2012 19th Asia-Pacific Software Engineering Conference

An Overview of Software Defect Density: A Scoping Study

Syed Muhammad Ali Shah
Politecnico di Torino, Corso Duca degli Abruzzi, 24 10129 Torino, Italy
syed.shah@polito.it

Maurizio Morisio
Politecnico di Torino, Corso Duca degli Abruzzi, 24 10129 Torino, Italy,
maurizio.morisio@polito.it

Marco Torchiano
Politecnico di Torino, Corso Duca degli Abruzzi, 24 10129 Torino, Italy
marco.torchiano@polito.it

***Abstract -- Context*: Defects are an ineludible component of software, Defect Density (DD) - defined as the number of defects divided by size - is often used as a related measure of quality. Project managers and researchers alike would benefit a lot from overview DD figures from software projects, the former for decision making the latter for state-of-the-practice assessment.**

***Objective*: In this paper, we collect and aggregate DD figures published in literature, in addition we characterize DD as a function of different project factors in terms of central tendency and dispersion. The factors considered include development mode –open vs. closed source–, programming language, size, and age.**

***Results*: We were able to identify 19 papers reporting defect density figures concerning 109 software projects. The mean DD for the studied sample of projects is 7.47 post release defects per thousand lines of code (KLoC), the median is 4.3 with a standard deviation of 7.99. Development mode, is characterized by statistically meaningful different DD, the same for Java vs. C. Besides, in the studied sample large projects exhibited lower DD than medium and small projects.**

***Conclusion*: The study is a first step in collecting and analyzing DD figures for the purpose of characterizing one important aspect of software quality. These figures can be used both by researchers and project managers interested to evaluate their projects. Further work is needed to extend the data set and to identify predictors of defect density.**

Keywords: defects density, overview, scoping study.

1. INTRODUCTION

Quality of software projects is of concern to all stakeholders e.g. users, practitioners, researchers etc. The very nature of software as a continuously evolving entity makes it possible for several different factors –e.g. size, complexity, age, environment etc. – to influence either directly or indirectly the quality of the software. A common method to measure the quality of any piece of software is to reveal the presence of defects in it, and usually the metric used is Defect Density (DD). The DD is defined as the total number of defects divided by the size of the software [1].

In recent works many researchers characterize the DD of software modules based on different factors like size, complexity etc. [2][3]. While such contributions are very important to understand the internal quality behavior of software, we believe they only tell one part of the story: the perspective of the whole software product instead of individual modules.

To the best knowledge of the authors, there is no work in literature dedicated to aggregate and analyze DD figures for software projects. We took conducted our work using a “scoping study” method to first underpin the need of research and then find the main sources of evidence in literature to understand the objective of this study [4].

The objective of this study help us to characterize the software projects DD based on different factors. The study has a two fold outcome, first it aggregate and analyze DD figures of software projects to answer very simple question, both from researchers and practitioners point of view, such as ‘what is the typical defect density in a project’? Second it answers the question, ‘what are the factors to characterize the defect density in a project’?

The paper is organized as follows; Section 2 discusses the related work. Section 3 presents the research design of the study. Section 4 presents the results. Finally, results are discussed in Section 5 and the conclusions presented in Section 6.

2. RELATED WORK

As for the first question mentioned above, what is the typical defect density of a project, the earliest study conducted by Akiyama’s [5] reports that a 1 KLoC seems to have approximately 23 defects. McConnell [6] reports 1 to 25 defects per thousand lines. Chulani [7] reports it to be 12 defects per thousand lines.

As for the second question, what are the factors to characterize DD. Fenton and Ohlsson study shows that size is a good factor to characterize defects and DD at module level [3]. Many other studies reports size as a factor to characterize the defects and defect proneness at module level with different implications for open and close source softwares [2][8][9].

Raghunathan et al. compared the quality aspects of both open source and close source software’s and they found no difference of quality of open source and close source software’s [10]. Phipps found that a typical C++ programs had two to three times as many defects per line of code as a typical Java programs [11]. Graves et al. stated that we can characterize the faults based on number of modifications, the size of the modifications and on the age of a file [12]. Zvegintzov stated that the quality of software also increases with the age of software [13]. Cotroneo et al. highlighted the significant correlation of defects with the software aging [14].

In most of the related work, the software factors were used to characterize the defects or defect proneness without considering DD. If in some cases DD was used, it was used at module level. This makes serious concern for the need of such study which characterized the software projects based on DD.

3. RESEARCH DESIGN

We followed the framework of Arksey and O'Malley [4] for conducting our scoping study. There are five stages in the adopted framework. We present the first four stages of the study in the current section (in subsection 3.1, 3.2, 3.3, and 3.4 respectively) while the fifth stage, containing the results, is illustrated in section 4.

3.1 Stage 1: Research Questions Definition

The present paper aims at answering the impact of some important product factors concerning DD.

RQ1: What are the typical figures of DD in software projects?

Such figures provide quality managers and project managers benchmarks to define quality goals upfront, to evaluate the quality of a project during development, and to assess the quality of a project post mortem.

RQ2: Is there a difference in DD between open and closed source project?

Since often the context, motivation, and development process differ between open source and proprietary projects, as the anecdotal story goes one would expect a different quality of products. We aim at finding some evidence, at least in terms of DD.

RQ3: Is there a difference in DD among programming languages?

