Relationship Between Design and Defects for Software in Evolution

MATIJA MILETIĆ, MONIKA VUKUŠIĆ, GORAN MAUŠA and TIHANA GALINAC GRBAC,
University of Rijeka, Faculty of Engineering

Successful prediction of defects at an early stage is one of the main goals of software quality assurance. Having an indicator of the severity of defect occurrence may bring further benefit to allocation of testing resources. Code churn in terms of modified lines of code was found to be a good indicators of bugs. However, that metric does not reveal the structural changes of the code design. The idea of this project is to analyze the relationship between the evolution of object oriented software metrics and the appearance of defects. To achieve this, the absolute and relative differences between the initial version of a class and its current version were calculated for each metrics as indicators of design change. Then, the correlation between these differences and the number of defects were analyzed. Our case study showed that certain metrics have no influence on defect occurrence, while several of them exhibit moderate level of correlation. In addition, we concluded that the relative differences were inappropriate indicator for determining the relationship.

Categories and Subject Descriptors: D.2.5 [**SOFTWARE ENGINEERING**]: Testing and Debugging—*Tracing*; D.2.9 [**SOFTWARE ENGINEERING**]: Management—*Software quality assurance (SQA)*; H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**] Information Search and Retrieval

Additional Key Words and Phrases: Design chage, defect occurrence, evolving software, correlation

1. INTRODUCTION

In software development, one of the main problems is predicting defects during the software product life cycle. In our analysis, we focused on that problem and investigated whether it can be tackled as early as in the design stage of evolving software. In the design stage of software life cycle, most of the effort is invested in emergence of a module. Then, with each upgrade of this module, it is possible to break concepts of good design. Numerous changes are introduced, and these changes can also cause defects. From this standpoint, we would like to find the design metrics that are critical to the appearance of defects. Product metrics have been successfully used to build prediction models [Basili et al. 1996], so a great number of such metrics is investigated in this study.

When developing a software product, it is essential to plan the necessary actions needed to upgrade a certain functionality. Often, through this process, developers need to modify, delete or add new lines of program code. The amount of program code that goes through these actions is called code churn. Developers should strive to minimize the quantity of program lines that will be modified, thus achieving a smaller code churn ratio [Munson and Elbaum 1998]. After the development phase, the software

This work has been supported in part by Croatian Science Foundation's funding of the project UIP-2014-09-7945 and by the University of Rijeka Research Grant 13.09.2.2.16.
Author's address: M. Miletić; email:mmiletic@riteh.hr; M. Vukušić; email: movukusic@riteh.hr; G. Mauša, Faculty of engineering, Vukovarska 58, 51000 Rijeka, Croatia; email: gmausa@riteh.hr; T. Galinac Grbac, Faculty of engineering, Vukovarska 58, 51000 Rijeka, Croatia; email: tgalinac@riteh.hr.

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2017: 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Belgrade, Serbia, 11-13.9.2017, Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

product also requires maintenance. Many changes are often introduced during maintenance, which are caused by various factors. Some of them include changes in the environment, unused code, rarely updated code, etc. With the aging of the software product and the consequent links of the above-mentioned factors, the software becomes increasingly difficult to maintain, leading to software design deterioration [Land 2002]. Code churn was used to predict defects in many studies but a finer investigation of source code changes yields stronger correlation [Giger et al. 2011].

Projects in evolution are consisting of a number of versions of program code. In the design stage of each version, new requirements are introduced into the existing system architecture. The effort put into this phase affects the effort required to fix defects [Nadir et al. 2016]. Different design metrics of the object-oriented program code can be measured then. It would be highly beneficial to consider the effect of the program design change on the severity of defect occurrence at this early stage. Their relationship will be analyzed as a first step to achieve this goal. The main contribution of this paper is the methodology for investigating the effect of the program design change on the severity of defect occurrence. The design change is represented by the relative and absolute differences of software metrics between consecutive releases of an evolving project. The number of defects is regarded as the severity of occurrence of defects, with higher values corresponding to greater severity.

