Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 7 (3): 114-120, 2013
ISSN 1819-4311 / DOI: 10.3923/jse.2013.114.120

Study on the Correlations Between Program Metrics and Defect Rate by a Controlled Experiment

Fuqun Huang and Bin Liu
Centre for Software Dependability, Beihang University, Beijing, China

Corresponding Author: Fuqun Huang, School of Reliability and System Engineering, Beijing, 100191, China

ABSTRACT

Software defect prediction is an important approach that helps practitioners to manage software projects effectively. A large numbers of the existing models are based on program metrics. But how much exactly these metrics can account for defect rate still remains a controversial problem. As is known to all, programming is a knowledge-intensive activity. The content of task and the expertise of individuals influence defect rates a lot. Thus, if these interference factors are not well controlled, it's hard to draw any conclusions on how program metrics affect defect rate alone. There is extremely limited evidence produced by controlled experiments on this problem. This study bridges the gap by conducting a controlled experiment, in which the programming problem is solved by subjects with the same background of academic performance, programming training and programming experience. Fifty four subjects participated in the experiment, with 51 different versions of programs produced in the same language C. The results demonstrate that program metrics can only account for 27.6% variability of defect rate.

Key words: Defect prediction, program metrics, human factors, controlled experiment

INTRODUCTION

Software defect prediction is an important technology to provide evidences for helping people make decisions on when to deploy software system and how to allocate testing resource more effectively. Thus, software prediction is craved by many organizations.

There are a great number of prediction models proposed by the current studies. Most of the existing models are based on the size and complexity metrics of programs. Some models use program metrics as one important dimension of predictors while some others predict defects entirely based on program metrics. However, just as Fenton states that, "the existing models are incapable of predicting defects accurately using size and complexity metrics alone" (Fenton and Neil, 1999). Though many studies demonstrate that there are strong correlations between program metrics and defect rates, some empirical evidences show that program metrics have no predictive power with respect to defect rate (Van der Meulen and Revilla, 2007). How much the program metrics can account for defect rate is still controversial and extremely lack of experimental evidences.

One critical problem confronts researcher is how to isolate other factors that affect defect rates from program metrics. As is known to all, programming is a knowledge-intensive activity. The human factors and the context of task influence defect rates a lot (Adelson and Soloway, 1985; Detienne, 2002; Reason, 1990; Visser and Hoc, 1990). Meanwhile, the complexity of task is always correlated to program complexity. If the tasks are different and the programmers are at different expertise levels, the corresponding programs are incomparable from a scientific rigorous viewpoint. It is hard to draw any scientific conclusion on how much program metrics correlate to defect rates.

To address the problem, a controlled experiment is designed to minimize the effects of task and individual expertise which provides an opportunity to study the correlations between program metrics and defect rate more precisely.

RELATED RESEARCH

There are mainly two groups of program metrics involved in defect prediction which are size and complexity. The size describes the static elements that constitute a program while the complexity represents the structural characteristics of a program.

Size metrics: Lines of Code (LOC), lines of executable code, Halstead volume, are widely used as size metrics in existing models, to predict program defects. Most models assume that larger modules have higher defect densities. However, the controversial phenomenon is confirmed that larger modules can have lower levels of defect density (Karl-Heinrich and Paulish, 1993). Hatton (1997) introduce working memory theories to explain this phenomenon. He holds the view that only those components that fit best into human short-term memory cache seem to use it effectively, thereby producing the lowest fault densities. Bigger and smaller average component sizes appear to degrade reliability.

Recently, in realization that the task affects program size, the use of Albrecht Function Points (FPs) becomes widespread (Fenton *et al*., 2008). In this study, the task is the same for all the subjects. Therefore, the function points are the same for all the subjects. Meanwhile, defect rate is defined as the defect number per function point. When the function points are the same for all subjects, the difference of defect rates between individuals are equivalent to the differences of defect numbers. Thus, in this study, the defect number can be used to represent defect rate. This study aims to investigate the correlations between Executable Line, Halstead Volume and defect number.

Complexity metrics: Researchers have realized that size-based metrics alone are poor general predictors of defect density, complexity metrics are introduced in defect prediction models. McCabe's Cyclomatic Complexity (McCabe, 1976) has been used in many studies (Kitchenham *et al*., 1990; Khoshgoftaar *et al*., 1990). Nesting depth is also found to be one of the most important dimensions that account for defect variability (Neil, 1990).