Different programming languages encompass e.g. varying styles, expressive power and abstraction level. Such differences are likely to influence the DD of projects.

In general (RQ2) & (RQ3) provide project managers evidence on the influence of programming language (RQ3) and reuse of OSS components (RQ2) on project quality.

RQ4: What is the relationship between DD and project size?

A lot of analysis has been conducted on the relationship between module size and DD, but we could not find any on project size and DD. RQ4 provide researchers the evidence of a relationship between project size and DD.

RQ5: What is the relationship of DD and project age?

A reasonable expectation is that the longer a project is released, more defects are found and therefore the higher the defect density. In addition it also provides researchers the evidence of evolution of reliability over time, not only on failure happening (traditional reliability definition) but also on DD.

3.2 Stage 2: Relevant Studies Identification

We identified the relevant studies by following a three phase search strategy.

Phase 1: The first phase is exploratory and considers papers published in top software engineering journals from 2000 to 2011. The search string is "Defect Density" OR "Fault Density" OR "Reliability". Fault density was used as a possible synonym of defect density, while reliability was used because related papers often present defect data. The search was applied through 'Science direct', 'Springer link', 'IEEE explorer' and 'ACM digital library'. The web site search function "search in all fields" including full text was used for every journal.

The journals considered are:

- System and Software
- IEEE Transaction on Software Engineering
- ACM Transaction on Software Engineering and Measurement
- IEEE Software
- Empirical Software Engineering
- Information and Software Technology

Inclusion and exclusion criteria are explained in 3.3.The results of Phase 1 are shown in Table 1.

Table 1. Distribution of papers in phase 1

Source	Scanned	Included
Systems and Software	654	1
IEEE Transaction on Software Engineering	83	0
IEEE Software	55	0
ACM Transaction on Software Engineering and Measurement	6	1
Empirical Software Engineering	173	4
Information and Software Technology	412	2

Phase 2: We search using two publication databases, IEEE explorer and ACM digital library. The search keywords are "Software Defect Density" and "Software Fault Density". We had to add "Software" to filter out excessive non relevant hits. The word 'Reliability' was dropped to narrow down the focus of search as it did not select any relevant paper in phase 1.

Same inclusion and exclusion criteria explained in 3.3 are used. The result of Phase 2 is shown in Table 2. The table does not consider papers already found in Phase 1.

Table 2. Distribution of papers in phase 2

Source	Scanned	Included

IEEE explorer	184	8
ACM digital library	2206	2

Phase 3: Phase 1 and phase 2 selected 18 papers. In Phase 3 we followed the references of selected 18 papers to identify other relevant studies. But we did not find any new relevant paper. In addition we particularly searched the Promise proceedings and found one study that indicates the availability of required data set at promise data repository.

This led to a selection of 19 papers in total, out of 3774: 8 from Phase 1 + 10 from Phase 2 + 1 from Phase 3. In total the 19 papers contain DD data about 110 projects. For space constraints the list of projects and their characteristics are not included in the paper but can be found as an electronic resource at: http://softeng.polito.it/syed/AppendixA.pdf. Table 3 reports the selected studies for the analysis.

Table 3. List of selected studies for DD

ID	Publication
S1	J.-H. Lo and C.-Y. Huang, "An integration of fault detection and correction processes in software reliability analysis," *Journal of Systems and Software*, vol. 79, no. 9, pp. 1312-1323, Sep. 2006.
S2	A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309-346, 2002
S3	E. Weyuker, T. Ostrand, and R. Bell, "Comparing the effectiveness of several modeling methods for fault prediction," *Empirical Software Engineering*, vol. 15, no. 3, pp. 277-295-295, Jun. 2010.
S4	S. Kpodjedo, F. Ricca, P. Galinier, Y.-G. Guéhéneuc, and G. Antoniol, "Design evolution metrics for defect prediction in object oriented systems," *Empirical Software Engineering*, vol. 16, no. 1, pp. 141-175-175, Feb. 2011.
S5	E. Weyuker, T. Ostrand, and R. Bell, "Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 539-559-559, Oct. 2008
S6	G. Koru, H. Liu, D. Zhang, and K. El Emam, "Testing the theory of relative defect proneness for closed-source software," *Empirical Software Engineering*, vol. 15, no. 6, pp. 577-598-598, Dec. 2010
S7	T. Illes-Seifert and B. Paech, "Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs," *Information and Software Technology*, vol. 52, no. 5, pp. 539-558, May 2010
S8	M. F. Ahmed and S. S. Gokhale, "Linux bugs: Life cycle, resolution and architectural analysis," *Information and Software Technology*, vol. 51, no. 11, pp. 1618-1627, Nov. 2009
S9	P. Abrahamsson and J. Koskela, "Extreme programming: a survey of empirical data from a controlled case study," in *Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on*, 2004, pp. 73-82
S10	P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz, "An empirical study of software reuse vs. defect-density and stability," in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, 2004, pp. 282-291.
S11	A. Mockus and D. Weiss, "Interval Quality: Relating Customer-Perceived Quality to Process Quality," in *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on*, 2008, pp. 723-732.
S12	A. Gupta, O. P. N. Slyngstad, R. Conradi, P. Mohagheghi, H. Ronneberg, and E. Landre, "A Case Study of Defect-Density and Change-Density and their Progress over Time," in *Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on*, 2007, pp. 7-16
S13	M. Cartwright and M. Shepperd, "An empirical investigation of an object-oriented software system," *Software Engineering, IEEE Transactions on*, vol. 26, no. 8, pp. 786-796, 2000
S14	Hongyu Zhang, "An investigation of the relationships between lines of code and defects," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, 2009, pp. 274-283.
S15	T. Dinh-Trong and J. M. Bieman, "Open source software development: a case study of FreeBSD," in *Software Metrics, 2004. Proceedings. 10th International Symposium on*, 2004, pp. 96-105
S16	N. Fenton, M. Neil, W. Marsh, P. Hearty, L. Radlinski, and P. Krause, "Project Data Incorporating Qualitative Facts for Improved Software Defect Prediction," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, 2007, p. 2.
S17	S. Wu, Q. Wang, and Y. Yang, "Quantitative analysis of faults and failures with multiple releases of softpm," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, Kaiserslautern, Germany, 2008, pp. 198-205.
S18	P. Mohagheghi, R. Conradi, and J. A. Borretzen, "Revisiting the problem of using problem reports for quality assessment," in *Proceedings of the 2006 international workshop on Software quality*, Shanghai, China, 2006, pp. 45-50.
S19	M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in Proceedings of the 6th International Conference on Predictive Models in Software Engineering, Timi\&scedil;oara, Romania, 2010, pp. 1–10.