The study analyzed the afore mentioned relationship between calculated differences and defect occurrence according to the methodology presented in section 2. This project performed a case study based on a sample of data from two Eclipse open source projects according to the description in section 3. The results of this analysis are represented in form of scatter plot diagrams and 3D histograms in section 4. Furthermore, the results were grouped for easier comprehension and interpretation discussed in section 5. Based on calculated data, it is possible to determine if there are some metrics that developers should pay special attention to in the design phase. In addition to these pieces of information, it should be possible to reduce the defect appearance. A future step would be to build a software defect prediction model based on the output data of our program. For example, Genetic Programming algorithms have recently been used to generate high-quality predictors that perform this task successfully [Mauša and Grbac 2017].

2. METHODOLOGY

Figure 1 displays data flow from our project and encapsulates all the steps in data processing. Input data includes several consecutive releases of open-source projects and contains the values of software metrics and the number of defect occurrences. The output data is consisted of *csv* files that contain program design change and graphical representations of correlation analysis.

To measure the program design change, static code attributes software metrics are used. The amount of change is expressed by the relative and absolute difference between the current and a previous release of a class. Relative code churn was found to have stronger correlation with defect density that absolute churn for lines of code (LOC) [Nagappan and Ball 2005] so this study will analyze both. The values of metrics and calculated differences for each metric are written in output file so that we can use them in the later analysis. The program also adds a binary indicator if the class is found in a previous version of program code, named ”Class found”. It is worth noting that a great number of software modules in a complex system is expected not to contain defects or code change [Fenton and Ohlsson 2000] [Runeson et al. 2013]. Since this project is interested in the severity of defect occurrence, the non-defective software modules are not considered. Furthermore, because of interest in the quantity of design change, the software modules without a previous version are also not considered.

Within this case study, two algorithms are prepared: ”compareFiles.py” and ”compareOldestFiles.py”. These algorithms calculate the amount of change for all the metrics of software modules in a selected (current) release. The ”compareFiles.py” script computes the differences between the current release

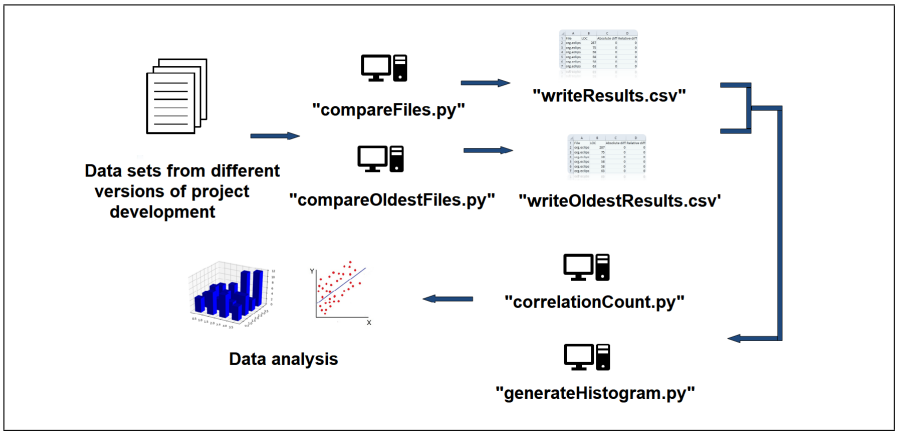

Fig. 1. Data flow in project development

and all the previous releases selected by the user, while the ”compareOldestFiles.py” script computes the differences between the current release of a software module and its first (oldest) appearance in the evolution of the project. The computed amount of change is added to input data and algorithms generate the ”writeResults.csv” or ”writeOldestResults.csv” output files.

These output files are afterwards used as input files for ”correlationCount.py” and ”generateHistogram.py” algorithms. The ”correlationCount.py” algorithm generates a scatter plot diagram between the amount of change of each metric and the number of defects, with corresponding correlation coefficients. The correlation analysis is a standard procedure in analyzing the relationship between a metric and defect count and it may even reveal a good candidate predictors for defect prediction models [Zhang et al. 2017]. The ”generateHistogram.py” algorithm generates a histogram with a 3D projection. To analyze the results, they are grouped according to Spearman’s correlation coefficient. The aim of this analysis is to detect which design metrics are critical to the severity of defect occurrence.