As the task is fixed for all the individuals, the Numbers of Procedures for different programs are comparable. Number of Procedures should be a good metric to reflect the granularity of the program structure. The same situation is applied to number of loops which can be obtained from the automatic tool. Therefore, the number of loops is also included as one of the metrics in this study.

To sum up, in this study, the programs will be measured by the following software metrics: Lines of Executable Code (LOC), the Halstead Volume (Volume), McCabe's Cyclomatic Complexity (CC), Nest Depth, Number of Procedures and Number of loops.

EXPERIMENT DESIGN

Background: The programming contest of Beihang University (BUAA-ICPC) is an annual programming contest which has been held for 7 years by 2011. The original purpose of the contest is to select candidates to participate in the Asian Qualifying of ACM International Collegiate Programming Contest (ACM-ICPC). Therefore, the procedure, the task specification styles and the Online Judge System involved in the contest are similar

to those used in ACM-ICPC. The subjects and the data were obtained from the 7th BUAA-ICPC. The contest score was given out in real-time by the Online Judge System.

Feedbacks provided by the online judge: The Online Judge is used to test the correctness of programs. Programmers can debug and fix their programs according to the information given by the system and resubmit updated versions until it passed the test. The submission results that the Online Judge fed back to the students include the following types:

- **Accepted (AC):** The program's output matches the Online Judge's output
- **Wrong answer (WA):** The output of the program does not match what the Online Judge expects
- **Presentation error (PE):** Presentation errors occur when the program produces correct output for the Online Judge's secret data but does not produce it in the correct format
- **Runtime error (RE):** This error indicates that the program performs an illegal operation when running on the Online Judge's input. Some illegal operations include invalid memory references such as accessing outside an array boundary
- **Time Limit Exceeded (TL):** The Online Judge has a specified time limit for every problem. When the program does not terminate in that specified time limit, this error will be generated
- **Compile Error (CE):** The program does not compile with the specified language's compiler

Scoring rules: The scoring rule was as follows:

- First the contestants' performance was ranked by their total scores which was equal to the total number of problems solved successfully
- If the total scores were the same, they were ranked by problem solving time: the less time, the higher rank. The time was the sum of the successful submission time and the penalty time for unacceptable submissions. And the penalty time was calculated by 20 min multiplied by the number of unacceptable submissions

Procedure and tools: The aim of this study is to observe empirical information as much as possible under the real circumstances. Thus, the task designs and arrangement of the programming contest was completely independent of this study. The researchers only involved after the contest was completed, accessed to the data and performed analysis.

Static analysis and code inspection for all the programming versions were performed by a senior test engineer from a professional testing centre.

The static analysis was implemented by Testbed C/C++ LDRA Testbed (Version 7.0.1). The code inspection was assisted by Source Insight V3.5 and file comparison tool-Diffuse. Defect types were summarized in this stage. And the statistical analysis was performed by SPSS Statistics V19.

Problem: The specification material of the problem is presented in Appendix A.

Participants: There have been 120 students participating in the contest, among whom 77 students were from BUAA and 43 participants were from other Universities. To make sure the subjects are with the similar backgrounds, the participants from other universities are excluded in our analysis. Among these 77 students, 55 have tried the "jiong" problem. All the selected subjects

in this study were recruited by National College Entrance Examination, in which the test subjects included math, Chinese, English, physics, chemistry and biology. In addition, the students were recruited across the Country, even distributed across all the provinces. Generally, high school students apply universities according to their scores in National College Entrance Examination and the requirement scores of the universities (normally related to the ranks of universities). Therefore, their overall academic achievements in earlier education and cognitive capacities are supposed to be at the same level.

All the students were in grade one, from the department of computer science or software engineering. All the students had just finished their course on C language, without any extra programming experience. Thus, their programming experiences are supposed to be at the same level.

All the students are male, between the ages of 18-21. Thus, their demographic characteristics are supposed to be at the same level.

Based on all the information above, the assumption can be made that all the subjects' demographic characteristic, education background, cognitive capacity and programming experience are at the same level.

Data collection

Defects found by code inspection: There were 26 different defects found in code inspection. The classification and sample of the defects is shown in Table 1.

