ALGORITHMIZATION AND REALIZATION OF THE SOFTWARE TOOL FOR THE SOFTWARE CODE QUALITY ASSESSMENT

Irakli Basheleishvili, Sergo Tsiramua, Avtandil Bardavelidze*

[1,3] Akaki Tsereteli State University, Kutaisi
[2] University of Georgia, Tbilisi
Georgia

* Corresponding Author: e-mail: basheleishvili.irakli@gmail.com

Abstract: The paper deals with the development of software for the program code quality assessment, which is based on the software code quality assessment metrics and multi-criteria solution analysis method. The software presented in the paper allows us to assess the quality of software code written in C, C ++, C # and Java programming languages, as well as, if necessary (when we have several software solutions for one task and want to choose the best among them), to rank the program source files based on quality quantitative indicators.

Key words: program code, quality, assessment, metrics.

1. INTRODUCTION

The rapid development that has taken place in information technology increases demand for its use in virtually all spheres of human activity. What is more, it is impossible today to conceive of them as being functioning effectively without using modern information technology. The rapid development of information technology and computing processes in recent decades has led to the existence of software in all areas of human activity. This leads to the increasingly high requirements for software quality. Software quality is a combination of the characteristics of the computer software product and their meanings that relate to the possibility to use it to meet established or expected requirements [2]. Quality in software means that there are no errors therein [1, 3]. Software errors can cause heavy material damage, so research in software quality management is very important today.

Software quality indicators can generally be divided into two parts: internal and external quality indicators. Software code quality is an internal quality indicator that is considered to be one of the most important parts of software quality management [3].

Based on the above, the aim of our paper is to develop software for program source-code quality assessment software using the metrics and multi-criteria solution analysis methods. This will allow us to quantify the quality of the source code written in C, C++, C# and Java programming languages, as well as to rank the source code files of the program when we have several software solutions for one task (Source Codes) and select the best quality ones among them. Such an approach to solving the set task is a research novelty, the assessment of quantitative quality of the software code is integrated in the current programs' integrated development environments, but they compute the values only by using some metrics of the software code and do not help us in decision making. In particular, when we have several software implementations of one particular problem and we want to choose the best one based on their quantitative assessment.

The software that we propose offers a set of tools that can be used to assess the quality of the source code files of the program and rank it, if necessary, which will help us make decision on the best choice. Which is a novelty of the study.

2. MAIN PART

2.1. Research methodology

The research methodology presented in the paper involves developing the algorithms and a software tool on their basis that analyzes software code with a view to determining its quality assessment metrics (number of codelines, cyclomatic complexity, Halstead complexity measures, and maintainability index). The methodology also involves developing the source code file ranking algorithm based on the multi-criteria solution analysis method -Topsis [7, 8]. The program source code file ranking task can be represented as a multi-criteria solution task, in which alternatives are the program source code files to be ranked, while the assessment criteria are as follows: Lines of Code, Cyclomatic complexity, Program vocabulary, Program length, Calculated estimated program length, Volume of Code, Software Difficulty, Software Effort, Time to Write Code and Estimated Number of Delivered Bugs[8, 9].

2.2. Code metrics and their definition algorithms

Metrics are the tools that are aimed to simplify the software quality assessment decision-making process, increase productivity and responsibility levels, based on the methods of collection, processing, and reflection of data related to issues of problem statement and solving. There are some common metrics listed below [3, 6]:

- **The number of the code lines** - is used to determine the amount of original text in a program based on its number of lines. This indicator is used to predict program development costs in a specific programming language or to assess labor productivity after program development.

To count the number of code lines, we use a very simple algorithm, which in fact is counting of all lines, in addition to empty lines and the commentary lines in the source code file.

❖ **Cyclomatic complexity** - is a software metric used to measure the complexity of a program or its topological measurer. The initial code of the program measures the number of linear, independent paths from beginning to end. For example, if the program does not contain a cycle or conditional branch, then the cyclomatic complexity is equal to 1. If the program contains one if/else block then the complexity is equal to 2 and so on.

