Correlations between Internal Software Metrics and Software Dependability in a Large Population of Small C/C++ Programs

Meine J.P. van der Meulen
Centre for Software Reliability
City University
London EC1V 0HB, UK

Miguel A. Revilla
Department of Applied Mathematics
University of Valladolid
47011 Valladolid, Spain

Abstract

Software metrics are often supposed to give valuable information for the development of software. In this paper we focus on several common internal metrics: Lines of Code, number of comments, Halstead Volume and McCabe's Cyclomatic Complexity. We try to find relations between these internal software metrics and metrics of software dependability: Probability of Failure on Demand and number of defects.

The research is done using 59 specifications from a programming competition—The Online Judge—on the internet. Each specification provides us between 111 and 11,495 programs for our analysis; the total number of programs used is 71,917. We excluded those programs that consist of a look-up table.

The results for the Online Judge programs are: (1) there is a very strong correlation between Lines of Code and Halstead Volume; (2) there is an even stronger correlation between Lines of Code and McCabe's Cyclomatic Complexity; (3) none of the internal software metrics makes it possible to discern correct programs from incorrect ones; (4) given a specification, there is no correlation between any of the internal software metrics and the software dependability metrics.

1 Introduction

Software metrics have been subject of research since the seventies, and expectations were high that metrics would exist to help managerial decision making during the software lifecycle. Software metrics come in many flavours (e.g. described by Fenton e.a. in [3]). Essentially any metric is an attempt to measure or predict some attribute (internal or external) of some product, process or resource. Normally, the internal attributes are those that we can directly measure, and the external ones those that we are interested in [2]. In this paper we concentrate on a few internal, product-related software metrics: Lines of Code, number of comments, Halstead Volume and McCabe's Cyclomatic Complexity. We contrast these with two external software metrics: number of defects and Probability of Failure on Demand (PFD).

There have been many attempts to use software metrics in the development of software. Kafura reports that a collection of software metrics can be used to identify those components which contain an unusually high number of errors or which require significantly more time to code than the average [5]. In general however, the results have been ambiguous at least. Fenton, in [1], states: "Specifically, we conclude that the existing models are incapable of predicting defects accurately using size and complexity metrics alone. Furthermore, these models offer no coherent explanation of how defect introduction and detection variables affect defect counts."

In this paper we investigate which correlations between internal software metrics and dependability metrics exist by analysing a very large collection of small C/C++ programs submitted to the "Online Judge".

2 The experiment

2.1 The UVa Online Judge

http://acm.uva.es, the "UVa Online Judge"-Website [8], is an initiative of one of the authors (Revilla). It contains program specifications for which anyone may submit programs in C, C++, Java or Pascal intended to implement them. The correctness of a program is automatically judged by the "Online Judge". Most authors submit programs repeatedly until one is judged correct. Tenthousands of authors contribute and together they have produced more than 3,000,000 programs for the approximately 1,500 specifications on the website (as of May 2004, the programs submitted at that date are used in this experiment).

1071-9458/07 $25.00 © 2007 IEEE
DOI 10.1109/ISSRE.2007.12

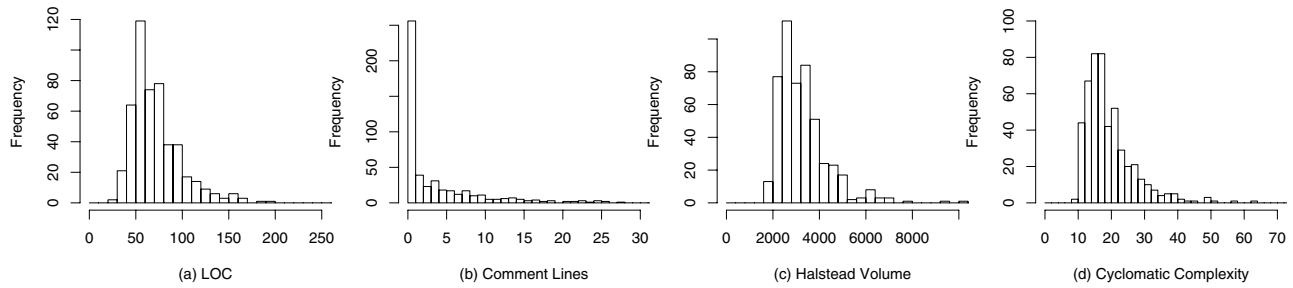


Figure 1. Histograms of internal software metrics of the correct submissions for the "Unidirectional TSP"-problem. The vertical axis presents the number of programs.

