УДК 004.051
DOI: 10.17277/vestnik.2016.02.pp.208-216

IMPROVEMENT OF THE PROGRAMS COMPLEXITY EVALUATION METRICS BASED ON THE ANALYSIS OF CONTROL FLOW GRAPH

A. M. Babichev[1], V. E. Podolsky[2], A. N. Gribkov[3], S. S. Tolstykh[4], S. G. Tolstykh[4]

Department "Automated Decision Support Systems" (1); Tambov Regional Centre of New Information Technologies (2), zhelkzhelk@yandex.ru; departments: "Design of Radio and Microprocessor Systems" (3); "Computer Science" (4), TSTU

Keywords: cloud computing; complexity metrics; large-block parallel computing; precision computations; precision-trust arithmetic.

Abstract: The article describes the methodological aspects of the complexity evaluation of large-block cloud computing with enhanced precision. The objective of our work is solving in the cloud of the mathematical simulation problems with special requirements for precision. In particular, we consider complexity evaluation of the programs that solve problems with complex connections between sub-problems in the form of large blocks and the computation time considerably exceeding the time of information transmission between them. We propose a method for improving the complexity evaluation metrics, with the use of control flow graph of the program for designing of optimal performance computing systems that operate in the cloud. The method consists in the use of the complexity of particular operations on the basis of experimental data.

Currently, there is a need for solving problems that require a large amount of nonstandard floating-point calculations based on precise mathematical modeling. Such problems cannot be solved in a single-processor mode as every arithmetic operation and, in particular, standard mathematical functions (sine, exponent, etc.) may require significant time, and while parallelizing there is a tendency of grouping calculations in blocks with computational complexity, that is significantly higher than the complexity of information transmission within a network. A mode of large-block distributed computing appears. To implement large-block computations it is beneficial to use modern cloud technologies to get access not only to the data storage, but also to computing resources. As an example, the transformation of the solution of problems of multidimensional optimization – from classical deterministic options, each of which is quite significant in terms of the calculations, to the consideration of parametric uncertainty, which appears when insufficient knowledge of the object of study. The amount of calculations in such problems increases sharply in comparison with the deterministic settings, and the solution must be found in a multidimensional region whose dimension exceeds the dimension of the original deterministic problem. To solve such problems, of course, it is necessary to organize high-performance parallelization. But with the high dimension of these problems and the complicated topology of the block interaction it becomes more difficult to evaluate the accuracy of the resulting solution, what stimulates the refusal from the standard floating-point arithmetic in favor to the special class libraries with adjustable size of the mantissa [1].

Along with the search for the optimal parallelization topology, it is necessary to look for the ways of the best aggregation of the simplest operations into blocks (the problem of optimal block enlargement) [2]. Generally speaking, the new methods of cloud parallelization are required along with special software complexity metrics (**SCM**): in this sense, there is an orientation on the program that carries out all calculations in the cloud.

First of all, it is logical to consider the possibility of the use of existing SCM. It should be noted, that the existing were initially designed to measure characteristics as a criteria of programs quality. SCM usually can only evaluate the overall picture of the program execution, that is, they help to analyze the complexity of the control flows, the volume of source code etc., but do not give a more complex evaluation of the data and operations that are used during the program execution. However, this can greatly affect the program execution result, as programs even with similar algorithms may vary considerably in resource consumption of operations they perform. It is obvious that the operation of addition is performed much easier and faster than operation of division or, for example, calculating the sine. Apart from different complexity of the operations themselves, the complexity of numbers that are involved can also vary, especially if they are fractional numbers with a large number of decimal places. Based on the foregoing, making of the dependencies of computation time on the type of operation and / or the length of the mantissa is an actual experimental issue, for which there is no analytical alternative. Existing SCM may serve as a basis for complexity evaluation of typical programs that implement the individual blocks in the large-block parallelization option, if there are any dependencies of the computation time from the dimension of problems solved in the block, and from the size of the mantissa.

There are two main types of SCM:

– metrics for evaluation of the software itself;

– metrics for evaluation of the software development environment.

We are interested in the use of the first group of metrics.

First, we consider the simplest and most common metrics of programs evaluation – quantitative metrics that give an idea of the overall complexity evaluation of the programs, which are solving problems in large-block distributed computing mode. The most elementary metric of quantitative evaluation is a trivial evaluation of the number of source code lines, which called SLOC metric [3]. Of course, in the modern conditions it is not rational at all to evaluate programs in terms of just its text volume.