Cyclomatic complexity can also be calculated within the program for the individual functions, modules, methods, or classes. Cyclomatic complexity of a program is determined by means of an oriented graph whose picks are the program blocks, with joined ribs, if the control can be moved from one block to another. . In this case, the complexity is determined by the formula:

$$\text{Cyclomatic complexity} = E - N + 2*P \quad (1)$$

where: E = the number of edges in the control flow graph; N = the number of nodes in the control flow graph; P = the number of connected components.

The algorithm for determining cyclomatic complexity of a source file includes the following steps:

Step 1. Analyze the source code file to identify cyclic and branch operators.

Step 2. Build an oriented graph for the source code file.

Step 3. Find the number of ribs and peaks in the graph.

Step 4. Determine the number of connection components.

Step 5. Compute cyclomatic complexity.

❖ **Halstead complexity measures -** this is a preliminary assessment of the complexity of software implementation at the design stage. It is one of the static and analytical methods of measuring software complexity, introduced by M. Halstead in 1977 [3]. His concept was to create an empirical science of software development. He found that software metrics should reflect implementation or representation of algorithms in different languages, but should be independent of their performance on a particular platform. These metrics, therefore, are computed statically from the code. The aim of Halstead was to determine the measurable properties of software and the relationships between them. Halstead metrics are based on the assumption that parts of an executable program consist of operators and operands. For example, variables and constants are treated as operands; keywords, logical and comparative operators, etc. as operators. We need to find the following basic dimensions for each program: $\eta 1$ = the number of distinct operators, $\eta 2$ = the number of distinct operands; N1 = the total number of operators; N2 = the total number of operands. From these numbers, several measures can be computed:

$$\text{Program vocabulary: } \eta = \eta 1 + \eta 2 \quad (2)$$

$$\text{Program length: } N = N1 + N2 \quad (3)$$

$$\text{Calculated estimated program length: } L = \eta 1 * \log2(\eta 1) + \eta 2 * \log2(\eta 2) \quad (4)$$

Volume of Code: $V=N * \log2(\eta)$ (5)

Software Difficulty: $D = (\eta1/2) * (N2 / \eta2)$ (6)

Software Effort: $E = D*V$ (7)

Time to Write Code: $T = (E/18)$ (8)

Estimated Number of Delivered Bugs: $B = V/3000$ (9)

The algorithm for determining Halstead complexity of the program's source code file includes the following steps:

Step 1. Analyze the source code file to determine the set of distinct operators;

Step 2. Analyze the source code file to determine the set of distinct operands;

Step 3. Determine the frequency for a single element of the set of distinct operators, or how many times it is found in the program code;

Step 4. Determine the frequency for a single element of the set of distinct operands, or how many times it is found in the program code;

Step 5. Compute the value of Holsted complexity.

❖ **Maintainability Index -** is a complex indicator of code quality. It is defined by the formula [4]:

$$MI = MAX (0, (171 - 5.2*\ln (V) - 0.23*CC - 16.2*\ln (LC))*100/171) \quad (10)$$

where: V – Halstead Volume, computational complexity. The value of metric increases in direct proportion to the number of operators used; CC – Cyclomatic Complexity. Structural complexity of the code or the number of different branches in the code. The higher the value, the more tests to be planned; LC – Number of code lines.

2.3. Example of program code evaluation

Evaluate the software code (which implements the Rabin-Karp algorithm) below according to quantitative metrics. The software code is written in the C++ programming language.

```
1. void search(string pattern, string text){
2.     int q = 100;
3.     const int d = 28;
4.     int pl = pattern.length();
5.     int tl = text.length();
6.     int i, j, hash_v_p = 0, hash_v_t = 0, h = 1;
7.     for (i = 0; i < pl - 1; i++){
8.         h = (h * d) % q;
9.     }
10.    for (i = 0; i < pl; i++){
11.        hash_v_p = (d * hash_v_p + pattern[i]) % q;
12.        hash_v_t = (d * hash_v_t + text[i]) % q;
13.    }
14.    for (i = 0; i <= tl - pl; i++){
15.        if (hash_v_p == hash_v_t){
16.            for (j = 0; j < pl; j++){
```

```
17.                         if (text[i + j] != pattern[j]){
18.                             break;
19.                         }
20.                     }
21.                     if (j == pl)
22.                         cout << "Find index: " << i << endl;
23.                 }
24.                 if (i < tl - pl){
25.                     hash_v_t = (d*(hash_v_t - text[i] * h) + text[i + pl]) % q;
26.                     if (hash_v_t < 0)
27.                         hash_v_t = hash_v_t + q;
28.                 }
29.     }
30. }
```

