected by the analyzed project or not. The reason for doing this separation is to distinguish origins of faults so that improvements are directed to the right area. The faults injected by the analyzed project are also divided after if they were injected before or after customer release (step 2c, 2d).

2a: When a project is enhancing an existing product (which commonly is the case), testers might find faults in the software that the company already has verified and released. If there are many such faults in a project, the quality of the product baseline is insufficient and might reduce the development speed of later releases (in addition to increased maintenance costs). However, note that in our experience this does not have to happen due to low quality of the delivered software; it might as well be because the next project starts already before testing of the previous release is finished. In our experience, the most common reason is however that the old product functionality is to be used in new way that has not been verified properly before.

2b: Sorting out third-party faults is only relevant for products using software from sub-contractors or external organizations within the company. If there are many faults in the third-party software, it is probably not feasible to do a detailed RCA on the faults but rather preferable to put pressure on the third-party provider to improve the quality of future deliveries.

2c: If the analyzed project has introduced many new faults that were found before release, the next step is to determine FST PIQ and POQ ratios of the faults found.

2d: If the analyzed project has introduced many new faults that were found after the release, only a FST POQ analysis is relevant as the next step since such an analysis is only relevant for test phases (since all faults should have been found before the release).
Analysis level 3: At this level, PIQ and POQ analysis should be performed according to the graphs described in Section 2.4. The purpose with this step is to pinpoint areas that in the next analysis level should be investigated further, e.g. if it is the test strategy or its current application in a test phase that needs to be improved.
3a: PIQ analysis should be performed on each test phase with reported anomalies. As can be seen in the figure, if the analysis result is not in the "OK" quadrant in the figure, a potential improvement area has been identified. This requires further analysis (described in level 4).
3b: POQ analysis should be performed in the same way as PIQ analysis except that POQ analysis can be performed on all test phases in a project (not only those with reported anomalies).
Analysis level 4: The purpose of level 4 is to perform an in-depth analysis of the sets of faults that indicated a need for improvement. Faults can be analyzed from many different angles but through the input to this level, it is possible to focus on the right area(s) as indicated by the FST quadrants. That is, phases with faults/FST ratios corresponding to the upper-right corners of the POQ or PIQ analysis should be prioritized. Additionally, high fault ratios among other fault types (1b, 1c, 1d, 2a, 2b) should also be considered when selecting focus areas. Differences in fault severities might also be important to weigh in, e.g. if two areas have similar fault quantities but one has more severe faults than the other, focus should be put on the former area. Depending on which improvement areas were selected, the steps 4a-f in the figure can then be used as support to decide what actions to take (further described below). Note that in addition to the suggested classifications in steps 4a-f, a traditional RCA might in some cases also be needed, i.e. in cases when the causes of problems are more complex than what just classifications can capture. However, in that case it is important to limit the number of analyzed faults, i.e. related work suggests selecting no more than 20 faults to analyze in a session (Card 1998).
4a: Having many legacy faults can be costly to address if they are spread out in the product. Sorting the faults after which modules they belong to might however pinpoint areas that are extra fault-prone.
4b: Any type of cause analysis is normally impossible on third-party faults but a trigger analysis can at least be performed to determine which test areas that need extra attention.
4c: With few FST and faults, a trigger analysis is suggested on all faults reported in the phase to determine which test areas were covered in relation to expected (since the wrong areas apparently were covered). Additionally, this result should also be compared to the POQ analysis that determined the fault slippage from this phase to later phases, i.e. if there is a high slippage to later phases, the analyzed phase has not completed its responsibilities.
4d: With many faults and few FST, it is the test strategy that needs to be improved, and the only way to determine where to improve is by sorting the faults found by using trigger classification. The trigger area(s) with the highest percentages of faults should be covered early in the test process.
4e: A high percent FST and many faults found is a typical situation of a project flooded with faults in testing. There might be several reasons but from a test perspective, a trigger analysis is probably the best way to determine where the problems are. A module classification might also be able to provide complementary information about what part of the product caused the faults.
4f: When a phase had a significant POQ, it is possible to pinpoint the areas to improve by sorting the FST of that phase into fault trigger classes, i.e. as described in detail in Damm and Lundberg (2005). As can be seen in the figure, the same analysis is preferred in all three areas. However, depending on which quadrant each analyzed phase belongs to, it should have different priorities when deciding where to improve. That is, the upper-right quadrant should have the highest priority since if an analyzed phase ends up there, it has probably caused the most fault slippages of all phases.
4g: If a significant ratio of the anomalies did not result in a correction, they should as motivated in Section 6.7 not be neglected since they might also have a significant impact on the total cost of rework. In our experience, the best way to identify actions to take to reduce the amount of these faults are either by expert opinions by project members or through a detailed RCA, i.e. when the cause is not obvious. Further classifications of these faults are in our experience not feasible.

