VERIFICATION OF THE SOFTWARE RELIABILITY MODELS

Dmitry A. Maevsky, Elena J. Maevskaya, Oleg P. Jekov, Ludmila N. Shapa
•
Odessa National Polytechnic University, Odessa, Ukraine
e-mail: Dmitry.A.Maevsky@gmail.com

ABSTRACT

The paper concerns the verification of the existing Software reliability models and their comparison to a new one based on the theory of Software system dynamics. A statistically significant number of observations over the process of fault detecting in the fifty different Software systems has been used for the verification. The results of comparison of estimation correctness of the nine most widely used reliability models to the new one based on the theory of Software system dynamics are represented. It has been proven the Software system dynamics model provided 2,7 times higher correctness of reliability estimation than the existing reliability models.

1 INTRODUCTION

The problem of providing and forecasting the Software reliability of informational systems is one of the most up to date in the modern program engineering. Nowadays the cost of program failure can be measured not only in million dollars but also in million human lives. Since the modern informational technologies have extensively entered all spheres of our life the majority of mankind becomes in a sense a hostage of its own creature – Software systems. These systems are relied on a considerable amount of functions of control in traffic, communications, power, economy, defense and other areas control. In order to immobilize a big city activity a failure in a system of control dealing with the work of traffic lights in its main transport lines is enough. The consequences of computer errors in mobile communications and power systems are much more dramatic. And a nuclear reactor management system failure at the nearby power station can be catastrophic for all the continent.

So the creation of the reliable computer systems and further keeping their reliability during operation is of vital importance. A new and comparatively young trend in the reliability theory – Software reliability (SR) – deals with estimation and forecasting the Software system reliability. Its task is to develop the theoretical base of software reliability as well as models, methods and practical technology for Software resources reliability determination.

2 THE PROBLEM NOWADAYS

At present there exist about twenty different Software reliability models (SRM). Such an abundance conditions the necessity to classify them, and now we have got several schemes of classification. The most popular one has been proposed by J.D.Musa and Okumoto [1]. It distinguishes such characteristics as:

— Model time. It determines a time counting system applied to the Model — either actual astronomic (calendar) time or processor time, spent on the work with the given Software by the moment of fault detection.

— Model category. It determines the amount of faults which can be detected when investigation time is infinite. According to this characteristic all models are classified into finite and infinite.

— Model type. It determines the probability distribution of random event occurrence, fault detection in our case. Two types of distribution are used in the models of reliability: the Poisson distribution and binomial distribution.

— Model class. This characteristic is only used for finite category models and determines a type of function describing the law of intensity change of fault appearing.

— Family. This characteristic is only used for infinite models and possesses the same meaning as the characteristic "class" for finite ones.

In the above classification a particular attention should be focused on such a characteristic as "model time" because it is a principal factor. First of all a time counting system is different for an individual analyzing Software systems and for an analyzed Software itself. A human lives in his (her) own time counting system breaking the stream of time into habitual time intervals – years, months, days, hours, etc. For a human being the time is uninterrupted. From the viewpoint of a Software system – if we are trying to imagine ourselves a program system – all events happen in absolutely another way. Assume a researcher detected and eliminated a program fault in the evening of November, 25, 2011 at 20 o'clock according to the local time. We see it is a natural way of time counting for a human. Assume the 25th of November is Friday and after fault detection and elimination the computer has been off over Saturday and Sunday. So the analyzed system was run and started operating only at 8 o'clock in the morning on Monday. The next fault was detected by the same researcher at 9 o'clock in the morning on Monday, November, 28. What is the time interval between these two sequential detections? From the human viewpoint it is 61 hours. And what is it for a Software system (SS)? While the computer was off the Software system was not downloaded in the memory and executed. It is possible to say that over sixty hours the system did not exist in general and all the processes in it were stopped! It "immersed itself in suspended animation" at 20 o'clock on Friday and "raised again" at 8 o'clock on Monday morning. So from the viewpoint of the system (you understand here we are trying to make the system "human" and suppose that it is able to have its own opinion) the period of time between two fault detections is an hour. We have proven that SS time was discontinuous and consisted of separate intervals during which it was active.