To measure the complexity of cyclomatic, build an oriented graph for a given program code. The oriented graph is represented in Fig.1 below:

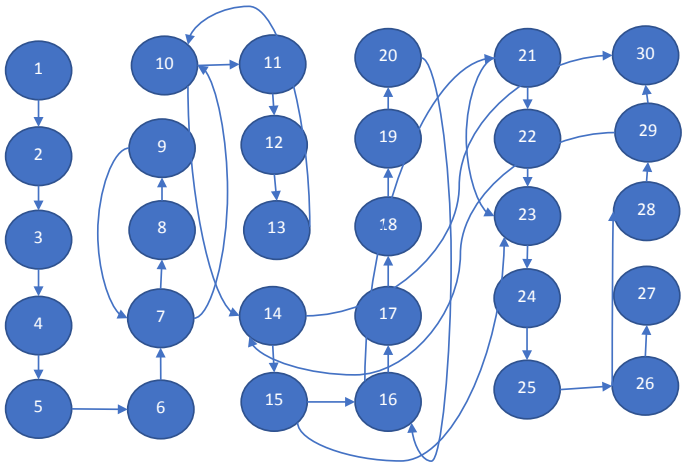


Fig. 1. Control flow graph

According to the given oriented graph Cyclomatic complexity=13.
Identify the operators, operands, and their frequencies in the given software code to determine the Halstead complexity measures (Table 1).

Program vocabulary - 26+16=42
Program length - 98+84=182
Calculated estimated program length - 26*log2(26) +16*log2(16)=186.2114
Volume of Code - 182*log2(42)=981.4018
Software Difficulty - (26/2)*(84/16)=68.25
Software Effort - 8.25*981.4018= 66980.67
Time to Write Code - 66980.67/18=3721.148
Estimated Number of Delivered Bugs - 981.4018/3000=0.327

Table 1 Operators and operands

Operators	Frequency	Operators	Frequency	Operands	Frequency
Void	1	[ ]	6	search	1
String	2	{ }	8	pattern	4
,	5	<=	1	text	6
( )	14	==	2	q	6
Int	5	If	5	100	1
=	2	!=	1	pl	8
const	1	break	1	tl	3
;	12	Cout	1	i	16
length	2	<<	2	j	7
.	2	Endl	1	hash_v_p	4
<	5	-	-	hash_v_t	9
-	4	-	-	h	4
++	4	-	-	0	7
For	4	-	-	28	1
*	5	-	-	1	2
%	3	-	-	d	5
		η1=26	N1=98	η2=16	N2=84

Once we have defined the number of the code lines, cyclomatic complexity and Halstead complexity measures, we can calculate the maintainability index:

MI = MAX (0, (171 - 5.2 * 6.889 - 0.23 * 13 - 16.2 * 3.401)) * 100 / 171) =45.

The values of the evaluation measures for a given program code are given in the Table 2.

Table 2. Evaluation results

#	Measure	Value
1	number of the code lines	30
2	Cyclomatic complexity	13
3	Program vocabulary	42
4	Program length	182
5	Calculated estimated program length	186.2114
6	Volume of Code	981.4018
7	Software Difficulty	68.25
8	Software Effort	66980.67
9	Time to Write Code	3721.148
10	Estimated Number of Delivered Bugs	0.327
11	maintainability index	45

3. SOFTWARE

The software presented in the paper was developed as a desktop application on the .NET platform, with a simple and flexible user interface. The software implemented using object-oriented programming paradigm. The program has a

database-based structure. The software database developed on a MYSQL database server. The main functionality of the program includes the following: Manage source code files(add, edit and delete); Source code file analysis to determine the number of lines of code, cyclomatic complexity, Halstead complexity measures and maintainability Index; ranking the source code files for the best selection.