3. CASE STUDY

The idea of this project was to analyze data from different datasets and try to find relationship between input data and defect occurrence in program code. To accomplish this, the following functionalities were implemented iteratively:

(1) Algorithm that computes absolute and relative difference between metrics for every class inside current project version and one or more selected files for comparing.
(2) Algorithm that computes relative and absolute differences for every class from current file with the oldest version of project that contains certain class.
(3) Graphical representation of data collected from previous algorithms:
   (a) Histogram within 3D projection showing relationship between relative or absolute differences, defect count and frequency of defect occurrences inside a certain range.
   (b) Scatter diagrams showing relationship between relative or absolute differences and defect count accompanied with calculated Pearsons and Spearman’s coefficient.

3.1 Datasets

Input datasets that are used in our research included 5 consecutive versions of Eclipse JDT (Java development tools) and PDE (Plug-in Development Environment) open source projects. The data were

collected using a systematically defined data collection procedure [Mauša et al. 2016] implemented in Bug-Code Analyzer tool. The JDT project provides the tool plug-ins that implement a Java IDE supporting the development of any Java application, including Eclipse plug-ins[1]. The Plug-in Development Environment (PDE) provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites, and RCP products[2]. The included releases are named: 2.0, 2.1, 3.0, 3.1 and 3.2.

The datasets are given in *csv* format with comma as the delimiter and point as decimal mark. First row contains description of metrics: column 1 is the file path, columns 2 49 are independent variables (software metrics) described with their abbreviations, column 50 is the dependent variable (number of defects). The datasets were cleared of files that contained .example or .tests in their path. Full list of metrics and the explanation of their abbreviations can be found in [Mauša and Grbac 2017] and on-line[3].

3.2 Evaluation

After the D'Agostino and Pearson's normality test was performed, it was found that input data is not normally distributed. This analysis is necessary to select an appropriate method for the calculation of correlation coefficients. Unlike Pearson's correlation, the Spearman's correlation analysis does not require the data to be normally distributed. Hence, this non-parametric statistical method had to be used for the purpose of correlation expression [D'Agostino and Pearson 1973].

Spearman's correlation coefficient is a statistical measure of the power of a monotonic relationship between paired data. It is calculated according to the formula presented in equation 1 [Myers and Well 1991].

$$r_s = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n^3 - n} \tag{1}$$

where r_s denotes its value in a sample s. The range of possible values is as follows [Lehman 2005] :

$$-1 \leq r_s \leq 1 \tag{2}$$

The interpretation is similar to Pearson's correlation, but closer to a stronger monotonic relationship. The relationships between variables may be determined by their correlative dependence as follows:

(1) Positive correlation - the small value of one variable corresponds to the small value of the second variable, also the high value of one variable corresponds to the high value of the other variable

(2) Negative correlation - the small value of one variable corresponds to the high value of the second variable and vice versa.

(3) Correlation does not exist when a value of a variable can not be inferred from the value of another variable based on the value of a variable. The points in such a graph are scattered [Evans 1996].

Correlation is the size of the effect and so we can verbally describe the correlation force using the Table I [Evans 1996].

[1] http://www.eclipse.org/jdt/
[2] http://www.eclipse.org/pde/
[3] http://www.seiplab.riteh.uniri.hr/wp-content/uploads/2016/12/Table-of-Metrics.pdf

Table I. Correlation ranges

Coefficient	Interpretation
0.00-0.19	Very poor correlation
0.20-0.39	Low correlation
0.40-0.59	Moderate correlation
0.60-0.79	Strong correlation
0.80-1.00	Very strong correlation

4. RESULTS

The first analysis counted the number of files that had its earliest version in each of the previous releases. Table II shows this number as follows: first column represents the number of files that had no previous release, the second column represent the number of files that had its earliest version in the previous release, and so on until the last column shows the number of files that had its earliest version in the oldest release.

Table II. Number of files within each release of open source project

Project	Release 3.2	Release 3.1	Release 3.0	Release 2.1	Release 2.0	Total
JDT	333	231	339	110	1220	2233
PDE	417	271	255	79	329	1351

In Table II we can see that in ”PDE” project the number of newly released files is approximately equally distributed in every release. For the ”JDT” project we can see that most of the files were created in the first release (2.0). Every following release is mainly built upon the previous release and the number of files depends on the newly added features. With the exception of Release 2.1, most of the other versions have similarly distributed newly released files.