The program specifications of the Online Judge contain a short description of the problem, mainly in natural language, and some example input and output. The formalism of the specifications differs from specification to specification, but most are more or less informal.

Note that we do not use the test results of the Online Judge, except that we only use those programs that the Online Judge *can* run, see Section 2.2. We test the programs ourselves for determining the external metric PFD as explained in Section 3.2.

2.2 Selection of specifications and programs

We selected 59 specification from the Online Judge from different domains: graph theory, string operations, mathematical puzzles, etc. To enable statistical analysis, we selected specifications for which many submissions existed. From all the programs submitted to the Online Judge for these 59 specifications, we use:

Programs running under the Online Judge. We only use those programs that the Online Judge can execute, i.e. the Online Judge was able to compile the program, was able to run it, and the program runs using prescribed time and memory resources. This first filter saves us much time, and also protects us from malicious programs like fork bombs.

Programs in C or C++. We only chose programs in C or C++, because our tools calculate the internal metrics for these programming languages. Apart from this practical reason, mixing programming languages in this research might invalidate the results, or at least complicate their interpretation.

Programs that succeed for at least one demand. We excluded the completely incorrect submissions, because there is obviously something wrong with these in a way that is outside our scope (these are often submissions to the wrong specification, or apply incorrect formatting to the output).

First submission of each author. We only used one program submitted by each author and discard all other submissions (except for the determination of the number of defects, see Section 3.2). Subsequent submissions have comparable fault behaviour and this dependence between submissions would invalidate the statistical analysis.

Programs smaller than 40kB. A manipulation of the data we allowed ourselves is that we remove those programs from the analysis that have a filesize over 40kB. This is the maximum size allowed by the Online Judge, but was not enforced for a small period of time, and during this time some authors managed to submit programs exceeding this limit. Imposing this restriction does therefore only enforce a restraint that already in principle existed.

No look-up tables. We also disregarded those programs that consist of look-up tables, because their software metrics are completely different (the Halstead Volume is in general more than ten times the average for all programs written to a specification, thus completely dominating statistical analysis). These programs are very easily distinguishable from others, because they combine a very high Halstead Volume with a very low Cyclomatic Complexity. In rare cases a look-up table has a very high Cyclomatic Complexity, more than a hundred; in these cases the table is programmed with if-then-else statements. We deleted these programs manually from the analysis. In total 314 of the 41,685 remaining correct programs (0.75%), and 132 of the 30,232 remaining incorrect programs (0.43%) were disregarded.

The total number of submissions to the 59 specifications used in our analyses was 71,917, on average 1,219 per specification.

