Complexity-based Prediction of Faults Number for Software Modules Ranking Before Testing: Technique and Case Study

Svitlana Yaremchuk[1], Vyacheslav Kharchenko[2]

[1]Danube Institute of National University "Odessa Maritime Academy",
8 Didrikhson St., 65029, Odessa, Ukraine

svetlana397@yandex.ru

[2] National Airspace University of N. E. Zhukovsky "The Kharkiv Aviation Institute",
17 Department, Chkalova St., 61070, Kharkiv, Ukraine

v_s_kharchenko@ukr.net

Abstract. The proposed method is based on estimating software modules' complexity by means of metrics. Indices of source code complexity are the input data for the method. Ranged selection software modules with faults form the output data for the method. The verification of the method has been performed on the basis of representative selection of experimental data for five systems. An appropriate coefficient is proposed and calculated for assessment of the method's efficiency. It shows the correlation between a part of revealed faults to a part of verified modules. The efficiency coefficient value increases on condition of increase in quantity of applied metrics. The proposed method enables to reach the required reliability by means of a choice of certain part of modules for identification as it is possible for the most part of faults. The method provides a software reliability increase in limited resources conditions.

Keywords. Reliability, efficiency, complexity metrics, software modules with faults

Key Terms. Software System, Metric, Software Engineering Process

1 Introduction

1.1 Motivation

Software system reliability (hereinafter referred to as the System) substantially depends on its complicatedness. Complicatedness of multiple and mutually related requirements leads to formation of erroneous, incomplete, contradicting specifications. System's architecture complicatedness causes fatal errors (critical faults) generations, which may result to project ruination. Complicated system implementation leads to errors' increase in the initial code. Various sources show that testing process reveals 10 – 20 errors per each 1,000 code lines as an average. Changes in initial code are required at the stage of system operation contributing to newer errors generation. Integrated approach is therefore essential to access and ensure reliability basing on system complicatedness evaluation throughout its lifetime. Actual software indicators/ metrics should be measured thoroughly and continuously to ensure reliability and handle development and verification procedures. Increasing systems' complicatedness, from the one side and limited resources, from the other side, generate contradictions between required systems' reliability and that achieved once development is completed. It is therefore critically important to develop and to research reliability assessment methods basing on system's complicatedness analysis, both in general and its components taking into note limited testing resources.

1.2 Related Works

Systems' reliability should be continuously controlled throughout their entire lifecycle. Therefore the process of required reliability progress should be controlled at all stages. Any process management demands indicative values evaluation. Multiple number of models, methods and tools are developed to evaluate various reliability indicators at various lifecycle stages.

Works [1-5] propose various models and methods to increase precision of evaluation of reliability indicators. In work [1] the authors analysed the impact of debugging time upon reliability evaluation and forecast. The debugging time causes an overestimation of the perceived software quality up to 15% in studied dataset; similarly, it causes the underestimation of testing time required to obtain a given quality for a software product.

Authors of [2] suggested a method based on matrix allowing known reliability models. Applying matrix enables to 1) form an allowance vector for software system under development to take into account development process peculiarities; 2) to choice appropriate model basing on allowances vector; and 3) calculate model parameters. However, the allowances matrix requires adding newer models and methods.

The most of expenditures are associated with the testing stage. Here the bulk of defects are also revealed. Authors of [3] proved that applying combined operation and debugging tests (OP-D) ensures, in general, addressing both faults occurring with high frequency at the operation time and those occurring with lower frequency. The objective of proposed OP-D technique lies in reliability improving only by means of operation testing revealing, at the same time, as many bugs, as possible, as may be achieved via debugging application. However, the proposed OP-D technique is unable to take into account rigid restrictions in testing resources.

Work [4] has been devoted to the basic theory of the of software systems' dynamics and established the theoretical basis for the reliability assessment of and proposed a new universal method for such assessment. This method takes into account effects of secondary faults, and improves the accuracy of software reliability indices more than twice. Proposed models and methods require experimentally obtained data of faults revealing time in the course of testing. The more data becomes available, the more precise are reliability evaluating indicators, shorter is the testing period, and the less are resources required to achieve required reliability level.