Tables III and IV show the obtained Spearman correlation coefficients for the following pairs:

—Correlation between the metric’s real value and number of defects (column ”value”)

—Correlation between the absolute difference of the metric and number of defects (column ”abs_diff”)

—Correlation between the relative difference of the metric and number of defects (column ”rel_diff”)

Under the column ”single_previous_rls” the correlation values are shown for the comparison between the fifth version of the open source project release and its previous version. Under the column ”oldest_rls” the same values are shown, but for the comparison between the fifth version and the oldest appearance of every class in it.

Using the previously mention tables we can see which of the metrics achieve the greatest correlation coefficient. These metric have the most significant impact on the occurrence of defects. For that reason, in Tables III and IV we showed only the metrics with the correlation coefficient value greater then ”0.20”. Metrics that fall into the category of ”Very poor correlation” in both datasets include the following: HCLOC, MI, DIT, MINC, S_R, FOUT, NSUP, NSUB, INTR, MOD. Metrics that achieve the ”Very poor correlation” only in ”JDT” project include HCWORD, LCOM, COH, EXT, MPC. Similarly, metrics that fall into that category only in ”PDE” project include CLOC, CWORD, AVCC, CBO, CCML, F_IN, HIER, SIX, NCO.

The degree of correlation between the absolute and relative differences and the number of defects are shown in a scatter plot diagram representing a two-dimensional graph. The y-axis represents the number of defects, while the x-axis represents calculated absolute or relative difference of the certain metric. Additionally, Spearman’s correlation coefficient is calculated and displayed above diagram. In order to make results more intuitive and better visualized, results are shown graphically on histogram in 3D projection. From them it is easier to bring conclusions about how the data is divided between

given ranges. Every histogram has x, y, and z axis. X-axis represents computed relative or absolute difference between metrics. Y-axis represents data about defect count from the given input file. Z-axis represents frequency of data appearance from y-axis in a range on x-axis. Due to limited space, these diagrams are shown only for SLOC_L metric which had the greatest value of correlation coefficient in both projects in figures 2 and 3. After absolute difference value higher than 500 the number of defect occurrences is negligible. The biggest concentration of defect occurrence with values between 0 and 5 are in range of absolute difference from 0 to 500, with frequency value from z axis of around 500. Similarly, on the right side of the figure the second histogram represents number of defect occurrences inside a certain range of relative difference on x axis. From the Figure 3 we can see that the biggest concentration of the defect occurrences are in a range of 0 to 1 values on x axis. After the relative difference reaches the value of 1 there is no significant defect occurrence.