3.3 *Stage 3: Study Selection*

The inclusion and exclusion criteria employed in stage 2 are defined below.

3.3.1 *Inclusion Criteria*

The inclusion criteria were applied at three subsequent levels. First we read the papers titles to select those relavant to our study. Then we read the abstracts of previously selected papers and kept the relevant papers only. As a third step we thoroughly read the papers and included only those studies which satisfied the folowing criteria:

- Are related to software engineering
- Are related to software projects
- Contain directly figures of DD, or contain data that allows to compute DD indirectly (such as number of post release defects and size in LOC)
- Mention that DD was computed after the particular release or the project (operation phase, post release phase, etc).

3.3.2 *Exclusion Criteria*

The studies that did not satisfy the inclusion criteria were excluded.

3.4 *Stage 4: Charting the Data*

DD is the key object of this study we only reported DD in LoCs on post release phase, but despite its apparent simplicity it can take slightly different forms.

Typically DD can be sampled at different times during the evolution of a project; in the meantime the code base may undergo complex transformations, e.g. code additions, changes, deletions. Therefore it is difficult to match a defect to corresponding code base. In this work we consider DD as a cumulative measure, i.e. we count defects since the first release; such a definition considers only post-release defects, therefore temporary problems happening before release should be filtered out. Moreover the data available in the articles is often not as accurate as desired. Such considerations led us to adopt as a reference the definition of defect density as a cumulative metric:

$$DD = \frac{CNDD}{Size}$$

Where CNDD is the cumulative number of post release defects in the observed period, and Size is measured at the end of the observed period in thousands of lines of code (KLoC). On the basis of the above referenced construct we performed a data extraction using either DD figures provided directly in the papers or values computed from the data available in the papers. In addition to the main dependent variable DD, we collected also some context variables:

- Type: whether the project was developed as open or closed source,
- Language: the main programming language used,
- Size: the size of the project in KLoC,
- Age: is the calendar time between first and last release on which DD is computed

3.4.1 *Data Analysis and Hypotheses*

To answer the research questions we report descriptive statistics, in particular distribution information, and when applicable we also statistically test some hypothesis. To represent the distribution of DD for the projects we use both cumulative distribution diagrams and box plots. The former report on the horizontal axis the DD values sorted in ascending orders and on the vertical axis the proportion of values not greater than the corresponding DD.

Research questions RQ2 and RQ3 lend themselves to be answered by means of hypothesis testing. The respective null and alternative hypotheses can be formulated as follows.

Concerning RQ2:

$H2_0$: There is no significant difference in term of DD between open source and close source projects.

$H2_a$: There is a significant difference in term of DD between open source and close source projects.

Concerning RQ3:

$H3.1_0$: There is no significant difference in terms of DD among projects adopting different languages.

$H3.1_a$: There is a significant difference in terms of DD among projects adopting different languages.

If the above null hypothesis can be rejected we can conduct a post-hoc investigation of the pair-wise differences; in this case the Bonferroni correction for multiple tests shall be applied. For any pair of languages L1 and L2 we can formulate the hypotheses as:

$H3.2_0$: Projects developed in L1 have not lower defect density as those developed in L2

$H3.2_a$: Projects developed in L1 have lower defect density as those developed in L2

From preliminary analysis we found that the data is not normally distributed, therefore we adopt non-parametric tests. According to the recommendations in [15] we use the Kruskal-Wallis test for differences between three or more groups and the Mann-Whitney test for pair-wise differences. When comparing different groups we will also evaluate the difference from a practical point of view. For this purpose we use the standardized effect size, measured as Cohen's d.

As far as RQ4 and RQ5 are concerned we will conduct a regression analysis. Such analysis helps to determine to which extent the dependent variable varies as a function of one or more independent variables. We will consider both the statistical significance of the model and the practical significance that is expressed by the R^2 statistic.

Since the size of a project may have a huge variability, focusing a pure linear correlation may yield no result. Therefore for RQ4, we perform an additional analysis focusing on size categories and their effect on DD. In order to avoid arbitrary thresholds we identify the classes by means of a clustering algorithm. In particular we use the K-means method to identify k=3 cluster corresponding to small, medium, and large projects.

