Hybrid metric to reduce labour costs in assessing the complexity of the programs

V V Kukartsev[1,2], V S Tynchenko[1,2], A A Boyko[1,2], M A Denisov[1], V A Kukartsev[1], E A Chzhan[1]

[1] Siberian Federal University, 79, Svobodny pr., Krasnoyarsk, 660041, Russia
[2] Reshetnev Siberian State University of Science and Technology, 31, Krasnoyarsky Rabochy Av., 660037 Krasnoyarsk, Russia

E-mail: vlad_saa_2000@mail.ru

Abstract. The article deals with the problem of assessing the complexity of software products. A review of widely used metrics (number of lines of code, average number of lines for functions, Halstead's metrics, Jilb's metrics, ABC-metric) is being conducted, on the basis of which a conclusion is drawn about their deficiencies. Also, indicators are analyzed, subtracted when assessing the complexity of programs, such as: program saturation with conditional operators or cycle operators, complexity of program understanding, program coding complexity, program content. A hybrid metric based on a composition of already well-known ones is proposed. As a result of the experiment to assess the complexity of the program, it is shown that the proposed hybrid metric allows us to more effectively evaluate the program. The use of such a metric can significantly reduce the computation time.

1. Introduction

One of the basic rules of programming is simplicity [1 - 3]. How to estimate the complexity of programs developed by programmers? To develop your own metrics, you should consider the existing ones [4]. There are different approaches to assessing the complexity of programs (metrics) [3, 5, 6]. Software metric is a measure that allows you to obtain a numerical value of a property of the software or its specifications [7-10].

There are the following groups of metrics: quantitative metrics, complexity metrics of the data flow program, software complexity metrics of the program's control flow, object-oriented metrics and the reliability metrics, the combined complexity metrics of control and data, hybrid metric [1, 5, 7, 11].

The set of metrics used includes:

- Order of growth.
- Number of lines of code.
- Functional point analysis.
- Cyclomatic complexity.
- Number of errors per 1000 lines of code.
- Requirements coverage.
- Degree of code coverage by testing.
- Number of classes and interfaces.
- Connectivity.

2. Description of existing metrics

Let us consider quantitative metrics. Quantitative characteristics of programs are simple, so they are usually considered first. [1, 3, 8]

Number of lines of code. Lines are divided into physical and logical (commands, operators). This measure does not make a special contribution to the analysis of the complexity of the programs. To build a complexity profile, a variation of this metric can be used, such as the number of instructions in the base block - the smaller the base blocks, the "thicker" the transfer of control within the code fragment. In order for this measure to increase with increasing complexity, it is necessary to take the inverse value.

Average number of lines for functions (classes, modules, files). The possible variation is the average size of the base block in the function (to estimate the complexity of the measure increase - the inverse value).

Halstead's metrics allow different number of rows and operators partially consider the possibility of recording the same functionality. They include a large number of quantitative indicators: the number of unique operands of the program, the number of unique operators of the program, the total number of operands, the total number of operators, the theoretical number of unique operands and the theoretical number of unique operators. Due to these indicators, the complexity of understanding the program is determined, the level of the quality of programming, the level of language expression, the complexity of the coding program, the information content of the program (the mental cost of creating a program), the assessment of intellectual effort in the development of the program (the number of required basic solutions in writing the program).

Jilb's metrics show the complexity of the software based on the intensity of the program conditional operators or loop operators. cl=CL/n, where CL is absolute complexity (number of control operators), n is the total number of program operators. The metric reflects the complexity of understanding the program and the complexity of the development. When you add an indicator of the maximum nesting level of conditional operators and cycles, the effectiveness of this metric increases significantly.

ABC-metric is based on the calculation of variable assignments, explicit transfers of control beyond the scope (function calls) and logical checks. Example of a measure view: ABC = <9, 5, 3> (three values). To estimate the complexity of the program, we calculate the number - the square root from the sum of squares A, B, C The metric is visual when visualizing (vector in three-dimensional space) and can be calculated for different code fragments. One of the disadvantages is that ABC-metric can have zero value for some non-empty program units.

3. Development of complexity metrics of program

Having considered the existing metrics to estimate the complexity of programs, we can conclude that there is no universal metric. Individually, each of the above metrics will not give an accurate answer to the question of program complexity, and therefore it is possible to create "hybrid" metrics. "Hybrid" metrics include a number of simple metrics to get the desired result. To better estimate the complexity of the programs among the quantitative metrics we propose to combine Jilb's metrics and part of the Halstead's metrics.

These metrics include appropriate formulas for solving the problem of assessing the complexity of programs. For clarity, we present the necessary formulas in tabular form (Table 1).

Used variables in formulas (Table 1):

- CL - absolute complexity (number of control operators), n-total number of program operators.
- V=N * log2n - volume of the program.
- L'= (2 n2)/(n1*N2) - the level of programming quality, based only on the parameters of the real program without taking into account the theoretical parameters.
- N=N1+N2 - the length of the program.
- N1 - total number of operators in the program.
- N2 - total number of operands in the program, n=n1+n2-program dictionary.
- n1 - number of unique operators in the program, including delimiter characters, the names of the procedures and operation signs (a dictionary of operators).

- n2 - number of unique operands of the program (dictionary of operands).

Table 1. Formulas used in the "hybrid" metric.