5. CASE STUDY APPLICATION OF THE MODEL

This section describes the results from applying the model described in Section 4 on one large software development project at Ericsson, i.e. to evaluate the correctness and usefulness of the model in a real industrial

context. Section 5.1 describes the result of applying the model on the project. After that, Section 5.2 describes the practical impact of the analysis, i.e. how the PDU (Product Development Unit) benefited from the analysis results.

5.1. Application of Anomaly Data on the Model

Fig. 3 illustrates the result of applying the model on the anomalies reported in the studied project (including the anomalies reported the first six months after release). The studied project developed new features for an existing product and about 100 persons at two locations were involved in it. The anomaly data in the figure are provided as percent of total number of anomalies analyzed in that step (except level three as motivated in the figure).

As can be seen in Fig. 3, the analysis highlighted several areas (paths) to consider for improvement. Since a significant proportion of the anomalies were legacy faults (30.5 percent), an analysis of the number of faults in each module was made to determine where to focus legacy improvements (see Graph 1 in the figure). As the graph shows, modules I and J had a majority of the faults. Further, the PIQ (Phase Input Quality) analysis showed that FT (Function Test) and ST (System Test) both found relatively many faults but had a relatively low FST (Faults-Slip-Through) from previous phases (22 percent and 26 percent respectively). As described in Section 2.4, this quadrant indicated that the test strategy needed improvements, i.e. the responsibility of some test areas in FT/ST should be moved to earlier phases. As shown in Graph 3 of Fig. 3, the configuration area had the highest proportion of pre-release faults, especially in ST. In relation to what was expected, ST also found relatively many robustness faults. The high proportions of node coverage and human interface faults were expected since those areas covered a larger part of the requirement scope. The POQ (Phase Output Quality) analysis determined that the pre-release slippage from UT (Unit Test) was significant enough to investigate further (43.7 percent). When as specified in the model sorting the FST against the fault trigger categories, the major category was ‘component coverage’. Additionally, some pre-release faults in human interfaces and configurations slipped from UT. Regarding the post-release faults, the POQ analysis determined that most post-release faults should have been found in ST (42.8 percent). The fault trigger analysis presented in Graph 4 shows that most post-release faults belonged to system documentation, configuration, robustness, and redundancy. Finally, the opinion/improvement anomalies were many enough to investigate further (10.1 percent as can be seen in analysis level 1). However, since the model does not recommend further classifications of this type of anomalies, the analysis result of opinion/improvement anomalies was brought as is to the workshops held afterwards (described in the next section). It should also be noted that the severity of the anomalies was also considered when conducting this analysis, i.e. the graphs were generated with the non-severe faults excluded. However, it was concluded that the non-severe faults only had a minor impact, e.g. there were fewer document triggers when removing the low severity faults. Additionally, the PDU determined severity after urgency to fix, not the potential damage if found in the field or cost to fix the fault. Therefore, the severity classification was not considered useful.