Finally we analyze the correlation among the context variables. As far as project size is concerned we consider the size categories identified through clustering. In particular, since we deal with categorical or ordinal variables, we build the contingency tables and apply the χ^2 test to detect statistically significant correlations.

In the statistical testing, the significance level is checked by the given p-value. For rejecting or accepting the null hypothesis we used the significance value α=5%.

4. RESULTS (Stage 5)

In performing the data extraction on the paper, we were able to use directly provided figures for 46% of the projects, in another 45% of the cases we computed it starting from a number of defects and code size, and in the remaining 9% of cases we had to extrapolate the values from average values.

We carried on a preliminary analysis to identify possible outliers. As a consequence we discarded a project having DD 120 defects per KLoC.

Table 4. Descriptive statistics of DD of projects

Group		N	Mean	Median	Std Dev
All		109	7.47	4.3	7.99
Type	Closed source	77	8.6	5.4	8.49
	Open source	32	4.66	2.75	5.84
Language	C	43	10.0	7.9	7.98
	Java	45	5.9	3.5	6.22
	C++	11	8.73	1.3	13.17
	Other	10	1.89	1.1	2.83

RQ 1: What are the typical figures of DD in software projects?

Figure 1 presents the cumulative distribution diagram for the defect density of the surveyed projects. The DD is reported on a logarithmic scale, we observe that it spans nearly three orders of magnitudes, from 0.05 to close to 50.0. Table 4, in its first row, reports the central tendency of DD (mean 7.47 , median 4.3, standard deviation 7.99). While industry experience is about 1 to 25 errors per 1000 lines of code for delivered projects according to [6], in our data set 89 out of 109 projects are located in that range. The figure also reports, in gray line, the fitted normal distribution. Also based on the results of the Shapiro-Wilk test ($p<0.001$) we confirm that the DD data is not normally distributed.

RQ2: Is there a difference in DD between open and closed source project?

Our data set contains 77 closed source projects, and 32 open source projects. Figure 2 shows the cumulative distribution by type (i.e. open vs. closed source).

In addition Figure 3 contains the box plot. The summary descriptive statistics are reported in Table 4 divided by type. To answer RQ2 we test the hypothesis $H2_0$. The Mann-Whitney test reports a p-value = 0.004, which is below the α threshold, therefore we can reject the null hypothesis. Open source projects in our sample have a DD that is on average 4 Defects/KLoC smaller than closed source ones. In practical terms the difference can be considered of medium size (Cohen's d = 0.5).

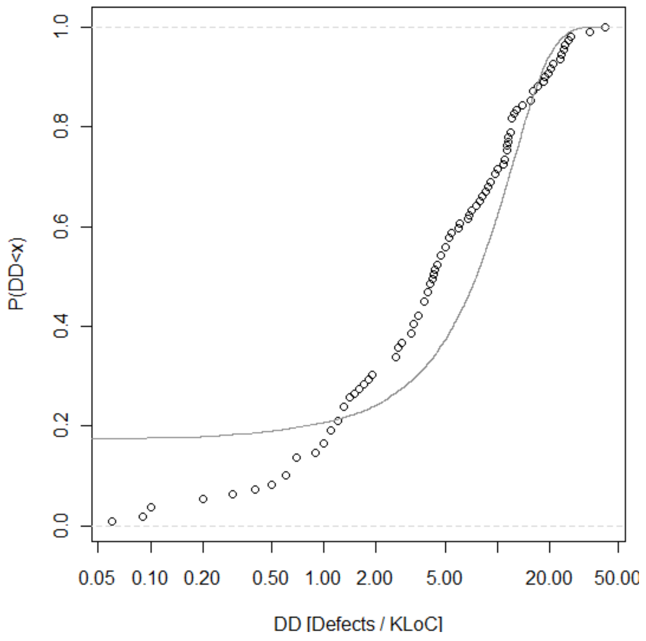

Figure 1. DD cumulative distribution

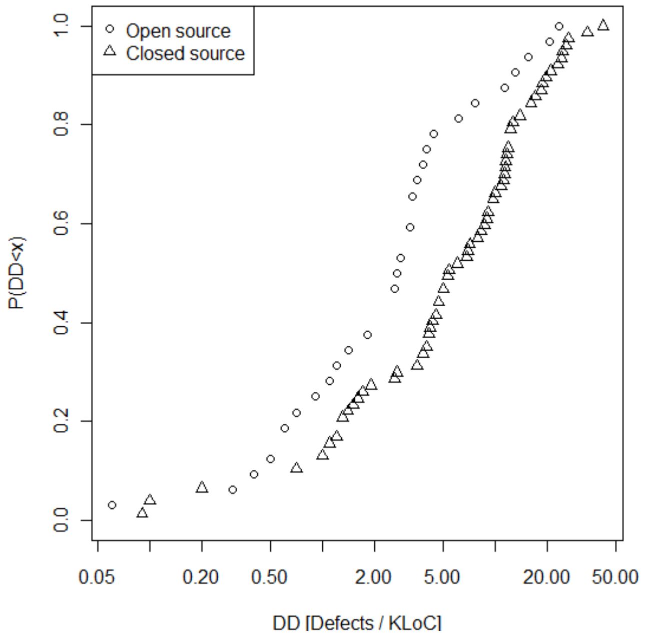

Figure 2. DD cumulative distribution for Close source and Open source projects

Such a result is confirmed by looking at the Figure 3 which shows close source having more DD and suggests a larger variation for closed source projects than open source ones.