The program flowchart is shown in Fig. 2 below:

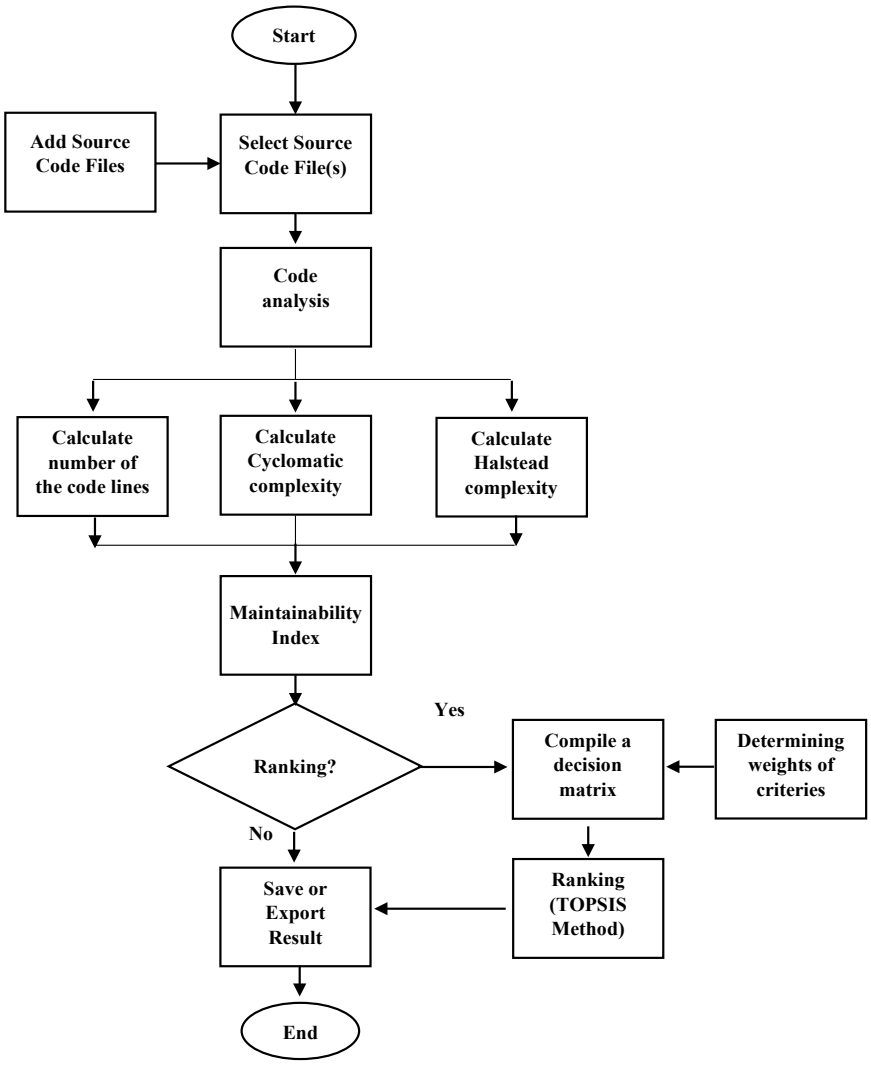


Fig. 2. Program flowchart

The program allows us to input and store the source code files for which we want to assess the quality of the code (Fig. 3).