FIGURE 3 should be placed here

Fig. 3. Application of the model in fig. 2 on a product release

5.2. Impact of the Analysis Results

When the analysis result was ready, i.e. the data provided in Fig. 3, the result was presented for the management team of the studied PDU. It was from this presentation concluded that the analysis result needed to be presented and discussed with everyone involved in the project so that a proper improvement program could be initiated. This meant that the data were presented for all development units involved in the projects and that every development unit after each presentation discussed the anomaly data. Every unit then compiled a list of the five most important areas to address to improve the efficiency in forth-coming projects. The reason for limiting the number of improvement areas was because no organization with a high market pressure can implement all identified improvements at once. From this output, an improvement program was started to first of all initiate investigations regarding what actions to take, then track the implementation, and finally follow-up the results of the implemented improvements. The major focus areas of the improvement program became to address the most important trigger areas (configuration and robustness), and improve requirements management to reduce the amount of opinion/improvement anomalies. Regarding the configuration and robustness triggers, a pragmatic type classification was also made to provide more specific information about how to reduce the amount of such triggers. That is, the faults were grouped based on patterns discovered when studying those anomaly reports. For example, the robustness triggers were divided into areas such as memory handling, database, thread handling, and security. From this analysis, it was decided to move the responsibilities of some of these areas to earlier phases. Additionally, the PDU initiated a separate investigation aimed at reducing the amount of legacy faults, i.e. by focusing on the

fault-prone modules I and J in Graph 1. In conjunction with this investigation, it was also decided to improve the testability on a module level to improve the UT environment, i.e. to reduce the slippage from UT.
Regarding the high amount of post-release documentation faults, it was concluded that they occurred due to a deliberate choice not to verify all system documentation before release since it could be managed shortly afterwards without affecting the customers. The high ratios of configuration and robustness triggers found after release confirmed the already identified need to do more test of these areas in early phases. Finally, the redundancy faults were found after the release because there was not enough time to fully test that area before as planned. Through the trigger analysis, the effect of this was highlighted, which showed that a similar decision should be avoided in the future.
The improvement program was at the time of writing this paper just started and expected to be performed in steps over a longer time. The measurement work has in any case already fulfilled one of the two primary factors for successful measurement program implementation, i.e. impact on decision making (Gopal 2002) (the other factor is impact on organizational performance).The impact on decision making was large since especially the managers had not before realized how large the improvement potential of some areas really was. Additionally, an even more important impact on decision making was that it is often hard to get resources for improvements that have a significant investment cost, i.e. it is not enough to ask for improvement budgets merely based on a subjective belief that the return on investment will be positive. With quantitative data on the possible benefits, managers could more easily weigh the required investment costs against the possible gains and take less risky decisions.
The improvement suggestions have resulted in concrete requirements that have been implemented in subsequent projects. Unfortunately, the studied PDU has at the same time as these improvements are under implementation initiated a larger process change that makes it impossible to distinguish the effects of the implemented improvements in relation to other changes, i.e. further described in Tomaszewski et al. (2007). Nevertheless, no matter how many of the suggested improvements the PDU actually implements, the analysis was seen as very valuable to them. For example, the usability of the metrics could be observed by noticing how people talked about what they learned from using them (Hartmann and Dymond 2006). Additionally, previous studies on isolated metrics, e.g. (Damm et al. 2006), have shown significant improvements after implementing similar improvements.

6. Model Evolution to the Current Contents

This section describes how the model described in this paper evolved. In particular, the section motivates the decisions taken regarding which metrics to include in the model and not. The evolution description is divided into eight parts, which Fig. 4 outlines. Then, the sub-sections below describe each part, including a reference in each heading stating which analysis level(s) in the model each section affected. Additionally, one major lesson learned is included in each part to highlight the most important finding from studying each area.

FIGURE 4 should be placed here

Fig. 4. Evolution of Measurements Resulting in the Model

6.1. Phase-Detection Effectiveness (Analysis level 3)

During the first period of the research project, the PDU (Product Development Unit) already used a measure for achieving earlier fault detection, i.e. Phase-Detection Effectiveness (PDE). As the name implies, the metric measures the ratio of faults found in subsequent phases. More specifically, it is measured as ‘number of faults found in phase X/(no faults found in phase X + number of faults found in phase X+1)’. This metric is sometimes also denoted defect removal efficiency (Jones 1986). The impact of this metric is that the higher percentage of faults found in the earlier phase, the lower percent fault slippage between those phases. Although this metric seemed to serve its purpose well, it was later abandoned because a few issues were discovered after using it a while in practice. The two major effects presented below might sound unlikely in practice, but they are not simply because ‘people work according to how they are measured’ (Austin 1996).