Work [5] describes software reliability model with complexity index based on the non-homogenous Poisson process (NHPP). The method of software reliability assessment has been developed on the basis of generalized NHPP model and the testing sufficiency criteria. Software application for software failures prediction using artificial neural networks has been developed. However, reliability forecasting by means of neural networks binds developers to use software known not so well and to study network characteristics. Complicatedness, reduced studying rate and high level of operation margin prevent this method from implementation into routine engineering practice.

It is statistically established that revealing and elimination of faults at the earlier stages of development are 10 – 100 times cheaper, than same actions performed at the ready product testing stage. Reliability indicators should be evaluated prior testing commencement to save expenditures. Developers should be aware of faults quantity in the software system. This knowledge enables to draw up testing process and to link it with available resources. However, this data is insufficient. It is more important to know which modules contain the highest quantity of faults. It enables to minimize testing group efforts and to maximize faults revealing which plays an essential; part in restricted resources situation. Works [6] through [13] propose certain methods of code complicatedness evaluation by means of metrics.

Work [6] describes analytic model to establish relation between faults quantity in the initial code and complicatedness indexes basing upon developed model. Application and statistical analysis of the proposed method showed that discrepancy between actual faults quantity and estimated one amounted to 11%. The work researches methods of faults localization in software modules. As a result, 9% discrepancy between obtained indicators and actual quantity of faults has been found. However, the problems associated with choice and ranging of the fault prone software modules still has not been solved.

The method proposed in [7] allows combining techniques so as to maximize the number of faults revealed for the tested software from those expected to be available. As for fault types the method refers to well-known orthogonal defects classification (ODC). As for testing techniques, the method applies techniques of functional, statistical, robustness and stress testing. The final result is a forecast of faults quantity of each ODC category each technique is capable to detect for a particular application. However, it still remains a question which modules should be tested if limited monetary and time resources make it impossible to carry on total testing.

Solutions for the problem in question are proposed in works [8] through [13]. Authors of work [8] developed three individual models forecasting fault-proneness in data set: one with ascending stepwise logistic regression, one with descending stepwise logistic regression and one without stepwise selection in logistic regression. The authors concluded that descending stepwise regression provides the best model. The level of false alarm rate is too high in all the models, while it should be contrary.

Complexity metrics in predicting fault-prone software modules have been intensively studied in the work [9]. The binary logistic regression method is applied in studying using as an example commonly available data on five commercial products. The study shows that (1) models generated using more data sets can improve the prediction accuracy but not the recall rate; (2) reducing the cut-off value may improve the recall rate, but the number of false positives will increase, resulting in higher maintenance efforts.

The authors of work [10] studied, whether metrics available in the early lifecycle (i.e. requirement metrics), combined with metrics available in the late lifecycle (i.e. code metrics), may be used to identify fault prone modules using genetic algorithm-based technique. However, applying multiple various metrics increases complicatedness of the method. It also requires running software, which is not well known.

Authors of research [11] have empirically evaluated performance of Hierarchical Clustering Technique (HCT) in predicting fault-prone modules using open source software and metrics. The proposed technique has shown 85% accuracy. However, developers should study thoroughly MatLab hierarchy clusterization algorithms to use this method in practice.

Research [12] addresses the problem of predicting fault prone modules using data mining techniques. In this study the authors applied different data mining rule-based classification techniques on several commonly available datasets. The newly proposed algorithm is an enhanced existing algorithm in terms of effectiveness (i.e. generating less number of rules) and accuracy (i.e. improving the results). Despite of rules reduction the method itself and its automatization possibilities are too complicated to enable its application.

The authors of work [13] have studied various metrics (requirement metrics, design metrics and code metrics) and techniques to identify fault prone modules. The proposed metrics are aimed to provide higher prediction results than existing techniques. However, various metrics combined with their application algorithms substantially increase complicatedness of this method.

The testing stage, aimed to improve reliability of a software system, is the most expensive and time-consuming one. Moreover, since dimensions of software systems have increased significantly during the past decades, effective utilization of limited testing resource has become even more important than before. A software system is typically composed by a number of modules. Each of them needs to be assigned with some testing resource before the testing stage commences. Hence, a natural question is how to allocate the testing resource to modules so that the reliability of a software system is could be maximized. Such a problem was formally defined by Ohtera and Yamada as the Optimal Testing Resource Allocation Problems (OTRAPs). In the works [14-16] it was proved that an optimal allocation scheme may lead to significant improvement in terms of the reliability of a software system. The available resource should be allocated among modules in a way enabling maximum number of faults to be removed from each module to achieve higher software reliability.