Another metric evaluating programs size is ABC metric [4]. It is calculated as

$$\sqrt{A^2 + B^2 + C^2},$$

where A is the number of the assignment operators (Assignment); B is the number of functions calls (Branch); C is the number of operators such as “if” (Condition).

Common metrics of quantitative evaluation is a Halstead metric [5]. This metrics are based on calculating a number of operators and operands in the source code of the program.

It is easy to guess that the use of quantitative metrics in this case is impossible, since they do not take into account programs topology and are not suitable for a complex evaluation of the software for the optimal allocation of tasks to resources, achieving a significant reduction in the total calculation time. In most cases of the optimal parallelization any metric should be used on the software design stage and all quantitative metrics are very dependent on the completeness of the source code. In this work we are interested in metrics that evaluate the digraph as a single object, constructed on the basis of the control flow of the program, that was first proposed by Thomas McCabe in 1976 [6].

Control graph is created as follows:

– vertices of the graph corresponds to the linear code fragment that does not contain control points, the vertices may correspond to one or more program operators without transmission flow management;

– control flows in the graph are denoted as directed arcs;

– the vertices are divided into two types: the operators, from which one arc comes out, and predicates – from which two arcs come out, what corresponds to the conditions of the program (for example, if).

Typically, two additional vertices are also present in the control flow graph (**CFG**) – the input and output. Each vertex of the graph should be accessible from the input vertex and the output vertex should be accessible from any other vertex. Complexity metric proposed by McCabe is called the cyclomatic complexity of the program and can be calculated using the formula

$$V(G) = E - N + 2P, \tag{1}$$

where E – the number of arcs in the control flow graph of the program; N – the number of vertices; P – the number of coherency components of the graph; for linear programs it is usually assumed to be equal to the number of arcs that should be added to the CFG to convert it to a strongly connected graph, i.e. to a graph, every vertex of which is reachable from any other. Graphs of the programs without inaccessible vertices from the input vertex and without "dangling" input and output points are converted into strongly connected by adding the arc, that closes input and output vertices, as shown in Fig. 1 by dotted line.

Main disadvantages of this metric are as follows:

– cyclomatic complexity is a metric of control complexity measurement, it does not depend on the input data for the calculation;

– parameterization of graphs is not considered in the complexity evaluation, so different programs with similar graphs will be evaluated equally;

– data flow is not monitored in the complexity evaluation, which can also give a distorted representation of the complexity of the program.

There are several metrics which improve the cyclomatic complexity evaluation. Let us briefly list them:

– G. Myers metric, which allows to distinguish the programs with the same CFG and predicates with various complexity. In practice, this method is rarely used [7];

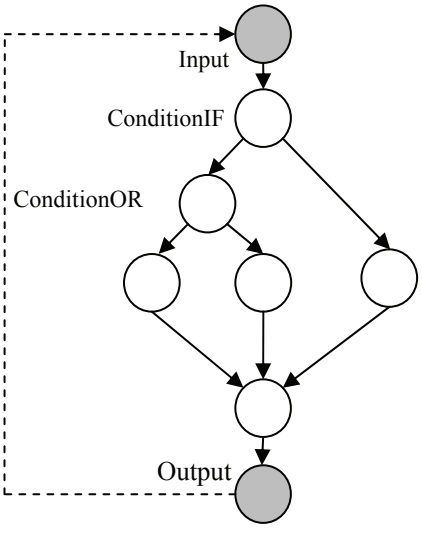

Fig. 1. An example of a control flow graph of a simple program with one condition of IF type and one Boolean operator of OR type

– Hansen's metric [8] evaluates the complexity of the programs with the use of a pair {cyclomatic complexity, the number of operators}. This metric is very sensitive to the structuring of the program code;

– Chen's topological metric evaluates complexity of the program by the number of intersections of the boundaries of the program graph [9]. Chen's metric is applicable only in the case of a structured program with a consecutive connection of control structures;otherwise, the metric significantly depends on presence in the program of conditional and unconditional transitions;

– for the evaluation of the length of the program Harrison-Magel's metric

is used [10]. This metric is based on an evaluation of the reduced complexity of the predicate vertex, which is calculated as the sum of the initial (defined by any assessment of the complexity metrics snippets previously described) or reduced complexities of the vertices within its sphere of influence, plus the initial complexity primary predicate vertex;