- To optimize the result of this metric, the easiest way is to test as much as possible in the earlier phase. Although this is basically good, this way of thinking contradicts the fact that different techniques are more cost-effective on finding different types of faults than others (Runeson et al. 2006), i.e. some faults are not cheaper to find early.
- To get a good result from this metric, i.e. a low fault-slippage ratio, the input quality to the first measured phase should be as low as possible so that many faults can be found there. A possible counter-productive result of this is

that less quality assurance is performed before delivery to the first measured test phase and then integration has to test and re-test for a long time before the product reaches a good enough quality. Solving this problem by starting to report anomalies in all test phases is not either a good solution since it causes an administrative overhead, i.e. it eliminates a lot of the gains from finding most faults early.
***Lesson learned**: "Just because a metric seems to serve its purpose, e.g. increased efficiency, people might react in a way that the metric causes counter-productive behavior"*

6.2. Phase-belonging Measurement (Analysis level 3)

To address the problems with the phase-detection metric, the metric called FST (Faults-Slip-Through) was developed based on existing concepts, e.g. (Berling et al. 2003). However, as the issues with the phase-detection metric were discovered, the PDU decided to replace the phase-containment metric with the new FST metric i.e. the PIQ metric described in Section 2.4. The metric was initially only used to monitor the projects in the studied PDU but after a while, it was spread to many other PDUs at the company. Based on conclusions drawn from the experiences obtained from several different projects in different contexts, several findings were made. Most importantly, it was concluded that FST should as most other anomaly measures not be connected to rewards/bonuses since that causes counter-productive behavior. Further, FST should not be used in isolation, e.g. it should be interpreted together with the number of faults found. That is, sometimes many faults are found late despite low FST ratios, i.e. because the test strategy allows most types of faults to be found late (see example in Section 5.1).
***Lesson learned**: "A project reporting many faults but at the same time has a low FST provides indications that the test strategy is the problem, not the compliance to it"*

6.3. Cause-oriented Type Classification (Analysis level 4)

It was soon realized that to improve from FST measurements, a scheme for fault types was needed to find areas to improve within a phase causing significant slippages to later phases. Most efforts to achieve this were put on trying to adopt the most widely known type classification scheme, i.e. ODC (Orthogonal Defect Classification) type classification. However, as also reported in related work (Henningsson and Wohlin 2004; El Emam and Wieczorek 1998), problems with classifying fault according to this scheme has been experienced, e.g. a lack of agreement between classifiers and overlapping categories. However, some degree of repeatability was not considered a major issue in practice. Instead, the major issue was that the fault classes were hard to connect to concrete improvements, e.g. having many 'assignment' faults is not easy to address. In fact, the most important characteristic of a metric is that it can be interpreted within the context of the measurement purpose (El Emam et al. 1993). Some attempts to solve this were made by making a tailored scheme that fits the PDU better. It however turned out that the PDU had so many different kinds of faults that creating a suitable scheme working across projects and products was too hard despite focusing on a specific domain. The only thing that worked well was project specific cause groupings as for example done on a sub-set of faults in the case study described in Section 5.
***Lesson learned**: "Classification by fault causes is hard since to obtain such a scheme with high interpretability, the categories easily becomes both product and process dependent"*

6.4. Test-oriented Type Classification (Analysis level 4)

With the discovered limitations of type classification, the focus was shifted to make ODC trigger classification work instead. The concept of classifying after test activities instead is technically easier since the fault classification can be made based on the tasks performed instead of varying underlying causes. However, as also reported in another study, applying the ODC triggers turned out to be inconsistent due to an overly complex scheme (Leszak et al. 2002). It was also common that one fault could fit two trigger types, i.e. as also reported in Lutz and Mikulski (2004). Additionally, as for ODC type classification, the interpretability of the results from applying ODC triggers was insufficient. To address these issues, a tailored version was developed to fit the context of the PDU better. This was achieved by iteratively trying candidate versions on sets of faults until satisfactory. Later the tailored scheme was adapted to fit a set of PDUs within the company. This more generic scheme was easier to develop since the PDUs had similar test processes and trigger categories turned out to be less product dependent than type-categories.
***Lesson learned**: "Trigger schemes are hard to use if not tailored to the test context of the organization"*

6.5. Module-oriented Type Classification (Analysis level 4)

A common characteristic of classification frameworks such as ODC is that that they only provide feedback on the development process, e.g. they do not consider which modules in the product had more or fewer faults. This is an important limitation because the goal should always be to support causal analysis of both product and process