We can ask the question: what time is more correct in estimating the hazard rate? Certainly, the system operation time, i.e. the processor time. But when using for astronomical time modeling, the cumulative curve has gaps on the axis of abscess. The data of the axis of abscess (time) have changed, and the ones of the ordinate axis (cumulative fault number) have not. It makes a false impression of dissimilar fault detection rate which distorts the modeling results and decreases the prediction ability of the model. It can be clearly seen in the diagram represented in Fig.1.

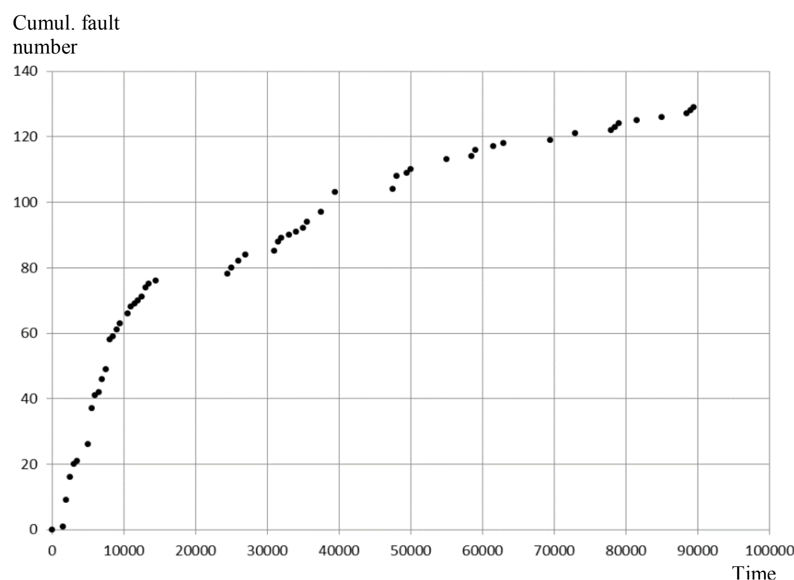

Fig. 1. The cumulative curve gaps in using the astronomical time

The data for the diagram have been taken from the appendix for [14] (Chapter 4, file Csr2.dat). Along the axis of abscess the moments of time are represented (the scale is not mentioned in the literature), along the ordinate axis – the cumulative fault number.

In the diagram we can see several gaps of mentioned type. For example, take the beginning site of the curve before the first gap. According to the diagram the initial point of the curve has got the coordinates (0, 0), а конечная – (14500, 76). Thus for 14500 time units 76 faults are detected, the average rate of detection is $\frac{76}{14500} = 0{,}0052$ fault per time unit. Then the curve demonstrates another distinct gap – the point has got the coordinates (24500, 78), i.e. for the next 10000 time units only two faults are detected. The fault detection rate in this site equals $\frac{2}{10000} = 0{,}0002$ fault per time unit, i.e. 26 times less than in the previous one! As we see in the diagram the fault detection rate is abruptly restored to its original value in the next third site again. It is obvious that the jumps of detection rate of such kind cannot be explained by anything but the ordinary interruptions in observation.

If the processor time were used the mentioned gap would not appear in spite of the interruption conditioned by some subjective factors. It is necessary to note though the instruction provides the astronomical time application in the model, the processor time can be used as well. And all the mathematical formula and equations remains the same.

In order to estimate the correctness of reliability index modeling the examination of the most popular models belonging to different branches of the described classification has been carried out. Let us consider the main characteristics of these models.

1. Jelinski-Moranda's Model [2]. Time – astronomical; category – finite; type – binomial; class – exponential. Assumptions: failure intensity is proportional to the actual fault number in the program and remains constant in the time interval between any two neighboring moments of fault detecting; detection of all the faults in the program is equiprobable and independent; all the faults have got similar degree of importance; time until the detection of the next Software fault is distributed exponentially.