Formula	Description
G = (cl + EC + D + I) / 4	General formula
cl=CL/n	Software complexity on the basis of saturation of the program
	conditional operators or loop operators
EC=V/(L')2	The complexity of understanding a program
D=1 / L'	The complexity of the program coding
I=V/D	Information content of the program

Software complexity on the basis of saturation of the program conditional operators or loop operators. This indicator provides information about the relative content of conditional operators such as if - then - else and loop operators (a language construct that determines the iteration of some execution sequence in the program). It should be noted that the actual record of conditions and loops in different programming languages can be presented in different forms while maintaining the specified meaning of operators.

The complexity of understanding the program. This indicator represents the relative simplicity of the program, taking into account its "overload". Some programs contain variables that do not increase the complexity of the program, but make it difficult to understand.

The complexity of the program coding shows how much work a programmer needs to create a program (writing code, scripts, in order to implement a certain algorithm in a particular programming language).

Information content of the program. This feature allows you to determine the mental costs of creating a program.

4. Results

Let us compare the calculation results of the following metrics: the number of lines of code, the average number of lines for functions, ABC metric, and hybrid metric. For the evaluation took part of the code on the Internet. The results are presented in Tables 2 and 3 respectively.

Table 2. Results based on metrics that are not part of a "hybrid" metric.

Formula	Calculation	Result
N (number of lines of code)	60	60
n (average number of lines for functions)	60/3	20
ABC	square root of the sum <9, 5, 2>	4

Based on the information presented in Tables 2 and 3, it can be concluded that the metrics presented in table 2 did not give us any useful information in relation to the evaluation of the complexity of the programs. This metric indicated only on the number of lines and the counting of assignments of values to variables, and not on the part of the program.

It can be concluded that separately these metrics should not be taken into account when assessing the complexity of programs, which can’t be said about the "hybrid" metric.

Table 3. Results based on the "hybrid" metric.

Formula	Calculation	Result
cl=CL/n	cl=6/263	0.0228
EC=V/(L')2	EC=((263 + 55) *log2(263))/((2 * 50)/(200 * 55)) 2	1.4
D=1 / L'	D=1/ 0.0182	0.5
I=V/D	I=V/D	0.6
G = (cl + EC + D + I) / 4	(0,0228 + 1,4 + 0,5 + 0,6) / 4	0.6

The results of the "hybrid" metric are following. The complexity of the software based on the saturation of the program conditional operators or loop operators is 0.0228, which indicates that the source code has a low complexity (263 text operators account for only 6 operators of conditions). The complexity of understanding the program - the average value of the result, which indicates the "simplicity" of the program, i.e. the code is not overloaded. The complexity of coding the program is 0.5, which indicates that a lot of labor programmer is not required to write code. The information content of the program is 0.6, which shows a small requirement of mental costs.

The overall complexity of a program is equal to 0.6. This indicator indicates the relative simplicity of the program.

5. Conclusion

The advantage of merging these metrics is that it covers the necessary aspects to assess the complexity of the programs from the available quantitative metrics, while there is a simplicity of calculations. These metrics can also be combined with metrics from other groups that are valuable when merged.

The "hybrid" metric can be used in any project where a software product is being developed. This metric allows to reduce labor costs by reducing the time that would be used to calculate all quantitative metrics that contain less important information to assess the complexity of programs.

References

[1] Plaskovitsky V A 2013 The use of software metrics to assess the complexity of executable code *Works BSTU. Series 3: Physics and mathematics and computer science* **6** 145-148

[2] Mironenko A N 2015 Using the metric characteristics of software code to identify plagiarism *Bulletin of Ugra State University* **S2(37)** 26-27

[3] Kafura D and Reddy G R 1987 The use of software complexity metrics in software maintenance *IEEE Transactions on Software Engineering* **3** 335-343

[4] Faidhi J A and Robinson S K 1987 An empirical approach for detecting program similarity and plagiarism within a university programming environment *Computers & Education* **11(1)** 11-19

[5] Diasamidze S V 2012 Method and methodology for identifying undeclared features of programs using structured complexity metrics *Proceedings of the Petersburg University of Communications* **3(32)** 117-122

[6] Milov A V, Tynchenko V S, Kukartsev V V, Tynchenko V V and Antamoshkin O A 2018 Classification of non-normative errors in measuring instruments based on data mining *Advances in Engineering Research* **158** 432-437

[7] Varenitsa V V 2011 Problems of calculating software complexity metrics during code security audit using manual review method *Bulletin of the Bauman Moscow State Technical University. Instrument making series* **S1** 79-84

[8] Berg O Yu 2005 Software quality assessment metrics *Proceedings of the International Symposium "Reliability and Quality"* **1** 321-322

[9] Bukhtoyarov V V, Tynchenko V S, Petrovsky E A, Kukartsev V V and Kuklina A I 2018 Evolutionary method for automated design of models of the vortex flowmeters transformation

function *Journal of Physics: Conf. Series* **1118** 012041.
[10] Kornienko A A and Diasamidze S V 2016 An approach to structuring complexity metrics to identify software security defects *Proceedings of the Petersburg University of Transport* **3 (48)** 421-429
[11] Avetisyan A I, Belevantsev A A and Chuklyaev I I 2014 Technologies of static and dynamic software vulnerability analysis *Cybersecurity Issues* **3(4)** 20-28