(Kelsey 1997). Therefore, some studies on the most common product-oriented classification technique, i.e. module classification, were performed on a few projects. The studies confirmed the previously established Pareto rule for modules i.e. that about 20 percent of the modules contains about 80 percent of the faults (Boehm and Basili 2001). However, it was also discovered that which modules that were fault prone tended to differ between projects. This was later confirmed to be because the fault-proneness is largely dependent on the degree of modification (Tomaszewski and Damm 2006), i.e. commonly also denoted 'code churn'. Therefore, it was concluded that results from module classifications should be used with care. In later studies, module classification of legacy faults turned out to be more reliable since such faults to a larger extent are in modules that are always used and thus are commonly executed in testing and operation.

***Lesson learned**: "Fault distributions across modules is largely dependent on the degree of modification"*

6.6. Combined FST and trigger classification (Analysis levels 3 and 4)

Since every trigger in ODC trigger classification is connected to a specific phase, the initial objective with trying to adopt the scheme was to use it as a replacement of the existing way of measuring FST. That is, since the trigger distribution would tell the degree of slippage between phases. However, it was discovered that even if using a tailored scheme, having triggers connected to only one phase was impossible without having a very large number of triggers (which would implicitly make the scheme practically infeasible). This resulted in a combination of the two metrics that more clearly could determine which areas that needed to be improved to reduce the amount of fault slippages (Damm and Lundberg 2005).

***Lesson learned**: "Combining different fault metrics is powerful because such combinations commonly create synergy effects"*

6.7. 'Not fault' anomalies (Analysis levels 1 and 2)

After a while, it was discovered that anomalies for which no corrective action was taken should not be neglected since they sometimes can provide important improvement input. This is because they commonly comprise a significant portion of the anomalies, e.g. in the studied projects between 10-30 percent. Further, they can cost as much as corrected faults, e.g. not reproducible faults are often at least as expensive as resolved faults since they require a lot of debugging and opinion faults commonly require time-consuming investigations before a decision is taken. Still, it is common that research studies and measurement frameworks fail to consider faults for which no action was taken (Westland 2004).

Lesson learned: *"Anomalies for which no corrective action is taken often have a significant effect on the cost of rework"*

6.8. Multi-Measurement Analysis (Analysis level 4)

As more and more projects, products, and PDUs were studied, the clearer it became that a set of metrics is required to get sufficient information on where to focus improvements, e.g. FST, fault types, triggers, and modules. A major reason for this was that even different projects belonging to the same product commonly needed different anomaly measures to pinpoint improvement areas adequately. Therefore, it was concluded that a more flexible approach for anomaly measurements is needed, which resulted in the model presented in this paper. The need for multiple measurements has also been reported in other industrial studies. That is, commonly an organization requests to start with only one measure to minimize costs. However, this can be misleading because a combination of metrics should be tracked and analyzed to obtain a more accurate picture (Daskalantonakis 1992).

Lesson learned: *"Different products/projects have different problems, which makes some metrics more useful than others in different situations"*

6.9. Other Considered but not Included Metrics

Some other metrics have been considered to be evaluated as a part of the research as well, but they were not since their benefit was in the context of the research considered marginal in relation to the cost of implementing them. Besides PDE measurement, the primary considered but excluded metric was to measure FST as when faults are injected instead, i.e. Phase-Containment Effectiveness (Hevner 1997; Six Sigma 2007). In our experience, most faults are injected in the coding phase (at least when not including requirements faults in the anomaly reports). Therefore, this measure did not provide much useful feedback to the test process and was discarded.

7. VALIDITY THREATS

This section discusses the validity threats to the work presented in this paper. The first two sections describe the validity threats to the model and case study presented in this paper. The third section describes to what degree the results can be generalized, i.e. the external validity.