2. Goel-Okumoto's Model [3]. Time – astronomical; category – finite; type – Poisson; class – exponential. Assumptions: all the SS faults are mutually independent; the detected faults are eliminated immediately; the fault detection process is a stream of homogeneous events and has got the Poisson distribution.

3. Schneidewind's Model [4]. Time – astronomical; category – finite; type – Poisson; class – exponential. The main distinctive peculiarity of this model – the failure intensity determining in the later time is supposed to be more correct for prediction of the further process development than the one measured at the earlier stages. Assumptions: the fault number in the given time interval is independent on the fault number in the other intervals; the detected fault number decreases from interval to interval; the failure intensity is proportional to the fault number detecting at that exact moment.

4. Musa's Model [5]. Time –processor; category – finite; type – Poisson; class – exponential. Assumptions: the fault detection process is the Poisson process; the fault detection is proportional to the number of faults which were not detected yet.

5. Weibull's Model [6]. Time – processor; category – finite; type –binomial; class – exponential. Assumptions: at the initial moment of observation there is a finite number of faults in SS; time before the fault detection is a stochastic value having probability subjected to Weibull distribution.

6. S-form Model [7]. Time – processor; category – finite; type – Poisson; class – gamma distribution. Assumptions: the fault detection process is the Poisson process; in detecting a fault it is immediately eliminated without entering new ones.

7. Duan's Model [8]. Time – astronomical; category – infinite; type – Poisson; family – gamma distribution. Assumptions: the cumulative fault detection is the Poisson process with the function of distribution $\mu(t) = \alpha \cdot t^{\beta}$, where α and β – are positive numbers.

8. Moranda's Geometrical Model [9]. Time – astronomical; category – infinite; type – Poisson; family – exponential. Assumption: the failure intensity is a geometric progression $\lambda(t) = D \cdot \phi^{i-1}$ with denominator $0 < \phi < 1$; the probability of detection of every certain fault is subjected to the exponential distribution law.

9. Musa-Okumoto's Logarithmic Model [10]. Time – astronomical; category – infinite; type – Poisson; family – exponential. Assumption: the failure intensity is decreased over time according to the exponential law; the fault detection process is the Poisson process.

10. Software System Dynamics (SSD) Model. The SSD theory fundamentals have been developed in [11] and [12] as an absolutely new deterministic approach to formulating the reliability parameters taking into account the secondary fault influence. SSD is different from the existing Software reliability theory because it is not based on the probability theory but on the non-equilibrium process theory, and it does not consider the fault appearing in Software system as an occasional event but as a result of deterministic fault flow impact.

SSD is based on the following assumptions:

1. SS is an open non-equilibrium system that interacts with its subject area ac-cording to the laws of the non-equilibrium processes. This is a new point of view on the program system. It is assumed that the properties of a software system are similar ones of other open systems.
2. The state of the SS is characterized by a special state function f(t) – the number of the defects containing in it. Here it means the number of primary or secondary defects.
3. Disappearing and appearing the defects in the SS is the result of the joint action of the direct (outcoming) and reverse (incoming) defect flows. It is implied that the primary defects are removed from the system by the direct flow and secondary defects are appeared in the system as a result of the reverse flow.
4. The intensity of each flow is proportional to the number of defects that this flow forms. This is a basic principle of the non-equilibrium processes theory. For a software system, this principle means that the reduction the number of defects causes the decrease of their detection rate.
5. All defects are equivalent, and participate in the formation of the flow in the same way, regardless of the causes, location, and type of defect (the principle of equivalence).
6. Function f(t) is differentiable on the whole domain (the principle of continuity).

The basic concept SSD is the one of software defect flows. Each defect is considered as an integral part of the total flow, which obeys not the laws of the probability theory but the laws of identification and evolution of flows in non-equilibrium systems. The identification of the defect flows in the SS is shown in Fig. 2.