– Pivovarskiy's metric is another modification of cyclomatic complexity metric and allows you to keep track not only of the differences between consecutive and embedded control structures, but also between structured and unstructured programs [11];

– another simple metric is T. Jilb's metric, which proposes to evaluate the complexity of the program for its saturation with the expressions `IF`, `THEN`, `ELSE` [12];

– another common complexity evaluation tool of the software is the boundary values metric [13]. For this metric directed graph $G(N, E)$ with a single input and single output vertices is used. For complexity evaluation graph G is divided into the maximum possible amount of subgraphs G', with whose help the adjusted complexity of selection vertex is calculated, which is equal to the number of vertices in its constituent subgraph, and then the relative boundary complexity of the program.

As we know, cyclomatic complexity metric does not account for the complexity of the individual vertices of the programs control flow graph when determining its complexity. Here is an example: given two source code, which has identical CFGs, but which do significantly different by complexity calculations (Fig. 2, 3). If you build a control flow graphs for these programs, they will be absolutely identical. Let's try to calculate theirs complexity with the use of cyclomatic complexity metric

$$V(G) = 12 - 10 + 2 = 4.$$

As you can see, this characteristic does not reflect differences in the complexity of operations and dimensions of numbers, what can cause unexpected and negative effects during the program execution. Therefore, we propose to modify the cyclomatic complexity metric, adding to it a complexity evaluation of the operations performed in the vertices of the CFG. To obtain experimental time required to perform standard mathematical operations, and to find the complexity of these operations, we developed a special Windows application.

Before starting the calculations the following parameters should be entered:

– i – the number of mantissa length increments;

– stepis – the number of symbols that will increase the length of the mantissa in the course of iteration;

– δ – relative error percentage of calculation time measurement;

– $N_z^{\max}$ – limit of the number of iterations with zero calculating time; if the limit is exceeded, we believe that further measurements should stop and set the zero time as result of computational experiment;

– ΔN – portion size;

– t_k – a critical calculation time for a single reference to the operation: if this value is exceeded, the influence of variation in time measurement is neglected, and the portion size is set to minimal, i.e. $\Delta N = 1$ [14].

To determine the average calculation time parameters were selected as follows: step was set as 18,

```
for (inti=0; i<30; i++)
{
 HRealG(_T("0.0"), 72);
 HRealN(_T("0.0"), 72);
 HReal T = (A + B) / C;

 if (T > 1)
 {
       G = sin(A);
 }
 else
 {
        G = sin(B);
 }
 for (int j = 0; j < 10; j++)
 {
       N += G * j;
 }
 A += 10;
 B += 15;
}
```

Fig. 2. Source code of the first program for verification of the precision calculation with the use of operations complexity

```
for (inti=0; i<40; i++)
{
      HRealG(_T("0.0"), 90);
      HRealN(_T("0.0"), 90);
      HReal T = sin(A) - sin(B);

      if (T > 0)
      {
            G = sin(A * C);

      }
      else
      {
             G = sin(B * C);

      }
      for (int j = 0; j < 20; j++)
      {
            N += (A + B) / G * j;
      }
      A += 10;
      B += 15;
      C += 1;
}
```

Fig. 3. Source code of the second program for verification of the precision calculation with the use of operations complexity

as this value gives the most significant time divergence, relative error percentage of calculation time measurement was set as 0.05, limit of the number of iterations with zero calculating time was set as 2000 and the portion size as 10000 since these number are large enough to provide a sufficiently precise value of the average calculation time. In addition, before each length of the mantissa increase two arrays are generated – $a[\Delta N]$ and $b[\Delta N]$, filled with random numbers that are used for the calculations in each portion. At the same time, each time the length of the mantissa is increased, the previous numbers remains the same, but a number of random digits equal to the specified step is added in the end, with the use of a standard function of C ++ srand(), which argument is set depending on the number of portion. As a result, the data about average calculation time of basic arithmetic, trigonometric and logarithmic functions were obtained.

The experimental data need to be approximated to obtain data about calculation time of operations with the use of numbers that have much greater length of the mantissa. It is important that the coefficients of the approximating polynomials are non-negative, otherwise there is a risk that with the large p the average time can become negative value, which is impossible. After finding the polynomial coefficients for each operation, they were checked on the possibility of negative values of the average calculation time. To do so, polynomials charts were built on a very long range of values of p, to ensure that the average time is always increasing (Fig. 4).