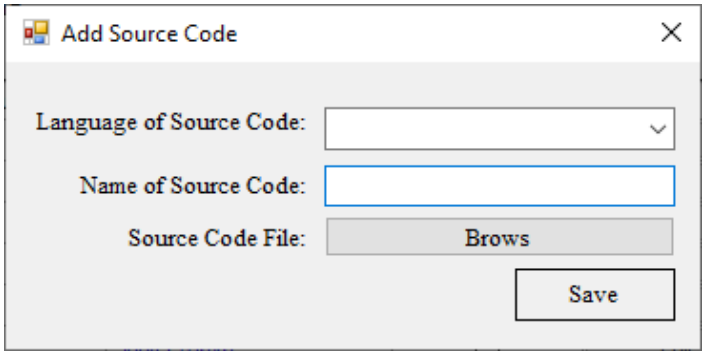


Fig. 3. Add Source Code

The image below shows (Fig. 4) the program window, which contains the source code files included in the program. To assess the quality of code for a specific code file, you need to select the appropriate file and click the "Measurements" button on the toolbar.

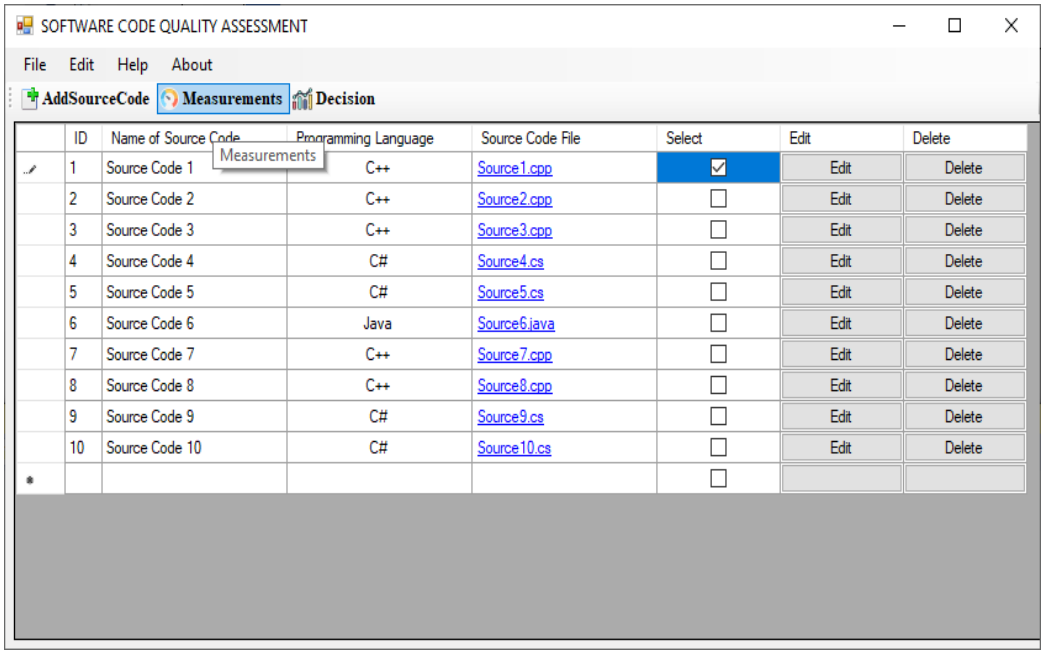
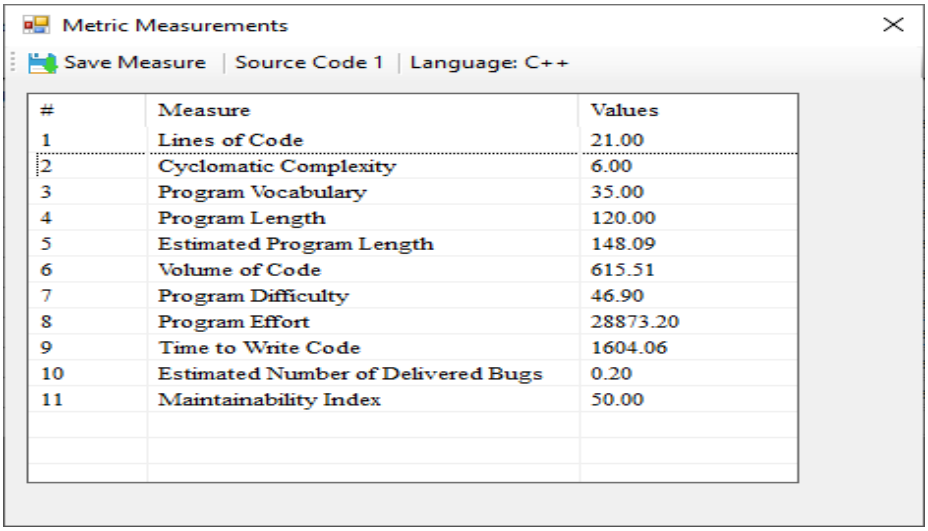