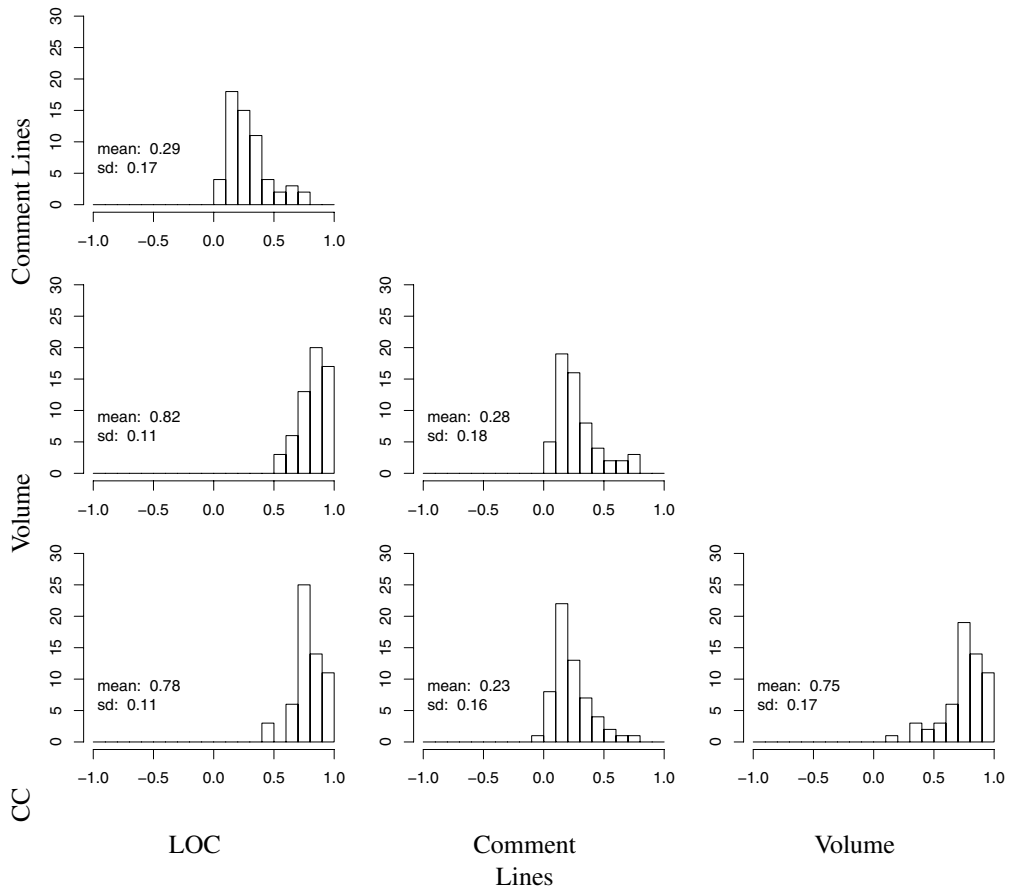


Figure 2. Histograms of the correlations between the various internal software metrics. The vertical axis depicts the number of specifications.

3 Measurement of software metrics

3.1 Internal metrics

We automatically measured the following internal software metrics: Lines of Code (LOC), the number of comment lines, the Halstead Volume (Volume) [4], and McCabe's Cyclomatic Complexity (CC) [6]. These four are very commonly used in the assessment of programs and many (commercial) tools give the possibility to measure them. The distributions of these metrics is similar for all specifications. Figure 1 presents a typical example for one specification, named "Unidirectional TSP".

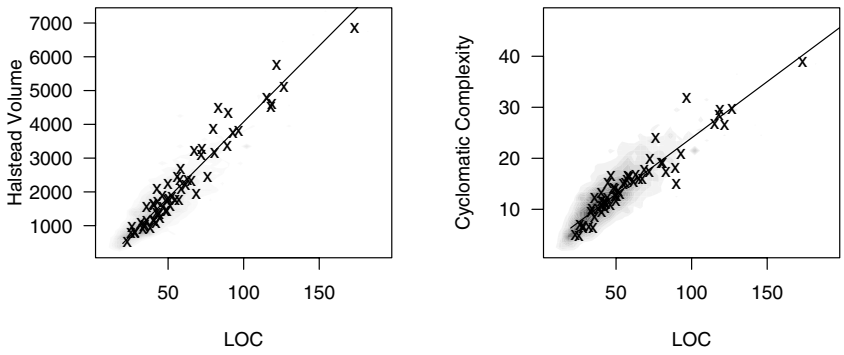
We determine the Cyclomatic Complexity as follows. We first measure the CC of the main body of the program and the subroutines and functions separately. We determine the CC of the entire program by summing the CCs of the constituent parts.

