Revisiting the Problem of Using Problem Reports for Quality Assessment

Parastoo Mohagheghi, Reidar Conradi, Jon Arvid Børretzen
Department of Computer and Information Science
Norwegian University of Science and Technology
No-7491, Trondheim- Norway
{parastoo, conradi, borretze}@idi.ntnu.no

ABSTRACT
In this paper, we describe our experience with using problem reports from industry for quality assessment. The non-uniform terminology used in problem reports and validity concerns have been subject of earlier research but are far from settled. To distinguish between terms such as defects or errors, we propose to answer three questions on the scope of a study related to what (problem appearance or its cause), where (problems related to software; executable or not; or system), and when (problems recorded in all development life cycles or some of them). Challenges in defining research questions and metrics, collecting and analyzing data, generalizing the results and reporting them are discussed. Ambiguity in defining problem report fields and missing, inconsistent or wrong data threatens the value of collected evidence. Some of these concerns could be settled by answering some basic questions related to the problem reporting fields and improving data collection routines and tools.

Categories and Subject Descriptors
D.2.8 [Software Engineering]: Metrics- product metrics, process metrics; D.2.4 [Software Engineering]: Software/Program Verification- reliability, validation.

General Terms
Measurement, Reliability.

Keywords
Quality, defect density, validity.

1. INTRODUCTION
Data collected on defect or faults (or in general problems) are used in evaluating software quality in several empirical studies. For example, our review of extant literature on industrial software reuse experiments and case studies verified that problem-related measures were used in 70% of the reviewed papers to compare quality of reused software components versus the non-reused ones, or development with systematic reuse to development without it. However, the studies report several concerns using data from problem reports and we identified some common concerns as well. The purpose of this paper is to reflect over these concerns and generalize the experience, get feedback from other researchers on the problems in using problem reports, and how they are handled or should be handled.

In this paper, we use data from 6 large commercial systems all developed by the Norwegian industry. Although most quantitative results of the studies are already published [4, 12, 18], we felt that there is a need for summarizing the experience in using problem reports, identifying common questions and concerns, and raising the level of discussion by answering them. Examples from similar research are provided to further illustrate the points. The main goal is to improve the quality of future research on product or process quality using problem reports.

The remainder of this paper is organized as follows. Section 2 partly builds on work of others; e.g., [14] has integrated IEEE standards with the Software Engineering Institute (SEI)’s framework and knowledge from four industrial companies to build an entity-relationship model of problem report concepts, and [9] has compared some attributes of a number of problem classification schemes (the Orthogonal Defect Classification-ODC [5], the IEEE Standard Classification for Software Anomalies (IEEE Std. 1044-1993) and a classification used by Hewlett-Packard). We have identified three dimensions that may be used to clarify the vagueness in defining and applying terms such as problem, anomaly, failure, fault or defect. In Section 3 we discuss why analyzing data from problem reports is interesting for quality assessment and who the users of such data are. Section 4 discusses practical problems in defining goals and metrics, collecting and analyzing data, and reporting the results through some examples. Finally, Section 5 contains discussion and conclusion.

2. TERMINOLOGY
There is great diversity in the literature on the terminology used to report software or system related problems. The possible differences between problems, troubles, bugs, anomalies, defects, errors, faults or failures are discussed in books (e.g., [7]), standards and classification schemes such as IEEE Std. 1044-1993, IEEE Std. 982.1-1988 and 982.2-1988 [13], the United Kingdom Software Metrics Association (UKSMA)’s scheme [24] and the SEI’s scheme [8], and papers; e.g., [2, 9, 14]. The intention of this section is not to provide a comparison and draw conclusions, but to classify differences and discuss the practical impacts for research. We have identified the following three
questions that should be answered to distinguish the above terms from one another, and call these as problem dimensions:

What- appearance or cause: The terms may be used for manifestation of a problem (e.g., to users or testers), its actual cause or the human encounter with software. While there is consensus on “failure” as the manifestation of a problem and “fault” as its cause, other terms are used interchangeably. For example, “error” is sometimes used for the execution of a passive fault, and sometimes for the human encounter with software [2]. Fenton uses “defect” collectively for faults and failures [7], while Kajko-Mattsson defines “defect” as a particular class of cause that is related to software [14].

