Author(s): Kozlov, Denis; Koskinen, Jussi; Sakkinen, Markku

Title: Fault-proneness of open source software: Exploring its relations to internal software quality and maintenance process

Year: 2013

Version:

Please cite the original version:

Kozlov, D., Koskinen, J., & Sakkinen, M. (2013). Fault-proneness of open source software: Exploring its relations to internal software quality and maintenance process. The Open Software Engineering Journal, 7, 1-23. https://doi.org/10.2174/1874107X01307010001

Send Orders of Reprints at Reprints@benthamscience.net

The Open Software Engineering Journal, 2013, *7*, 1-23

Open Access

Fault-Proneness of Open Source Software: Exploring its Relations to Internal Software Quality and Maintenance Process

Denis Kozlov, Jussi Koskinen and Markku Sakkinen*

Department of Computer Science and Information Systems, University of Jyväskylä, P.O. Box 35, 40014 Jyväskylä, Finland

Abstract: The goal of this study is to explore how fault-proneness of open source software (OSS) could be explained in terms of internal quality attributes and maintenance process metrics. We reviewed earlier studies and performed a multiple case study of eight Java-based projects based on data available in the Source Forge repository. Overall, we studied 342 releases of those systems. As is usual, software quality was regarded as a set of internal and external quality attributes. A total of 76 internal quality attributes were measured from the source code of the selected systems via the tool SoftCalc. Two external quality attributes contributing to fault-proneness were in turn obtained from the Source Forge Issue Tracking System. The framework for assessing the maintenance process was adopted from our previous studies. Its distinguishing feature is that it takes into account the peculiarities of OSS development. We included 23 maintenance process metrics to this study. Relationships between the metrics under study were identified by means of correlation analysis, multiple regression analysis and factor analysis. The obtained results give an interesting insight into interpretation of the earlier results of other researchers, regarding especially their generalizability. The strengths of our study include the following: 1) we studied a greater number of metrics than most of the related studies, 2) we studied a greater number of OSS-systems than most of the studies, and 3) we focused on the fault-proneness of modern Java-based systems and investigated them as an aggregated sample.

Keywords: Software quality, software maintenance process, open source software, software metrics, fault-proneness.

1. INTRODUCTION

Software quality is becoming increasingly important for modern society; software faults and other quality problems can have severe consequences. The problems are caused by inadequate quality management, the human factor and other reasons in all stages of the software development process and in particular in the maintenance stage. Revealing the causes of quality problems is extremely important from the practical viewpoint, but there is still a gap in the research in this area. The majority of the existing studies focus either on software quality or on the software maintenance process alone. Most of them do not study the relationships between those two aspects.

The aim of this study is to explore how external quality attributes contributing to fault-proneness of open source software (OSS) can be explained in terms of internal quality attributes and maintenance process metrics (maintenance being a phase of the OSS life cycle).

We understand an *attribute* as a measurable physical or abstract property of an entity [1]. A *quality attribute* is a management-oriented attribute of software that contributes to its quality [1]. *Software quality* is a collection of quality attributes, such as portability, efficiency, reliability, functionality, usability, maintainability and testability [2, 3]. *Internal quality attributes* are those that can be measured purely in terms of the software product itself, i.e., by examining the software product on its own, separately from its behaviour [4]. *External quality attributes* are those that can be measured only with respect to how the software product relates to its environment. Here the behaviour of the software product is important rather than the product itself [4]. *Fault-proneness* is sometimes defined e.g. as the probability that an artifact contains a fault [5]. We will give a detailed definition of this attribute later in the paper.

Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes or to adapt the product to a modified environment [1]. There are four types of software maintenance, i.e. corrective, adaptive, perfective and preventive. The corrective maintenance includes diagnosis and correction of errors. The adaptive maintenance includes modifications of a software product to properly interface it with a changing environment. The perfective maintenance covers enhancements of a software product to add new capabilities or modify existing functions. Finally, the preventive maintenance consists of changes which are made to improve future maintainability of the software system. Source code or some of its parts may be reengineered to achieve this. Various reverse engineering techniques may be applied to improve the safety of making changes to source code related to all four types of maintenance. In our study we have focused on small and medium-sized OSS projects. The

*Address correspondence to this author at the Department of Computer Science and Information Systems, University of Jyväskylä, P.O. Box 35, 40014 Jyväskylä, Finland; Fax: +358142603011;
E-mail: markku.j.sakkinen@cs.jyu.fi

1874-107X/13

maintenance process activities performed in those projects are related mostly to corrective and perfective types of maintenance.

We chose the maintenance process as our target because of its importance. From the 1970s to the 1990s the costs of maintenance ranged from 49% [6] to 75% [7] of the total software costs. Nowadays the proportion of costs caused by system maintenance and evolution may in some cases be more than 90% [8], and about 75% of the maintenance costs are caused by enhancements, i.e., adaptive and perfective maintenance [9-11]. Although those numbers represent closed source software (CSS), there is no doubt that maintenance issues are very important for OSS as well.

There were three main reasons for choosing OSS as the target of our study: 1) *Importance of OSS* - Open source has gained a strong position, e.g., in some countries the Linux operating system is prescribed for governmental organizations. 2) *Lack of studying OSS*. There is relatively little previous research on OSS, as compared to conventional proprietary software. It is a particularly open question how OSS project activities influence OSS quality. 3) *Potential for studying OSS.* There exists a huge amount of source code and related documents that can be studied.

Today many OSS projects are becoming more organized efforts as companies initiate and lead them to gain business advantages. Full-time company employees often participate in managing and developing these projects [12]. For instance, the office suite OpenOffice [13] and the integrated Java development environment Netbeans [14] are successful OSS projects carried out by Sun Microsystems. OpenOffice is a case in which a parallel version of an originally proprietary software product (StarOffice, recently renamed to Oracle Open Office) has been released to the OSS domain; and there are many others. In such cases, the initial implementation and early evolution of the software has probably conformed to the normal process of the company. We have therefore chosen for our study only 'pure' OSSs, i.e. systems which have been developed by an open source community without significant participation of any company.

In this study we use an enhanced version of the research method used in our previous studies [15, 16]. Firstly, we raise a number of hypotheses based on a literature review. Those hypotheses concern specific metrics and quality attributes to be analyzed. Secondly, we test the relations between the metrics under study by means of statistical methods. We use a combination of correlation analysis and multiple regression analysis.

The rest of the paper is organized as follows. Firstly, we give an in-depth analysis of the research background in Section 2. The research objectives and methods used are presented in Section 3. Section 4 presents the results of the study. In Section 5 we discuss the results and the limitations of the study. Finally, Section 6 summarizes the paper and suggests some issues for future research. The acronyms that are used for several metrics in this paper are explained in Appendix **A**.

2. RELATED WORKS

This section reviews the studies that are the most relevant ones to this paper. Firstly, we summarize those studies that discuss the relationships between various internal quality attributes and fault–proneness, either in OSS or CSS. Secondly, we provide an overview of the major studies about the peculiarities of the OSS maintenance process, including papers relevant to fault-proneness. Thirdly, we summarize the main characteristics of all these studies in Table **1**.

2.1. Relationships between Internal Quality Attributes and Fault-Proneness

2.1.1. Open Source Software Systems

Briand *et al.* [17] carried out a case study on quality factors of object-oriented design based on an analysis of the open multi-agent system development environment LALO. They found out that a number of metrics from the Chidamber and Kemerer (CK) metrics suite [18] were statistically related to the fault-proneness of classes. Later Briand *et al.* performed a replication of this study [19]. The results differed in some aspects, including the relations between *DIT*, *NOC* and fault-proneness of classes. In both studies logistic regression analysis was used.

Ferenc *et al.* [20] used their framework called Columbus to calculate the object-oriented metrics that were identified earlier by Briand *et al.* [21] and Basili *et al.* [22] to be indicators of fault-proneness. Their case study was carried out on seven releases of Mozilla. In general, the results supported those of Briand *et al.* [21] and Basili *et al.* [22]. However, in contrast to those previous studies, here *NOC* did not change significantly over time and *LCOM* got worse (i.e., increased) over time.

Gyimóthy *et al.* [23] carried out an empirical validation of eight object-oriented metrics for fault prediction of the OSS project Mozilla. The authors used such methods as logistic and linear regression, decision trees and neural networks. The results indicated strong statistically significant correlations between most of the CK metrics and fault-proneness.

Li *et al.* [24] attempted to find out predictors (i.e. metrics available before release) for field defects (i.e. customer-reported software problems requiring developer intervention to resolve) of OSS by scrutinizing nine releases of OpenBSD. The collected predictors included 101 product metrics, 22 development metrics, 9 deployment and usage metrics and 7 software and hardware configuration metrics. Those metrics were collected from request tracking systems, concurrent versioning system (CVS) and mailing lists, as well as by measuring source code using a set of special tools such as Campwood SourceMonitor. They were evaluated by means of statistical methods. Many internal quality attributes were identified as good predictors of defects in terms of Spearman's and Kendall's rank coefficients, e.g. *Total Number of Methods*, *Number of Public Methods*, and *Number of Inline Comments*.

Zhou and Leung [25] analyzed the relationships between the CK metrics suite augmented with SLOC and fault-proneness, taking into account the severity of faults. Their study was conducted on a public domain data set from the NASA Metrics Data Program. That data was collected from a storage management system implemented in C++. Zhou and Leung classified the severity ratings for each defect. Three types of predicting models were used in the study: high-severity fault model, low-severity fault model and ungraded-severity fault model. As analysis methods the authors used univariate and multivariate logistic regression. The re-

Table 1. Summary of the Related Works

Study (in a Chronological Order)	Focus Areas	Analyzed Software Products	Metrics used
Khoshgoftaar and Munson [29], 1990	2b	3 data sets from the earlier studies of other researchers	20 complexity metrics
Basili *et al.* [22], 1996	2b	8 medium-sized systems	Chidamber and Kemerer (CK) metrics suite
Briand *et al.* [17], 1999	2a	An open multi agent system development environment	CK metrics suite
Fenton and Ohlsson [33], 2000	2b	A large legacy project	2 complexity metrics and LOC
Briand *et al.* [19], 2001	2a	An open multi agent system development environment	28 coupling metrics, 10 cohesion metrics and 11 inheritance metrics
El Emam *et al.* [34], 2001	2b	1 commercial Java application	2 metrics from CK metrics suite, several Briand et al's metrics
Ferenc *et al.* [20], 2004	2a	1 OSS, i.e. Mozilla	Several metrics proposed by Briand *et al.* and Basili *et al.*
Koru and Tian [12], 2004	1	75 OSSs	A number of quantitative attributes measuring the OSS maintenance process
Gyimóthy *et al.* [23], 2005	2a	1 OSS, i.e., Mozilla	8 metrics from CK metrics suite
Hassan and Holt [45], 2005	3	6 large OSSs	4 heuristics
Ostrand *et al.* [35], 2005	2b	2 large industrial software systems	6 metrics for fault-proneness
Koponen [39], 2006	1	5 medium-sized OSSs	12 metrics measuring the OSS maintenance process
Lintula *et al.* [40], 2006	1	4 medium-sized OSSs	No metrics
Zhou and Leung [25], 2006	2a	Public domain data set from the NASA metrics data program	CK metrics suite
Olague *et al.* [26], 2007	2a	1 OSS, i.e. Mozilla Rhino	CK, MOOD and QMOOD metrics suites
Koponen and Hotti [41], 2008	1	2 large OSSs	No metrics
Li *et al*. [24], 2008	2a, 3	1 OSS, i.e. OpenBSD	101 product metrics, 22 development metrics, 9 deployment and usage metrics, 7 software and hardware configuration metrics
This study	1, 2a	8 medium-sized OSSs	76 internal quality attributes, 2 external quality attributes accounting for fault-proneness, 23 maintenance process metrics

1: Software maintenance process for OSS
2a: Relationships between internal quality attributes and software faults - the case of OSS
2b: Relationships between internal quality attributes and software faults - the case of CSS
3: Relationships between software maintenance process and fault proneness

sults indicate a great number of strong correlations between the CK metrics and high, low and ungraded severity faults.

Olague *et al.* [26] empirically validated three object-oriented metrics suites for their ability to predict software fault-proneness: CK, Brito e Abreu's [27] MOOD metrics suite and Bansiya and Davis' [28] QMOOD metrics suite. The case study was carried out on six releases of the Mozilla Rhino OSS project implemented in Java. All the metrics were collected by means of a special tool – Software System Markup Language tool chain developed by the authors. The univariate binary logistic regression and collinearity analysis were used to determine the relations between the metrics under study. The results of that study indicated that some of the CK metrics and the QMOOD metrics are consistent predictors of class error-proneness.