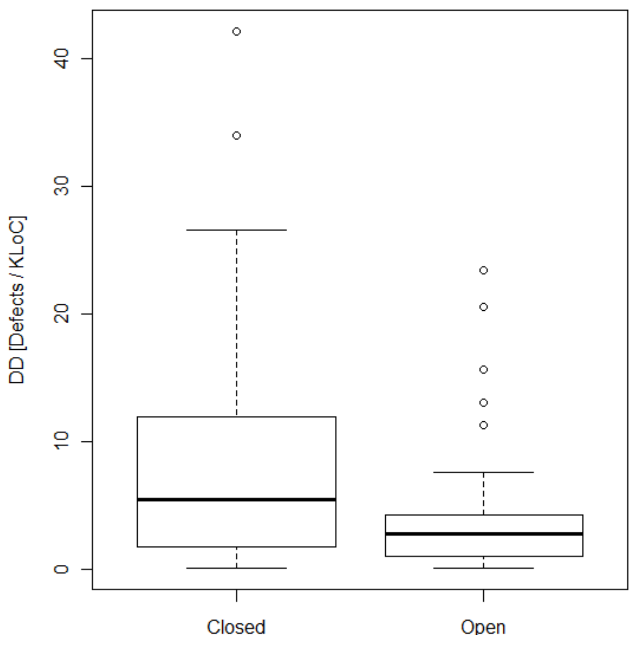

Figure 3. Box plot of Close Source vs. Open Source DD

RQ3: Is there a difference in DD among programming languages?

Our data set contains 43 C projects, 45 Java projects and 11 C++ projects. We excluded from the analysis 10 projects that were coded in other languages (i.e. Perl) or for which it was not possible to identify a clear major language. Figure 4 presents the cumulative distribution diagram for the three languages under study, Figure 5 shows the box plots. Descriptive statistics are in Table 4, rows 2 and 3. In this case visual analysis does not give clear suggestions.

The first hypothesis concerning RQ3, H3.1_0 can be tested using the Kruskal-Wallis test. The returned p-value is 0.009, therefore the null hypothesis can be rejected.

Given the above result we proceed with the pair-wise comparisons. In particular we test the H3.2_0 for the three possible pairs of languages, by means of the Mann-Whitney test. In assessing this test we adopt an α divided by 3 according to the Bonferroni rule.

For the pair (Java, C) we obtained a p-value of 0.003, therefore we can reject the null hypothesis. For the pairs (Java, C++) and (C++, C) we obtained the p-values 0.375 and 0.119, respectively, therefore we cannot reject the corresponding null hypotheses.

The significant difference can be considered of medium size (Cohen's d = 0.5), C projects have a DD that is an average 4.1 defect per KLoC higher than Java ones. In summary, as regards programming languages, there is evidence that the defect density in Java is lower than in C.

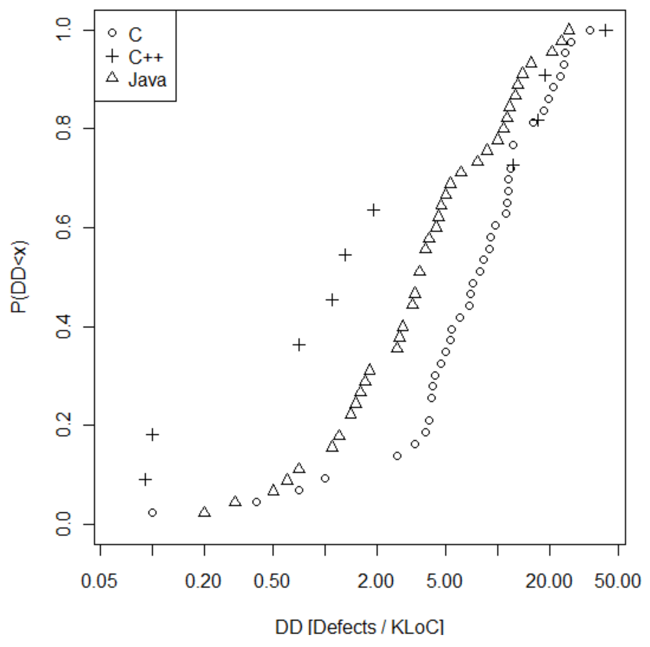

Figure 4. DD cumulative distribution by programming language

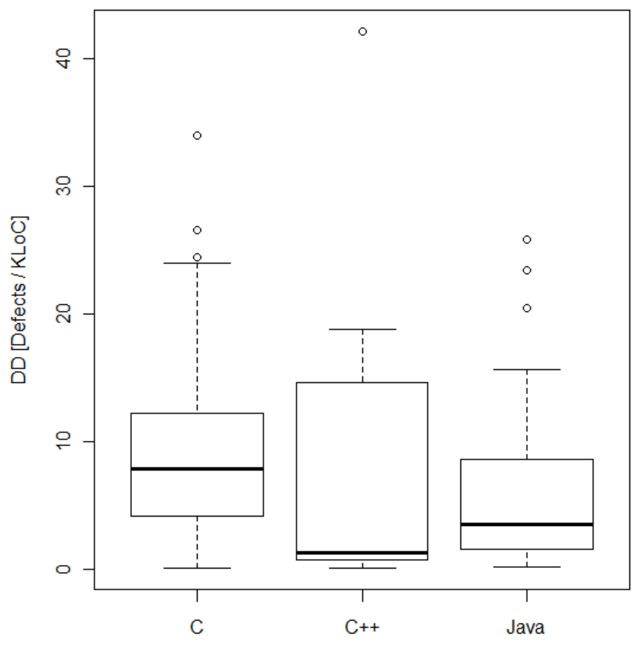

Figure 5. Box plot of DD per programming language

RQ4: What is the relationship between DD and project size?