The optimization problems are formulated in work [14] as nonlinear programming problems (NLPP), which are solved by means of software reliability improving model based on a non-homogenous Poisson process which incorporates Log-logistic testing-effort function. Work [15] suggests solving the OTRAPs by means of Multi-Objective Algorithms known as Hierarchy Particle Swarm Optimization Algorithm Experimental results show that the proposed algorithm has overcome the drawbacks of the existing algorithm, and is more efficient. The main goal of the article [16] is to examine the resource allocation plan for fault detection and correction process of the software to save costs during testing and operational phases. The authors developed a model for fault detection and correction process in pursue of the said aim. Methods proposed in works [14-16] are complicated enough. Their application requires highly qualified personnel.

Analysis of described methods shows situation, as follows. Methods [1] through [5] don't permit to plan and evaluate testing resources, since they require testing data. Methods [6] through [16] require data on projects developed previously. Differences between systems being currently developed and those already existing are inevitably accompanied by substantial margin in evaluation. Allowances methods in realistic development processes don't work, thus reducing precision in reliability indicators' evaluation. Methods are complicated in application. Input data processing requires specific not commonly known software and efforts of skilled highly qualified personnel. All these items increase systems' development costs and reduce popularity of the methods. Methods don't take into account rigid restrictions of the testing resources. Analysis of described defects and drawbacks enables to offer general approaches to reliability evaluation and management throughout systems' lifetime taking into consideration their complicatedness and limited resources assigned for development basing on comparatively simple and more precise methods.

The rest of the work is composed, as follows. Chapter 2 describes general approaches to reliability assessment and management. Chapter 3 presents the check assumption and the description of the complexity-based prediction technique. Chapter 4 describes the case study in terms of the experimental verification methods, results' analysis and discussion. Conclusions are represented in chapter 5.

2 Approaches to Reliability Assessment and Management

There are two key principles of reliability evaluation and reliability management. The first one is technology-oriented. It means that proposed evaluation methods and aids are based on complicatedness recording and may be applicable with various systems development methodologies. Adjustable software may ensure their flexibility and simplify their implementation into business – software corporate processes. The second aspect of reliability management is based on resource considerations and enables to calculate and put foundation under resources allocation for reliability evaluation and ensuring. It comprises determining costs associated with achieving required reliability and comparative analysis of expenditures and incomes derived from faults detection in all the created products (specifications demands, project, initial code, etc.) throughout the entire development stage and risks and losses reduction at the operation stage. The technological- and resources-oriented approach to reliability evaluation and reliability management supposes following methods to be applied.

Complicatedness evaluation method with appropriate metrics enables to identify the most complicated and the most fault-prone requirements. These activities may contribute to design faults identification and elimination at the earliest stages. Their analysis will contribute to earlier faults identification and elimination in specifications. The project complicatedness should be also evaluated using metrics at the development stage. Such an approach enables to analyse project, to identify most complicated subsystems, components, interfaces, carry on their decomposition and improve links. Such activities may contribute to earlier identification and elimination of design faults.

One of proposed ways to reduce complicatedness and to improve program code reliability lies in its optimization (also known as refactoring). A large number of authors of publications in software systems development state that the initial complicatedness cannot be reduced. From their point of view, simplifying any software constructions leads to other ones becoming more complicated. However, the problem of code complicatedness quantification both at pre-optimization and post-optimization stages remains still unsolved.

Once the initial code is written, the testing procedure should be scheduled and resources should be allocated. Here total faults number in the system should be evaluated basing on code metrics. Since lack of resources is a common situation they should be spent in a most efficient manner. Selection and ranking modules with the highest fault number method should be developed. Testing of such modules will enable to reveal maximum faults with minimum number of modules subject to testing.

Conditions analysis for development, testing and allowance for reliability models enables to select a suitable model for evaluating achieved reliability indicators. Profile of system application should be evaluated at the operation stage by means of multiple metrics. Such a measure enables to improve efficient resources allocation to monitor the system to improve its reliability.