2.1.2. Closed Source Software Systems

Khoshgoftaar and Munson [29] studied three available sets of data, i.e. 1) Lennselius' [30] data set, which is based on an analysis of a software project in the telecommunication domain, 2) Harrison and Cook's [31] data set, which is based on an analysis of a number of modules from a medium-sized C-based project, and 3) Akiyama's [32] data set. The authors concluded that complexity metrics such as *McCabe's Cyclomatic Complexity* correlated highly with software errors and fault-proneness. However, they warned that this high correlation by itself was an unreliable indicator of the predictive quality of models based on those complexity metrics, because the correlations indicated also multicollinearity between the metrics.

Basili *et al.* [22] analyzed eight medium-sized systems developed by students in C++. They examined the effect of the CK metrics suite on fault-proneness of classes by means of logistic regression analysis. Most of the CK metrics were identified as explaining factors of fault-proneness of classes.

Fenton and Ohlsson [33] analyzed two major releases of a large legacy project in switching telecommunication systems. They raised and tested a number of hypotheses. Six of them were related to the ability of size and complexity metrics to predict software faults. The complexity metrics were collected automatically from the design documents using a special tool ERIMET. For testing the hypotheses they used Alberg diagrams to evaluate the independent variables' ability to rank the dependent variable, and scatter plots. The

study found no strict evidence that size and complexity metrics are good predictors of fault-proneness of modules.

El Emam *et al.* [34] used object-oriented design metrics to construct prediction models for identification of faulty classes. The study used data collected from one version of a commercial Java application to construct a prediction model. The analyzed metrics included *DIT* and *NOC* from the CK suite and several coupling metrics proposed by Briand *et al.* distinguishing the types of relationships among classes, different types of interactions and the locus of impact of the interaction. The statistical modeling technique used was logistic regression analysis. The results indicated that an inheritance metric (in terms of *DIT* and *NOC*) and an export coupling metric were strongly associated with fault-proneness.

Ostrand *et al.* [35] developed a negative binomial regression model to predict the expected number of faults in each file of the next release of a system. The prediction is based on the code of the file in the current release, and the fault and modification history of the file from previous releases. The model was applied to two large industrial software systems with 17 and 9 releases respectively. The predictor variables of the model included e.g. *the logarithm of the number of lines of code* and *the file's change status*. The constructed model was capable of successfully identifying the most fault-prone files in multiple releases of the studied software systems.

2.2. OSS Maintenance Process

Lehman *et al.* [36] studied software evolution and maintenance on a long time perspective. Although their study was conducted on CSS, their results and insights are relevant to OSS researchers as a point of comparison.

Vixie [37] claimed that in the case of OSSs such activities as requirements definition, unit and system testing and support are not carried out in a manner similar to traditional software engineering. Vixie also stressed that quality assurance activities are unorganized, but extensive field testing helps to improve quality. This fact is well known as Raymond's principle: "given enough eyeballs, every bug is shallow" [38, 2].

The most comprehensive study of OSS maintenance was conducted by Koru and Tian [12], who studied the maintenance process of OSS from the viewpoint of defect handling. They did not analyze the software, but sent a questionnaire to persons working in OSS projects. They received answers from 119 individuals (largely either developers or testers) who contributed to 52 medium and large OSS projects. The researchers found out various aspects of defect handling in OSS development and maintenance, including reasons for defect reporting, subjects of defect reports, reporting of pre- and post-release defects, initial employment of defect database systems, consistency of defect recording, and completeness of defect reports.

Koponen [39] developed a framework for evaluation of OSS maintenance process based on several quantitative attributes: software type, intended audience, number of opened defects, number of source code changes, etc. The framework was validated by five case studies.

Lintula *et al.* [40] analyzed the maintenance processes of four medium-sized OSS systems from the viewpoint of defect reporting, user support and feature requesting. It was found out that in those projects discussion forums were very active. A reason for this can be that a large number of users can respond faster to user requests than a limited number of developers.

Koponen and Hotti [40] studied the maintenance processes of two large open source software projects, Apache HTTP server and Mozilla web browser. They came to the conclusion that most of those processes, e.g. problem and modification analysis, modification implementation and modification review and acceptance, were similar to the common vision of the maintenance process defined in the standards ISO/IEC 12207 [42] and ISO/IEC 14764 [43, 44]. However, the study also revealed a number of differences between the standards and the OSS maintenance process. For example, instead of Migration activities in OSS projects there is Release management, which consists of pre-release testing, packaging and release announcement tasks. The latter findings are in line with Vixie's [37] claims.

2.2.1. Relationships between Maintenance Process and Fault-Proneness

Hassan and Holt [45] proposed a framework to identify the ten most susceptible subsystems (i.e. directories) of the source code to have a fault. The study was based on an analysis of six large OSS projects. The results indicate that the following four heuristics should be applied by OSS maintainers to define their top-ten list of susceptible directories: 1) most frequently modified, 2) most recently modified, 3) most frequently fixed, and 4) most recently fixed subsystems.

The case study of Li *et al.* [24] (presented already in Section 2.1.1) provides a number of results also on the relations between fault-proneness and maintenance process metrics. Several metrics accounting for maintenance support (in terms of numbers of messages in various mailing lists) were identified as good predictors of defects by rank correlation analysis.

2.3. Summary

The studies discussed in the previous subsections are summarized in Table **1** in a chronological order. The current paper appears as the last one in the table. Information related to the following aspects is shown: 1) focus areas of the study, 2) analyzed software products, and 3) metrics used. As can be seen from the table, this study has the following advantages over others: 1) it covers two areas that have not much yet been discussed in conjunction with each other; 2) it is based on a greater (or equal) number of analyzed software products than the previous studies except that of Koru and Tian [12]; and 3) it is based on a substantially greater number of metrics than the other studies except that of Li *et al.* [24].

3. RESEARCH OBJECTIVES AND METHODS

3.1. Research Objectives

Our main research objective is to explore if and how selected external quality attributes representing fault-proneness

Table 2. Analyzed OSSs

Project	J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia
Number of releases	19	22	22	28	30	39	75	108
Max. *NSL*	127	103.3	87.9	133.3	25.8	120.9	34.5	104.2
Avg. *NSL*	120	77.2	47.6	44.7	14.5	91.2	27.5	50.5
Avg. *NM*	8.6	21.8	21.5	59.9	9.8	37.8	22.2	10
Development time (years)	2.0	6.5	4.5	4.0	4.0	4.5	1.0	2.5
Maturity of the latest release	Beta	Mature	Mature	Stable	Stable	Stable	Stable	Beta
Number of downloads per month	10 - 25	400 - 800	1500 - 3000	1000 - 8000	40 -110	180 -620	300 -850	250 - 400

can be explained quantitatively by maintenance process metrics and internal quality attributes for OSS. We studied these relations quantitatively based on a set of metrics.

We chose to measure fault-proneness by the two metrics *Rate of Bug Reports (RBR)* and *Average Bug Priority Level* (*ABPL*). The applied maintenance process metrics are adopted from our previous studies [15, 16] and are presented in Appendix **B**, Table **B-1**. The chosen palette of analyzed internal quality attributes is large; it includes, for example, size attributes like *Number of Source Lines in All* (*NSL*), *Number of Modules* (*NM*), *Number of Comment Lines* (*NCL*), and *Number of Object-Points* (*NOP*), as well as complexity attributes like *Data Complexity Chapin Metric (DCCM)* and *Interface Complexity Henry Metric (ICHM*). The complete list of these 76 attributes is available as Kozlov *et al.* [46].

3.2. Analyzed Software

In the choice of OSS systems for analysis, we applied six main criteria: 1) They should cover different kinds of applications, to eliminate commonalities that might be valid only for OSS of some specific kind. 2) They should have a common implementation language, so that differences between languages would not affect the results. For current OSS, Java was the first choice. 3) Each project should have a reasonable number of releases, and the numbers should preferably be nearly the same. 4) The projects should be relatively popular (in terms of number of downloads per month) and active during their lifetime so far (average project activity). 5) They should be of medium size, especially concerning source code. Very large OSS projects would be tackled in a possible later study. 6) For each project, at least the source code and relatively detailed information about bugs and feature requests should be available in a repository.

Based on those criteria, we selected the following eight Java-based projects and analyzed their all releases:

1) J – text editor [47];
2) Art of Illusion – full-featured 3D modeling, rendering, and animation studio [48];
3) jEdit – programmer's text editor [49];
4) TVBrowser – TV guide [50];
5) Jaxe – XML editor [51];
6) DrJava – lightweight programming environment for Java [52];
7) Buddi – simple budgeting program targeted for users with little or no financial background [53];
8) KoLmafia – cross-platform desktop tool which interfaces with the online adventure game Kingdom of Loathing [54].

Table **2** shows the main information about the selected projects. We were able to fulfil the criteria otherwise, but there is a rather large variation in the number of releases (criterion 3). The chosen projects cover various domains, including text and programming editors and multimedia and gaming applications. They all are available in the online Source Forge Issue Tracking System (SFITS) [55]. SFITS contains information on various kinds of bugs and feature requests related to specific releases of OSS products.

Software size is given in Table **2** in terms of maximum and average (over releases) *Number of Source Lines in All* (*NSL*, in thousands of lines) and average *Number of Modules* (*NM*). The development time is considered to be the time between the release dates of the first and last OSS versions available. The maturity of each project is estimated on the 6-point scale used on the Source Forge website (1 – Planning, 2 – Pre-Alpha, 3 – Alpha, 4 – Beta, 5 – Production/Stable, 6 – Mature). Of course, this scale is more or less subjective, since the OSS developers have to estimate the maturity level of their projects based on spent effort, added features, etc. However, no better metric is available. The number of downloads (average over the lifetime) is extracted from the corresponding project website in the Source Forge OSS repository.

3.3. Hypotheses

The approach of the paper is 1) to raise a number of hypotheses about the relations between the metrics under study, based on earlier research, 2) to test those hypotheses against the empirical results obtained from analysis of the selected OSS projects, and 3) to raise further hypotheses based on those results. The focus of this section is the first set of hypotheses.

We divided our hypotheses in two groups according to the objectives of our study, i.e. 1) hypotheses concerning the relations between internal quality attributes and fault–proneness (Subsection 2.1.1) and 2) hypotheses concerning the relations between maintenance process metrics and fault-proneness (Subsection 2.1.2). Note that when we talk about a positive correlation in a hypothesis, we implicitly assume that the correlation is statistically significant. Some metrics used in the papers reviewed in Section 2 differ from those

used here. Therefore we define our hypotheses in more general terms. The metrics used in the previous studies and in our study can be considered as operationalizations of the more abstract concepts used in the hypotheses.

It should be noted that some of the hypotheses are based on results from CSS studies. We used them when there were no OSS studies that treated the same issues.

3.3.1. Hypotheses about the Relations between the Internal Quality Attributes and Fault-Proneness

We raised the following hypotheses concerning the relations of the internal quality attributes and fault-proneness based on the literature review:

H1: *Software size correlates positively with fault-proneness*. Based on Briand *et al.* [21], Zhou and Leung [25], Khoshgoftaar and Munson [29] and Ostrand *et al.* [35] — but not supported by Fenton and Ohlsson [33].

H2: *The number of methods correlates positively with fault-proneness*. Based on Li *et al.* [24] and Olague *et al.* [26].

H3: *The number of decisions correlates positively with fault-proneness*. Based on Khoshgoftaar and Munson [29] but not supported by Fenton and Ohlsson [33].

H4: *Inheritance correlates positively with fault-proneness*. Based on Briand *et al.* [19], Ferenc *et al.* [20], Briand *et al.* [21], Basili *et al.* [22], Gyimóthy *et al.* [23] and El Emam *et al.* [34].

H5: *Lexical or textual complexity of software correlates positively with fault-proneness*. Based on Khoshgoftaar and Munson [29].

H6: *The amount of nesting correlates positively with fault-proneness*. Based on Khoshgoftaar and Munson [29].

3.3.2. Hypotheses about the Relations between the Maintenance Process Metrics and Fault-Proneness

We raised the following hypotheses concerning the relations of the maintenance process metrics and fault-proneness, based on the literature review:

H7: *The number of system modifications correlates positively with fault-proneness*. Based on Hassan and Holt [45].

H8: *The number of system fixes correlates positively with fault-proneness*. Based on Hassan and Holt [45].

H9: The number of messages in the mailing list correlates positively with fault-proneness. Based on Li *et al.* [24].