It is also important to check how the found polynomials correspond with the real data. To do so, instead of a single operation, we obtain data of the calculation time of the tested expression, and compare it with the theoretical estimates. We are taking several points $p' = np_{\max}$, $n = 1, ..., 10,$ where $p_{\max}$ is the accuracy of the last experimentally

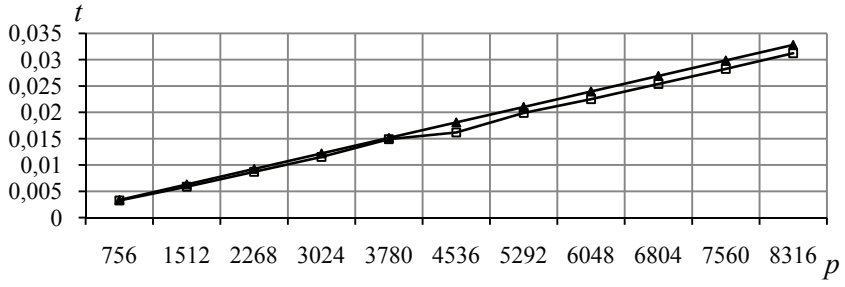

Fig. 4. Polynomial charts for the addition operation on a long range of values of *p*: —▲— – polynomial chart; —□— – experimental data

Table 1

Calculated polynomial and experimental data for addition operation

p'	Average calculated time	Average experimental time
756	0.00173889	0.001729
1512	0.003277071	0.003124
2268	0.004815251	0.00468
3024	0.006353431	0.006106
3780	0.007891611	0.007502
4536	0.009429792	0.009093
5292	0.010967972	0.010424
6048	0.012506152	0.011842
6804	0.014044333	0.01344
7560	0.015582513	0.01465
8316	0.017120693	0.01625

obtained value, and calculating in these points as the polynomials values, and calculating the experimental data with thegiven length of the mantissa [15].

Then we compare the experimental and calculated data to determine how precisely the polynomial data describes the average computation time of each operation. Table 1 shows the data for the addition operation, and as a result of these tests, it was found that, for example, for the addition operation with $p' = 8316$ match with the experimental data is equal $\frac{0.01625}{0.017120693}100 = 94.9\ \%$, therefore, we believe that coefficients are found precisely enough.

Obtained data is used to determine the complexity of the operations performed in the CFG vertices, and the total operations complexity is calculated as

$$S_0 = \lg \sum_{N}^{i=1} S_i p_i, \tag{2}$$

where i is operation number; N is total number of program operations; S is complexity of i-th operation; p is maximum precision of numbers involved in the i-th operation. Modified complexity of the program is calculated based on the complexity of its CFG and the total operations complexity

$$V' = S_0 V(G). \tag{3}$$

Thus, we propose the method of improving the metrics using the control flow graph for a more precise definition of the programs complexity.

The work has been performed within the Project No 1346 from the register of state tasks to higher education institutions and scientific organizations in the field of scientific research.

References

1. Tolstykh S.S., Podol'skii V.E. [Evaluation of complexity of the large-block cloud computing using arithmetic with enhanced accuracy], *Trudy ISP RAN* [Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS)], 2014, vol. 26, no. 5, pp. 29-64. (In Russ., abstract in Eng.)

2. Egorov S.Ya., Fursov I.S., Sharonin K.A., Nemtinov K.V. [The use of parallel computing for tracing algorithm wave of technological pipelines], *Transactions of Tambov State Technical University*, 2012, vol. 18, no. 2, pp. 361-366. (In Russ.)

3. Kulakov A.Yu. *Otsenka kachestva programm EVM* [Evaluation of the quality of the computer program], Kiev: Tekhnika, 1984, 167 p. (In Russ.)

4. Fitzpatrick J. Applying the ABC Metric to C, C++, and Java, *C++ Report*, 1997, pp. 245-264.

5. Kholsted M.Kh. *Nachalo nauki o programmakh* [Beginning of Science Programs], Moscow: Finansy i Statistika, 1981, 128 p. (In Russ.)

6. McCabe T.J. A complexity measure, *IEEE Transactions on Software Engineering*, 1976, vol. SE-2, no. 4, pp. 308-320.

7. Myers G. An Extension to the Cyclomatic Measure of Program Complexity, *SIGPLAN Not*., 1977, vol. 12, pp. 61-74.

8. Hansen W.J. Measurement of program complexity by the pair (cyclomatic number, operator count), *SIGPLAN Not*., 1978, vol. 13, no. 3, pp. 61-64.