The Cyclomatic Complexity appears to have a broad range for every specification, Figure 1(d) gives it for the correct programs for the specification "Unidirectional TSP". The CC goes well beyond 50, and many correct programs have a CC above 20. This observation is valid for all specifications.

3.2 Dependability metrics

We also measured two external software metrics related to dependability: the Probability of Failure on Demand (PFD) and the number of defects (D).

To determine the PFD, we used three different testing strategies. For some specifications, a complete test is possible. For other specifications this is not the case, we then completely tested part of the demand space or we did a random test. The number of demands is either 2,500 or 10,000 for all specifications, except for those for which we did a complete test, in those cases the number of demands is equal to the number of possible demands.



(a) Halstead Volume (b) Cyclomatic Complexity

Figure 3. Density plots of the LOC/Volume and LOC/CC distribution of all correct programs of all specifications combined. The x's are the means for the 59 specifications. The lines are unweighted regressions of these means. The background of the figure shows a density plot of the Volume-LOC and CC-LOC pairs of all the programs involved, with every specification having the same weight.

The PFD is the fraction of demands for which the program fails, i.e. we assume that every demand is equiprobable. This assumption seems to be as good as any other assumption, and we do not have any indication that it influences our results.

We determined the number of defects as follows. In our approach this is only possible for those authors who manage to submit a correct program (we disregard all submissions after a first correct submission). We assumed that the number of defects in this correct submission is zero, and in the penultimate submission is one. Then, we assess the submission before the penultimate submission of this author; if its behaviour is different from that of the penultimate submission, we add one to the defect count, otherwise we assume the defect count is the same. We repeat this procedure until the first submission of the author. The defect count of this submission is used for the analysis.

4 Correlations between internal software metrics

For every specification, we determined the correlation between the internal software metrics. This gives us 59 measurements for every pair of metrics. These are presented in histograms in Figure 2.

There are a some very strong correlations: CC vs. LOC (mean=0.78), Halstead Volume vs. CC (0.75), Halstead Volume vs. LOC (0.82).

The correlations between Comment Lines and LOC/Volume/CC (mean=0.29, 0.28 and 0.23) are rather unexpected. They might be explained by assuming that programmers who write comments, also tend to write more elaborate code. Writing comments in code for the Online Judge is completely voluntary.

5 Lines of Code vs. Halstead Volume and Cyclomatic Complexity

The correlations in Figure 2 suggest a very strong relationship between LOC on the one hand, and Volume/CC on the other. We would like to investigate this a little bit further.

We plot the mean Volume and CC of all the specifications against the mean LOC, see Figure 3. In both cases, the correlation is very strong (0.97 and 0.95). For the means, we determined the following regression lines:

$$Volume = 45 \times LOC - 428 \quad (1)$$

$$CC = 0.22 \times LOC + 1.9 \quad (2)$$

For CC this can informally be interpreted as: on average, programmers write a branch in almost every five lines of C/C++ code.

6 Are internal software metrics different for incorrect programs?

To discover whether there is a relationship of any of the internal metrics with correctness of program, we determined the ratio between the means of the internal metrics of the correct programs and those of the incorrect programs for every specification. This gives us 59 measurements for the