3.3.3. Testing the Hypotheses

We tested the hypotheses on the operationalization metrics used in this study by means of correlation analysis and multiple regression analysis. We considered a hypothesis to be supported if the correlation analysis showed a relatively strong and statistically significant correlation between the metrics in question.

3.4. Research Methods

In our study we used an approach adopted from the studies of Ostrand *et al.* [35] and Anderson and Felici [56]. According to it the relations between the metrics under study can be revealed from a substantial set of software releases available for measuring and analysis.

3.4.1. Approach to Measure the Internal Quality Attributes

We measured the internal software characteristics with the static analysis tool SoftCalc and its module JavaAnal dedicated to assessing Java source code [57, 58]. SoftCalc is listed among the efficient Computer-Assisted Software Measurement and Evaluation Tools (CAME tools) supporting different phases of the software development process [59] and has been used in our previous studies [15, 16]. In total we measured 76 internal quality attributes, which can be divided into the following categories: 1) basic internal quality attributes, e.g. *Number of Source Lines in All* (*NSL*), *Number of Modules* (*NM*) and *Number of Comment Lines* (*NCL*), and 2) complexity metrics, e.g. *Data Complexity Chapin Metric* (*DCCM*) and *Interface Complexity Henry Metric* (*ICHM*).

3.4.2. Approach to Measure the External Quality Attributes

We define fault-proneness informally as the rate of new faults (bugs) found as the software is being used and maintained, weighted by their severities. For this purpose we use two external quality attributes: *Rate of Bug Reports (RBR)* and *Average Bug Priority Level (ABPL)*. Our *single* metric for fault-proneness is simply the product of *RBR* and *ABPL*, but calculating them separately first yields more information.

RBR is the inverse of the more common metric *Mean Time to Bug Report (MTTBR)*, but more convenient because it is always finite. Note further that *MTTBR* is not the same as the metric *Mean Time To Failure* (*MTTF*), which is commonly used to measure *reliability*. The point is that *RBR* counts every different fault only once, when it is first reported, while *MTTF* counts all failures of the software caused by all faults.

We calculated *RBR* as the total number of bugs reported for a specific OSS release divided by the life time of the corresponding release. The total number of bugs for a specific release was calculated from the information available in SFITS by taking into account the dates of the releases. The accuracy of measurement was sufficient, since 1) manual review of SFITS revealed that most of the bugs reported after the release of a newer software version (i+1) were related to that release rather than to the previous one (i), and 2) the bugs posted after the release of a newer software version (i+1) and related to the previous version *(*i) are likely to be valid also for the newer version (i+1).

The metric *Average Bug Priority Level (ABPL)* is the arithmetic average of the priority levels of all bugs reported for a specific release. The priority of each bug is estimated from the subjective viewpoint of developers on a 10-point scale, where 1 is the lowest priority level and 10 the highest one. The information about bug severity levels was obtained from SFITS. We inspected bug reports, and the severity levels of the bugs seemed correct; thus we regard *ABPL* as a reliable metric. There were some releases with no bug reports, and we took 0 as their *ABPL*. This choice has no effect on the product $ABPL \times RBR$, because *RBR* is then 0.

The priority scale is only an ordinal scale, and the highest priority levels are usually regarded as significantly more important than the lower ones. ABPL is therefore not an optimal metric. Instead of it and RBR we could have computed the rate of bugs of each level separately, but that would have

made the results more difficult to process further and to assess intuitively.

3.4.3. Approach to Measure the Maintenance Process

In this study we used the approach presented in our previous work [15], as summarized in Appendix **B**, to measure the maintenance process. The conventional standards and evaluation models for maintenance process, such as ISO/IEC 12207 [42], ISO/IEC 14764 [43] and IEEE 1219 [60], have been defined with closed-source software in mind. Many metrics defined in these standards, e.g. metrics related to documentation and testing, are not often used by OSS developers. It is thus hardly possible to evaluate OSS based on the ISO/IEC metrics. In turn, as noted earlier, the main disadvantage of Koponen and Hotti [41] is that the authors do not propose any particular metrics for the maintenance process that would be suitable for many OSS developers.

Our approach is based both on the ISO/IEC [42] and IEEE [60] standards and on the study of Koponen and Hotti [41]. We defined a set of substitutes for ISO/IEC and IEEE metrics, as listed in Appendix **B**. The original metrics and their substitutes are mapped to each other by groups. For instance, *ROB*, *ROFR*, *ABPL* and *AFRPL* are substitutes to the corresponding ISO metrics related to the 'Implementation phase'. Some substitutes, e.g. *TNB*, are used in several groups. We regard the *complexity* metrics as internal quality attributes in this study, but some of them could also be treated as maintenance process metrics. The detailed information about all the substitutes and the grounds for choosing them can be found in paper [15].

A significant advantage of our approach is that it can be easily and fruitfully applied to many OSS projects, since it is based on "primitive" metrics measured by a majority (according to our observations) of OSS practitioners. In turn, a limitation of our approach is that we do not take into account the human factor of the maintenance process, i.e., metrics characterizing OSS developers and OSS communities.

3.4.4. Statistical Methods Used

We intended to use a combination of correlation analysis and multiple regression analysis to find out quantitative relations between the metrics under study. We treated the two external quality attributes as dependent variables and the others as explaining variables. This does not necessarily imply *causal* relationships, but it seems safe to assume that the external quality attributes are affected by the other metrics more than vice versa.

The analyses were first carried out on the releases of one project at a time, as we had done in our previous studies. After considering the results (Subsection 4.1.1) we decided to analyze the "*grand sample*" consisting of all releases from all the studied OSS projects in order to get more generalizable results. All analyses were performed using the statistical software package Statistica 8.

The correlation analysis was used to identify significant relations between the studied metrics. Following a common recommendation [61], we decided to regard only those explaining variables as *interesting* that had statistically significant ($p <= 0.05$) and moderate or strong ($|r| >= 0.5$) correlations with the dependent variable. The multiple regression analysis would then have been performed, using for each dependent variable only those explaining variables that were interesting for it.

It turned out that all interesting explaining variables were very strongly collinear (see Subsection 4.1.2), and thus multiple regression analysis was not meaningful. Khoshgoftaar and Munson [29] had run to the same situation in their study, and performed factor analysis instead. We decided to follow their example.

4. RESULTS

We classify our results into two groups corresponding to the goals of our study. Firstly, we provide the results about relations between internal quality attributes and fault-proneness. Secondly, we present the results about relations between maintenance process metrics and fault-proneness. In the rest of the section we explain whether our results support or do not support the hypotheses raised earlier (Section 3.3). As explained earlier (Subsection 3.4.2) we model fault-proneness with the two metrics *ABPL* and *RBR*.

4.1. Relations between the Internal Quality Attributes and Fault-Proneness

4.1.1. Correlation Analysis

We calculated the correlations first for *each individual OSS project*. They turned out to be very different for different projects. Only 20 out of the 76 quality attributes (QAs) had non-conflicting statistically significant correlations with *ABPL* or *RBR* in more than one project. All correlations of those attributes are presented in Appendix **D** (Tables **D-1**, **D-2**, **D-3**, **D-4**). Even the most consistent correlation that between *ABPL* and *Number of Loop Statements* (*NLS*) is valid only for four of the eight projects If we consider only *interesting* (see Subsection 3.4.4) correlations, none of them is valid for more than two projects.

Regarding *ABPL,* it has interesting correlations with all except 5 QAs in jEdit. It has statistically significant correlations with almost all QAs also in Buddi and KoLmafia, but most of them are weak ($|r| < 0.5$). For two QAs (*NCI* and *NNML*) the correlations in jEdit and KoLmafia are in conflict. Art of Illusion and Jaxe each have one statistically significant but weak correlation, and in the remaining three projects there are no such correlations with *ABPL*.

Regarding *RBR,* it also has interesting correlations with all except the same 5 QAs as *ABPL* in jEdit. Almost all the same correlations are valid also in J. In Art of Illusion there are almost as many statistically significant correlations, but they are mostly weak. Interestingly, none of those correlations has the same sign as in jEdit and J. *RBR* has statistically significant correlations with only three QAs in KoLmafia and with two in Buddi; all these are weak. In the remaining three projects there are no such correlations.

From the analysis of *the grand sample* we obtained interesting correlations between the external quality attribute *ABPL* and the following internal quality attributes: *NSL*, *NCD*, *NDS*, *NDVI*, *NDP*, *NFR* and *NLiS* (Table **3**). The descriptive statistics for each of the above metrics by itself are provided in Appendix **C** (Table **C-1**, Figs. **C-1**, **C-2**). The single-correlation coefficients of the explaining variables with the dependent variable varied from 0.506 to 0.537 (Table **3**).

Table 3. Correlations between *Average Bug Priority Level* (*ABPL*) and Internal Quality Attributes

ABPL	r	p
NSL	0.522	0.000 (***)
NCD	0.516	0.000 (***)
NDS	0.537	0.000 (***)
NDVI	0.509	0.000 (***)
NDP	0.525	0.000 (***)
NFR	0.502	0.000 (***)
NLiS	0.506	0.000 (***)

*** $p < 0.001$

As can be seen from Table **D-2**, on the project level the correlations of the above QAs are statistically significant only in two or three of the studied OSS projects. For five of them, the correlations in jEdit are stronger than in the grand sample, and those in Buddi and KoLmafia mostly weaker (but statistically significant). However, for *NCD* and *NLiS* the correlations in jEdit are low and not significant, but those in Buddi and KoLmafia significant (although rather low).

In the grand sample there were no interesting correlations between *RBR* and any quality attribute; the strongest one was between *RBR* and *Data Flow Complexity Elshoff Metric* (*DFCEM*, $r = 0.197$) As noted in Subsection 3.4.2, the composite metric *ABPL×RBR* is a better indicator of fault-proneness than the two initial metrics. Therefore we computed also its correlations with the internal quality attributes, but those were very low. The strongest correlations were found with *DFCEM* ($r = 0.194$, $p = 0.000$), and *Data Complexity Chapin Metric* (*DCCM,* $r = 0.171$, $p = 0.002$). On the project level the correlations between *ABPL×RBR* and other internal quality attributes were slightly higher, but statistically significant only for some OSSs, e.g. jEdit (Appendix **D**, Tables **D-5**, **D-6**).

4.1.2. Regression Analysis

We proceeded in our analysis on the grand sample. Because *RBR* and *ABPL×RBR* had no significant correlations with any internal quality attributes, further analysis was meaningful only for *ABPL*.

We computed next the pairwise correlations between the identified seven attributes (Table **3**), and they were very high (minimum 0.772, maximum 0.992 and mean 0.901, Appendix **F**, Table **F1**). Thus, the regression coefficients produced by multiple regression analysis would not have been meaningful. We performed it nevertheless to get the multiple-correlation coefficient R, and it was 0.597. The coefficient of multiple determination, R^2 is thus 0.357; the *adjusted* R^2, which takes into account the number of degrees of freedom, is 0.343. This indicates that 34.3% of the variability of *ABPL* can be explained by the seven explaining variables The coefficient of single determination (r^2) of the best predictor *NDS* (*Number of Data Structures*) alone is 0.288, which means that the multiple-regression model explains only 5.5% of the variability of *ABPL* over single regression with *NDS.*Thus, multiple regression did not give as much additional information as we had hoped; the main reason for that is that the explaining variables are highly collinear.

4.1.3. Factor Analysis

It is obvious that the seven explaining variables in Table **3** either can be taken to represent the size of the software, or at least tend to grow with size. On the other hand, it is also obvious that many other things beside those measured in this study have significant effects on fault-proneness. Therefore, we could not expect any other *strong* factor than size to appear in factor analysis. To detect possible weaker factors we included also those quality attributes with $0.3 < |r| < 0.5$. This lead to 41 variables for the factor analysis (Appendix **G**).

As expected, there came out one very strong factor, on which all variables except one had loadings from −0.577 to −0.992, and which can be regarded as size (Appendix **G**, Tables **G1**, **G2**). The exceptional variable was *LCHM* (*Language Complexity Halstead Metric*), whose loading was 0.443. The second factor was already much weaker; its loadings were rather symmetrically distributed around 0, only 11 larger than 0.3 in absolute value, and the maximum absolute value was 0.706. The third factor was still weaker, and considering the large number of variables in proportion to the sample size (335) we decided to ignore it.

The results of the factor analysis indicate that there can be at least one other underlying, orthogonal factor behind the studied internal quality attributes, in addition to size. However, it is not easy to name this factor or have an intuitive understanding of it.