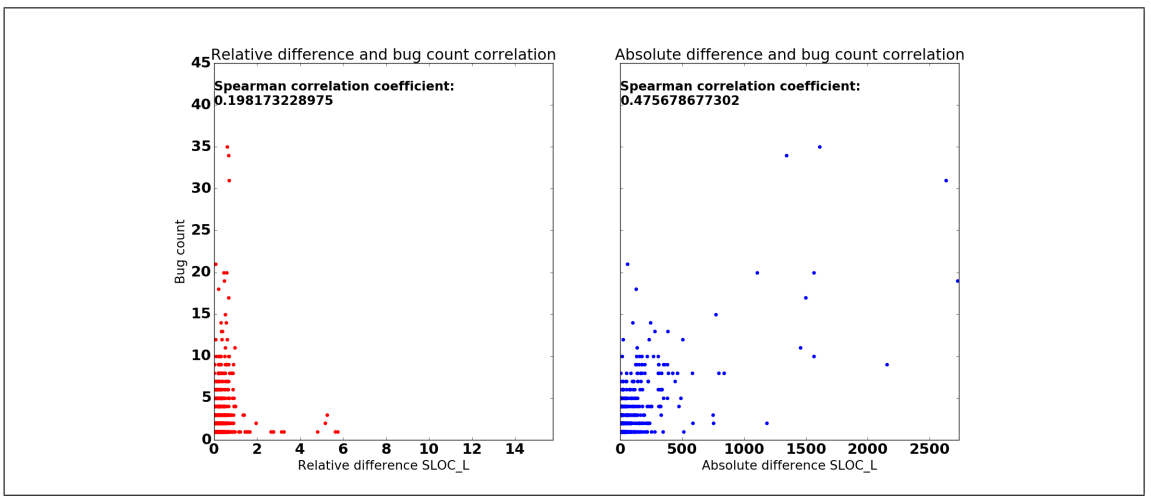

Fig. 2. Scatter diagrams of the correlation between differences and number of defects for the SLOC_L metric, with the corresponding Spearman's correlation coefficient.

5. DISCUSSION

In this chapter we will give a more detailed overview of the obtained results and show them on the example of specific metrics. Further more, we will discuss results obtained from two different datasets, that refer to "JDT" and "PDE" open source project releases.

Analysis of the obtained results have shown that certain metrics from different datasets fall into the same correlation range. Analyzing both datasets, we noticed that the greatest correlation coefficient are achieved for the difference between the value of the metric itself and the number of defects. This confirms the general appropriateness of metrics for software defect prediction and indicates their capability to distinguish the severity of defect occurrence. However, this study was motivated to investigate whether similar effect can be discovered for the quantity of change. When sorting the data by the "value" column we can observe that most of the metrics fall into the category of moderate correlation. The lowest value of correlation coefficients are achieved between the relative difference and the number of defects. Hence, we used the metrics from the "abs_diff" column for this discussion.

When analyzing the data from the Tables III and IV we can see that the highest correlation coefficient are most often achieved by the same metrics. For example, we can see from the Table IV that

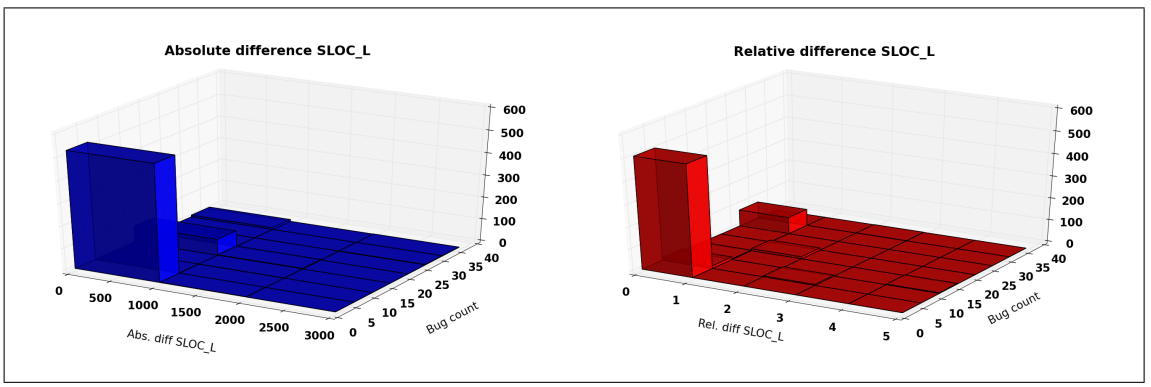

Fig. 3. The correlation between the differences of the SLOC_L metric and number of defects shown in 3D projection.

metrics SLOC_L and SLOC_P achieve similar correlation coefficients in single and oldest releases. In contrast, metric R_R falls into ”Moderate correlation” in the single version, but in the oldest version it falls into negative ”Very poor correlation”. Hence, we can conclude that both of the proposed approaches may contribute to the research performed in this project.

Tables III and IV do not contain the same set of metrics because the ones with low correlation coefficient were omitted. However, there are some metrics with higher correlation coefficient. Metrics LOC (Total Lines of code), SLOC_P (Pyisical executable source lines of code) and SLOC_L (Logical source lines of code) achieve the greatest positive correlation coefficients in both datasets. The correlation of these metrics is intuitive because the greater amount of code is introduced, the greater is the possibility for a defect to occur. It is also important to mention the cyclomatic complexity metric which is used to indicate the complexity of a program [McCabe 1976]. The correlation coefficients of metrics MVG (McCabe VG Complexity), AVCC (Average Cyclomatic Complexity of all the methods in the class) and MAXCC (Maximum Cyclomatic Complexity of any method in the class) fall into the category of positive ”moderate correlation”. This shows that more independent paths through an algorithm, i.e. higher cyclomatic complexity, increase the possibility of defect occurrence. Furthermore, MVG exhibits the highest correlation coefficient in JDT project when a file is compared to its oldest version.