The proposed approach is directed to achieve required reliability in the most cost-saving manner. With testing being the most expensive and extended development stage initially, within the framework of general approach, method of improving testing efficiency should be developed. It determines the *researches goal – complexity-based fault number forecasting technique development and verification for ranking software modules prior testing.*

3 The Complexity-Based Prediction Technique of Fault Number for Ranking of Software Modules

3.1 Check Assumption of the Prediction Technique

The proposed method is based on an assumption that the most complicated modules (classes, components) contain the bulk quantity of faults. This assumption is logical and is often applied in scientific publications, requiring, however, practical confirmation. Five sets of experiment data have been used for verification [17]. The mentioned data was placed with Internet resource of the PROMISE (PRedictOr Models In Software Engineering) depository. The data was collected by numerous researchers and experts in software faults forecasting and published as freeware. The data contains evaluations based on various metrics criteria for various software systems for certain periods of time. Selected software systems are developed for various purposes, are implemented by means of various programming languages and methodologies, have various initial code dimensions, various number of active developers, development period, versions quantity, etc. The sole essential factor common for all reviewed software system consists in their being non-commercial projects with freeware initial code. The data in question represent complicatedness indicators by metrics and number of faults revealed in the course of modules' testing for five various object-oriented systems. Systems characteristics are shown in Table 1. Total volume of explored data exceeds 1.000.000 initial code lines, contains 4.330 modules and 4.449 faults. Comparative analysis of data grouped in the Table 1 shows that the researched system very substantially in their characteristics. Table 1 contains also complicatedness measures per one module in metrics. Multiple metric of choice $M = \{RFC, WMC, LCOM, LOC, NPM, CE, CBO\}$ is explained in the work [6]. Table 1 data analysis shows that complicatedness of software systems differs substantially in a number of metrics. Significant total volume and specified differences in the abovementioned Systems enable to regard this sample as a representative sample. Actions described below are proposed to identify probable correlations between complicatedness and faults quantity in an individual module. Total modules' multitude SET_{mod} for each system has been indexed seven (7) times (according to applied metrics number) in decreasing sequence of complicatedness numerical indicator per each metric. A certain part has been derived from each multitude (10 %, … 50 %) of the most complicated modules with highest complicatedness level under each metric. As a result seven basic sub-collections have been obtained $SUBSET^{RFC}_{mod}$, $SUBSET^{WMC}_{mod}$, $SUBSET^{LCOM}_{mod}$, $SUBSET^{LOC}_{mod}$, $SUBSET^{NPM}_{mod}$, $SUBSET^{CE}_{mod}$, $SUBSET^{CBO}_{mod}$.

Table 1. Data on Systems being under Research

Systems characteristics	Abbreviated systems names and versions				
	Luc 2.4	Xer 1.4	Ant 1.7	Xal 2.7	Cam 1.4
Modules quantity	340	588	746	910	1 746
Faults quantity	632	1 596	338	1213	670
Fault modules ratio	40%	26%	78%	1%	83%
Number of faults in a module	1.86	2.71	0.45	1.33	0.38
Faults density per 1,000 lines	6.14	11.30	1.62	2.83	3.42
RFC Metric	25	19	34	29	21
WMC Metric	10	10	11	11	9
LCOM Metric	69	75	89	126	73
LOC Metric	303	240	280	471	112
NPM Metric	7	8	8	9	7
CE Metric	5	3	6	7	6
CBO Metric	11	6	11	12	11

Intersection of the sets by seven metrics formed newer modules' sub-collections $SUBSET^{7}_{mod}$, by six metrics $SUBSET^{6}_{mod}$, by five metrics $SUBSET^{5}_{mod}$, by four metrics $SUBSET^{4}_{mod}$, by three metrics $SUBSET^{3}_{mod}$, by two metrics $SUBSET^{2}_{mod}$, by one metric $\mathrm{SUBSET}^{1}_{\mathrm{mod}}$. Actual quantity of faults per one module has been counted for each enlisted sub-collection. The results are displayed in Table 2.