Where- Software (executable or not) or system: The reported problem may be related to software or the whole system including system configuration, hardware or network problems, tools, misuse of system etc. Some definitions exclude non-software related problems while others include them. For example, the UKSMA’s defect classification scheme is designed for software-related problems, while SEI uses two terms: “defects” are related to the software under execution or examination, while “problems” may be caused by misunderstanding, misuse, hardware problems or a number of other factors that are not related to software. Software related problems may also be recorded for executable software or all types of artefacts: “Fault” is often used for an incorrect step, logic or data definition in a computer program (IEEE Std. 982.1-1998), while a “defect” or “anomaly” [13] may also be related to documentation, requirement specifications, test cases etc. In [14], problems are divided into static and dynamic ones (failures), where the dynamic ones are related to executable software.

When- detection phase: Sometimes problems are recorded in all life cycle phases, while in other cases they are recorded in later phases such as in system testing or later in field use. Fenton gives examples of when “defect” is used to refer to faults prior to coding [7], while according to IEEE Std. 982.1-1998, a “defect” may be found during early life cycle phases or in software mature for testing and operation [from 14]. SEI distinguishes the static finding mode which does not involve executing the software (e.g., reviews and inspections) from the dynamic one.

Until there is agreement on the terminology used in reporting problems, we must be aware of these differences and answer the above questions when using a term.

Some problem reporting systems cover enhancements in addition to corrective changes. For example, an “anomaly” in IEEE Std. 1044-1993 may be a problem or an enhancement request, and the same is true for a “bug” as defined by OSS (Open Source Software) bug reporting tools such as Bugzilla [3] or Trac [23]. An example of ambiguity in separating change categories is given by Ostrand et al. in their study of 17 releases of an AT&T system [20]. In this case, there was generally no identification in the database of whether a change was initiated because of a fault, an enhancement, or some other reason such as a change in the specifications. The researches defined a rule of thumb that if only one or two files were changed by a modification request, then it was likely a fault, while if more than two files were affected, it was likely not a fault. We have seen examples where minor enhancements were registered as problems to accelerate their implementation and major problems were classified as enhancement requests (S5 and S6 in Section 4).

In addition to the diversity in definitions of a problem, problem report fields such as Severity or Priority are also defined in multiple ways as discussed in Section 4.

3. QUALITY VIEWS AND DEFECT DATA
In this section, we use the term “problem report” to cover all recorded problems related to software or other parts of a system offering a service, executable or non-executable artefacts, and detected in phases specified by an organization, and a “defect” for the cause of a problem.

Kitchenham and Pfleeger refer to David Garvin’s study on quality in different application domains [15]. It shows that quality is a complex and multifaceted concept that can be described from five perspectives: The user view (quality as fitness for purpose or validation), the product view (tied to characteristics of the product), the manufacturing view (called software process view here or verification as conformance to specification), the value-based view (quality depends on the amount a customer is willing to pay for it), and the transcendental view (quality can be recognized but not defined). We have dropped the transcendental view since it is difficult to measure, and added the planning view (quality as conformance to plans) as shown in Figure 1 and described below (“Q” stands for a Quality view). While there are several metrics to evaluate quality in each of the above views, data from problem reports are among the few measures of quality being applicable to most views.

Figure 1. Quality views associated to defect data, and relations between them

Q1. Evaluating product quality from a user’s view. What truly represents software quality in the user’s view can be elusive. Nevertheless, the number and frequency of defects associated with a product (especially those reported during use) are inversely proportional to the quality of the product [8], or more specific to its reliability. Some problems are also more severe from the user’s point of view.

Q2. Evaluating product quality from the organization’s (developers’) view. Product quality can be studied from the organization’s view by assuming that improved internal quality indicators such as defect density will result in improved external behavior or quality in use [15]. One
example is the ISO 9126 definition of internal, external and quality-in-use metrics. Problem reports may be used to identify defect-prone parts and take actions to correct them and prevent similar defects.

Q3. Evaluating software process quality. Problem reports may be used to identify when most defects are injected, e.g., in requirement analysis or coding. Efficiency of Verification and Validation (V&V) activities in identifying defects and the organization’s efficiency in removing such defects are also measurable by defining proper metrics of defect data [5].

Q4. Planning resources. Unresolved problems represent work to be done. Cost of rework is related to the efficiency of the organization to detect and solve defects and to the maintainability of software. A problem database may be used to evaluate whether the product is ready for roll-out, to follow project progress and to assign resources for maintenance and evolution.

Q5. Value-based decision support. There should be a trade-off between the cost of repairing a defect and its presumed customer value. Number of problems and criticality of them for users may also be used as a quality indicator for purchased or reused software.