Table 1: Defects summarized by types

Identifier	Defect type	Description
Dt1	Array size	The size of array defined is smaller than required
Dt2	Printing defect	The blank line after the "jiong" is missing
		Mistook the symbol "!" for" | "
		The symbol '+'in the last line of "jiong" was missing
Dt3	Array initialization	Array or variables were used without being initialized, or initialized by the wrong values
Dt4	Loop	The variable was used as the upper limit in "for" loop, without initialization. As "n" might be in huge value which cause the iteration time exceed limit
		The array used to store the "jong" word was initialized by a 2-depth loop which confronted the problem of time efficiency
		Mistook the counter "n++" for "n--" in "for" loop
Dt5	Syntax	The program was in c++ while the file is ended as ".c"
		The quoting of array was mistaken, the initiation of array was mistaken as "char c[1000][1000] = {'0'}", where the author just initialized the first value of the array to be "0"
		The "memset" function was misused
		Mistook the "\" for "\\"
		"y< = m && m<y+b/2" was expressed as "y< = m<y+b/2" by mistake
Dt6	Algorithm	Used enumeration algorithm, failed to enumerate all possible "jiong"s
		Misunderstood the requirement, printed out all the "jiong"s after all the inputs were entered by the user while the requirement demanded that print each "jiong" after the nest-level was entered by the user
		Mistook the nest level iteration "n = n/2" for "n = n-1" in "for" loop
Dt7	Computing	The location for the symbol "\" was wrong, mistook f4 [32-i][65-i] = '\\' for f4[63-i] = '\\'
		The relationship between the height and nest level of the "jiong" was deduced wrongly, the wrong expression was h = 8n which was supposed to be h = 2^{n+2}
Dt8	Interface	Array citing slips, mistook map[l][1] for map[l][i]

Program measurement data: The descriptive statistics of program measurements are shown in Table 2.

RESULTS

Internal correlations between various metrics: The correlations between metrics are represented in Table 3. Results show that the most widely used metrics-Executable line of code, Halstead Volume and McCabe's Cyclomatic Complexity-are strongly correlated to each other (Mean = 0.904).

These three metrics are also strong correlated to Number of Loops (Mean = 0.788). They are moderately correlated to Nest Depth (Mean = 0.325).

Correlations between Number of Procedures, Number of Loops and Nest Depth are generally moderate: Number of Procedures vs. Number of Loops (-0.334); Number of Procedures vs. Nest Depth (-0.512); Number of Loops vs. Nest Depth (0.372).

Correlations between program metrics and defects: The Correlations between program metrics and defects are presented in the last column of Table 3. From the results we can see that, only Halsteads Volume, Nest Depth and the Number of Procedures are significantly related to defect rate, with the correlations at around 0.3. No other significant correlations were found.

Table 2: Descriptive statistics of program measurement

	Valid N	Min.	Max.	Mean	Std. deviation
LOC	51	64	706	207.96	134.890
Procedures	51	1	4	1.84	0.860
CC	51	5	69	20.51	15.110
Volume	51	290	4021	1587.90	766.940
Nest Depth	51	1	5	2.86	0.825
Loops	51	1	61	12.63	10.060

Table 3: Correlations between various metrics and defects

		LOC	Procedure	CC	Volume	No. of loops	Nest	Defects
LOC	r	1	-0.181	0.944**	0.899**	0.861**	0.318*	0.166
	p		0.203	0.000	0.000	0.000	0.023	0.246
Procedures	r			-0.128	-0.136	-0.334*	-0.512**	-0.341*
	p			0.371	0.341	0.017	0.000	0.014
CC	r			1.000	0.820**	0.698**	0.322*	0.115
	p				0.000	0.000	0.021	0.422
Volume	r				1.000	0.805**	0.334*	0.288*
	p					0.000	0.016	0.040
Loops	r					1.000	0.372**	0.174
	p						0.007	0.221
Nest depth	r						1.000	0.276*
	p							0.050

**Pearson correlation is significant at the 0.01 level (2-tailed), n = 51, *Pearson correlation is significant at the 0.05 level (2-tailed), n = 51

Table 4: Regression results between program metrics and defects

Model	Unstd. coefficient B	Std. error	Std. coefficients beta	t	Sig.
(Constant)	0.973	1.035		0.939	0.353
LOC	0.004	0.009	0.425	0.492	0.625
Procedure	-0.571	0.248	-0.367	-2.306	0.026
CC	-0.054	0.050	-0.610	-1.081	0.285
Volume	0.001	0.001	0.748	2.436	0.019
Nest Depth	0.153	0.264	0.095	0.580	0.565
Loops	-0.070	0.050	-0.526	-1.387	0.172

Dependent variable: Defects

Regression analysis: To explore how all these program metrics can account for defect variability, multiple linear regression analysis is performed. Combined all these metrics together, program metrics can totally explain 27.6% of the variance in defect rates, the multiple correlation $R^2 = 0.276$, $F = 2.789$, $p = 0.022$. The correlation coefficients for each variable are shown in the Table 4.