Table 2. Average actual faults quantity per module

Modules complicatedness	Abbreviated systems names and versions				
	Luc 2.4	Xer 1.4	Ant 1.7	Xal 2.7	Cam 1.4
$\mathrm{SUBSET}^{7}_{\mathrm{mod}}$	13,5	20,9	3,1	3,0	4,6
$\mathrm{SUBSET}^{6}_{\mathrm{mod}}$	5,4	13,2	1,9	2,3	2,0
$\mathrm{SUBSET}^{5}_{\mathrm{mod}}$	3,1	14,2	1,1	2,0	1,1
$\mathrm{SUBSET}^{4}_{\mathrm{mod}}$	2,1	5,8	1,4	1,7	1,2
$\mathrm{SUBSET}^{3}_{\mathrm{mod}}$	2,0	4,3	1,0	1,7	1,9
$\mathrm{SUBSET}^{2}_{\mathrm{mod}}$	2,3	3,8	1,0	1,5	1,6
$\mathrm{SUBSET}^{1}_{\mathrm{mod}}$	1,5	2,8	0,4	1,7	0,4

Data contained in Table 2 shows that the most complicated software system modules have the greatest number of faults. As modules complicatedness decreases, the average fault number in module reduces confirming thus the validity of allowance method.

The proposed method should be run once the initial code is written, prior its testing commences. Complicatedness indicators of individual modules in metrics form the method's input data. Should the developers lack the corporative and practically checked set of metrics commonly available and most informative for faults forecasting object-oriented metrics considered in the work [6] may be applied as starters. The set of applicable metrics is indicated as $M = \{M_1, ..., M_n\}$. The method supposed proceeding in five steps, as described below.

Step 1. Complicatedness indicators calculation split into metrics for individual modules in a system being developed using standard or corporate software. ...

Step 2. Total modules set $\mathrm{SET}_{\mathrm{mod}}$ should be indexed n times as the complicatedness indicator descends with each metric $\mathrm{SET}^{M_1}_{\mathrm{mod}}, ..., \mathrm{SET}^{M_n}_{\mathrm{mod}}$.

Step 3. A certain part of the most complicated modules by a specific metric should be drawn from each indexed set. Dimensions of such a part should be determined referring to testing resources' restrictions. The less are these resources, the less is the part of drawn modules. As a result *n* basic sub-collections have been obtained for *n* metrics $\mathrm{SUBSET}^{M_1}_{\mathrm{mod}}, ..., \mathrm{SUBSET}^{M_n}_{\mathrm{mod}}$.

Step 4. Once intersection for basic sub-collections $\mathrm{SUBSET}^{M_1}_{\mathrm{mod}}, ..., \mathrm{SUBSET}^{M_n}_{\mathrm{mod}}$ is determined, intermediate sub-collections should be formed with top complicatedness indicators by the metrics $\mathrm{SUBSET}^{n}_{\mathrm{mod}} = \mathrm{SUBSET}^{M_1}_{\mathrm{mod}} \cap ... \cap \mathrm{SUBSET}^{M_n}_{\mathrm{mod}}$. All the modules within this sub-collections should be ranked as *n*. Choosing all and any *n-1* metrics intersections for subset $\mathrm{SUBSET}_{\mathrm{mod}}^{\mathrm{n-1}}$ should be determined with all the modules to be ranked as *n – 1*. Similar procedure should be applied until subset for any single metric is built as $\mathrm{SUBSET}_{\mathrm{mod}}^{1}$ with all the modules in it ranked as *1*. Number of ranks should be the same as metrics number.

Step 5. Determining the sum of subsets for $\mathrm{SUBSET}_{\mathrm{mod}}^{\mathrm{n}}, \mathrm{SUBSET}_{\mathrm{mod}}^{\mathrm{n-1}}, ..., \mathrm{SUBSET}_{\mathrm{mod}}^{1}$ the resulting ranked modules sample should be formed as $\mathrm{SUBSET}_{\mathrm{mod}}^{\mathrm{R}} = \mathrm{SUBSET}_{\mathrm{mod}}^{\mathrm{n}} \cup \mathrm{SUBSET}_{\mathrm{mod}}^{\mathrm{n-1}} \cup ... \cup \mathrm{SUBSET}_{\mathrm{mod}}^{1}$.

The resulting sample should include modules with top complicatedness indicators simultaneously by *n* metrics (*rank 7* in our example), *n-1*metrics (*rank 6*), … and, finally, one metric (*rank 1*). Modules ranking in resulting sample is necessary to establish sequence of their testing. Top rank modules are the most complicated, supposed to contain the most of faults, and are subject to be tested at the first turn. The proposed method should be applied once the initial code is written and prior its testing starts. Complicatedness indicators of individual modules within the developed system form the input data with output data being the resulting ranked sample of modules with faults. Basic sample dimensions should be determined by restrictions imposed by testing resources.