Table 1. Relation between quality views and problem dimensions

<table>
<thead>
<tr>
<th>Quality view</th>
<th>Problem Dimension</th>
<th>Examples of problem report fields to evaluate a quality view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1-user, Q4-planning and Q5-value-based</td>
<td>what-external appearance where-system, executable software or not (user manuals), when-field use</td>
<td>IEEE Std. 1044-1993 sets Customer value in the recognition phase of a defect. It also asks about impacts on project cost, schedule or risk, and correction effort which may be used to assign resources. The count or density of defects may be used to compare software developed in-house with reused.</td>
</tr>
<tr>
<td>Q2-developer and Q3-process</td>
<td>what-cause, where-software, executable or not, when-why all phases</td>
<td>ODC is designed for in-process feedback to developers before operation. IEEE Std. 1044-1993 and the SEI’s scheme cover defects detected in all phases and may be used to compare efficiency of V&amp;V activities. Examples of metrics types of defects and the efficiency of V&amp;V activities in detecting them.</td>
</tr>
</tbody>
</table>

Table 1 relates the dimensions defined in Section 2 to the quality views. E.g., in the first row, “what-external appearance” means that the external appearance of a problem is important for users, while the actual problem cause is important for developers (Q2-developer). Examples of problem report fields or metrics that may be used to assess a special quality view are given. Mendonça and Basili [17] call identifying quality views as identifying data user groups.

We conclude that the contents of problem reports should be adjusted to quality views. We discuss the problems we faced in our use of problem reports in the next section.

4. INDUSTRIAL CASES

Ours and other’s experience from using problem reports in assessment, control or prediction of software quality (the three quality functions defined in [21]) shows problems in defining measurement goals and metrics, collecting data from problem reporting systems, analyzing data and finally reporting the results. An overview of our case studies is shown in Table 2.

Table 2. Case Studies using data from problem reports

<table>
<thead>
<tr>
<th>System Id. and description</th>
<th>Approximate size (KLOC) and programming language</th>
<th>No. of problem reports</th>
<th>No. of releases reported on</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1- Financial system</td>
<td>Not available (but large) in C, COBOL and COBOL II</td>
<td>52</td>
<td>3</td>
</tr>
<tr>
<td>S2- Controller software for a real-time embedded system</td>
<td>271 in C and C++</td>
<td>360</td>
<td>4</td>
</tr>
<tr>
<td>S3- Public administration application</td>
<td>952 in Java and XML</td>
<td>1684</td>
<td>10</td>
</tr>
<tr>
<td>S4- a combined web system and task management system</td>
<td>Not available (but large), in Java</td>
<td>379</td>
<td>3</td>
</tr>
<tr>
<td>S5- Large telecom system</td>
<td>480 in the latest studied release in Erlang, C and Java</td>
<td>2555</td>
<td>2</td>
</tr>
<tr>
<td>S6- a reusable framework for developing software systems for oil and gas sector</td>
<td>16 in Java</td>
<td>223</td>
<td>3</td>
</tr>
</tbody>
</table>

4.1 Research Questions and Metrics

The most common purpose of a problem reporting system is to record problems and follow their status (maps to Q1, Q4 and Q5). However, as discussed in Section 3, they may be used for other views as well if proper data is collected. Sometimes quality views and measurement goals are defined top-down when initiating a measurement program (e.g., by using the Goal-Question-Metric paradigm [1]), while in most cases the top-down approach is followed by a bottom-up approach such as data-mining or Attribute Focusing (AF) to identify useful metrics when some data is available; e.g., [17, 19, 22]. We do not intend to focus on the goals more than what is already discussed in Section 3 and refer to literature on that. But we have encountered the same problem in several industrial cases which is the difficulty of collecting data across several tools to answer a single question. Our experience suggests that questions that need measures from different tools are
difficult to answer unless effort is spent to integrate the tools or data. Examples are:

- In S6, problems for systems not based on the reusable framework were not recorded in the same way as those based on it. Therefore it was not possible to evaluate whether defect density is improved or not by introducing a reusable framework [12].

- In S5, correction effort was recorded in an effort reporting tool and modified modules could be identified by analyzing change logs in the configuration management tool, without much interoperability between these tools and the problem reporting tool. This is observed in several studies. Although problem reporting systems often included fields for reporting correction effort and modifications, these data were not reliable or consistent with other data. Thus evaluating correction effort or the number of modified modules per defect or type of defect was not possible.

Graves gives another example on the difficulty of integrating data [11]. The difference between two organizations’ problem reporting systems within the same company lead to a large discrepancy in the fault rates of modules developed by the two organizations because the international organization would report an average of four faults for a problem that would prompt one fault for the domestic organization.

To solve the problem, researchers often collect or mine industrial data, transform it and save it in a common database for further analysis. Examples are given in the next section.