Fig. 4. Main Form

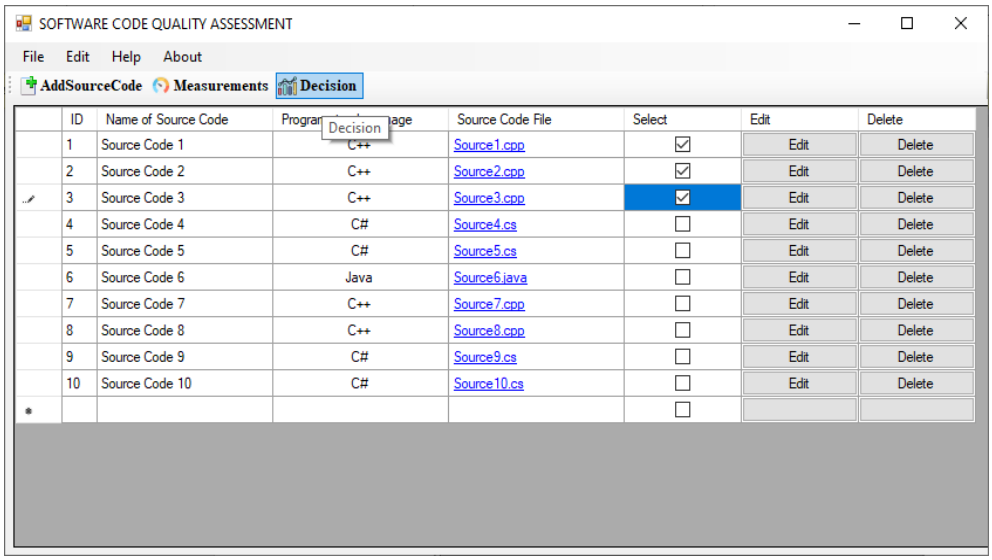
Clicking the "Measurements" button brings up a new window with quantitative indicators for assessing the quality of the selected file code (Fig. 5).



#	Measure	Values
1	Lines of Code	21.00
2	Cyclomatic Complexity	6.00
3	Program Vocabulary	35.00
4	Program Length	120.00
5	Estimated Program Length	148.09
6	Volume of Code	615.51
7	Program Difficulty	46.90
8	Program Effort	28873.20
9	Time to Write Code	1604.06
10	Estimated Number of Delivered Bugs	0.20
11	Maintainability Index	50.00