9. Chen J.Y., Lu J.F. A new metric for object-oriented design, *Information and Software Technology*, 1993, vol. 35, no. 4, pp. 232-240.

10. Harrison W., K. Magel. A complexity measure based on nesting level, *SIGPLAN Not*., 1981, vol. 16, no. 3, pp. 63-74.

11. Abran A. *Software Metrics and Software Metrology Hoboken*, NJ: Wiley-IEEE Computer Society Publ, 2010, 328 p.

12. Fenton N.E., Pfleeger S.L. *Software Metrics: A Rigorous and Practical Approach*, London: International Thomson Computer Press, 1996, 671 p.

13. Lipaev V.V. *Obespechenie kachestva programmnykh sredstv. Metody i standarty* [Quality assurance software. Methods and Standards], Moscow: Sinteg, 2001, 380 p. (In Russ.)

14. Tolstykh S.S., Podol'skii V.E., Babichev A.M., Tolstykh S.G. [The computational complexity of solving systems of linear algebraic equations: stage of the experiment], *Vestnik nauchnykh konferentsii* [Journal of scientific conferences], 2015, no. 1-7 (1), pp. 42-51. (In Russ.)

15. Luk'yanov O.V., Smirnov E.S., Khrapov I.V. [Evaluation system using the information processing resources in the information management], *Transactions of Tambov State Technical University*, 2011, vol. 17, no. 2, pp. 321-326. (In Russ.)

Усовершенствование метрик оценки сложности программ, основывающихся на анализе графа потока управления

А. М. Бабичев[1], В. Е. Подольский[2], А. Н. Грибков[3], С. С. Толстых[4], С. Г. Толстых[4]

Кафедра «Системы автоматизированной поддержки принятия решений» (1); Тамбовский областной центр новых информационных технологий (2), zhelkzhelk@yandex.ru; кафедры: «Конструирование радиоэлектронных и микропроцессорных систем» (3); «Информатика» (4), ФГБОУ ВО «ТГТУ»

Ключевые фразы: крупноблочные параллельные вычисления; метрики сложности; облачные вычисления; прецизионно-доверительное решение задач; прецизионные вычисления.

Аннотация: Рассмотрены методологические аспекты оценки сложности крупноблочных облачных вычислений с повышенной точностью. Данная разработка направлена на решение задач математического моделирования с особыми требованиями точности. В частности речь идет об оценке сложности программ, решающих задачи со сложными связями между подзадачами в виде крупных блоков и временем счета, значительно превышающим время передачи информации между ними. Предложен метод улучшения метрик оценки сложности, использующих для этого граф потока управления программ, для построения оптимальных по производительности вычислительных систем, функционирующих в облачной среде. Суть метода заключается в использовании сложности отдельных операций на основе экспериментальных данных.

Список литературы

1. Толстых, С. С. Оценка сложности крупноблочных облачных вычислений, использующих арифметику повышенной точности / С. С. Толстых, В. Е. Подольский // Труды ИСП РАН. – 2014. – Т. 26, № 5. – С. 29 – 64.

2. Применение параллельных вычислений для трассировки технологических трубопроводов волновым алгоритмом / С. Я. Егоров [и др.] // Вестн. Тамб. гос. техн. ун-та. – 2012. – Т. 18, № 2. – С. 361 – 366.

3. Кулаков, А. Ю. Оценка качества программ ЭВМ / А. Ю. Кулаков. – Киев : Техніка, 1984. – 167 с.

4. Fitzpatrick, J. Applying the ABC Metric to C, C++, and Java / J. Fitzpatrick // C++ Report. – 1997. – P. 245 – 264.

5. Холстед, М. Х. Начало науки о программах / М. Х. Холстед. – М. : Финансы и Статистика, 1981. – 128 с.

6. McCabe, T. J. A Complexity Measure / T. J. McCabe // IEEE Transactions on Software Engineering. –1976. – Vol. SE-2, No. 4. – P. 308 – 320.

7. Myers, G. An Extension to the Cyclomatic Measure of Program Complexity / G. Myers // SIGPLAN Not. –1977. – Vol. 12. – P. 61 – 74.

8. Hansen, W. J. Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count) / W. J. Hansen // SIGPLAN Not. – 1978. – Vol. 13, No. 3. – P. 61 – 64.