4 Case Study

The method has been verified by means of commonly accessible experimental data [17] and specially developed software. Number of selected modules has been calculated as well, as faults number in these modules for each system. As this data has been being calculated the quantity of involved metrics has been varied as well, as the modules number with top complicatedness indicators corresponding to these metrics. Obtained data has been analysed. The first subtask of the analysis was to define relations between faults number and modules' complicatedness and was estimated simultaneously by a number of metrics. The second subtask was to define relation between faults number and that of selected modules. Selected ratio values encompassed 10%, …, 50% including the most complicated modules. The parts dimensions were governed by probable restrictions in testing resources. Since the researched systems substantially varied in their characteristics, relative percentage indicators have been calculated.

4.1 The Experiment Performance Technique

99% modules within one system under research contained faults making it remarkably distinguished among others. Data processing and analysis showed that modules selection based on their complicatedness had not given increase in a quantity of faults in them. Average data for four systems is represented in Table 3. Data is allocated in two lines. The first line displays modules ratio from their total quantity. The lower ratio shows faults ratio contained in these modules in relation to their total quantity.

Values highlighted as examples in the last column in Table 1 should be interpreted, as follows. Base sample 20% of the most complicated modules per each metric forms a resulting ranked modules sample 41,8% ratio of their total number, containing 74,8% ratio of total faults' quantity.

The data in question should be applied in a manner, as follows. Tests are developed for each selected module. These tests take a certain amount of labour expressed in person-hours. Time consumed by running all such tests for all selected modules enables to calculate testing expenditures required to reveal 74,8% of the total faults quantity. At the further stage, restrictions imposed by labour, time, financial, hardware and software resources should be considered. Should there be a lack in resources required to test selected modules, basic sample dimensions should be reduced.

Factor $k = \frac{d_m}{m}$, with m being modules ratio in their total quantity and faults ratio of their total number contained in appropriate modules is proposed to reduce the number of analysed indicators and to facilitate the method's efficiency analysis. The higher is the k value, the more is the number of faults within the selected modules.

Table 3. Average calculated data on four systems under research

Modules' ratio	Indicators	Rank 7	Rank 6	Rank 5	Rank 4	Rank 3	Rank 2	Rank 1	$\sum$
10 %	mod., %	2,0	1,3	2,5	2,5	3,8	3,3	8,0	23,3
	faults, %	15,0	6,5	7,0	8,0	6,5	4,8	7,8	55,5
13 %	mod., %	2,3	1,8	2,8	3,5	4,0	3,8	8,8	26,8
	faults, %	17,3	8,8	8,3	7,0	6,8	5,8	8,5	62,3
17 %	mod., %	2,5	3,0	3,5	5,3	5,0	4,8	11,0	35,5
	faults, %	19,8	13,0	7,3	9,5	6,3	5,8	9,0	70,5
$\boxed{20\ \%}$	mod., %	4,5	4,0	4,5	5,5	6,0	5,5	11,8	$\boxed{41,8}$
	faults, %	23,5	13,3	8,3	7,8	6,5	6,8	8,8	$\boxed{74,8}$
25 %	mod., %	6,3	5,0	5,5	7,0	7,0	6,3	13,5	50,5
	faults, %	30,0	12,0	9,0	9,0	5,8	5,3	9,0	80,0
33 %	mod., %	9,0	7,5	8,0	8,8	6,5	8,3	14,5	62,5
	faults, %	36,0	14,8	9,3	8,5	5,3	6,5	7,0	87,3
50 %	mod., %	17,8	13,3	10,5	9,8	9,0	9,3	11,5	81,0
	faults, %	47,0	19,3	10,8	5,8	4,3	4,8	4,5	96,3

Basing on the k application, Table 3 data was transformed and represented in Table 4 format. The k values in Table 4 show how the selected and ranked modules share exceeds the part of faults they contain.