DISCUSSION AND CONCLUSION

Size and complexity metrics have been used widely as the predictors of software defects. Despite the many efforts to predict defects, there appears to be little consensus on how much program metrics can really account for defect rate. Controlled experimental evidences are extremely limited.

A controlled experiment has been designed to investigate the correlations between program metrics and defect rate. The experiment has controlled the interference of the most critical factors affecting programming performances which are task and individual expertise. The experiment provides an opportunity to reflect on this problem in a scientific way.

The results demonstrate that three most widely used metrics (Executable line of code, Halstead Volume and McCabe's Cyclomatic Complexity) are strongly correlated to each other. Only Halsteads Volume, Nesting depth and Number of Procedures are moderately correlated to defect rate. The regression results show that all the metrics together can only account for 27.6% of the variance in defect rate.

The limit of this study is that the programs are limited to small-scale programs. It bears the threats to the external validity, that whether the results can be generalized to large-scale programs. This is the inherent shortcomings suffered by every experimental study, as the experimental conditions are commonly different from real industry environment. However, a large-scale program is always constituted of small modules. Thus, results of this study have provided evidences at the basic level, as a valuable benchmark for all the defect prediction models based on program metrics. It also has important implications for software practitioners who concerns developing new models and improve software quality management.

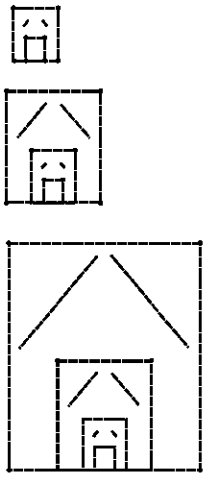
APPENDIX

Appendix A:

Sample input
3
1
2
3

Appendix A: Continue

Sample output



REFERENCES

Adelson, B. and E. Soloway, 1985. The role of domain experience in software design. Software Engin. Trans., 11: 1351-1360.

Detienne, F., 2002. Software Design-Cognitive Aspects. Springer-Verlag, New York, Inc., USA.

Fenton, N. and M. Neil, 1999. A critique of software defect prediction models. IEEE Trans. Software Eng., 25: 675-689.

Fenton, N., M. Neil, W. Marsh and P. Hearty, 2008. On the effectiveness of early life cycle defect prediction with Bayesian Nets. Empirical. Softw. Engg., 13: 499-537.

Hatton, L., 1997. Re-examining the fault density-component size connection. Software, 14: 89-97.

Karl-Heinrich, M. and D.J. Paulish, 1993. An empirical investigation of software fault distribution. Proceedings of the First International Symposium on Software Metrics, May 21-22, 1993, IEEE CS Press, pp: 82-90.

Khoshgoftaar, M. Taghi and J.C. Munson, 1990. Predicting software development errors using complexity metrics. J. Selec. Areas. Comm., 8: 253-261.

Kitchenham, B.A., L.M. Pickard and S.J. Linkman, 1990. An evaluation of some design metrics. Software Eng. J., 5: 50-58.

McCabe, T.J., 1976. A complexity measure. IEEE Trans. Software Eng., 2: 308-320.

Neil, M.D., 1990. Multivariate assessment of software products. J. Software. Testing, Verific. Reliab., 1: 17-37.

Reason, J., 1990. Human Error. Cambridge University Press, Cambridge, New York.

Van der Meulen, M.J.P. and M.A. Revilla, 2007. Correlations between internal software metrics and software dependability in a large population of small C/C++ programs. Proceedings of the 18th International Symposium on Software Reliability, (ISSRE '07), IEEE Comput. Soc. Washington, DC., USA., pp: 203-208.

Visser, W. and J.M. Hoc, 1990. Expert Software Design Strategies. In: Psychology of Programming. Hoc, J.M., T.R.G. Green, R. Samurcay and D. Gilmore (Eds), Acad. Press. Limit., pp: 235-249.