In our data set, the size is reported for 108 projects. Figure 6 plots DD vs. size of projects in KLoCs. We used a logarithmic scale for both axes to be able to discern the individual points, which would appear flattened against the lower and left borders if a linear scale were used. We performed the regression analysis to find the relation between DD and size of project. The regression equation is:

$$DD = 8.03 - 0.000002\ Size$$

The p-value of the regression is 0.013 and the corresponding adjusted R^2 is 5.5%. There is a negative correlation but it has a limited practical impact.

Since both the DD and size distributions are extremely skewed – actually requiring a dual log scale to have a discernible representation – we also conducted a regression analysis on the log of DD and Size. In this case the regression equation is:

$$\log(DD) = 5.12 - 0.341 \log(Size)$$

The regression's p-value is smaller than 0.001 and the corresponding adjusted R^2 is 22%. The negative correlation, for the log-values has a higher, though still small practical relevance.

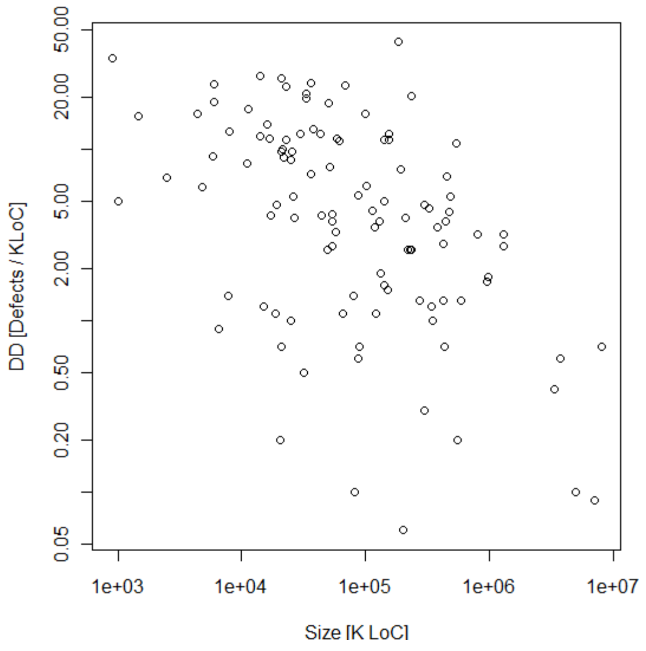

Figure 6. DD vs. size of project in logarithm scale

Finally we investigated a possible relationship between defect density and the category of projects (e.g. small, medium, large). To explore this possibility we identify project clusters by size using the K-means cluster analysis algorithm. The analysis identified 3 clusters that are described in Table 5. The ranges of DD for each Size category are plotted in Figure 7. Visual analysis shows a significant difference among the three groups. We conducted a Kruskal-Wallis test and we obtained a p-value of 0.0002, indicating a significant difference.

A pairwise comparison was then conducted, by means of Mann-Whitney tests and adopting a α divided by 3 according to the Bonferroni rule.

Table 5. Project clusters by size: descriptive statistics

		Defect Density		
Size range	**N**	**Mean**	**Median**	**Std Dev**
0 to 400K LoC	88	8.63	5.3	8.4
400K to 2M LoC	15	3.3	2.8	2.72
Above 2M LoC	5	0.38	0.40	0.28

For all pairs (Small, Medium), (Medium, Large) and (Small, Large) we obtained a p-value of 0.0146, 0.003 and 0.0006 respectively, which can be considered significant. The significant differences can be considered large, Cohen's d is being 0.83, -1.5 and -1.3 respectively.

In summary we cannot find a statistical meaningful direct relationship between DD and size. However by clustering projects we find evidence that the larger the projects, the lower the defect density. In particular we find statistically significant evidence that large projects have a lower defect density than medium and small projects.

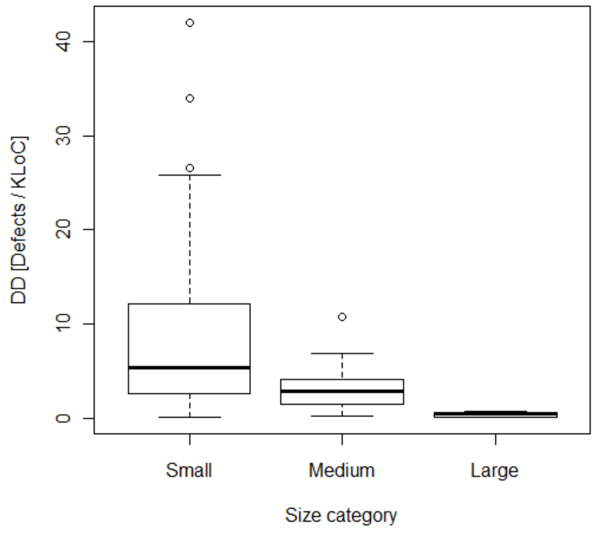

Figure 7. DD for different Size clusters

RQ5: What is the relationship between DD and Age?