In the SS operation defects are the causes that the result which is produced by SS does not correspond to the result expected by the subject area. This discrepancy is detected by the user which is in contact with the SS on the one hand and with its subject area on the other. Thus, firstly the user acts as an error detector, and secondly – a kind of "contact surface" between the SS and its subject area. We assume that the user is ideal, that is, he detects and records each defect at the time of its identification.

In the process of correcting the defect disappears from the SS due to changes made in its code. This disappearance can be supposed as a result of the of defects removal from the SS. Considering this process in time, we obtain the flow of defects from the SS through the "contact surface", i.e. the user. This flow is shown by arrows "Detection" and "Correction" in Fig. 2.

It is possible to insert additional "secondary" defects in the process of correcting defects in the SS. The process of inserting the secondary defect may be regarded as the second, counter-flow of defects, which operates in the direction from the subject area to the SS.

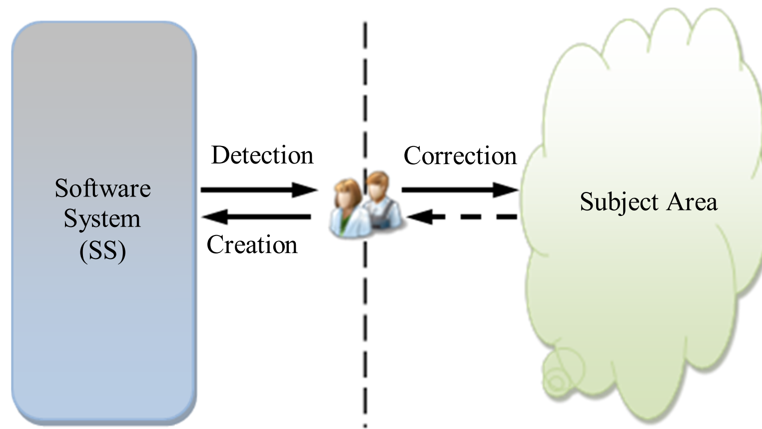

Fig. 2. Defect emergence in the SS

We will numerically characterize the flow of defects by the rate (intensity) of the flow, which can be determined by hypothesis 6 (principle of continuity). Taking into account the outcoming flow only, SS is characterized by the number of defects, which are contained in the system – coordinate f1(t). The defects leave the system for the subject area. It has just one degree of freedom, and is described by the differential equation of first order. In the case of taking into account of the second process (insertion of secondary defects), its coordinate is their current number – f2(t). Thus we obtain two coordinates – f1(t) and f2(t). SS in this case is a system with two degrees of freedom and described by differential equations of second order.

SSD is describes by the following autonomous system of differential equations:

$$\begin{cases} \dfrac{df_1}{dt} = -A_1 \cdot f_1 + A_2 \cdot f_2 \\ \dfrac{df_2}{dt} = A_2 \cdot f_1 - A_1 \cdot f_2 \end{cases}$$

She's solution allows to determine the time variation of the primary and secondary defects existing in SS:

$$f_1 = F_0 \cdot e^{-A_1 t} \cdot cosh(A_2 t)$$
$$f_2 = F_0 \cdot e^{-A_1 t} \cdot sinh(A_2 t)$$

The presented mathematical equations give the opportunity to estimate the number of the primary and secondary faults in the Software system and carry out a comparative analysis of correctness of the described reliability models.

3 THE RESULTS OF INVESTIGATION

As the initial data for the described reliability model verification we have used the data of the fault detection process analyzed in fifty different-type Software system. All of them were scripted in different computer languages and have got different functions. The information about these systems are given in Table 1. All the used series are inhomogeneous: because of the changes making in the system the law of fault detection curve changes is different over time. That is why in order to increase the correctness each of the series have been divided into sites with the help of the unchangeable law of fault detection.