Fig. 5. Metric Measurements

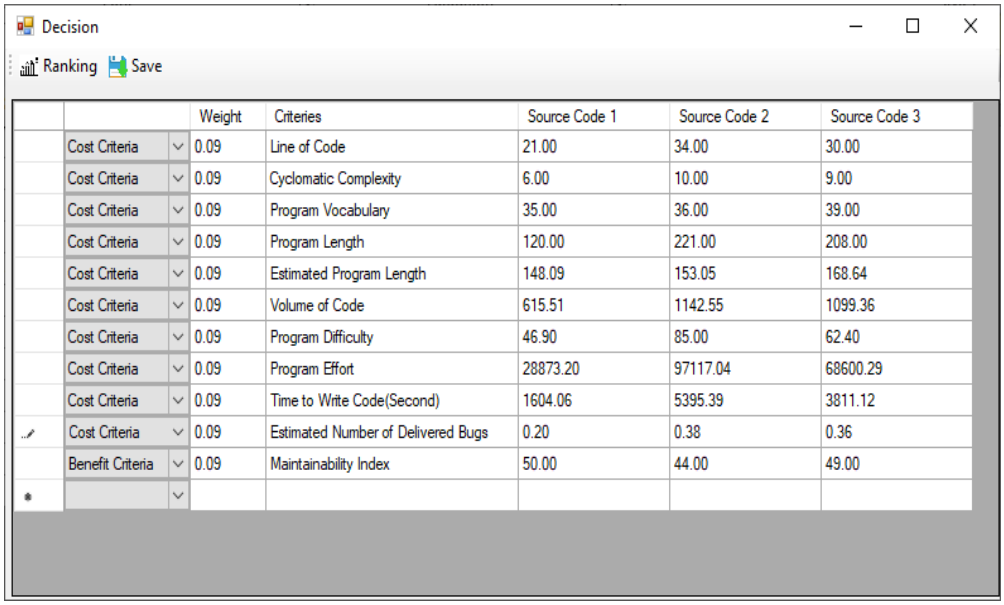
In case we have several programs for solving one task and we want to select the best program code according to the quality indicators, first it is necessary to upload these files to the program, then mark and click the "Ranking" button on the toolbar (Fig. 6). In our case such files are: the Source Code 1, Source Code 2 and Source Code 3, which are written in the programming language C ++, and they implement different versions of binary search in the array.



ID	Name of Source Code	Program Language	Source Code File	Select	Edit	Delete
1	Source Code 1	C++	Source1.cpp	☑	Edit	Delete
2	Source Code 2	C++	Source2.cpp	☑	Edit	Delete
3	Source Code 3	C++	Source3.cpp	☑	Edit	Delete
4	Source Code 4	C#	Source4.cs	☐	Edit	Delete
5	Source Code 5	C#	Source5.cs	☐	Edit	Delete
6	Source Code 6	Java	Source6.java	☐	Edit	Delete
7	Source Code 7	C++	Source7.cpp	☐	Edit	Delete
8	Source Code 8	C++	Source8.cpp	☐	Edit	Delete
9	Source Code 9	C#	Source9.cs	☐	Edit	Delete
10	Source Code 10	C#	Source10.cs	☐	Edit	Delete

Fig. 6. Ranking

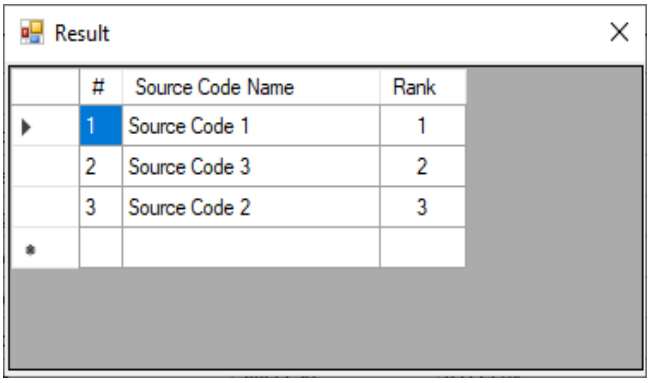
Clicking the “Decision” button brings up a window representing the resolution matrix (Fig. 7). For the assessment criteria by the implication principle, there have been defined the equal weights, which can be changed, if necessary, given that their sum must be equal to one. In the window that appears, we need to define the type of separate assessment criterion: Benefit or Cost.



	Weight	Criteries	Source Code 1	Source Code 2	Source Code 3
Cost Criteria	0.09	Line of Code	21.00	34.00	30.00
Cost Criteria	0.09	Cyclomatic Complexity	6.00	10.00	9.00
Cost Criteria	0.09	Program Vocabulary	35.00	36.00	39.00
Cost Criteria	0.09	Program Length	120.00	221.00	208.00
Cost Criteria	0.09	Estimated Program Length	148.09	153.05	168.64
Cost Criteria	0.09	Volume of Code	615.51	1142.55	1099.36
Cost Criteria	0.09	Program Difficulty	46.90	85.00	62.40
Cost Criteria	0.09	Program Effort	28873.20	97117.04	68600.29
Cost Criteria	0.09	Time to Write Code(Second)	1604.06	5395.39	3811.12
Cost Criteria	0.09	Estimated Number of Delivered Bugs	0.20	0.38	0.36
Benefit Criteria	0.09	Maintainability Index	50.00	44.00	49.00

Fig. 7. Decision Matrix

By clicking the "Ranking" button on the toolbar(Fig. 8), we can rank the alternatives using the TOPSIS method and get a ranked list of alternatives.