For better representation of data Figures 2 and 3 are displaying computed results for SLOC_L metric. The figures clearly show higher Spearman’s correlation coefficient for relationship between the absolute difference value and the number of defect occurrence, with coefficient value around 0.48, than for the relative difference value. Spearman’s correlation coefficient between the relative difference and defect occurrence is around 0.20, thus is very low. This is a strong indicator that absolute differences are more relevant for measuring the afore mentioned relationship of defect occurrence.

From the ”JDT” dataset the following metrics achieved the greatest correlation coefficients in the single previous version: HCLOC, MVC, LOC, SLOC_P, SLOC_L. Similarly in the oldest version, metrics MVG, LOC, SLOC_P, SLOC_L and C_SLOC achieved the greatest correlation coefficients. Metrics with the greatest correlation coefficient in the ”PDE” dataset from the single previous version include UWCS, LMC, BLOC, MVG and No_Methods metrics. From the oldest version they include BLOC, RFC, LMC, No_Methods and UWCS metrics.

5.1 Threat to Validity

The validity of this small scale case study is affected by the choice of data. This is a first preliminary study, so it is based only on a sample of data from five consecutive releases of two open source projects from Eclipse community, namely, JDT and PDE. Thus, external validity is the main threat to this study

Table III. Spearman's correlation coefficient for JDT input data

	single_previous_rls			**oldest_rls**		
Metrics	**value**	**abs_diff**	**rel_diff**	**value**	**abs_diff**	**rel_diff**
LOC	0.58	0.42	0.02	0.53	0.47	0.22
SLOC_P	0.58	0.42	-0.01	0.52	0.47	0.18
SLOC_L	0.58	0.41	-0.04	0.52	0.48	0.20
MVG	0.62	0.39	-0.19	0.56	0.48	0.08
BLOC	0.49	0.19	-0.23	0.44	0.37	0.09
C_SLOC	0.58	0.18	-0.48	0.49	0.35	-0.25
CLOC	0.40	0.24	-0.09	0.35	0.27	0.06
CWORD	0.51	0.44	0.14	0.45	0.38	0.16
No_Methods	0.50	0.26	-0.25	0.41	0.40	0.08
AVCC	0.47	0.07	-0.15	0.41	0.23	0.05
NOS	0.53	0.35	-0.12	0.47	0.42	0.14
HBUG	0.53	0.42	-0.04	0.48	0.47	0.19
HEFF	0.52	0.44	-0.02	0.48	0.47	0.17
UWCS	0.51	0.24	-0.26	0.41	0.35	0.05
INST	0.28	0.09	-0.27	0.28	0.24	-0.09
PACK	0.13	0.13	-0.02	0.11	0.24	0.14
RFC	0.49	0.23	-0.26	0.40	0.39	0.08
CBO	0.10	0.47	0.09	0.39	0.32	-0.32
CCML	0.45	0.38	-0.14	0.36	0.38	0.07
NLOC	0.52	0.35	-0.10	0.45	0.42	0.14
F_IN	0.10	0.38	0.00	0.36	0.36	-0.27
R_R	0.24	-0.20	-0.31	-0.07	-0.25	-0.23
LMC	0.54	0.13	-0.47	0.42	0.36	-0.13
LCOM2	0.42	0.36	-0.12	0.33	0.38	0.12
MAXCC	0.54	0.22	-0.32	0.47	0.28	-0.10
HVOL	0.52	0.41	-0.03	0.48	0.47	0.20
HIER	0.38	0.27	-0.19	0.21	0.25	0.07
NQU	0.21	0.10	-0.15	0.18	0.23	0.05
SIX	-0.30	-0.08	0.11	-0.42	-0.24	0.20
TCC	0.57	0.38	-0.16	0.49	0.45	0.11
NCO	0.51	0.21	-0.34	0.42	0.38	0.03
CCOM	0.53	0.35	-0.27	0.44	0.40	-0.01
HLTH	0.53	0.40	-0.03	0.48	0.46	0.20