4.2. Relationships between the Maintenance Process Metrics and Fault-Proneness

4.2.1. Correlation Analysis

As in Subsection 4.1.1, we calculated the correlations first for each individual OSS project, and they were quite different for different projects. In all projects except J and Art of Illusion some correlations could not be computed because the variance of the explaining variable was 0.

ABPL (Appendix **E**, Table **E-1**) had statistically significant correlations only with *Number of Closed Bugs (NBC)* in jEdit, Jaxe, Buddi and KoLmafia, and with *Number of Closed Feature Requests (NCFR)* in J.

RBR (Appendix **E**, Table **E-2**) had statistically significant correlations with three of the four metrics in jEdit and Jaxe, with two metrics in J and Buddi, and none in the remaining four projects.

The composite metric *ABPL×RBR* had statistically significant medium-sized correlations with a number of internal quality attributes for the OSSs jEdit and Art of Illusion (Appendix **D**, Tables **D-5**, **D-6**). However, the signs of those correlations are not consistent, i.e. positive in the case of jEdit and negative in the case of Art of Illusion.

The correlation analysis of the grand sample did not yield interesting (strong or medium-strong and statistically significant) correlations between the maintenance process metrics and external quality attributes. The strongest obtained correlations were between *ABPL* and *Rate of Unassigned Bugs* (*RUB,* $r = 0.329$, $p = 0.000$(**)) and *Rate of Unassigned Feature Requests* (*RUFR*, $r = 0.306$, $p = 0.000$). With respect to the composite metric *ABPL×RBR* the correlations were even weaker (Appendix **E**, Table **E3**). The strongest correla-

tion was found between *ABPL×RBR* and *RUB* ($r = 0.197$, $p = 0.000$(***)).

4.2.2. Regression Analysis

Since most of the obtained correlations between the maintenance process metrics and fault-proneness were very low, it was meaningless to carry out regression analysis for those metrics.

4.2.3. Factor Analysis

Similarly, most of the obtained correlations between the maintenance process metrics and fault-proneness were very low. Thus it did not make sense to carry out factor analysis for those metrics.

4.3. Relation of the Results to the Hypotheses

In this section we analyze the results of our study with respect to the raised hypotheses (Section 3.3). The results are presented according to the initial groups of hypotheses. Like earlier in the paper, we consider those correlations to be *interesting* that are moderate or strong and also statistically significant.

4.3.1. Hypotheses about the Relations between the Internal Quality Attributes and Fault-Proneness

H1: Software size correlates positively with fault-proneness. We used several operationalization metrics related to software size, i.e. Number of Source Lines in All (NSL), Number of Genuine Code Lines (NGCL), Number of All Control Statements (NACS), Number of If Statements (NIS), Number of Switch Statements (NSS), Number of Case Statements (NCS) and Number of Loop Statements (NLS). None of these correlated significantly with ABPL×RBR or RBR alone The external quality attribute ABPL correlates strongly positively with Number of Source Lines in All (NSL) (Table **3**), but ABPL alone does not really represent fault-proneness. Thus the hypothesis is at most weakly supported.

H2: *Amount of methods correlates positively with fault-proneness*. We used two operationalization metrics related to methods, i.e. *Number of Methods Declared* (*NMD*) and *Number of Methods Inherited* (*NMI*). Neither one of those metrics had interesting correlations with fault-proneness. Thus the hypothesis is not supported.

H3: *Amount of decisions correlates positively with fault-proneness*. We used two operationalization metrics related to decisions, i.e. *Decisional Complexity McClure Metric* (*DCMM*) and *Control Flow Complexity McCabe Metric* (*CFCMM*). There were no interesting correlations between the metrics *DCMM* and *CFCMM* and fault-proneness. Thus the hypothesis is not supported.

H4: *Inheritance correlates positively with fault-proneness*. We used three operationalization metrics related to inheritance, i.e. *Number of Classes Inherited* (*NCI*), *Number of Methods Inherited* (*NMI*) and *Number of Data Variables Inherited* (*NDVI*). The external quality attribute *ABPL* correlated strongly positively with *NDVI* (r = 0.509, p = 0.000(***)). However, there is no significant correlation with *ABPL×RBR*. The hypothesis is thus at most weakly supported.

H5: *Lexical or textual complexity of software correlates positively with fault-proneness*. We used one operationalization metric related to lexical complexity, i.e. *Language Complexity Halstead Metric* (*LCHM*). There were no interesting correlations between *LCHM* and external quality attributes. Thus the hypothesis is not supported.

H6: *Amount of nesting correlates positively with fault-proneness*. We have used one operationalization metric related to nesting, i.e. *Number of Nesting Levels Maximum* (*NNLM*). There were no interesting correlations between the metric *NNLM* and external quality attributes. Thus the hypothesis is not supported.

As explained earlier (Subsection 3.4.4), we performed the statistical analyses also on each individual OSS separately. There are more statistically significant correlations on that level, but none of them is valid for more than four of the eight studied OSS projects (Appendix **D**).

4.3.2. Hypotheses about the Relations between the Maintenance Process Metrics and Fault-Proneness

H7: Number of system modifications correlates positively with fault-proneness. We used two operationalization metrics related to system modifications, i.e. Number of Closed Feature Requests (NCFR) and Number of Closed Bugs (NCB). We did not obtain any interesting correlations between those metrics and external quality attributes. Thus the hypothesis is not supported.

H8: Number of system fixes correlates positively with fault-proneness. We used one operationalization metric related to system fixes, i.e. Number of Closed Bugs (NCB). We did not obtain any interesting correlations between the metric NCB and external quality attributes. Thus the hypothesis is not supported.

H9: Number of messages in the mailing list correlates positively with fault-proneness. We used two operationalization metrics related to mailing lists, i.e. Number of Messages in the Open Discussion Forum (NMODF) and Number of Messages in the Help Forum (NMHF). We did not obtain any interesting correlations between those metrics and external quality attributes. Thus the hypothesis is not supported.

On the level of individual OSSs, there are more statistically significant correlations (Appendix **E**).

4.3.3. Results not directly related to the hypotheses

As Table **3** shows, we identified a number of relations between the internal quality attributes and fault-proneness that go beyond the initial set of hypotheses. In particular, we found medium-strong statistically significant correlations between each of the following internal quality attributes and *ABPL*: *Number of Classes Declared* (*NCD*), *Number of Data Structures* (*NDS*), *Number of Data Points* (*NDP*), *Number of Function References* (*NFR*) and *Number of Literals in Statements* (*NLiS*). However, none of them had significant correlations with *RBR* or *ABPL×RBR*.

5. DISCUSSION

5.1. Why were Most Previous Research Results not Confirmed?

The positive correlations between various internal quality attributes and metrics of fault-proneness that were found in

most previous studies are intuitively very plausible. Those QAs largely measure the size and complexity of software, which are generally believed to affect fault-proneness. Therefore, we took these correlations as our hypotheses H1 – H6.

There was much less previous research on the relationships between maintenance process metrics and fault-proneness. However, the positive correlations found in two studies seemed plausible, and so we took them as our hypotheses H7-H9.

We used two rather orthogonal metrics for fault-proneness: *Average Bug Priority Level* (*ABPL*) and *Rate of Bug Reports (RBR)*. We considered these to be the best metrics available for all the studied OSS systems, and their product the best single metric to represent fault-proneness. Previous studies had used somewhat different metrics for this. Likewise, different concrete metrics had been used for many internal quality attributes in the previous studies.

A striking observation from our results is that the correlations are highly different among the eight projects. Indeed, jEdit is the only one where both *ABPL* and *RBR* have consistently significant and at least medium-strong positive correlations with almost all QAs. In J this is the case only for *RBR*. In Buddi and KoLmafia *ABPL* has significant but weaker positive correlations with almost all QAs. In Art of Illusion the results for *RBR* are directly *opposite to* the hypotheses*:* almost all correlations are *negative*, statistically significant and medium-strong or nearly so. In the remaining three projects there is only one significant correlation altogether.

The picture about the correlations between fault-proneness and maintenance process metrics is similar. Unfortunately, all four of those metrics were available from only two projects, and only one of them from one project (Appendix **E**, Tables **E-1** and **E-2**). Observations like this were not even possible in the earlier metric-based studies, because each of them focused on some specific software products or on a set of releases of the same product. Our decision to choose a set of highly dissimilar software products proved to be fruitful: our study suggests that some of the results of the earlier studies may not be widely generalizable. However, with the one exception mentioned above, our results were not directly opposite to them.

Our results on the "grand sample", where all releases of all projects were treated as a single set, suggest the low generalizability of the results from the earlier studies. Seven of the nine hypotheses were not supported at all, and the two others (H1 and H4) at most marginally: there were statistically significant correlations only between *ABPL* and some QAs.

5.2. OSS Maintenance Process

The SFITS contains information about bugs which resembles the defect handling patterns revealed by Koru and Tian [12]. Therefore our approach to measure the OSS maintenance process based on the data extractable from the SFITS can be regarded as a further step towards a practical usage of those defect handling patterns.

In our study we followed Koponen and Hotti [41], who claimed that the approach for measuring the maintenance process presented in ISO/IEC 12207 [42] and ISO/IEC 14764 [43] is applicable also to OSS. A limitation of their study is that they did not analyze experimental data. We have not encountered any crucial obstacles in implementing the above standards for OSS. Our approach entails collecting those maintenance process metrics that are easily extractable from real OSS projects. Although our approach was derived from the standards and the approach of Koponen and Hotti, the metrics used in those standards have been just linked to the metrics used in our study. However, it would be wise to carry out a strict validation of our approach against the ISO standard. It could be achieved e.g. by gathering and analyzing software maintenance process metrics (Appendix **B**, Table **B-1**) from closed source software development projects and comparing those results with the results of this study.

Our framework to measure the OSS maintenance process can be considered complementary to the framework of Koponen [39]. Both frameworks use many similar metrics, e.g. those that are related to the numbers of bugs of different types. Some of the attributes considered by Koponen, e.g. type of defect management system and intended audience, have not been taken into consideration in our framework, since they were not available in SFITS.

By collecting the metrics related to the maintenance process of the studied OSSs we found out that in all projects the number of messages in the open discussion forums was high. This confirms the results of Lintula *et al.* [40].

A distinguishing feature of our approach to measure the OSS maintenance process is its simplicity and usage of the data that has been gathered and used in real OSS projects. Usually OSS developers have collected only quite primitive information, which is not used or is used restrictedly by the available advanced and sophisticated frameworks, such as IEEE 1219 [60] and Boehm [62].

5.3. Further Research Avenues

We have studied a rather diverse set of eight OSS projects. It would be highly interesting to continue on that track and study an even larger and more diverse set of projects. Probably there would appear some further patterns of the relationships between quality attributes and fault-proneness within a single project. We might find some common factors that would explain (partially) why a project follows a certain pattern. It is also possible that in such a very large sample there would be some significant correlations that did not appear in our "grand sample".

6. SUMMARY AND CONCLUSIONS

This has been an empirical multiple case study. We have explored to what extent and how fault-proneness could be explained by means of internal quality attributes and maintenance process metrics. We first conducted a literature survey as a basis for taking into account the main findings of other researchers. Next we analyzed eight OSS systems and their 342 releases.

Software quality was measured in terms of 76 internal quality attributes, using the static analysis tool SoftCalc. There were 23 maintenance process metrics obtained from Source Forge Issue Tracking System (SFITS). Fault-proneness was measured in terms of *Rate of Bug Reports* (*RBR*) and *Average Bug Priority Level* (*ABPL*). That data was also obtained from SFITS.

We investigated the relationships between the metrics under study by the combination of correlation analysis, regression analysis and factor analysis where appropriate. Nine hypotheses based on the results of the reviewed studies were raised.

Firstly, we applied a bottom-up approach; i.e. strong correlations between the metrics under study were identified at the level of individual OSS projects. Such correlations, supporting the hypotheses, emerged for only two or three of the eight projects. Most correlations were weak and statistically insignificant.

Secondly, we applied a top-down approach; i.e. strong correlations between the metrics under study were identified at the level of the *grand sample* consisting of all versions of all projects. At this level we obtained only medium-strong statistically significant correlations between *ABPL* and some of the internal quality attributes. These results supported two hypotheses at most weakly, and the other seven hypotheses were not supported at all. The results of correlation analysis revealed a strong multicollinearity between the metrics obtained from the analysis of the grand sample. Consequently it was not possible to apply multiple regression analysis to gain a fine-grained insight into the obtained correlations, as we had intended. However, the results of the conducted factor analysis revealed that the studied metrics can be interpreted in terms of two factors, one of which represents system size.

We noted additionally that the correlations valid for individual OSS projects became insignificant at the level of the grand sample. Conversely, the few correlations valid in the grand sample were insignificant at the level of the individual projects.