#	Source Code Name	Rank
1	Source Code 1	1
2	Source Code 3	2
3	Source Code 2	3

Fig. 8. Ranking result

4. CONCLUSION

As result of the research presented in the paper, the program quality evaluation software has been developed, which is a tool that can be used to assess the quality of the software code written in C, C ++, C # and Java programming languages, as well as rank the program source code files based on quality indicators. This tool will help us to analyze the quality of the program code and improve it. The presented program works at the level of the program source code file, it is planned that the program will be updated and it will work at the program project level.

REFERENCES

[1] G. Daniel. *Software quality: concepts and practice*. ISBN: 978-1-119-13449-7, John Wiley & Sons, 2018(720 p).

[2] S. Dalla Palma, D. Nucci, D. Palomba, F. & Tamburri, D. A. Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software*, ISSN: 110726, Vol.70, 2018, pp. 8, https://doi.org/10.1016/j.jss.2020.110726

[3] K. Gill, C. Kemerer. Cyclomatic complexity density and software maintenance productivity. *IEEE transactions on software engineering.* Vol.17, No.12, 1991,pp. 1284-1288.

[4] G. Chogovadze, G. Surguladze, M. Gulitashvili and S. Dolidze, *Software application quality management: testing and optimization*. ISBN 978-9941-8-0629-2, Georgian Technical University, Georgia 2020 (pp. 366).

[5]Kan, Stephen H. *Metrics and models in software quality engineering*. ISBN: 0-201-72915-6, Addison-Wesley Professional, 2003 (250 p.).

[6] Yu, Sheng, and Shijie Zhou. A survey on metric of software complexity. In 2010 *2nd IEEE International conference on information management and engineering*. 2010, pp. 352-356.

[7] I. Basheleishvili. Developing the expert decision-making algorithm using the methods of multi-criteria analysis. *Cybernetics and Information Technologies*, Vol.20, 2020, pp. 22-29, DOI: 10.2478/cait-2020-0013

[8] I. Basheleishvili, A. Bardavelidze. Designing the Decision-Making Support System for the Assessment and Selection of the University's Academic Staff. *International Journal on Information Technologies & Security*, Vol.11, No.2, 2019, pp. 51-58.

[9] G. Fraser, J.Rojas. *Software testing. In Handbook of Software Engineering*. Springer Nature Switzerland, 2019 (p. 123–192), DOI: 10.1007/978-3-030-00262-6_4

[10] Savchenko, Dmitrii, Timo Hynninen, Ossi Taipale, Kari Smolander, and Jussi Kasurinen. Early warning system for software quality issues using maintenance

metrics. *International Journal on Information Technologies & Security*. Vol. 12, No.4, 2020, pp. 35-46

[11] Ateşoğulları, Dilara, and Alok Mishra. Automation testing tools: a comparative view. *International Journal on Information Technologies & Security*. Vol. 12, No.4, 2020, pp.63-76.

Information about the authors:

Irakli Basheleishvili – Doctor of informatics. Assistant Professor at the Akaki Tsereteli State University of Department of Computer Technology. Area of scientific research: Computer Sciences.

Sergo Tsiramua – Doctor of Technical Sciences, Professor at the University of Georgia of Informatics Department of School of Science and Technology. Area of scientific research: Computer Sciences.

Avtandil Bardavelidze – Doctor of Technical Sciences. Professor at the Akaki Tsereteli State University of Department of Computer Technology. Area of scientific research: Computer Sciences.

Manuscript received on 12 April 2022