### 4.2 Collecting and Analyzing Data

Four problems are discussed in this section:

1. Ambiguity in defining problem report fields even when the discussion on terminology is settled. A good example is the impact of a problem:

   - The impact of a problem on the reporter (user, customer, tester etc.) is called for Severity in [24], Criticality in [8] or even Product status in IEEE Std. 1044-1993. This field should be set when reporting a problem.

   - The urgency of correction from the maintenance engineer’s view is called Priority in [24], Urgency in [8] or Severity in IEEE Std. 1044-1993. It should be set during resolution.

Some problem reporting systems include the one or the other, or even do not distinguish between these. Thus, the severity field may be set by the reporter and later changed by the maintenance engineer. Here are some examples on how these fields are used:

   - For reports in S1 and S4 there was only one field (S1 used “Consequence”, while S4 used “Priority”), and we do not know if the value has been changed from the first report until the fault has been fixed.

   - S2 used the terms “Severity” and “Priority” in the reports.

   - S3 used two terms: “Importance” and “Importance Customer”, but these were mostly judged to be the same.

In [14], it is recommended to use four fields for reporter and maintenance criticality, and reporter and maintenance priority. We have not seen examples of such detailed classification. In addition to the problem of ambiguity in definitions of severity or priority, there are other concerns:

- Ostrand et al. reported that severity ratings were highly subjective and also sometimes inaccurate because of political considerations not related to the importance of the change to be made. It might be downplayed so that friends or colleagues in the development organization “looked better”, provided they agreed to fix it with the speed and effort normally reserved for highest severity faults [20].

- Severity of defects may be downplayed to allow launching a release.

- Probably, most defects are set to medium severity which reduces the value of such classification. E.g., 90% of problem reports in S1, 57% in S2, 72% in S3, 57% in S4, and 57% in release 2 of S5 (containing 1953 problem reports) were set to medium severity.

2. A second problem is related to release-based development. While most systems are developed incrementally or release-based, problem reporting systems and procedures may not be adapted to differ between releases of a product. As an example, in S6 problem reports did not include release number, only date of reporting. The study assumed that problems are related to the latest release. In S5, we experienced that the size of software components (used to measure defect density) was not collected systematically on the date of a release. Problem report fields had also changed between releases, making data inconsistent.

3. The third problem is related to the granularity of data. Location of a problem used to measure defect density or counting defects may be given for large components or subsystems (as in S6) or fine ones (software modules or functional modules as in S4) or both (as in S5). Too coarse data gives little information while collecting fine data needs more effort.

4. Finally, data is recorded in different formats and problem reporting tools. The commercial problem reporting tools used in industry in our case studies often did not help data collection and analysis. In S1, data were given to researchers as hardcopies of the problem reports, which were scanned and converted to digital form. In S2, the output of the problem reporting system was a HTML document. In S3 and S4, data were given to researchers in Microsoft Excel spreadsheet, which provides some facilities for analysis but not for advanced analysis. In S5, problem reports were stored in text files and were transferred to a SQL database by researchers. In S6, data were transferred to Microsoft Excel spreadsheets for further analysis. Thus, researchers had to transform data in most cases. In a large-scale empirical study to identify reuse success factors, data from 25 NASA software projects were inserted by researchers in a relational database for analysis [22]. One plausible conclusion is that
the collected data were rarely analyzed by organizations themselves, beyond collecting simple statistics.

The main purpose for industry should always be to collect business-specific data and avoid "information graveyards". Unused data are costly, lead to poor data quality (low internal validity) and even animosity among the developers. Improving tools and routines allows getting sense of collected data and giving feedback.

### 4.3 Validity Threats

We have identified the following main validity threats in the studies:

1. **Construct validity** is related to using counts of problems (or defects) or their density as quality indicators. For example, high defect density before operation may be an indicator of thorough testing or poor quality. Since this is discussed in the papers in the reference list, we refer to them and [21] on validating metrics.

2. **Internal validity**: Missing, inconsistent or wrong data is a threat to internal validity. Table 3 shows the percentages of missing data in some studies. In Table 3, “Location” gives the defect-prone module or component, while “Type” has different classifications in different studies.