7.1. Validity of Model

The primary validity threat to the suggested model is construct validity, i.e. if a method measures what you think it measures (Robson 2002). For construct validity, it must be considered how well the metrics used in the study reflect the real-world. For example, is it certain that a low FST ratio combined with many faults found in a phase implies that the test strategy needs to be improved? The primary approach applied to increase the construct validity of the model was through a series of case studies in the industrial setting, i.e. industry-as-laboratory. Thereby, it was tested if the results of applying the model really fit the reality. Additionally, from the case study in Section 5, the people involved in various parts of the studied project all confirmed that the outputs from the model provided an aggregated and accurate picture of major sources of rework. Finally, through the close and prolonged research involvement, the risk for not getting a comprehensive understanding of the context to interpret it correctly could be minimized.

7.2. Validity of the Case Study

The major validity concern of the case study is whether is it possible to draw correct conclusions from the results or not, i.e. if the measurement results were reliable (Robson 2002). Two types of reliability threats are particularly important to this type of case study:

Researcher bias that involves assumptions and preconceptions that affect the way the researcher behaves, e.g. how to perform the measurements (Robson 2002). The risk for biases or errors in the data collection process was low since the measurement data were collected with automated tools. Regarding the metrics classifications, the project members made the preliminary classifications. Thus, there was little room for researcher bias when validating the measurement data.

Respondent Bias that occurs when the respondents for various reasons do not provide the right information (Robson 2002). Since the procedure for anomaly reporting involves a pair validation of reported fields, e.g. the testers validate that the developers have classified the anomalies correctly, the risk for respondent bias was considered low. To identify eventual remaining incorrect classifications, the researcher post-validated the anomalies as well (see Section 5.1).

7.3. Generalizability

Since the measures in the model have been applied on several different products and projects, the model can be transferred to other parts of the company. Additionally, the model should have some degree of external generalizability to other companies as well since the studied projects that provided input to the model were developed in rather different contexts developing software for different products using different processes. For example, the size of the studied projects varied significantly and some projects were developed at one site whereas others included distributed development. However, the measurement data from the case study presented in this chapter are only valid within the context of the studied product.

Regarding transferability to other organizations, two limitations have been identified. First, there is a prerequisite that needs to be fulfilled to achieve a successful application of the suggested model. That is, the candidate organization must have an established anomaly process with sufficient tool support. Additionally, the more anomaly reports an organization has, the more useful the model is, i.e. organizations with very few anomaly reports will gain more from using traditional RCA instead.

Additionally, in some contexts, adaptations to the model are most likely required. For example, a very large project with several sub-projects should probably sort the anomalies both after test phases and which sub-project the anomalies came from. Finally, as stated in the introduction, it is important to keep in mind that the model is not intended to serve as a generic solution bur rather as a starting point to adapt to local needs and circumstances.

8. CONCLUSIONS

This paper describes an anomaly metrics model for identifying improvements that can reduce the amount of rework in software development projects. More specifically, a model for anomaly analysis has been developed to identify groups of anomalies that could have been prevented or at least been found and fixed less expensively. The model

evolved from several case studies conducted during a few years of research at Ericsson. The model includes several levels of analysis, starting with sorting out which anomalies that resulted in fault corrections or not. Then, the detailed fault analysis is primarily based on fault slippage analysis and fault trigger analysis. The practical usefulness of the model was evaluated through a case study on a product release. In the case study, the model turned out to be very useful for identifying the primary improvement areas that the PDU needed to address to reduce the amount of rework. The major contributions of this paper are:

- Industry-as-laboratory research in several industrial products and projects during a few years that resulted in a model describing a combination of metrics to use when the organizational goal is to achieve earlier anomaly detection (including how the metrics interrelate, order of usage, and how to interpret them). For organizations with similar challenges, the model can serve as a starting point for how to work with rework reduction.
- A case study application of the suggested model
- An evolutionary description of why some metrics were considered better drivers to achieve early anomaly detection than others

9. 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

Austin, R., 1996. Measuring *and Managing Performance in Organizations*, Dorset House, NY.

Basili, V.R., 1994. “Software Modeling and Measurement: The Goal Question Metric Paradigm”. Comp. Science Tech. Report Series. CS-TR-2956 University of Maryland.

Basili, V.R., McGarry, F.E., Pajerski R., and Zelkowitz, M., 2002. “Lessons learned from 25 years of process improvement: the rise and fall of the NASA software engineering laboratory”, *Proceedings of the 24th International Conference on Software Engineering*, pp. 69-79.