Most of the earlier studies in this area are based on only relatively small sets of OSS systems and releases despite the fact that OSS projects are very diverse and heterogeneous. The main conclusion to be drawn from our study is that the results of those earlier studies might not be well generalizable beyond their initial research settings due to their relatively limited nature.

APPENDIX A: ACRONYMS USED IN THE PAPER

The acronyms used in the main part of the paper are related to the primary focus of the paper, i.e. internal, external quality attributes and maintenance process metrics. The acronyms are provided in the alphabetical order. Also definitions are provided for those items, whose meaning is not necessarily intuitively clear.

ABPL - Average Bugs Priority Level (i.e. the severity and priority of a reported bug from the subjective viewpoint of developers)

AFRPL - Average Feature Request Priority Level (i.e. the priority of a reported feature request from the subjective viewpoint of developers)

BCSM - Branching Complexity Sneed Metric

CBO - Coupling between Object Classes

CFCMM - Control Flow Complexity McCabe Metric

CSS – Closed Source Software (i.e. computer software with restrictions on use or private modification, or with restrictions judged to be excessive on copying or publishing of modified or unmodified versions)

DACCM - Data Access Complexity Card Metric

DCCM - Data Complexity Chapin Metric

DCMM - Decisional Complexity McClure Metric

DFCEM - Data Flow Complexity Elshoff Metric

DIT – Depth of Inheritance Tree

ICHM - Interface Complexity Henry Metric

LCHM - Language Complexity Halstead Metric

LCOM – Lack of Cohesion of Methods

LOC – Number of Lines of Code

LOCD - LOC Difference between Two Adjoining Releases

LTR - Life Time of the Release

MTPV - Mean Time for Problem Validation (i.e. the time required by a developer to review a reported issue and mark it as a bug with specific priority, see *ABPL*)

MTTBR - Mean Time To Bug Report (i.e. the amount of bugs reported with respect to a specific release (*TNB*) divided by the life time of that release calculated in minutes (*LTR*))

MTTF – Mean Time To Failure

NACS - Number of All Control Statements

NAIV - Number of Arguments or Input Variables

NAM - Number of Assertions Made

NAV - Number of Arrays or Vectors

NCB - Number of Closed Bugs (i.e. number of those bugs, which were reviewed and fixed by the developers)

NCD - Number of Classes Declared

NCFB - Number of Control Flow Branches

NCFR - Number of Closed Feature Requests (i.e. number of those feature requests, which were reviewed and implemented by the developers)

NCI - Number of Classes Inherited

NCL - Number of Comment Lines

NCS - Number of Case Statements

NCUB - Number of Closed Unassigned Bugs (i.e. number of those bugs, which were reviewed and fixed by the developers without being assigned to any particular developer for fixing)

NCUFR - Number of Closed Unassigned Feature Requests (i.e. number of those feature requests, which were reviewed and implemented by the developers without being assigned to any particular developer for implementation)

NDB - Number of Deleted Bugs (i.e. those bugs that were deleted by a developer after problem validation; see *MTPV*)

NDBA - Number of Data Bases Accessed

NDCED - Number of Data Constants or Enums Declared

NDD - Number of Defined Definitions

NDFR - Number of Deleted Feature Requests

NDP - Number of Data-Points

NDR - Number of Data References

NDS - Number of Data Structures

NDST - Number of Different Statement Types

NDVD - Number of Data Variables Declared

NDVI - Number of Data Variables Inherited

NEC - Number of Exception Conditions

NFD - Number of Files Declared

NFDA - Number of File and Database Accesses

NFFR - Number of Foreign Functions Referenced (i.e. number of functions accessed from the source code and written in a programming language different to the main programming language used in the source code)

NFP - Number of Function-Points

NFR - Number of Function References

NGCL - Number of Genuine Code Lines (i.e. number of code lines without taking into account comments and blank lines)

NI - Number of Includes

NID - Number of Interfaces Declared

NII - Number of Interfaces Implemented

NIO - Number of Input Operations

NIS - Number of If Statements

NLiS - Number of Literals in Statements (i.e. the amount of characters in a specific statement)

NLS - Number of Loop Statements

NMCVS - Number of Messages in the CVS Archive (i.e. number of messages in the CVS of a specific OSS project exported and displayed in the SFITS)

NMD - Number of Methods Declared

NMDA - Number of Messages in the Developers' Archive (i.e. number of replies made by developers for an initial topic started by a developer facing a software development issue)

NMHF - Number of Messages in the Help Forum (i.e. number of replies made by developers for an initial topic started by a user asking for help or maintenance support)

NMI - Number of Methods Inherited

NMODF - Number of Messages in the Open Discussion Forum (i.e. number of replies made by developers or users for an initial topic started by a developer or a user discussing issues not directly related to development or maintenance)

NMR - Number of Macro References

NMRd - Number of Macros Referenced

NNLM - Number of Nesting Levels Maximum

NOC – Number of Children

NOP - Number of Object-Points

NOSR - Number of Open Support Requests (i.e. number of those support requests, which were not yet reviewed and fulfilled by the developers)

NPCD - Number of Predicates or Conditional Data

NPFA - Number of Parameters or Function Arguments

NROV - Number of Results or Output Variables

NRP - Number of Reports Produced

NRS - Number of Return Statements

NS - Number of Statements

NSL - Number of Source Lines in All (i.e. total number of source code lines of a specific OSS including blank lines and comments)

NSMA - Number of Source Members Analyzed (i.e. number of files comprising the source code of a specific OSS)

NSS - Number of Switch Statements

OSS – Open Source Software

OSSs – Open Source Software Products

RBR – Rate of Bug Reports, see *MTTBR*

RFC – Response for a Class

ROB - Rate of Open Bugs (i.e. percentage of those bugs, which were not yet reviewed and fixed by the developers)

ROFR - Rate of Open Feature Requests (i.e. percentage of those feature requests, which were not yet reviewed and implemented by the developers)

RUB - Rate of Unassigned Bugs (i.e. percentage of those bugs, which were not assigned to any particular developer for fixing; the total number of unassigned bugs includes both open and closed unassigned bugs)

RUFR - Rate of Unassigned Feature Requests (i.e. percentage of those feature requests, which were not assigned to any particular developer for implementing; the total number of unassigned feature request includes both open and closed feature requests)

SFITS – Source Forge Issue Tracking System

SLOC – Source Lines of Code

TNB - Total Number of Bugs (i.e. total amount of bugs reported with respect to a specific release)

TNFR - Total Number of Feature Requests

TNSR - Total Number of Support Requests

WMC – Weighted Methods per Class

APPENDIX B: THE APPROACH USED TO MEASURE THE OSS MAINTENANCE PROCESS

Table B-1. The Approach used to Measure the OSS Maintenance Process (Adopted from [15])

	Metrics Used in the IEEE Standard [60]	Metrics Used in this Study
Problem Identification	*Number of Omissions on Modification Requests* (*NOMR*)	*Number of Deleted Bugs* (*NDB*) *Number of Closed Unassigned Bugs* (*NCUB*) *Number of Deleted Feature Requests* (*NDFR*) *Number of Closed Unassigned Feature Requests* (*NCUFR*)
	Number of Modification Request Submittals (*NMRS*)	*Total Number of Bugs* (*TNB*) *Total Number of Feature Requests* (*TNFR*)
	Number of Duplicate Modification Requests (*NDMR*)	*Number of Deleted Bugs* (*NDB*) *Number of Deleted Feature Requests* (*NDFR*)
	Time Expended for Problem Validation (*TEPV*)	*Mean Time for Problem Validation* (*MTPV*)

Table B-1. contd….

	Metrics Used in the IEEE Standard [60]	Metrics Used in this Study
Analysis	Requirement Changes Documentation Error Rate Effort per Function Area Elapsed Time (schedule)	*Rate of Unassigned Bugs* (*RUB*) *Rate of Unassigned Feature Requests* (*RUFR*)
	Error Rates Generated by Priority and Type	*Average Bug Priority Level* (*ABPL*) *Average Feature Request Priority Level* (*AFRPL*)
Design	Software Complexity	*Data Complexity Chapin Metric* (*DCCM*) *Data Flow Complexity Elshoff Metric* (*DFCEM*) *Data Access Complexity Card Metric* (*DACCM*) *Interface Complexity Henry Metric* (*ICHM*) *Control Flow Complexity McCabe Metric* (*CFCMM*) *Decisional Complexity McClure Metric* (*DCMM*) *Branching Complexity Sneed Metric* (*BCSM*) *Language Complexity Halstead Metric* (*LCHM*)
	Design Changes Effort per Function Area	*Number of Messages in the CVS Archive* (*NMCVS*) *Number of Messages in the Developers' Archive* (*NMDA*)
	Elapsed Time	*Life Time of the Release* (*LTR*)
	Test Plans and Procedure Changes	-
	Error Rates Generated by Priority and Type	*Average Bug Priority Level* (*ABPL*) *Average Feature Request Priority Level* (*AFRPL*)
	Number of Lines of Code Added, Deleted, Modified, Tested	*LOC Difference Between Two Adjoining Releases* (*LOCD*)
	Number of Applications	-
Implementation	Volume or Functionality (function points or SLOC)	*Rate of Open Bugs* (*ROB*) *Rate of Open Feature Requests* (*ROFR*)
	Error Rates Generated by Priority and Type	*Average Bug Priority Level* (*ABPL*) *Average Feature Request Priority Level* (*AFRPL*)
Test	Error Rates by Priority and Type Generated Corrected	*Total Number of Bugs* (*TNB*) *Total Number of Feature Requests* (*TNFR*) *Number of Closed Bugs* (*NCB*) *Number of Closed Feature Requests* (*NCFR*)
Delivery	Documentation changes (i.e. version description documents, training manuals, operation guidelines)	*Number of Messages in the Help Forum* (*NMHF*) *Number of Messages in the Open Discussion Forum* (*NMOD*) *Total Number of Support Requests* (*TNSR*) *Number of Open Support Requests* (*NOSR*)

APPENDIX C. DESCRIPTIVE STATISTICS FOR THE STUDIED METRICS RELATED TO THE HYPOTHESES

Table C-1. Descriptive Statistics for the Studied Metrics at the Level of the Grand Sample

	Mean	Minimum	Maximum	Standard Deviation
ABPL	3.370	0.000	7.000	2.416
RBR	5382.136	0.000	341280.000	20495.502
NCD	286.551	30.000	707.000	182.378
NDP	7698.353	621.000	22754.000	5296.506
NDS	322.160	32.000	745.000	206.273
NDVI	191.227	15.000	464.000	134.239
NFR	13115.560	884.000	38892.000	9599.499
NLiS	7484.327	376.000	22152.000	6511.630
NSL	52557.988	4649.000	133337.000	35792.421

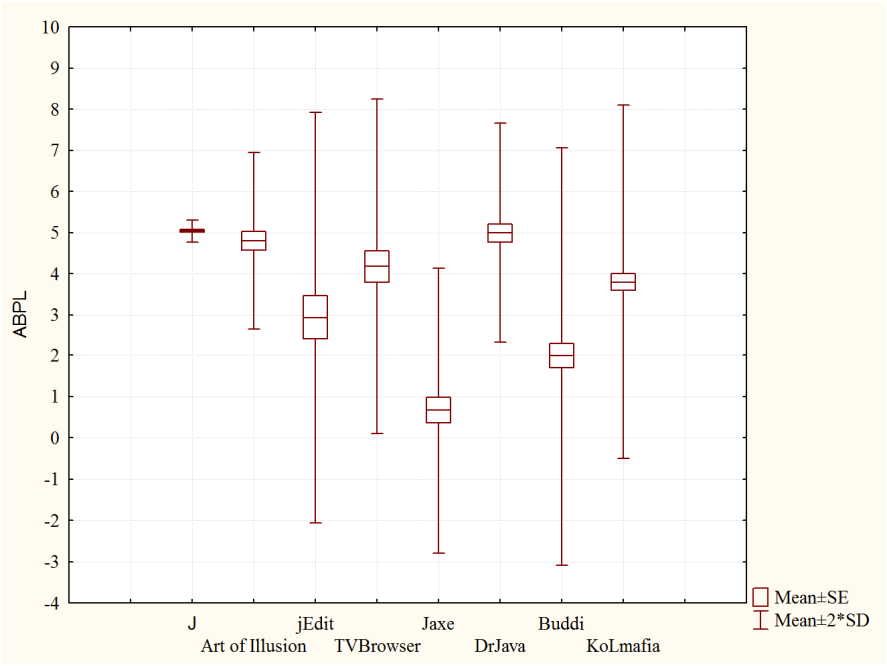

Fig. (C-1). Mean plot of *Average Bug Priority Level* (*ABPL*) values for the individual OSSs.