Table 1

№	Data Sources	Information	Number of points	Numb er of inter vals
1	[13]	OS «Android», version 2.3	765	
2	[14], Chapter 4, file Csr1.dat	No data	397	
3	[14], Chapter 4, file Csr2.dat	No data	129	
4	[14], Chapter 4, file Csr3.dat	No data	104	
5	[14], Chapter 4, file SS3.dat	No data	278	
6	[14], Chapter 4, file Sys1.dat	No data	136	
7	[14], Chapter 7, file Sys1.dat	No data	136	
8	[14], Chapter 7, file Sys2.dat	No data	86	
9	[14], Chapter 7, file Sys3.dat	No data	207	
10	[14], Chapter 7, file J1.dat	No data	62	
11	[14], Chapter 7, file J2.dat	No data	181	
12	[14], Chapter 7, file J3.dat	No data	41	
13	[14], Chapter 7, file J4.dat	No data	114	
14	[14], Chapter 7, file J5.dat	No data	73	
15	[14], Chapter 8, file 8.txt	Multiprocessor System	186	
16	[14], Chapter 9, file Odc1.dat	Large IBM Project	1207	
17	[14], Chapter 9, file Odc3.dat	No data	400	
18	[14], Chapter 10, file S2.dat	No data	54	
19	[14], Chapter 10, file S27.dat	No data	41	
20	[14], Chapter 10, file SS4.dat	No data	197	
21	[14], Chapter 10, file SS1.dat	The Language of Assembler	81	
22	https://github.com /AArnott/dotnetopenid	C#	55	5
23	https://github.com /activescaffold/active_scaffold	Ruby	97	7
24	https://github.com /adamzap/landslide	Python	89	9
25	https://github.com /addyosmani/backbone-fundamentals	JavaScript	25	3
26	https://github.com /AFNetworking/AFNetworking	Objective-C	155	12
27	https://github.com /ai/r18n	Ruby	36	5
28	https://github.com /akzhan/jwysiwyg	JavaScript	194	18
29	https://github.com /alankligman/gladius	JavaScript	43	6
30	https://github.com /AlanQuatermain/AQGridView	Objective-C	88	11
31	https://github.com /alecgorge/jsonapi	Java	111	15
32	https://github.com /alohaeditor/Aloha-Editor	JavaScript	391	27
33	https://github.com /amatsuda/kaminari	Ruby	191	11
34	https://github.com /andreasgal/B2G	Rust	136	13
35	https://github.com /andreasronge/neo4j	Ruby	112	14
36	https://github.com /andrewplummer/Sugar	JavaScript	88	15
37	https://github.com /andris9/Nodemailer	JavaScript	50	6
38	https://github.com /andymatuschak/Sparkle	Objective-C	132	5
39	https://github.com /antirez/hiredis	C	70	11
40	https://github.com /apneadiving/Google-Maps-for-Rails	Ruby	138	12
41	https://github.com /Araq/Nimrod	Nimrod	91	10
42	https://github.com /arsduo/koala	Ruby	160	19
43	https://github.com /asual/jquery-address	C#	126	18
44	https://github.com /away3d/away3d-core-fp11	JavaScript	195	28
45	https://github.com /bartaz/impress.js	Java	65	12
46	https://github.com /BaseXdb/basex	JavaScript	271	23
47	https://github.com /Baystation12/Baystation12	No data	302	39
48	https://github.com /bbatsov/ruby-style-guide	Ruby	73	12
49	https://github.com /benbarnett/jQuery-Animate-Enhanced	JavaScript	67	14
50	https://github.com /bengottlieb/Twitter-OAuth-iPhone	Objective-C	102	17

We have counted 522 intervals with the same law of change of detected fault amount over time. Along all these intervals for the ten analyzed models both 5220 estimations of reliability are made and their correctness determined. The correctness is obtained according to standard deviation (SD) criterion observed and calculated with the help of the fault value model and SD value dispersion for different Software systems. SD values are calculated on the formula:

$$SD = \frac{\sum_{i=1}^{n} \left(f_{io} - f_{ic}\right)^2}{n}$$

where n is the number of points in series, f_{io} is the observed value and f_{ic} is the calculated value.
The results of comparing the reliability estimation correctness by different models are represented in Table 2. Besides this Table demonstrates the number of series (in percentage to the total number of series) unprocessed by each of the model, minimum and maximum SD values obtained for a model, the average value according to the model as well as the logarithm for dispersion. The analysis of the Table has shown that only two of the investigated nine models – SSD and S-form model – are able to carry out the reliability estimation for all of 522 time intervals.