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

Boehm, B. W. and Basili, V., 2001. “Software Defect Reduction Top 10 List”, *IEEE Computer*, Vol. 34, No. 1, pp. 135-138.

Boehm, B.W., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Winsor Brown, A., Chulani, S., and Abts, C., 2000. *Software Cost Estimation with Cocomo II*, Prentice Hall, Upper Saddle River, New Jersey.

Boehm, B.W., *Software Engineering Economics*, Prentice-Hall, 1981.

Butcher, M., Munro, H., and Kratschmer, T., 2002. “Improving Software Testing via ODC: Three Case Studies”. *IBM Systems Journal* Vol. 41, no. 1, pp. 31–44.

Card, D.N., 1998. “Learning from our Mistakes with Defect Causal Analysis” *IEEE Software*, Vol. 15, no.1, pp. 56–63.

Chillarege, R., Bhandari, I., Chaar, M., Halliday, D., Moebus, B., Ray, B., and Wong, M.Y., 1992. “Orthogonal Defect Classification—A concept for In-Process Measurement”, *IEEE Transactions on Software Engineering*, Vol. 18, pp. 943-956.

Damm L.-O., and Lundberg, L., 2005. “Identification of Test Process Improvements by Combining ODC Triggers and Faults-Slip-Through”, *Proceedings of the 4th International Symposium on Empirical Software Engineering*, IEEE, pp. 152-161.

Damm, L.-O., and Lundberg, L., 2006. ”Results from Introducing Component-Level Test Automation and Test-Driven Development”, *Journal of Systems and Software*, Elsevier, Vol. 79, pp. 1001-1014.

Damm, L.-O., and Lundberg, L., 2007. “Company-wide Implementation of Metrics for Early Software Fault Detection”, Proceeding of the 29th International Conference on Software Engineering(ICSE), *IEEE*, Minneapolis, USA.

Damm, L.-O., Lundberg, L., and Wohlin, C., 2006. “Faults-Slip-Through – A Concept for Measuring the Efficiency of the Test Process”, *Journal of Software: Process Improvement and Practice*,, Wiley InterScience, Vol. 11, no. 7, pp. 47-59.

Daskalantonakis, M., 1992. “A practical view of software measurement and implementation experience within Motorola”. *IEEE Transactions on Software Engineering*, Vol. 18 , no. 11, pp 998-1010.

El Emam, K., Moukheiber, N., and Madhavji, N. H., 1993. “Empirical Evaluation of the G/Q/M Method”. *Proceedings of the 1993 CAS Conference*, Toronto, pp. 265-289.

El Emam K., and Wieczorek, I., 1998. The Repeatability of Code Defect Classifications, Tech. Report. International Software Engineering Research Network, *ISERN-98-09*.

Fenton, N., and Neil, M., 1999. “Software Metrics: Successes, Failures and New Directions”, *Journal of Systems and Software*, Elsevier, Vol. 47, no. 1, pp. 149-157.

Fredericks M., and Basili, V.R., 1998. “Using Defect Tracking and Analysis to Improve Software Quality”, *Tech. Rep. DACS*-SOAR-98-2, DoD Data Analysis Center for Software.

Fuggetta, A., Lavazza, L., Morasca, S., Cinti, S. Oldano, G., and Orazi, E., 1998. “Applying GQM in an industrial software factory”, *ACM Transactions on. Software Engineering Methodology*, Vol. 7, no. 4, pp. 411-448.

Gopal, A., Mukhopadhyay, T., Krishnan, M.S., and Goldenson, D.R., 2002. “Measurement Programs in Software Development: Determinants of Success,” *IEEE Transactions on Software Engineering*, vol. 28, no. 9, pp. 863-875.

Grady, R., 1992. Practical Software Metrics for Project Management and Process Improvement, Prentice Hall, Upper Saddle River, NJ.

Hartmann, D. and R. Dymond, 2006. “Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value”. In: proceedings of Agile 2006 conference, *IEEE*, pp. 126–134.

Henningsson, K., and Wohlin, C., 2004. Assuring Fault Classification Agreement – An Empirical Evaluation, *Proceedings of the 3rd International Symposium on Empirical Software Engineering*, *IEEE*, pp. 95-104.

Hevner, A., 1997. “Phase Containment for Software Quality Improvement”, *Information and Software Technology* Vol. 39. pp. 867-877.