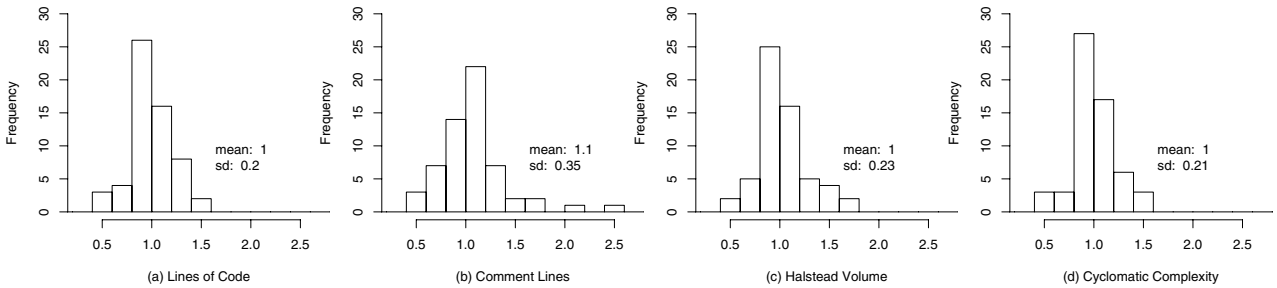


Figure 4. Histograms of the ratio of internal metrics of correct programs and incorrect programs for 59 specifications, for (a) LOC, (b) Comment Lines, (c) Halstead Volume, (d) Cyclomatic Complexity.

four internal metrics, which we depict in histograms in Figure 4. It appears that although the change in internal metrics can be large for some specifications, these changes go either way, and on average the internal metrics are practically the same for correct and incorrect programs.

Number of Comment Lines. There is no requirement to insert comments in the code submitted. Beforehand, we conjectured that the number of comment lines in submissions would correlate with lower PFD (also reported by Runeson [7]), the rationale being that someone who voluntarily adds comments, thinks about the program more carefully. However, only a small change in this metric was found, correct programs have on average 10% more comment lines; the spread on the distribution is very large.

Halstead Volume and Cyclomatic Complexity. On average, the Volume and the CC of correct and incorrect programs are the same.

7 Correlations between internal metrics and dependability

We determined the correlation between the internal software metrics and the two dependability software metrics. We depict this information in eight histograms in Figure 5.

Lines of Code. Figure 5 shows that in our experiment the mean correlation between LOC and number of defects or PFD is close to zero.

Number of Comment Lines. Figure 5 shows that the average correlation of the number of comment lines with the number of defects and PFD is very close to zero. This adds to our finding in Section 6: although correct programs have slightly more comment lines, it now appears that the *number* of comment lines is not correlated to the *number* of defects.

Cyclomatic Complexity and Halstead Volume. In our experiments, there is no correlation between these metrics and PFD nor number of defects.

8 Discussion

This research suffers from one major point of criticism: the programs are not 'real', they are most probably written by students and not by professional programmers. Even so, we argue that if there is a relationship between one of these internal software metrics and PFD and number of defects, this relationship would even be stronger if less rigorous programming methods are followed. A higher Cyclomatic Complexity is more problematic in a 'trial and error program' then in a rigorously developed program and becomes irrelevant if the correctness of a program is formally proved.

Another point of criticism is the size of the programs, varying between several dozens and several hundreds of lines of code. This is comparable to the size of subroutines, and the results should be interpreted at that level. We can only conjecture that these results are applicable to bigger, more realistic programs.

In our approach to counting defects, we assume that programs of the same author that have the same behaviour have the same number of defects and that the number of defects removed in one step is one. Both assumptions most probably lead to underreporting of defects. A possible other way to measure the number of defects is to simply use the number of attempts to the first correct submission. This approach would probably lead to overreporting, assuming that authors on average remove less than one fault per attempt. We also tried this approach, and the results are virtually the same, and for that reason we do not publish these.

9 Conclusion

We analysed the relations between various internal and external software metrics in a large collection of C/C++ programs submitted to a programming competition, the Online Judge.