Table2. The Results of Verification

Model	SSD	Jel.-Mor	NHPP	Schneiderw.	Musa	Weib.	S-form.	Duan	Moranda’s Geom	Musa-Okum.
Unprocessed %	0,00	54,2	15,9	36,8	55,6	0,96	0,00	28,2	9,4	70,9
Min. SD	0,00	0,23	0,02	0,00	0,01	0,00	0,00	0,09	0,14	0,00
Max. SD	54,5	379	800	928	246	559,7	245,9	996	556,3	417,8
Aver. SD	1,5	12,1	18,7	26,5	4,1	5,3	4,5	25,3	7,8	10,3
$\lg\left(\sigma^2\right)$	3,37	7,59	8,47	8,75	5,91	6,86	5,75	9,09	7,27	7,72

According to this index the worst is Musa-Okumoto’s Logarithmic Model that was not able to process almost 71 % intervals. According to the SD and dispersion value the best turns out SSD Model. It shows 2,7 times more correct results than Musa’s Model which takes the second place as to the correctness. According to the dispersion value SSD Model demonstrates two and more orders of magnitude less than the value of the other analyzed models. The results of verification are represented graphically in Fig. 3.

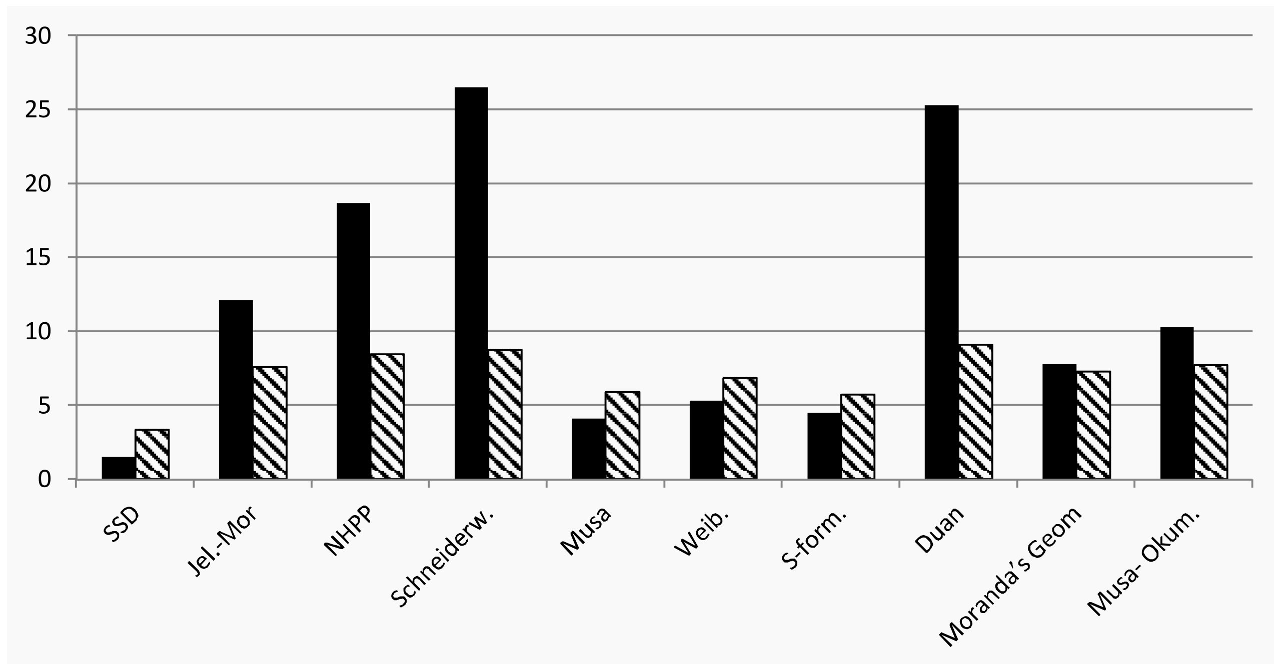

Fig. 3. The results of verification

In this diagram the solid part shows the SD values, and the hatch – the dispersion value logarithms. We have used the logarithmic scale for dispersion because its absolute values vary in orders from model to model.