9. Chen, J. Y. A New Metric for Object-Oriented Design / J. Y. Chen, J. F. Lu // Information and Software Technology. – 1993.– Vol. 35, No. 4. – P. 232 – 240.

10. Harrison, W. A Complexity Measure Based on Nesting Level / W. Harrison, K. Magel // SIGPLAN Not. – 1981. – Vol. 16, No. 3. – P. 63 – 74.

11. Abran, A. Software Metrics and Software Metrology Hoboken / A. Abran. – NJ : Wiley-IEEE Computer Society Publ, 2010. – 328 p.

12. Fenton, N. E. Software Metrics: A Rigorous and Practical Approach / N. E. Fenton, S. L. Pfleeger. – 2nd ed. – London : International Thomson Computer Press, 1996. – 671 p.

13. Липаев, В. В. Обеспечение качества программных средств. Методы истандарты / В. В. Липаев. – М. : Синтег, 2001. – 380 с.

14. Вычислительная сложность решения систем линейных алгебраических уравнений: этап эксперимента / С. С. Толстых [и др.] // Вестник научных конференций. –2015. – № 1-7 (1). – С. 42 – 51.

15. Лукьянов, О. В. Оценка использования ресурсов системы обработки информации в информационном менеджменте / О. В. Лукьянов, Е. С. Смирнов, И. В. Храпов // Вестн. Тамб. гос. техн. ун-та. – 2011. – Т. 17, № 2. – С. 321 – 326.

Verbesserung der Metriken der Einschätzung der Komplexität der auf der Analyse des Graphstroms der Steuerung gründenden Programme

Zusammenfassung: Im Artikel sind die methodologischen Aspekte der Einschätzung der Komplexität der wolkigen Großblockberechnungen mit der erhöhten Genauigkeit betrachtet. Diese Entwicklung ist auf die Lösung der Aufgaben der mathematischen Modellierung mit den besonderen Forderungen der Genauigkeit gerichtet. Unter anderem handelt es sich um die Einschätzung der Komplexität der Programme, die die Aufgaben mit den komplizierten Beziehungen zwischen den Teilaufgaben in Form der Großblöcken und der die Zeit der Sendung der Informationen zwischen ihnen wesentlich übertretenden Zählenzeit lösen können. Es wird die Methode der Verbesserung der Metriken der Einschätzung der Komplexität der für diesen Graphstrom verwendenden Steuerung der Programme für die Konstruktion der nach der Produktivität in der wolkigen Umgebung funktionierenden optimalen Computersysteme vorgeschlagen. Das Wesen der Methode besteht in der Nutzung der Komplexität der einzelnen Operationen aufgrund der experimentalen Daten.

Perfectionnement des métriques de l'évaluation de la complexité des programmes fondés sur l'analyse d'un graphe du flux de la commande

Résumé: Dans l’article sont examinés les aspects méthodologiques de l'évaluation de la complexité du cloud computing de grands blocs avec une précision accrue. Ce travail vise à résoudre des problèmes de la modélisation mathématique ayant des besoins de précision. En particulier, il s'agit d'évaluer la complexité des programmes résolvant des problèmes avec des relations complexes entre sous-problèmes sous la forme de gros blocs et le temps du calcul dépassant celui de la transmission de l’informations. Est proposée la méthode du perfectionnement des métriques de l'évaluation de la complexité des programmes utilisant un graphe du flux de la commande pour la construction des systèmes optimaux fonctionnant dans le cloud. L'essence de la méthode est d'utiliser la complexité de certaines opérations à la base des données expérimentales.

Авторы: ***Бабичев Антон Михайлович*** – аспирант кафедры «Системы автоматизированной поддержки принятия решений»; ***Подольский Владимир Ефимович*** – доктор технических наук, профессор, директор Тамбовского областного центра новых информационных технологий; ***Грибков Алексей Николаевич*** – кандидат технических наук, доцент кафедры «Конструирование радиоэлектронных и микропроцессорных систем»; ***Толстых Сергей Степанович*** – кандидат технических наук, доцент, заведующий кафедрой «Информатика»; ***Толстых Светлана Германовна*** – кандидат технических наук, доцент кафедры «Информатика», ФГБОУ ВО «ТГТУ».

Рецензент: ***Немтинов Владимир Алексеевич*** – доктор технических наук, профессор, заведующий кафедрой «Компьютерно-интегрированные системы в машиностроении», ФГБОУ ВО «ТГТУ».