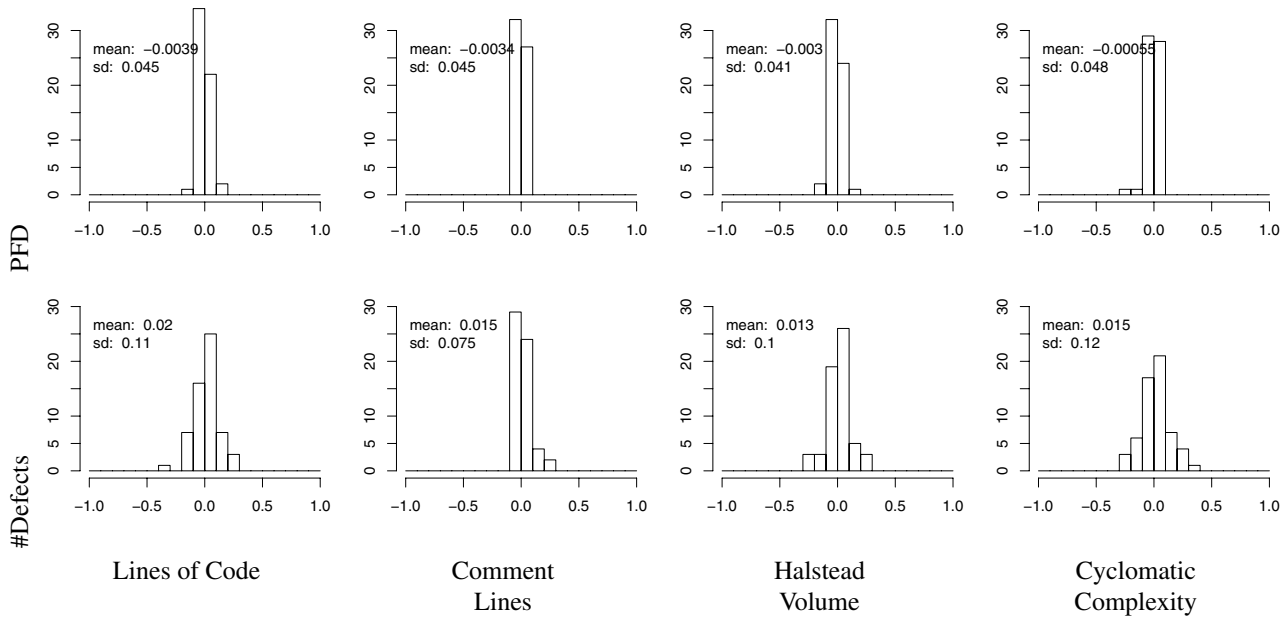


Figure 5. Histograms of the correlations between various internal software metrics and PFD or Number of Defects. The vertical axis reflects the number of specifications.

The experiment shows very strong correlations between the internal software metrics Lines of Code (LOC), Halstead Volume (V) and Cyclomatic Complexity (CC). We derived the following relations between the means of their distributions: $V = 45 \times LOC - 428$ and $CC = 0.22 \times LOC + 1.9$. These give the best estimates for V and CC when LOC is given.

In the experiment, the internal software metrics have no predictive power with respect to the Probability of Failure on Demand nor the number of defects of programs. The internal metrics are on average the same for correct and incorrect programs.

Acknowledgement

This work was supported in part by the U.K. Engineering and Physical Sciences Research Council via the Interdisciplinary Research Collaboration on the Dependability of Computer Based Systems (DIRC), and via the Diversity with Off-The-Shelf Components (DOTS) project, GR/N23912.

References

[1] N.E. Fenton and M. Neill. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, SE-25(5):675–89, 1999.

[2] N.E. Fenton and M. Neill. Software metrics: Roadmap. In *Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland*, pages 357–70, 2000.

[3] N.E. Fenton and S.L. Pfleeger. *Software Metrics: A Rigourous and Practical Approach*. International Thomson Computer Press, 1996.

[4] M.H. Halstead. *Elements of Software Science*. Elsevier North-Holland, New York, 1977.

[5] D. Kafura and J. Canning. A validation of software metrics using many metrics and two resources. In *Proceedings 8th International Conference on Software Engineering, London*, pages 378–85, 1985.

[6] T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4), December 1976.

[7] P. Runeson, M. Holmstedt Jöhnsson, and F. Scheja. Are found defects an indicator of software correctness? an investigation in a controlled case study. In *The 15th IEEE International Symposium of Software Reliability Engineering, 2–5 November 2004, St. Malo, France*, pages 91–100, 2004.

[8] S. Skiena and M. Revilla. *Programming Challenges*. Springer Verlag, March 2003.