4 CONCLUSIONS

Thus the verification results show that nowadays SSD Model is the best as to the correctness of reliability estimation by a model. The low dispersion values demonstrate the fact that SSD Model constantly shows the most correct results in all the analyzed Software and can be considered as a universal model. Besides a doubtless advantage of SSD Model lies in its capability to predict the appearing of secondary faults in a Software system. At present SSD is the only model which can not only take into account the secondary fault impact but also foresee their number.

5 REFERENCES

1. Musa, J. D. Software Reliability Models: Con-cepts, Classification, Comparisons, and Practice / J. D. Musa, K. Okumoto // Electronic Systems Effectiveness and Life Cycle Costing, NATO ASI Series, F3, Springer-Verlag, Heidelberg, p. 395 – 424.
2. Moranda P.B. Final Report of Software Relia-bility Study. — / P.B. Moranda, J. Jelinski // McDonnel Douglas Astronautics Company. MDC Report № 63921, dec. – 1972. – 51 c.
3. Goel, A.L., Time-Dependent Error-Detection Rate Model for Software and Other Performance Measures / A.L. Goel, K. Okumoto // IEEE Transactions on Reliability, v. R-28, № 5, August. – 1979. – P. 206 – 211.
4. Schneidewind, N.F. Software Reliability Model with Optimal Selection of Failure Data / N.F. Schneidewind //IEEE Transactions on Software Engi-neering. – Vol. 19. – No. 11. Nov. – 1993. P. 1095 – 1104
5. Musa J.D. Validity of Execution time theory of software reliability // IEEE Trans. on reliability. – 1979. – № 3. – P.199–205.
6. Quadri, S. M. K. Software Reliability Growth Modeling with New Modified Weibull Testing–effort and Optimal Release Policy / S. M. K. Quadri, N. Ahmad // International Journal

of Computer Applications. – Vol. 6. – 2010. – № 12. – C. 1 – 10.

7. Yamada, S. S-Shaped Reliability Growth Mod-eling for Software Error Detection / S. Yamada, M. Ohba, S. Osaki //IEEE Trans-actions on Reliability. Vol. R-32. No. 5, Dec. – 1983. – P. 475 – 478.

8. Duan J.T. Learning Curve Approach to Relia-bility Monitoring // IEEE Trans. on Aerospace. – 1964. – Vol. 2. – P. 563 – 566.

9. Moranda, P.B. Event-Altered Rate Models for General Reliability Analysis / P.B. Moranda // IEEE Transactions on Reliability. Vol.R-28. – No. 5. – 1979. – C.376 – 381

10. Musa, J.D. A Logarithmic Poisson Time Model for Software Reliability Measurement / J.D.Musa, K. Okumoto //Proc. Sevent International Conference on Software Engineering. – Orlando, Florida: – 1984. – P. 230 – 238.

11. Maevsky D. A. A New Approach to Software Reliability / Dmitry A. Maevsky // Lecture Notes in Computer Science: Software Engineering for Resilient Systems. – № 8166. – Berlin: Springer, 2013. – pp. 156 – 168.

12. Maevsky, D. A. Fundamentals of software stability theory [Electronic resource] / D. A. Maevsky // Reliability: Theory & Applications. – 2012. – Vol.7. – № 4(27). – p. 31 – 40. Access mode: http://www.gnedenko-forum.org/Journal/2012/ RTA_4_2012.pdf

13. Android – An Open Handset Alliance Project [Electronic resource]. Access mode: http://code. google. com/p/android/issues/list

14. Lyu, M. R. Handbook of Software Reliability Engineering / M. R. Lyu. – New York: McGraw-Hill Company. – 1996. – 805 p.