Humphrey, W.S., 2002. *Winning with Software – An Executive Strategy*, Pearson Education, Upper Saddle River, NJ.

IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology.

IEEE Std. 1044-1993, IEEE Standard Classification for Software Anomalies.

IEEE Std 1061-1998. IEEE Standard for a Software Quality Metrics Methodology.

Jones, C., 1986. *Programming Productivity*, McGraw-Hill, Inc., NY.

Kelsey, R. B., 1997. Integrating a defect typology with containment metrics, *ACM SIGSOFT Software Engineering Notes*, Vol. 22, no. 2, pp 64-67.

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

Leszak, M., Perry, D., and Stoll, D., 2002. “Classification and Evaluation of Defects in a Project Retrospective”, *Journal of Systems and Software*, Elsevier, Vol. 61, no. 3, pp. 173-187.

Lindvall, M., Donzelli, P., Asgari, S., and Basili, V.R., 2005. Towards Reusable Measurement Patterns, *Proceedings of the 11th IEEE International Software Metrics Symposium*, IEEE, pp. 21.

Lutz R., and Mikulski, C., 2004. “Empirical Analysis of Safety-Critical Anomalies During Operations”, *IEEE Transactions on Software Engineering*, Vol. 30, no. 3, , pp. 172-180.

Mashiko, Y., and Basiili, V.R., 1997. “Using the GQM Paradigm to Investigate Influential Factors for Software Process Improvement”, *Journal of Systems and Softwa*re, Elsevier, Vol. 36, no. 1, pp. 17-32.

McGarry, J., D. Card, C. Jones, B. Layman, E. Clark, J. Dean, and F. Hall, 2001. *Practical Software Measurement: Objective Information for Decision Makers*. AddisonWesley Professional, Boston, MA.

McQuaild, A., Dekkers, C.A., 2004. ”Steer Clear of Hazards on the Road to Software Measurement Success”, *Software Quality Professional*, Vol. 6, no. 2, pp. 27 - 33.

Mendonca, M. G., and Basili, V. R., 2000. Validation of an Approach for Improving Existing Measurement Frameworks, *IEEE Transactions on Software Engineering*, Vol. 26, no. 6, pp. 484-499.

Pfleeger, S.L., 1993. “Lessons Learned in Building a Corporate Metrics Program,” *IEEE Software*, Vol. 10, no. 3, pp. 67-74.

Pfleeger, S.L., 1997. ”Status Report on Software Measurement”, *IEEE Software*, Vol. 14 , no. 2, pp 33-43.

Potts, C., 1993. ”Software-Engineering Research Revisited”, *IEEE Software*, Vol. 10, no. 5, pp. 19-28.

Robson, C., 2002. *Real world research*, 2nd ed., Blackwell Publishers, Oxford, UK

Runeson, P., Andersson, C., Thelin, T., Andrews A., and Berling, T., 2006. “What Do We Know about Defect Detection Methods?”, *IEEE Software*, Vol. 23, no. 3, pp. 82- 90.

Six Sigma, 2007. http://software.isixsigma.com, Last accessed: 2007-04-02.

Tomaszewski, P., and Damm, L-O. 2006. "Comparing the Fault-Proneness of New and Modified Code - An Industrial Case Study" Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2006), pp.2-7, Rio de Janeiro, Brazil, 21-22.

Tomaszewski, P., P. Berander, and L.-O. Damm, 2007. From Traditional to Streamline Development - Opportunities and Challenges. To be published i*n: Software: Process Improvement and Practice*, DOI: 10.1002/spip.355.

Voas, J., 1997. “Can Clean Pipes Produce Dirty Water?” IEEE Software Vol. 14, no. 4, pp. 93-95.

Voas, J., 1999. “Software Quality’s Eight Greatest Myths”, *IEEE Software* Vol. 16, No. 5, pp. 118–121.

Westland, J. C. 2004. “The Cost Behavior of Software Defects”, *Decision Support Systems*, Elsevier, Vol. 37, no. 2, pp 229-238.

Weyuker, E. J., 2001. “Transitioning from Academia to Industrial Research”, *Journal of Systems and Software*, *Elsevier*, Vol. 59, no. 1.

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