Table 4. Method's average efficiency factors k for four systems

Modules' ratio	k for rank 7	k for rank 6	k for rank 5	k for rank 4	k for rank 3	k for rank 2	k for rank 1
10 %	7,50	5,20	2,80	3,20	1,73	1,46	0,97
13 %	7,67	5,00	3,00	2,00	1,69	1,53	0,97
17 %	6,58	4,33	2,07	1,81	1,25	1,21	0,82
20 %	5,22	3,31	1,83	1,41	1,08	1,23	0,74
25 %	4,80	2,40	1,64	1,29	0,82	0,84	0,67
33 %	4,00	1,97	1,16	0,97	0,81	0,79	0,48
50 %	2,65	1,45	1,02	0,59	0,47	0,51	0,39

For example, k = 7,5 means that testing of 1% of the selected modules enables to reveal 7,5% of faults. The following factor interpretation is proposed. If $k < 1$ the modules' selection is inefficient from the point of view of their complicatedness evaluated by metrics. If $1 \le k < 1,5$ modules are selected with minor efficiency. Finally, if $k \ge 1.5$ the modules are selected efficiently. E.g. if 13 % modules are selected their testing may be efficient if they are characterized by high complicatedness levels simultaneously by seven, six, five, four, three, and two metrics. If 20 % modules are selected, testing of modules with high complicatedness values by seven, six, and five metrics will be efficient.

4.2 Results of Analysis

Analysis of Table 4 data enabled to identify two tendencies. The first tendency may be followed in each horizontal data line. Efficient modules selection will be maximum for modules with top complicatedness indications involving simultaneously all the involved metrics. The k decreases as modules' complicatedness reduces. The k is less than 1 in the Table 4, last column. Thus, modules selection by a single metric is not advantageous for faults detection. The second tendency may be followed in each data column. Selection of the least part of the most complicated modules is the most efficient. The k value decreases as the selected module number increases. Consequently, the less is selected modules number, the higher is effect or modules complicatedness evaluation by metrics.

Thus the method's efficiency depends on metrics' quantity determining simultaneously high complicatedness and on modules sample dimensions. The coefficient value increases with the metrics' number increase and sample dimensions decrease. Reverse statement is valid, too. Maximum coefficient value amounted to 7,5 with average value 2,34 and minimum value 0,39.

Selecting modules and tests for them enables to estimate testing expenditures to detect a certain quantity of faults. Since restricted resources prevent testing all the modules, the unrevealed faults may cause risk of losses. Risks calculation and comparing them with testing expenditures may enable managers to find adequate and economically grounded solutions. Tables 3 and 4 data may be helpful not only for efficient testing efforts allocations, but also for efficient application of other code verification methods, e.g. inspections, statistical analysis, etc.

The proposed set of known metrics for code complicatedness evaluation is an initial, or starting, one. Software corporations' experts may apply their unique corporate metrics set, proven at practice. Discussable aspect is using obtained results not only for code testing but its applicability for other verification methods, such as survey, control, inspections, audits, statistical analysis. It is not still clarified, whether selected modules contain fault-free modules, what is their quantity, how it depends on complicatedness level.

5 Conclusions

Elements are developed for technological and resources oriented approach to reliability management at all the lifecycle stages in restricted resources environment. The technique of complexity-based prediction of faults number for ranking software modules is proposed within the framework of general approach. Supposed statement that the most complicated modules contain the bulk quantity of faults is proved experimentally. The method is applicable after initial code is written prior its testing commencement. The method's input data consists of complicatedness indicators for individual modules of the system being in development by metrics. The output data represents a sample of ranked modules of certain dimensions with a certain number of faults. Sample dimensions are only restricted by testing resources.

The proposed method is aimed to efficient testing with restricted resources. Verification of the method using a representative sample of experimental data demonstrated its efficient operability. Proposed efficiency factor depends on number of metrics by which modules have simultaneous high complicatedness rates and on modules sample dimensions. The achieved results enable to control process of achieving required reliability with restricted resources by means of allocating the testing efforts to a certain number of modules with the highest faults quantity.

Sampling such modules and selecting tests for them enables to estimate *a priori* testing costs by means of summing up testing time for all the selected modules. Maximization of revealed faults number and improved systems' operating reliability may reduce expenditures and increase developers' revenues. Risks of faults triggering at the operation stage may be mitigated. Testing expenditures will be efficient investments into the systems' reliability.