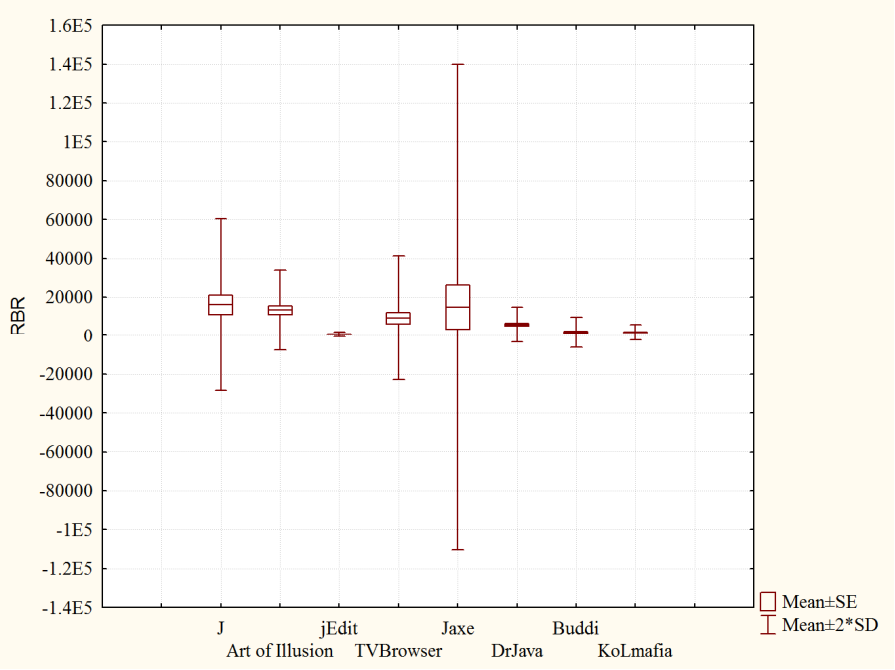

Fig. (C-2). Mean plot of *Rate of Bug Reports* (*RBR*) values for the individual OSSs.

APPENDIX D. MAIN CORRELATIONS BETWEEN FAULT-PRONENESS (ABPL, RBR) AND INTERNAL QUALITY ATTRIBUTES

Table D-1. Correlations between *Average Bug Priority Level* (*ABPL*) and Internal Quality Attributes (Part 1)

ABPL		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand Sample
NGCL	r	-0.151	0.389	0.665	0.199	-0.085	0.007	0.300	0.428	0.484
	p	0.537	0.074	0.001(**)	0.311	0.654	0.969	0.009(**)	0.000(**)	0.000(**)
NCI	r	-0.191	0.333	-0.442	0.194	-0.112	0.011	0.150	0.496	0.483
	p	0.434	0.130	0.040(*)	0.324	0.555	0.946	0.200	0.000(**)	0.000(**)

Table D-1. contd….

ABPL		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand Sample
NMD	r	-0.241	0.378	0.764	0.198	-0.096	0.056	0.264	0.420	0.462
	p	0.321	0.083	0.000(**)	0.313	0.613	0.735	0.022(*)	0.000(**)	0.000(**)
NMI	r	-0.238	0.326	0.741	0.193	-0.053	-0.171	-0.157	0.501	0.498
	p	0.326	0.139	0.000(**)	0.325	0.780	0.298	0.178	0.000(**)	0.000(**)
NIS	r	-0.220	0.417	0.724	0.196	-0.082	0.032	0.319	0.406	0.394
	p	0.367	0.054	0.000(**)	0.317	0.665	0.845	0.005(**)	0.000(**)	0.000(**)
NSS	r	-0.203	0.294	0.547	0.197	-0.473	0.014	0.251	0.387	0.064
	p	0.405	0.184	0.008(**)	0.315	0.008(**)	0.931	0.030(*)	0.000(**)	0.239
NCS	r	-0.220	0.325	-0.181	0.187	-0.082	0.007	0.281	0.359	0.162
	p	0.366	0.140	0.420	0.342	0.669	0.965	0.015(*)	0.000(**)	0.003(**)
NLS	r	-0.236	0.429	0.662	0.206	-0.063	0.054	0.288	0.419	0.411
	p	0.331	0.046(*)	0.001(**)	0.294	0.740	0.745	0.012(*)	0.000(**)	0.000(**)
NACS	r	-0.223	0.420	0.708	0.198	-0.081	0.037	0.318	0.403	0.407
	p	0.360	0.051	0.000(**)	0.313	0.672	0.823	0.005(**)	0.000(**)	0.000(**)
NNML	r	0.059	0.062	-0.766	0.166	0.009	0.009	-0.200	0.603	0.087
	p	0.809	0.785	0.000(**)	0.400	0.964	0.956	0.086	0.000(**)	0.109
CFCMM	r	-0.045	-0.364	-0.787	0.076	0.228	-0.017	0.174	0.011	0.267
	p	0.855	0.096	0.000(**)	0.702	0.225	0.917	0.136	0.910	0.000(**)
DCMM	r	-0.381	0.272	0.873	0.227	-0.082	0.032	0.282	0.452	0.062
	p	0.107	0.221	0.000(**)	0.245	0.666	0.845	0.014(*)	0.000(**)	0.249
LCHM	r	-0.139	0.157	-0.331	-0.167	0.038	-0.148	-0.287	-0.585	-0.429
	p	0.570	0.485	0.133	0.397	0.842	0.368	0.012(*)	0.000(**)	0.000(**)

*$0.01 \leq p \leq 0.05$; ** $0.001 \leq p \leq 0.01$; *** $p < 0.001$

Table D-2. Correlations between *Average Bug Priority Level* (*ABPL*) and Internal Quality Attributes (Part 2)

ABPL		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand sample
NSL	r	-0.069	0.384	0.615	0.198	-0.097	-0.005	0.299	0.444	0.522
	p	0.781	0.078	0.002(**)	0.313	0.611	0.976	0.009(**)	0.000(**)	0.000(**)
NCD	r	-0.271	0.335	0.223	0.194	-0.117	0.008	0.301	0.495	0.517
	p	0.262	0.127	0.318	0.322	0.539	0.963	0.009(**)	0.000(**)	0.000(**)
NDS	r	0.011	0.376	0.718	0.192	-0.118	0.011	0.257	0.505	0.538
	p	0.966	0.084	0.000(**)	0.328	0.536	0.945	0.026(*)	0.000(**)	0.000(**)
NDVI	r	-0.254	0.296	0.622	0.197	-0.022	-0.126	-0.226	0.534	0.510
	p	0.294	0.181	0.002(**)	0.316	0.910	0.447	0.052	0.000(**)	0.000(**)
NDP	r	-0.274	0.380	0.789	0.197	-0.103	0.033	0.315	0.471	0.525
	p	0.257	0.082	0.000(**)	0.316	0.589	0.841	0.006(**)	0.000(**)	0.000(**)
NFR	r	-0.215	0.417	0.623	0.196	-0.095	0.028	0.330	0.447	0.502
	p	0.377	0.054	0.002(**)	0.317	0.617	0.866	0.004(**)	0.000(**)	0.000(**)
NLiS	r	-0.102	0.398	0.316	0.201	-0.090	0.037	0.313	0.413	0.508
	p	0.679	0.066	0.152	0.305	0.635	0.825	0.006(**)	0.000(**)	0.000(**)

*$0.01 \leq p \leq 0.05$; ** $0.001 \leq p \leq 0.01$; *** $p < 0.001$.

Table D-3. Correlations between *Rate of Bug Reports* (*RBR*) and Internal Quality Attributes (Part 1)

RBR		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand sample
NGCL	r	0.586	-0.474	0.645	-0.151	0.245	0.150	0.209	0.128	0.098
	p	0.008(**)	0.026(*)	0.001(**)	0.442	0.193	0.364	0.073	0.188	0.069
NCI	r	0.691	-0.406	-0.038	-0.152	0.219	0.174	0.122	0.178	-0.012
	p	0.001(**)	0.061	0.865	0.441	0.245	0.289	0.299	0.065	0.830
NMD	r	0.586	-0.476	0.701	-0.148	0.230	0.157	0.204	0.130	0.094
	p	0.008(**)	0.025(*)	0.000(**)	0.451	0.221	0.340	0.079	0.182	0.084
NMI	r	0.566	-0.390	0.667	-0.150	0.256	-0.219	-0.073	0.185	0.004
	p	0.012(*)	0.073	0.001(**)	0.447	0.173	0.180	0.533	0.055	0.943
NIS	r	0.560	-0.505	0.679	-0.146	0.246	0.158	0.218	0.113	0.122
	p	0.013(*)	0.017(*)	0.001(**)	0.457	0.189	0.337	0.060	0.244	0.024(*)
NSS	r	-0.245	-0.469	0.639	-0.192	-0.047	-0.055	0.167	0.082	-0.061
	p	0.312	0.028(*)	0.001(**)	0.327	0.806	0.741	0.152	0.399	0.258
NCS	r	-0.223	-0.484	-0.073	-0.178	-0.109	-0.077	0.176	0.079	-0.060
	p	0.360	0.023(*)	0.746	0.364	0.567	0.644	0.131	0.418	0.267
NLS	r	0.530	-0.515	0.620	-0.160	0.267	0.182	0.197	0.120	0.117
	p	0.020(*)	0.014(*)	0.002(**)	0.418	0.153	0.267	0.090	0.217	0.031(*)
NACS	r	0.557	-0.507	0.666	-0.147	0.247	0.159	0.215	0.111	0.120
	p	0.013(*)	0.016(*)	0.001(**)	0.456	0.187	0.333	0.064	0.251	0.026
NNLM	r	0.498	-0.135	-0.744	-0.151	0.169	-0.056	0.059	0.348	0.024
	p	0.030(*)	0.549	0.000(**)	0.443	0.371	0.735	0.618	0.000(**)	0.656
CFCMM	r	-0.651	0.517	-0.632	-0.274	-0.057	0.114	0.089	0.122	0.089
	p	0.003(**)	0.014(*)	0.002(**)	0.159	0.764	0.489	0.446	0.209	0.102
DCMM	r	0.408	-0.243	0.693	-0.134	0.216	0.227	0.178	0.133	0.146
	p	0.083	0.277	0.000(**)	0.497	0.251	0.165	0.127	0.170	0.007(**)
LCHM	r	0.404	-0.219	0.166	0.263	-0.196	0.208	-0.199	-0.204	0.090
	p	0.086	0.328	0.461	0.177	0.300	0.204	0.087	0.035(*)	0.098

*$0.01 \leq p \leq 0.05$; ** $0.001 \leq p \leq 0.01$; *** $p < 0.001$.

Table D-4. Correlations between *Rate of Bug Reports* (*RBR*) and Internal Quality Attributes (Part 2)

RBR		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand Sample
NSL	r	0.513	-0.476	0.644	-0.152	0.231	0.193	0.205	0.141	0.058
	p	0.025(*)	0.025(*)	0.001(**)	0.440	0.220	0.240	0.078	0.145	0.286
NCD	r	0.563	-0.429	0.447	-0.149	0.206	0.179	0.148	0.170	0.004
	p	0.012(*)	0.046(*)	0.037(*)	0.450	0.274	0.277	0.207	0.079	0.945
NDS	r	-0.058	-0.455	0.531	-0.158	0.203	0.189	0.084	0.174	-0.005
	p	0.814	0.034(*)	0.011(*)	0.421	0.281	0.250	0.475	0.072	0.929
NDVI	r	0.564	-0.385	0.622	-0.152	0.270	-0.051	-0.153	0.247	0.014
	p	0.012(*)	0.077	0.002(**)	0.441	0.149	0.756	0.190	0.010(*)	0.800
NDP	r	0.505	-0.463	0.624	-0.153	0.223	0.181	0.208	0.157	0.078
	p	0.028(*)	0.030(*)	0.002(**)	0.437	0.237	0.271	0.073	0.104	0.147

Table D-4. contd….

RBR		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand Sample
NFR	r	0.599	-0.511	0.590	-0.145	0.233	0.154	0.231	0.142	0.089
	p	0.007(**)	0.015(*)	0.004(**)	0.462	0.216	0.350	0.046(*)	0.144	0.101
NLiS	r	0.659	-0.513	0.350	-0.151	0.240	0.112	0.241	0.116	0.054
	p	0.002(**)	0.015(*)	0.111	0.444	0.202	0.499	0.038(*)	0.234	0.319

*0.01 ≤ p ≤ 0.05; ** 0.001 ≤ p ≤ 0.01; *** p < 0.001.