#### Table 3. Percentages of missing data

<table>
<thead>
<tr>
<th>System Id</th>
<th>Severity</th>
<th>Location</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>4.4</td>
<td>25.1</td>
<td>2.5</td>
</tr>
<tr>
<td>S3</td>
<td>20.0</td>
<td>20.0</td>
<td>8.6*(4.3)</td>
</tr>
<tr>
<td>S4</td>
<td>0</td>
<td>0</td>
<td>9.0*(8.4)</td>
</tr>
<tr>
<td>S5</td>
<td>0**</td>
<td>22 for large subsystems, 46 for smaller blocks inside subsystems **</td>
<td>44 for 12 releases in the dataset</td>
</tr>
</tbody>
</table>

Notes:

*These are the sum of uncategorized data points (unknown, duplicate, not fault). In parentheses are “unknown” only.

** For release 2

The data in Table 3 shows large variation is different studies, but the problem is significant in some cases. Missing data is often related to the problem reporting procedure that allows reporting a problem or closing it without filling all the fields. We wonder whether problem reporting tools may be improved to force developers entering sufficient information. In the meantime, researchers have to discuss the introduced bias and how missing data is handled, for example by mean substitution or verifying random missing. One observation is that most cases discussed in this paper collected data at least on product, location of a fault or defect, severity (reporter or developer or mixed) and type of problem. These data may therefore base a minimum for comparing systems and release, but with sufficient care.

3. **Conclusion validity**: Most studies referred to in this paper have applied statistical tests such as t-test, Mann-Whitney test or ANOVA. In most cases, there is no experimental design and neither is random allocation of subjects to treatments. Often all available data is analyzed and not samples of it. Preconditions of tests such as the assumption of normality or equal variances should be discussed as well. Studies often chose a fixed significance level and did not discuss the effect size or power of the tests (See [6]). The conclusions should therefore be evaluated with care.

4. **External validity or generalization**: There are arguments for generalization on the background of cases, e.g., to products in the same company if the case is a probable one. But “formal” generalization even to future releases of the same system needs careful discussion [10]. Another type of generalization is to theories or models [16] which is seldom done. Results of a study may be considered as relevant, which is different from generalizable.

### 4.4 Publishing the Results

If a study manages to overcome the above barriers in metrics definition, data collection and analysis, there is still the barrier of publishing the results in major conferences or journals. We have faced the following:

1. The referees will justifiably ask for a discussion of the terminology and the relation between terms used in the study and standards or other studies. We believe that this is not an easy task to do, and hope that this paper can help clarifying the issue.

2. Collecting evidence in the field needs comparing the results across studies, domains and development technologies. We tried to collect such evidence for studies on software reuse and immediately faced the challenge of inconsistent terminology and ambiguous definitions. More effort should be put in meta-analysis or review type studies to collect evidence and integrate the results of different studies.

3. Companies may resist publishing results or making data available to other researches.

### 5. DISCUSSION AND CONCLUSION

We here described our experience with using problem reports for quality assessment in various industrial studies. While industrial case studies assure a higher degree of relevance, there is little control of collected data. In most cases, researchers have to mine industrial data, transform or recode it, and cope with missing or inconsistent data. Relevant experiments can give more rigor (such as in [2]), but the scale is small. We summarize the contributions of this paper in answering the following questions:

1. What is the meaning of a defect versus other terms such as error, fault or failure? We identified three questions to answer in Section 2: **what-** whether the term applies to manifestation of a problem or its cause, **where-** whether problems are related to software or the environment supporting it as well, and whether the problems are related to executable software or all types of artifacts, and **when-** whether the problem reporting system records problems detected in all or some life cycle phases. We gave examples on how standards and schemes use different terms and are intended for different quality views (Q1 to Q5).

2. How data from problem reports may be used to evaluate quality from different views? We used the model described
in [15] and extended in Section 3. Measures from problem or defect data is one the few measures used in all quality views.

3. How data from problem reports should be collected and analyzed? What is the validity concerns using such reports for evaluating quality? We discussed these questions with examples in Section 4. The examples show challenges that researchers face in different phases of research.

One possible remedy to ensure consistent and uniform problem reporting is to use a common tool for this - cf. the OSS tools Bugzilla or Trac (which stores data in SQL databases with search facilities). However, companies will need local relevance (tailoring) of the collected data and will require that such a tool can interplay with existing processes and tools, either for development or project management - i.e., interoperability. Another problem is related to stability and logistics. Products, processes and companies are volatile entities, so that longitudinal studies may be very difficult to perform. And given the popularity of sub-contacting/outsourcing, it is difficult to impose a standard measurement regime (or in general to reuse common artifacts) across subcontractors possibly in different countries. Nevertheless, we evaluate adapting an OSS tool and defining a common defect classification scheme for our research purposes and collecting the results of several studies.

6. REFERENCES
[23] The Trac project: http://projects.edgewall.com/trac/