The proposed method is simple enough to be applied in practice. The method does not demand any additional data except the code complicatedness evaluation for the system being in development. The method may be completely automated. Revealed application restriction concerns systems with 99% faulty modules ratio. Only in such a case method application may be inefficient. Prospective direction of further researching may be implementation of the proposed method into business processes involving various methodologies of software systems developments. Testing expenditures calculation model and method of comparing expenditures with unrevealed faults risk evaluation should be developed. Software implementation of the proposed method should be described.

References

1. Cinque, M., Gaiani, C., Stradis, D., Pecchia, A., Pietrantuono R., Russo, S.: On the Impact of Debugging on Software Reliability Growth Analysis: A Case Study. Computational Science and Its Applications (ICCSA), pp 461—475, LNCS-8583 (2014)
2. Kharchenko, V. S., Tarasyuk, O. M., Sklyar, V.V.: The Method of Software Reliability y Growth Models Choice Using Assumptions Matrix. Proceedings of 26-th Annual Int. Computer Software and Applications Conference, COMPSAC, Oxford, England, pp. 541--546 (2002)
3. Cotroneo, D., Pietrantuono, R., Russo, S.: Combining Operational and Debug Testing for Improving Reliability. The Journal of Systems and Software, vol. 62, no. 2, pp. 408-- 423 (2013)
4. Mayevsky, D. A.: A New Approach to Software Reliability. Lecture Notes in Computer Science. Software Engineering for Resilient Systems. Berlin, Springer, no. 8166, pp. 156--168 (2013)
5. Yakovyna, V. S.: Influence of RBF neural network input layer parameters on software reliability prediction. 4-th International Conference on Inductive Modelling ICIM'2013, Kyiv, pp. 344--347 (2013)
6. Yaremchuk, S. O.: Models, methods and technology of a priori estimation reliability indices of accounting and analytical information systems. Dissertation for scientific degree of candidate of technical sciences in specialty Information technology. – Odessa National Polytechnic University, Odessa, Ukraine, 210 p. (2015) (In Ukrainian)
7. Cotroneo, D., Pietrantuono, R., Russo, S.: Testing techniques selection based on ODC fault types and software metrics. The Journal of Systems and Software, no. 86, pp. 1613--1637 (2013)
8. Mausa, G., Grbac, T. G., Basic, B. D.: Multivariate Logistic Regression Prediction of Fault-Proneness in Software Modules. MIPRO 2012/CTI, pp. 813--818 (2012)
9. Yu1, L., Mishra, A.: Experience in Predicting Fault-Prone Software Modules Using Complexity Metrics. Quality Technology & Quantitative Management, vol. 9, no. 4, pp. 421--433 (2012)
10. Sandhu, P. S., Khullar, S., Singh, S., Bains, S. K., Kaur, M., Singh, G.: A Study on Early Prediction of Fault Proneness in Software Modules using Genetic Algorithm. International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 4, no. 12, pp. 1891--1896 (2010)
11. Kaur, S., Mahajan, M., Sandhu, P. S.: Identification of Fault Prone Modules in Open Source Software Systems using Hierarchical based Clustering. ISEMS, Bangkok, pp. 238--244 (2011)
12. Najadat, H., Alsmadi, I.: Enhance Rule Based Detection for Software Fault Prone Modules. International Journal of Software Engineering and Its Applications, vol. 6, no. 1, pp. 75--86 (2012)
13. Kaur, I.: A Compound Metric for Identification of Fault Prone Modules. IOSR Journal of Computer Engineering (IOSR-JCE), vol. 17, Issue 6, Ver. V, pp. 31--35 (2015)
14. Ahmad, N., Khan, M., Islam, S.: Optimal Allocation of Testing Resource for Modular Software based on Testing-Effort Dependent Software Reliability Growth. ICCCNT-12, Coimbatore, India (2012)
15. Pavithra, M.: Optimal Testing Resource Allocation Problems in Software System using Heuristic Algorithm. Bonfring International Journal of Software Engineering and Soft Computing, vol. 2, no. 4, pp. 1--9 (2012)

16. Nasar, M., Johri, P.: Testing and Debugging Resource Allocation for Fault Detection and Removal Process. International Journal of New Computer Architectures and their Applications, no. 4, pp. 193--200 (2014)
17. The PROMISE Repository of empirical software engineering data, http://openscience.us/repo/defect/ck/