Table D-5. Correlations between *ABPL*RBR* and Internal Quality Attributes (Part 1)

ABPL*RBR		J	ArtOfIllusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmfia	Grand Sample
NGCL	r	0.270	-0.473	0.643	-0.164	0.245	0.061	0.220	0.128	0.120
	p	0.263	0.026(*)	0.001(**)	0.404	0.193	0.712	0.058	0.187	0.056
NCI	r	0.362	-0.405	-0.032	-0.165	0.219	0.083	0.128	0.179	0.064
	p	0.128	0.062(*)	0.887	0.401	0.245	0.617	0.274	0.064	0.306
NMD	r	0.239	-0.475	0.698	-0.161	0.230	0.071	0.215	0.130	0.086
	p	0.324	0.026(*)	0.000(**)	0.414	0.221	0.669	0.065	0.180	0.169
NMI	r	0.262	-0.389	0.666	-0.162	0.256	-0.179	-0.076	0.186	0.095
	p	0.278	0.074	0.001(**)	0.411	0.173	0.275	0.519	0.055	0.132
NIS	r	0.254	-0.504	0.676	-0.159	0.246	0.067	0.230	0.114	0.146
	p	0.295	0.017(*)	0.001(**)	0.420	0.189	0.685	0.047(*)	0.242	0.020(*)
NSS	r	-0.346	-0.467	0.639	-0.207	-0.047	-0.062	0.176	0.082	-0.044
	p	0.147	0.028(*)	0.001(**)	0.291	0.806	0.706	0.131	0.397	0.483
NCS	r	-0.310	-0.482	-0.076	-0.193	-0.109	-0.075	0.185	0.079	-0.035
	p	0.196	0.023(*)	0.738	0.325	0.567	0.650	0.112	0.415	0.584
NLS	r	0.232	-0.513	0.618	-0.172	0.267	0.094	0.208	0.120	0.136
	p	0.339	0.015(*)	0.002(**)	0.381	0.153	0.567	0.074	0.216	0.030(*)
NACS	r	0.247	-0.505	0.664	-0.159	0.247	0.073	0.227	0.112	0.150
	p	0.309	0.016(*)	0.001(**)	0.420	0.187	0.660	0.050(*)	0.249	0.016(*)
NNLM	r	0.375	-0.136	-0.743	-0.163	0.169	-0.060	0.062	0.348	0.012
	p	0.114	0.545	0.000(**)	0.408	0.371	0.717	0.599	0.000(**)	0.852
CFCMM	r	-0.358	0.515	-0.632	-0.242	-0.057	0.036(*)	0.095	0.122	0.098
	p	0.132	0.014(*)	0.002(**)	0.215	0.764	0.828	0.420	0.210	0.119
DCMM	r	0.142	-0.243	0.691	-0.126	0.216	0.133	0.189	0.133	0.196
	p	0.563	0.277	0.000(**)	0.522	0.251	0.419	0.105	0.169	0.002(**)
LCHM	r	0.190	-0.217	0.171	0.275	-0.196	0.122	-0.205	-0.204	0.073
	p	0.437	0.332	0.447	0.156	0.300	0.458	0.078	0.034(*)	0.248

*0.01 ≤ p ≤ 0.05; ** 0.001 ≤ p ≤ 0.01; *** p < 0.001.

Table D-6. Correlations between *ABPL*RBR* and Internal Quality Attributes (Part 2)

ABPL*RBR		J	ArtOfIllusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmfia	Grand Sample
NSL	r	0.220	-0.474	0.642	-0.165	0.231	0.092	0.215	0.142	0.082
	p	0.366	0.026(*)	0.001(**)	0.402	0.220	0.577	0.064	0.144	0.195
NCD	r	0.234	-0.427	0.448	-0.162	0.206	0.086	0.156	0.170	0.059
	p	0.335	0.048(*)	0.036(*)	0.409	0.274	0.602	0.182	0.078	0.345

Table D-6. contd….

ABPL*RBR		J	ArtOfIllusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmfia	Grand Sample
NDS	r	-0.048	-0.453	0.528	-0.173	0.203	0.096	0.089	0.174	0.048
	p	0.847	0.034(*)	0.012(*)	0.378	0.281	0.561	0.450	0.071	0.444
NDVI	r	0.244	-0.384	0.621	-0.164	0.270	-0.095	-0.160	0.248	0.102
	p	0.315	0.078	0.002(**)	0.404	0.149	0.564	0.169	0.010(*)	0.106
NDP	r	0.177	-0.461	0.621	-0.167	0.223	0.089	0.218	0.158	0.093
	p	0.468	0.031(*)	0.002(**)	0.396	0.237	0.590	0.060	0.103	0.139
NFR	r	0.278	-0.510	0.588	-0.157	0.233	0.062	0.244	0.142	0.115
	p	0.250	0.015(*)	0.004(**)	0.425	0.216	0.707	0.035(*)	0.142	0.066
NLiS	r	0.272	-0.512	0.347	-0.163	0.240	0.033	0.253	0.116	0.111
	p	0.259	0.015(*)	0.114	0.408	0.202	0.842	0.028(*)	0.232	0.076

*0.01 ≤ p ≤ 0.05; ** 0.001 ≤ p ≤ 0.01; *** p < 0.001.

APPENDIX E. MAIN CORRELATIONS BETWEEN FAULT-PRONENESS (ABPL, RBR) AND MAINTENANCE PROCESS METRICS

Table E-1. Correlations between *Average Bug Priority Level* (*ABPL*) and Maintenance Process Metrics

ABPL		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand Sample
NCB	r	0.376	0.342	0.493	0.243	0.942	0.260	0.803	0.515	0.171
	p	0.113	0.119	0.020(*)	0.214	0.000(**)	0.110	0.000(**)	0.000(**)	0.006(**)
NCFR	r	0.864	0.213	0.192	---	---	0.151	0.194	0.327	0.234
	p	0.000(**)	0.342	0.392	---	---	0.358	0.095	0.001	0.000(**)
NMHF	r	0.044	0.263	0.313	---	0.116	---	---	---	0.105
	p	0.859	0.237	0.156	---	0.541	---	---	---	0.096
NMODF	r	0.019	0.308	---	---	0.195	0.135	0.054	---	0.193
	p	0.939	0.163	---	---	0.302	0.412	0.644	---	0.002(**)

*0.01 ≤ p ≤ 0.05; ** 0.001 ≤ p ≤ 0.01; *** p < 0.001.

Table E-2. Correlations between *Rate of Bug Reports* (*RBR*) and Maintenance Process Metrics

RBR		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand sample
NCB	r	0.106	-0.257	0.538	-0.128	0.407	-0.278	0.365	-0.028	-0.031
	p	0.667	0.249	0.010(*)	0.517	0.026(*)	0.087	0.001(**)	0.774	0.620
NCFR	r	-0.031	0.066	0.560	---	---	-0.239	0.464	0.065	0.009
	p	0.900	0.771	0.007(**)	---	---	0.143	0.000(**)	0.502	0.888
NMHF	r	0.710	-0.249	0.761	---	0.421	---	---	---	-0.010
	p	0.001(**)	0.264	0.000(**)	---	0.020(*)	---	---	---	0.871
NMODF	r	0.937	-0.341	---	---	0.599	0.108	0.173	---	0.051
	p	0.000(**)	0.120	---	---	0.000(**)	0.514	0.139	---	0.413

*0.01 ≤ p ≤ 0.05; ** 0.001 ≤ p ≤ 0.01; *** p < 0.001.

Table E-3. Correlations between *ABPL*RBR* and Maintenance Process Metrics

ABPL*RBR		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand Sample
NCB	r	0.785	-0.254	0.535	-0.136	0.407	-0.232	0.379	-0.028	0.002
	p	0.000(**)	0.254	0.010(*)	0.491	0.026(*)	0.155	0.001(**)	0.777	0.969

Table E-3. contd….

ABPL*RBR		J	Art of Illusion	jEdit	TVBrowser	Jaxe	DrJava	Buddi	KoLmafia	Grand Sample
NCFR	r	0.642	.0663	0.563	---	---	-0.213	0.489	0.066	0.055
	p	0.003(**)	0.770	0.006(**)	---	---	0.193	0.000(**)	0.497	0.382
NMHF	r	0.628	-0.248	0.764	---	0.421	---	---	---	0.002
	p	0.004(**)	0.266	0.000(**)	---	0.020(*)	---	---	---	0.972
NMODF	r	0.103	-0.338	---	---	0.599	0.089	0.184	---	-0.015
	p	0.675	0.124	---	---	0.000(**)	0.595	0.114	---	0.809

*0.01 ≤ p ≤ 0.05; ** 0.001 ≤ p ≤ 0.01; *** p < 0.001.

APPENDIX F. PAIRWISE CORRELATIONS BETWEEN THE MAIN METRICS UNDER STUDY

Table F-1. Pairwise Correlations between the Metrics Under Study at the Level of the Grand Sample

		ABPL	*RBR*	*NSL*	*NCD*	*NDS*	*NDVI*	*NDP*	*NFR*	*NLiS*
ABPL	r	1.000	0.187	0.522	0.517	0.538	0.510	0.525	0.502	0.508
	p	---	0.001(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)
RBR	r	0.187	1.000	0.058	0.004	-0.005	0.014	0.078	0.089	0.054
	p	0.001 (**)	---	0.286	0.945	0.929	0.800	0.147	0.101	0.319
NSL	r	0.522	0.058	1.000	0.914	0.902	0.810	0.977	0.981	0.937
	p	0.000(**)	0.286	---	0.000(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)
NCD	r	0.517	0.004	0.914	1.000	0.992	0.879	0.884	0.879	0.919
	p	0.000(**)	0.945	0.000(**)	---	0.000(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)
NDS	r	0.538	-0.005	0.902	0.992	1.000	0.870	0.882	0.861	0.914
	p	0.000(**)	0.929	0.000(**)	0.000(**)	---	0.000(**)	0.000(**)	0.000(**)	0.000(**)
NDVI	r	0.510	0.014(*)	0.810	0.879	0.870	1.000	0.772	0.823	0.906
	p	0.000(**)	0.800	0.000(**)	0.000(**)	0.000(**)	---	0.000(**)	0.000(**)	0.000(**)
NDP	r	0.525	0.078	0.977	0.884	0.882	0.772	1.000	0.975	0.909
	p	0.000(**)	0.147	0.000(**)	0.000(**)	0.000(**)	0.000(**)	---	0.000(**)	0.000(**)
NFR	r	0.502	0.089	0.981	0.879	0.861	0.823	0.975	1.000	0.944
	p	0.000(**)	0.101	0.000(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)	---	0.000(**)
NLiS	r	0.508	0.054	0.937	0.919	0.914	0.906	0.909	0.944	1.000
	p	0.000(**)	0.319	0.000(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)	0.000(**)	---

*0.01 ≤ p ≤ 0.05; ** 0.001 ≤ p ≤ 0.01; *** p < 0.001.

APPENDIX G. RESULTS OF THE FACTOR ANALYSIS AT THE LEVEL OF THE GRAND SAMPLE

Table G-1. Results of the Factor Analysis at the Level of the Grand Sample for the Selected Metrics

	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5
LCHM	0.423	0.105	0.142	0.549	0.053
NSMA	-0.907	0.195	0.271	-0.194	0.065
NSL	-0.981	0.152	-0.023	-0.002	-0.044
NGCL	-0.985	-0.112	-0.006	0.110	-0.035
NCL	-0.618	0.706	0.061	-0.123	-0.171
NI	-0.779	0.263	0.286	-0.179	0.385
NCD	-0.892	0.236	-0.330	-0.100	0.059
NCI	-0.811	0.238	-0.507	-0.047	-0.002

Table G-1. contd….