because it cannot discover general conclusions. Nevertheless, the obtained results are a motivation to put more effort in this research direction. Projects from different background, like different communities, development methodologies or written in different programming language need to be included to obtain more general conclusions. As industrial data are difficult to obtain, the construct validity is also threatened. That is why the chosen projects are complex and long lasting ones that may approximate industrial projects. The statistical analyses are threats to internal validity, but they are all based on well known and widely used tests. The conclusions about the level of correlation and the importance of software metrics for defect prediction lacks a precise explanation, hence threatening the conclusion validity. The causality of the conclusions is unknown and it remains open to speculations.

6. CONCLUSION

When developing a project it is very important to pay attention to the appearance of defects in program code. Since the design is the phase in which the requirements are introduced into the existing system architecture, it is particularly important to pay attention to the occurrence of malfunctions that

Table IV. Spearman's correlation coefficient for PDE input data

	single_previous_rls			**oldest_rls**		
Metrics	**value**	**abs_diff**	**rel_diff**	**value**	**abs_diff**	**rel_diff**
LOC	0.45	0.38	0.13	0.47	0.39	0.12
SLOC_P	0.43	0.35	0.08	0.46	0.41	0.13
SLOC_L	0.45	0.37	0.07	0.46	0.43	0.16
MVG	0.51	0.32	-0.06	0.43	0.35	0.01
BLOC	0.52	0.33	-0.03	0.51	0.33	-0.07
C_SLOC	0.33	0.18	-0.17	0.38	0.34	-0.20
HCWORD	-0.28	0.06	0.06	-0.29	-0.28	-0.29
No_Methods	0.50	0.25	-0.18	0.48	0.26	-0.12
LCOM	-0.25	-0.16	0.12	-0.29	-0.26	0.01
NOS	0.45	0.35	0.06	0.46	0.38	0.06
HBUG	0.45	0.34	0.03	0.45	0.36	0.03
HEFF	0.42	0.33	0.03	0.43	0.40	0.09
UWCS	0.53	0.29	-0.11	0.48	0.34	-0.03
INST	0.36	0.20	-0.17	0.37	0.21	-0.18
PACK	0.29	0.17	-0.07	0.26	0.22	0.07
RFC	0.50	0.25	-0.18	0.49	0.27	-0.12
NLOC	0.44	0.35	0.06	0.45	0.36	0.07
R_R	0.00	0.58	0.33	0.13	-0.26	-0.36
COH	-0.27	-0.19	0.01	-0.32	-0.20	-0.01
LMC	0.53	0.24	-0.32	0.49	0.29	-0.23
LCOM2	0.43	0.32	-0.17	0.43	0.34	-0.11
MAXCC	0.31	0.10	-0.14	0.33	0.26	-0.02
HVOL	0.45	0.35	0.05	0.46	0.38	0.07
NQU	0.34	0.13	-0.26	0.32	0.31	-0.06
EXT	0.00	-0.77	-0.32	0.01	-0.20	-0.20
TCC	0.45	0.32	0.01	0.44	0.35	0.02
MPC	0.00	-0.77	-0.32	0.01	-0.20	-0.20
CCOM	0.33	0.35	-0.04	0.31	0.33	-0.06
HLTH	0.44	0.35	0.06	0.45	0.37	0.07

may arise due to necessary changes. The goal of this project was to establish the correlation between changes in design metrics and the appearance of defects.

This paper provides a methodology for estimating the relationship between the quantity of change and the severity of defect occurrence. The methodology was used in a small scale case study which used 5 subsequent releases of "JDT" and "PDE" Eclipse open source community projects as datasets.