In our data set the age (defined as number of years from the first release to the evaluation date mentioned in research studies or the number of years from the first release to the next release) is available for 47 projects. Figure 8 contains the plot of DD vs Age.

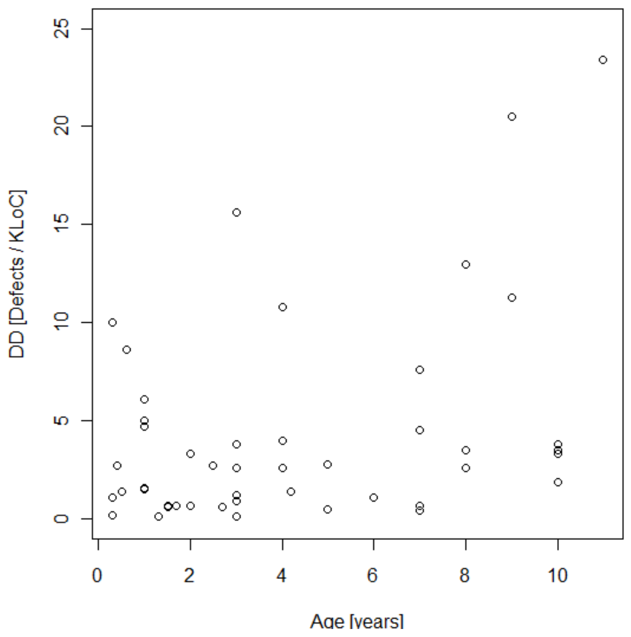

Figure 8. DD vs. age of projects

We performed the regression analysis to find the relation between DD and Age of project. The regression equation is:

$$DD = 1.86 + 0.59 \cdot Age$$

The p-value of the regression is 0.011 and the corresponding adjusted R^2 is 13.8%. There is a positive correlation but it has a limited practical impact.

Context variables correlation

We test the presence of correlations among context variable by means of pair-wise contingency tables between: Size category, Type, and Language. Table 6 reports the contingency table for Size category and Type. We can notice how closed source projects are significantly more skewed towards the small projects w.r.t. open source ones (χ^2 test p-value < 0.001).

Table 6. Contingency table for Size category and Type

Size \ Type	Closed		Open	
Small	63	82%	25	78%
Medium	12	15%	4	12%
Large	2	3%	3	10%
	77	100%	32	100%

As far as programming language and size are concerned we found no statistically significant correlation (χ^2 test p-value > 0.05). Finally, Table 7 reports the contingency table for Type and Language. We observe a significant difference, in our sample, between the languages used in open vs. closed source projects (χ^2 test p-value < 0.05).

Table 7 Contingency table for Type and Language

Lang Type\	C		C++		Java		Other	
Closed	39	91%	11	100%	20	44%	7	70%
Open	4	9%	0	0%	25	56%	3	30%
	43	100%	11	100%	45	100%	10	100%

5. DISCUSSION

5.1 Summary of Results

The extraction of DD data from systematically selected articles in the literature allowed us to publish a summary of DD data available. The essential descriptive statistics are presented in Table 4. This is a result per-se, which can be of use to both researchers and practitioners.

In addition we asked ourselves a few research questions whose answers can be summarized for our particular smaple of projects with the following pieces of evidence:

- There exists a statistically significant medium sized difference between open and closed source projects: the former have a DD that is 4 defects per KLoC lower than the latter.
- Java projects exhibit a significantly lower DD than C projects, 4.1 defects per KLoC on average
- In general the Size appears to be negatively correlated to DD: the larger the project the lower the DD. In particular, large projects are 10 times less defective than medium ones.

We can offer a few explanations for the above differences, although they are just speculative hypotheses that ought to be verified with further investigations.

The surprising, though not large, the difference between open and closed source projects can be explained by an heavy bias in our sample (see Table 6): the proportion of small projects studied in the literature is larger among closed source projects (82%) than for open source ones (78%).

As far as programming languages are concerned, the difference between C and Java could be explained by the different level of detail and expressive power between the two languages. Moreover the unbalanced use of languages between open and closed source projects (see Table 7) may have some influence.

Finally, the statement that "size is negatively correlated with defect density" is in (only) apparent contrast with several previous studies, e.g. [3] ignoring other additional factors. But we should keep in mind that most of previous studies referred to size and defect density of modules *within* a project and not of whole projects. Our result can be explained considering that large projects typically need to put a relevant effort on testing while small ones often neglect that phase, the result is a significantly lower post-release DD.

5.2 Threats to validity

We discuss in this section validity threat using the classification proposed by [15].

As for internal validity, the key issue is about the soundness in applying the scoping study approach. In this regard we have used known and reliable databases and journals, and repeatable search strings. We believe that inclusion and exclusion of papers in this case is easily repeatable since in the end it consists of checking that a defect density figure is available or not. This may give an impression that the reported data to calculate DD would be "survivorship bias". The selection of DD figures from important software engineering journals and electronic databases assure that nothing relevant has missed However we stress the main goal of our work is to provide an overview from data collected in real projects, as in any overview the composition is subject of debate [16], and actually we acknowledge that this is not the definitive: it is deemed to evolve indefinitely.