NMD	-0.915	0.257	0.276	0.044	-0.100
NMI	-0.826	-0.005	-0.517	0.009	0.065
NII	-0.933	0.205	-0.069	0.112	0.111
NOO	-0.948	0.143	0.248	-0.015	-0.111
NPP	-0.577	0.036	0.003	-0.519	0.583
NRP	-0.691	0.230	-0.466	0.404	0.120
NFD	-0.729	0.501	0.312	0.287	-0.004
NDS	-0.884	0.213	-0.338	-0.187	0.010
NDVD	-0.882	-0.365	0.070	-0.243	-0.131
NDVI	-0.822	-0.024	-0.495	-0.007	0.021
MDCED	-0.857	-0.280	0.134	-0.347	0.012
NEDE	-0.591	0.029	-0.186	-0.588	-0.322
NDDTU	-0.945	-0.231	0.130	-0.104	-0.002
NDR	-0.897	-0.418	0.053	0.072	-0.098
NAIV	-0.866	-0.467	0.069	0.026	-0.134
NROV	-0.870	-0.460	0.069	0.034	-0.129
NPCD	-0.898	-0.360	-0.020	0.235	0.005
NPFA	-0.960	0.129	0.151	-0.090	-0.061
NDP	-0.992	0.043	0.072	-0.070	-0.008
NS	-0.982	-0.146	0.004	0.102	-0.047
NIO	-0.831	-0.182	0.409	0.113	0.219
NOO	-0.885	-0.181	0.379	-0.060	-0.054
NFR	-0.992	0.008	-0.019	0.090	-0.005
NFFR	-0.991	0.016	0.065	0.108	0.015
NIS	-0.895	-0.240	-0.038	0.318	0.095
NLS	-0.847	-0.477	-0.015	-0.037	-0.141
NEC	-0.702	0.630	0.099	0.044	-0.288
NRS	-0.887	-0.157	-0.131	0.257	0.198
NCFB	-0.921	-0.230	-0.059	0.267	0.081
NLiS	-0.939	-0.020	-0.266	-0.022	-0.088
NDST	-0.983	-0.141	0.008	0.097	-0.045
NAM	-0.754	0.567	0.125	0.158	-0.214
NFP	-0.879	0.341	0.121	-0.017	0.279

Table G-2. General Results of the Factor Analysis Based on the Selected Metrics at the Level of the Grand Sample

Factors	Eigenvalue	% Total Variance	Cumulative Eigenvalue	Cumulative %
1	30.479	74.334	30.479	74.339
2	3.568	8.702	34.047	83.041
3	2.193	5.348	36.240	88.389
4	1.909	4.656	38.148	93.045
5	1.099	2.680	39.247	95.725

CONFLICT OF INTEREST

The authors confirm that this article content has no conflicts of interest.

ACKNOWLEDGEMENTS

We thank Harry M. Sneed for providing us with the Soft Calc tool, which was essential for this research.

REFERENCES

[1] IEEE, "IEEE Std. 1061-1998, Standard for a software quality metrics methodology, revision", Piscataway, N.J., *IEEE Standards Dept*., 1998.

[2] R. Glass, *"Facts and fallacies of software engineering*", Addison-Wesley: USA, 2003, pp. 174-177.

[3] ISO/IEC 9126-1, "Software Engineering. Product Quality. Part 1: Quality Model", *International Organization for Standardization (ISO),* Geneva, 2001.

[4] N.E. Fenton, and M. Neil,"A critique of software defect prediction models", *IEEE Trans. Softw. Eng.,* vol. 25, no. 5, pp. 675-689, 1999.

[5] G.J. Pai, and J.B. Dugan, "Empirical analysis of software fault content and fault-proneness using Bayesian methods", *IEEE Trans. Softw. Eng.*, vol. 33, no. 10, pp. 675-686, 2007.

[6] G. Alkhatib, "The maintenance problem of application software: an empirical analysis", *J. Softw. Maint. Evol. Res. Pract.*, vol. 4, no. 2, pp. 83-104, 1992.

[7] J.L. Elshoff, "An analysis of some commercial PL/I programs", *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 2, pp. 113-120, 1976.

[8] L. Erlikh, "Leveraging legacy system dollars for E-business" *(IEEE) IT Pro*, 2000, pp. 17-23, 2000.

[9] J. Martin, and C. McClure. "*Guidance on Software Maintenance*". *Software Maintenance: The problem and Its Solutions*. Prentice-Hall, Inc: Englewood Cliffs, New Jersey, p. 189, 1983.

[10] J. Nosek, and P. Palvia, "Software maintenance management: changes in the last decade", *J. Softw. Maint. Evol. Res. Pract.*, vol. 2, no. 3, pp. 157-174, 1990.

[11] H. van Vliet, "*Software Engin: Principles and Practice"*, Wiley: USA 2000.

[12] A.G. Koru, and J. Tian, "Defect handling in medium and large open source projects", *IEEE Softw.,* vol. 21, no. 4, pp. 54-61, 2004.

[13] *Open Office*. [Online]. Available: http://www.openoffice.org [Accessed Nov. 30, 2012].

[14] *Netbeans.* [Online] Available: http://www.netbeans.org [Accessed Nov. 30, 2012].

[15] D. Kozlov, J. Koskinen, J. Markkula, and M. Sakkinen, "Evaluating the impact of adaptive maintenance process on open source software quality", In *Proceedings of the 1st International Symposium on Empirical Software Engineering (ESEM)*, pp. 186-195*,* 2007.

[16] D. Kozlov, J. Koskinen, M. Sakkinen, and J. Markkula, "Assessing maintainability change over multiple software releases", *J. Softw Maint. Evol. Res. Pract.*, vol. 20, no. 1, pp. 31-58, 2008.

[17] L.C. Briand, J. Wust, S.V. Ilkonomovski, and H. Lounis, "Investigating quality factors in object-oriented design: an industrial case study", In *Proceedings of the 21 International Conference on Software Engineering (ICSE),* pp. 345-354, 1999.

[18] S.R. Chidamber, C.F. Kemerer, "A metrics suite for object-oriented design", *IEEE Trans. Softw. Eng.*, vol. SE-20, no. 6, pp. 476-493, 1994.

[19] L.C. Briand, J. Wust, and H. Lounis, "Replicated case study for investigating quality factors in object-oriented designs", *J. Empir. Softw. Eng.,* vol. 6, no. 1, pp. 11-58, 2001.

[20] R. Ferenc, I. Siket, and T. Gyimóthy, "Extracting facts from open source software", In *Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM),* 2004, pp. 60-69.

[21] L.C. Briand, J. Wust, J.W. Daly, and D.V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems", *J. Syst. Softw.*, vol. 51, no. 3, pp. 245-273, 2000.

[22] V.R. Basili, L.C. Briand, and W.L. Melo, "A validation of object-oriented design metrics as quality indicators", *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, pp. 751-761, 1996.

[23] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction", *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 897-910, 2005.

[24] P.L. Li, J. Herbsleb, and M. Shaw, "Finding predictors of field defects for open source software systems in commonly available data sources: a case study of Open BSD", In *Proceedings of the 11th IEEE International Symposium on Software Metrics*, pp 32-41*,* 2005.

[25] Y. Zhou, and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults", *IEEE Trans. Softw. Eng.*, vol. 32, no. 10, pp. 771-789, 2006.

[26] H.M. Olague, L.H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes", *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 402-419, 2007.

[27] F. Brito e Abreu, "Metrics for object-oriented environment", In *Proceedings of the 3rd International Conference on Software Quality (ICSQ),* pp. 67-75, 1993.

[28] J. Bansiya, and C.G. Davis, "A hierarchical model for object-oriented design quality assessment", *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4-17, 2002.

[29] T.M. Khoshgoftaar, and J.C. Munson, "Predicting software development errors using software complexity metrics", *IEEE J. Select Areas Commun*, vol. 8, no. 2, pp. 253-261, 1990.

[30] B. Lennselius, *"*Software complexity and its impact on software handling processes*",* In *Proceedings of the 6th International Conference on Software Engineering for Telecommunication Switching Systems (ICSETSS)*, pp. 148-153, 1986.

[31] W. Harrison, and C.R. Cook, "A micro/macro measure of software complexity", *J. Syst. Softw.*, vol. 7, no. 3, pp. 213-219, 1987.

[32] F. Akiyama, "An Example of Software System Debugging", In *Proceedings of the IFIP Congress,* pp. 353-359, 1972.

[33] N. Fenton, and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system", *IEEE Trans. Softw. Eng.*, vol. 26, no. 8, pp. 797-814, 2000.

[34] K. El Emam, W. Melo, and J. Machado, "The prediction of faulty classes using object-oriented design metrics", *J. Syst. Softw.*, vol. 56, no. 1, pp. 63-75, 2001.

[35] T.J. Ostrand, E. Weyuker, and R.M. Bell, "Predicting the location and number of faults in large software systems", *IEEE Trans. Softw. Eng.*, vol. 31, no. 4, pp. 340-355, 2005.

[36] M. Lehman, D. Perry, and J. Ramil, "Implications of evolution metrics on software maintenance", In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pp. 208-217*,* 1998.

[37] P. Vixie, "*Open Sources: Voices from the Open Source Revolution"*, O'Reilly & Associates: USA, pp. 91-100, 1999.

[38] E. Raymond, *"Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary"*, p. 30, O'Reilly: US, 1999.

[39] T. Koponen, "Evaluation framework for open source software maintenance", *In Proceedings of the International Conference on Software Engineering Advances (ICSEA),* paper 52, 2006.

[40] H. Lintula, T. Koponen, and V. Hotti, "Exploring the maintenance process through the defect management in the open source projects – four case studies", In *Proceedings of the International Conference on Software Engineering Advances (ICSEA)*, paper 53, 2006.

[41] T. Koponen, and V. Hotti, "Open source software maintenance process framework", *ACM SIGSOFT Softw Eng Notes*, 2005, vol. 30, no. 4, pp. 1-5, In *Proceedings of the Fifth Workshop on Open Source Software Engineering 5-WOSSE,* 2005.

[42] ISO/IEC 12207*:* 2008*,* "System and Software Engin — Software life cycle processes", *International Organization for Standardization,* Geneva (Switzerland), 2008.

[43] *ISO/IEC 14764*: 2006, "Software Engin — Software Life Cycle Processes", Maintenance, Geneva, *International Organization for Standardization*, 2006.

[44] IEEE, "Guide to the Software Engineering Body of Knowledge (SWEBOK)", *IEEE Comput Soc.,* 2001.

[45] A.E. Hassan, and R.C. Holt, "The top ten list: dynamic fault prediction", In *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM)*, pp. 263-272, 2005.

[46] D. Kozlov, *"Table of metrics measurable by SoftCalc"*. [online] Available*:* http://users.jyu.fi/~dekozlov/research/Table_of_metrics_Soft Calc.doc [Accessed Nov. 30, 2012].

[47] "*J text editor written in Java"*. [Online] Available: http://sourceforge.net/projects/armedbear-j/ [Accessed Nov. 30, 2012].

[48] *"Art of Illusion"*. [Online] Available: http://source-forge.net/projects/aoi/ [Accessed Nov. 30, 2012].

[49] "*J Edit programmer's text editor written in Java"*. [Online] Available: http://sourceforge.net/projects/jedit/ [Accessed Nov. 30, 2012].

[50] *"TV Browser Java-based TV guide".*[Online] Available: http://sourceforge.net/projects/tvbrowser/ [Accessed Nov. 30, 2012].

[51] *"Jaxe Java XML editor"*. [Online] Available: http://source-forge.net/projects/jaxe/ [Accessed Nov. 30, 2012].

[52] *"DrJava"*. [Online] Available: http://sourceforge.net/projects/-drjava/ [Accessed Nov. 30, 2012].

[53] *"Buddi"*. [Online] Available: http://sourceforge.net/projects/buddi/ [Accessed Nov. 30, 2012].

[54] *"KoLmafia"*. [Online] Available: http://sourceforge.net/projects/-kolmafia [Accessed Nov. 30, 2012].

[55] *"Source Forge"*. [Online] Available: http://sourceforge.net [Accessed Nov. 30, 2012].

[56] S. Anderson, and M. Felici, "Quantitative aspects of requirements evolution", In *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC)*, pp. 27-32, 2002.

[57] H.M. Sneed, "Analysis, measurement and evaluation of existing legacy systems", In *Tool Demonstrations of the 8th European Conference on Software Maintenance and Reengineering (CSMR)* Tampere, Finland, pp. 2-3, March 24-26, 2004.

[58] H.M. Sneed, "Estimating the costs of software maintenance tasks", In *Proceedings of the International Conference on Software Maintenance (ICSM),* pp 168-181, 1995.

[59] C. Ebert, and R. Dumke, *Software Measurement: Establish – Extract - Evaluate – Execute*, Springer: NY 2007.

[60] IEEE 1219. *"IEEE Standard for Software Maintenance",* IEEE Computer Society Press, 1998.

[61] D.S. Moore, and G.P. McCabe, *Introduction to the Practice of Statistics*, W.H. Freeman: USA, 2007.

[62] B.W. Boehm, *Software Engineering Economics*, Prentice-Hall: USA, 1981.

Received: August 29, 2012 Revised: September 25, 2012 Accepted: September 27, 2012