The algorithms that implement the proposed methodology have been applied to all software metrics in the datasets. Generated output data is used as the input file to algorithms that computes metric correlation and graphically display obtained relationships. The results were afterwards grouped according to Spearman's correlation coefficient, as described in section 3.1. The analysis showed a very week or week correlation between changes in metric value and defect occurrence in design stage for most of the metrics. Furthermore, there was no strong correlation found between any of the design metrics and the defect occurrence. However, these results were somewhat expected. The aim of this project was to find design metrics that could bring additional information in the prediction of defects. For example, SLOC_P, SLOC_L and HEFF metrics exhibit a moderate correlation between the changes in the value of a particular metric and defect occurrence. The moderate level of correlation is an indication enough that the proposed metric calculation may improve the categorization of severity of defect occurrence and possibly the software defect prediction. The fact that the values of metrics themselves, which are traditionally already used for software defect prediction, have similar values of correlation,

encourages us to continue this research. When these metrics are known it is possible to pay more attention to them when introducing new requirements, or re-designing the existing software modules, and thus reduce the possibility of defect occurrence.

To summarize, this research came to following conclusions:

(1) Metrics like HCLOC, MI, DIT, MINC, S_R, FOUT, NSUP, NSUB, INTR and MOD exhibit very poor correlation in both projects and may have lower impact on the level of defect occurrence.
(2) Metrics SLOC_L and SLOC_P achieve the highest correlation coefficients in both input datasets. Hence, we can conclude that they have the greatest impact on defect occurrence.
(3) Spearman's correlation coefficients between the relative difference and defect occurrence are very low. Thus, we can conclude that they are not good indicators of design change quality.

REFERENCES

Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. 1996. A Validation of Object-Oriented Design Metrics As Quality Indicators. *IEEE Trans. Softw. Eng.* 22, 10 (Oct. 1996), 751–761.

Ralph D'Agostino and Egon S. Pearson. 1973. Tests for Departure from Normality. Empirical Results for the Distributions of b_2 and $\sqrt{b_1}$. *Biometrika* 60, 3 (1973).

James D. Evans. 1996. *Straightforward Statistics for the Behavioral Sciences*. Brooks/Cole Publishing Company.

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

Emanuel Giger, Martin Pinzger, and Harald C. Gall. 2011. Comparing Fine-grained Source Code Changes and Code Churn for Bug Prediction. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*. ACM, New York, NY, USA, 83–92.

Rikard Land. 2002. Software Deterioration And Maintainability A Model Proposal. In *In Proceedings of Second Conference on Software Engineering Research and Practise in Sweden (SERPS), Blekinge Institute of Technology Research Report 2002:10*.

Ann Lehman. 2005. *JMP for Basic Univariate and Multivariate Statistics: A Step-by-step Guide*. SAS Institute.

Goran Mauša and Tihana Galinac Grbac. 2017. Co-evolutionary multi-population genetic programming for classification in software defect prediction: An empirical case study. *Appl. Soft Comput.* 55 (2017).

Goran Mauša, Tihana Galinac Grbac, and Bojana Dalbelo Bašić. 2016. A systematic data collection procedure for software defect prediction. *Comput. Sci. Inf. Syst.* 13, 1 (2016).

Thomas J. McCabe. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2, 4 (Dec. 1976), 308–320.

John C. Munson and Sebastian G. Elbaum. 1998. Code Churn: A Measure for Estimating the Impact of Code Change. In *Proceedings of the International Conference on Software Maintenance (ICSM '98)*. IEEE Computer Society, Washington, DC, USA.

Jerome L. Myers and Arnold D. Well. 1991. *Research Design and Statistical Analysis*. HarperCollins.

Shahab Nadir, Detlef Streitferdt, and Christina Burggraf. 2016. Industrial Software Developments Effort Estimation Model. In *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. 1248–1252.

Nachiappan Nagappan and Thomas Ball. 2005. Use of Relative Code Churn Measures to Predict System Defect Density. In *Proceedings of the 27th International Conference on Software Engineering (ICSE '05)*. ACM, New York, NY, USA, 284–292.

Per Runeson, Tihana Galinac Grbac, and Darko Huljenić. 2013. A Second Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. *IEEE Transactions on Software Engineering* 39 (2013), 462–476.

Feng Zhang, Ahmed E. Hassan, Shane McIntosh, and Ying Zou. 2017. The Use of Summation to Aggregate Software Metrics Hinders the Performance of Defect Prediction Models. *IEEE Trans. Softw. Eng.* 43, 5 (May 2017), 476–491.