On the other hand we may have missed papers that report defect density data but calling it by another name. To increase reliability, the authors have cross checked all major steps in data collection and analysis.

As for construct validity, we have little concern about attributes programming language and development mode (open or closed source), that are hardly subject to ambiguities. On the contrary size, defect density and age are easily subject to ambiguities. It is well known that measuring size in lines of code is subject to variations due to programming language and modes of measuring (with or without comments, with or without blank lines, including libraries or not, etc). In nearly all selected papers the authors do not provide any information on how size was measured, so this poses a threat, of course on the values of size, and indirectly on the values of defect density. Defect density also depends on the measure of the number of defects. Here too there may be differences in the precision of measurement processes used, and on the definition of a defect used. This lack of precision of defect data may give estimates with larger error but following [16] we believe that this is better than having no data at all and relying on intuition. And again the authors of papers hardly describe the measurement process used. A procedural feature in measuring defects is the criterion according to which classify an issue as a defect. Unfortunately there is not a single technique adopted throughout the literature, when a procedure is described at all, therefore we have no way of estimating or balancing the issue, we can but accept it as an additional source of error. Another problem is when in the development process the defect number is computed. We have set as inclusion criterion papers that publish defect density in post release phase. Also at this regard we have no way of double checking whether the authors of papers all use the same meaning for it. The same applies to the attribute age.

As for external validity we underline two problems. On one hand published papers may be subject to publication bias, that probably skews data toward projects with lower defect density, reducing the representativeness of the sample. On the other hand the sample is clearly smaller. 109 projects is not a negligible number, but is a very limited percentage of the number of projects released overall.

6. CONCLUSIONS

This study has mined the literature for DD figures that had not been gathered and analyzed before. On 109 software projects the mean DD is 7.47 defects per KLOC with the dispersion (standard deviation) of 7.99. These values are useful for overview purposes.

Besides we have analyzed if size, age, programming language and development mode of project (close vs. open) could be factors for DD. We found that development mode is a factor (open source projects in our sample have a lower defect density), and programming language is sometimes a factor (Java projects have lower DD than C projects, but C++ and Java, C and C++ projects have a similar DD). In addition we found that projects size is relevant (large projects have lower DD figures), while Age is not a factor.

The main limitation of this study is the lack of process variables (testing effort and quality effort in general, code churn and code history, experience of the team and project manager) that are probably key factors for DD.

As a future work we plan to validate the speculative explanations we devised for the empirical results we found during this work.

7. REFERENCES

[1] Ronald B. Finkbine, Ph.D.. 1996. Metrics and Models in Software Quality Engineering. *SIGSOFT Softw. Eng. Notes* 21, 1 (January 1996), 89-.

[2] A. Gunes Koru, Dongsong Zhang, and Hongfang Liu. 2007. Modeling the Effect of Size on Defect Proneness for Open-Source Software. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering* (PROMISE '07). IEEE Computer Society, Washington, DC, USA, 10-.

[3] Norman E. Fenton and Niclas Ohlsson. 2000. Quantitative Analysis of Faults and Failures in a Complex Software System. *IEEE Trans. Softw. Eng.* 26, 8 (August 2000), 797-814.

[4] Hilary A. and Lisa O'Malley. 2005, Scoping studies: towards a methodological framework. *International Journal of Social Research Methodology.* vol. 8, no. 1. 19–32.

[5] Akiyama F. 1971. An Example of Software System Debugging. *Information Processing*, vol. 71, 353–379.

[6] Steve McConnell. 2004. *Code Complete, Second Edition*. Microsoft Press, Redmond, WA, USA.

[7] Sunita C. 1999. Constructive Quality Modeling for Defect Density Prediction: COQUALMO.*" IBM Research, Center for Software Engineering.*

[8] Gunes Koru, Hongfang Liu, Dongsong Zhang, and Khaled Emam. 2010. Testing the theory of relative defect proneness for closed-source software. *Empirical Softw. Engg.* 15, 6 (December 2010), 577-598.

[9] Gunes Koru, Khaled El Emam, Dongsong Zhang, Hongfang Liu, and Divya Mathew. 2008. Theory of relative defect proneness. *Empirical Softw. Engg.* 13, 5 (October 2008), 473-498.

[10] Raghunathan S, Prasad A, Mishra B. K, Hsihui C. 2005. Open source versus closed source: software quality in monopoly and competitive markets. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol.35, no.6. 903- 918.

[11] Geoffrey Phipps. 1999. Comparing observed bug and productivity rates for Java and C++. *Softw. Pract. Exper.* 29, 4 (April 1999), 345-358.

[12] Todd L. Graves, Alan F. Karr, J. S. Marron, and Harvey Siy. 2000. Predicting Fault Incidence Using Software Change History. *IEEE Trans. Softw. Eng.* 26, 7 (July 2000), 653-661.

[13] Nicholas Zvegintzov. 1998. Counterpoint: Software Should Live Longer. *IEEE Softw.* 15, 4 (July 1998), 19-.

[14] Cotroneo D, Natella R, Pietrantuono R. 2010. Is software aging related to software metrics?," in *Software Aging and Rejuvenation (WoSAR), IEEE Second International Workshop on*. 1–6.

[15] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. 2000. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA.

[16] Walter F. Tichy. 1998. Should Computer Scientists Experiment More?. *Computer* 31, 5 (May 1998), 32-40