Lappeenranta University of Technology, Department of Information Technology

USING TIME AND STOCHASTIC MODELS FOR SOFTWARE RELIABILITY FORECASTING AND ANALYSIS

The topic of the master's thesis has been confirmed by the Department Council of the Department of Information Technology on 14 June 2000

Examiner: Prof. Jan Voracek
Supervisor: Prof. Jan Voracek

Lappeenranta 6 November 2000

Evgenya Salganik
Ruskonlahdenkatu 13-15 B 14
53850 LAPPEENRANTA
+358 40 7435506

TIIVISTELMÄ

Tekijä	Evgenya Salganik
Työn nimi	**Ajallisten ja stokastisten mallien käyttäminen ohjelmien luotettavuuden ennustamista ja analyysia varten**
Osasto	Tietotekniikan osasto
Vuosi	2000
Paikka	Lappeenranta, Finland

Diplomityö. Lappeenrannan teknillinen korkeakoulu. 78 lehteä, 12 kuvaa, 10 taulukkoa ja 3 liitettä. Tarkastajana apulaisprofessori Jan Voracek.

Hakusanat: software reliability, reliability models, software testing

Tämä työ luo katsauksen ajallisiin ja stokastisiin ohjelmien luotettavuus malleihin sekä tutkii muutamia malleja käytännössä. Työn teoriaosuus sisältää ohjelmien luotettavuuden kuvauksessa ja arvioinnissa käytetyt keskeiset määritelmät ja metriikan sekä varsinaiset mallien kuvaukset. Työssä esitellään kaksi ohjelmien luotettavuusryhmää. Ensimmäinen ryhmä ovat riskiin perustuvat mallit. Toinen ryhmä käsittää virheiden ”kylvöön” ja merkitsevyyteen perustuvat mallit.

Työn empiirinen osa sisältää kokeiden kuvaukset ja tulokset. Kokeet suoritettiin käyttämällä kolmea ensimmäiseen ryhmään kuuluvaa mallia: Jelinski-Moranda mallia, ensimmäistä geometrista mallia sekä yksinkertaista eksponenttimallia. Kokeiden tarkoituksena oli tutkia, kuinka syötetyn datan distribuutio vaikuttaa mallien toimivuuteen sekä kuinka herkkiä mallit ovat syötetyn datan määrän muutoksille. Jelinski-Moranda malli osoittautui herkimmäksi distribuutiolle konvergaatioongelmien vuoksi, ensimmäinen geometrinen malli herkimmäksi datan määrän muutoksille.

ABSTRACT

Written by	Evgenya Salganik
Title	**Using Time and Stochastic Models for Software Reliability Forecasting and Analysis**
Department	Department of Information Technology
Year	2000
Place	Lappeenranta, Finland

Diploma Thesis, Lappeenranta University of Technology, 78 pages, 12 figures, 10 tables and 3 appendices. Examiner Associate Professor Jan Voracek.

Keywords: software reliability, reliability models, software testing

The aim of the present study was to provide a survey of time and stochastic software reliability models, and also to investigate some of these models in practice. The theoretical part of the study contains basic definitions and software metrics, used to describe and evaluate software reliability, and the description of the models as such. The paper provides a survey of two software reliability model groups: risk function based models – the first group, and error ”seeding” and tagging and input data structure based models – the second group.

The practical part contains the description and results of experiments, which were done using three models of the first group – Jelinsky-Moranda model, the first geometrical model and the simple exponential model. The purpose of the experiments was to investigate, how the input data distribution affects the model’s functionality, and also how sensitive the models are to the input data size changes. Jelinsky-Moranda model turned out to be the most critical to the distribution because of convergence problems, and the first geometrical model – the most sensitive to the data size changes.

ACKNOWLEDGEMENTS

I would like to thank Associate Professor Vladimir Kirjanchikov from Saint-Petersburg Electrotechnical University for being, in fact, my unofficial supervisor in the present work.

Also I would like to thank Dr. Jan Voracek from Lappeenranta University of Technology for supervising my work and giving valuable comments, that helped me to complete this master's thesis in time.

Besides, I would like to thank Maija Hulkkonen and her company "Kirjansitoja Maija Hulkkonen" for helping me to get the thesis bound within a short time and at a relatively low price.

And finally, last, but not least, I would like to thank Mari Pajunen for translating my abstract into Finnish, and also my roommate Mia Niemi, for recommending Mari to me.

Lappeenranta 6 November 2000

Evgenya Salganik

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

a	A risk function parameter, used in the simple exponential model
A	A convergence criteria in Jelinsky-Moranda model
b	Another risk function parameter, used in the simple exponential model
B	Number of errors, remaining in the program
$\hat{B}$	An estimation of B
C	A parameter, used for matrix form representation of the simple exponential model
CM_i	Number of modules with number of corrections less than 10 in the i-th version
COR_i	Total number of corrections, made in the modules in the i-th version
d	A scaling coefficient in Scheidewind model risk function
d_i	Error detection rate in Schneidewind model
D	a risk function parameter, used by some of the models
D_1	A designation, used to calculate variance in Shick-Wolverton model
E	Software input data space
E_i	A subset of software input data space
f	A function, used to simplify calculations in Jelinsky-Moranda model¨
f_i	Number of errors, detected on the i-th interval

$F(E_i)$	Required output of the program, obtained with an input from E_i
$F'(E_i)$	Actual output of the program, obtained with an input from E_I
$F_M = \sum_{i=1}^{M} f_i$	Total number of errors, detected within M testing intervals
F_{s-1}	Summarized number of errors, detected from the first to (s-1)-th intervals
g	A function, used to simplify calculations in Jelinsky-Moranda model
i	Error detection interval's number
K	A scaling coefficient, used by some of the models
$\hat{K}$	An estimation of K
L	The likelihood function
m	Number of "seeded" detected errors in Mills model
$m_{1,2}$	Number of detected errors in Rudner model
M	Total number of detected errors in Mills model
M_i	Number of modules in the i-th version
MCM_i	Multiple correction modules in the i-th version
n	Total number of error detection intervals
n_i	number of errors, corrected on the i-th interval
N	Number of operators in Beisin model
N(t)	The number of errors, detected by moment t
NM_i	System growth (number of new modules) in the i-th version
OCM_i	Number of old corrected modules in the i-th version

P	Probability of the fact, that no error will occur on a time interval
q	Density of fault probability
Q	Probability of at least one software fault during a time interval
$\hat{Q}(m)$	Statistical estimation of software fault probability during m runs
r	Software purity level in the first geometrical model
R	Risk function on a time interval
s_i	The number of successful tests in La Padula growth model
t	Time
t_{av}	Average time of software working before a fault occurrence
t_i	i-th error detection moment
T_M	Total duration of testing
v	Number of "natural" detected errors in Mills model
X_i	Length of i-th error detection interval
$\hat{X}_{n+1}$	An estimation of (n+1)-th interval between errors
β	A parameter of Schneidewind model risk function
δ	Acceptable $\hat{Q}$ estimation error of fault probability Q
Δ_i	Acceptable limit of software output deviation
φ_i	Binary indicator of a program fault
ϕ	A scaling constant coefficient, used in the third geometrical model

CM	Number of modules with number of corrections less than 10
COR	Total number of corrections, made in the modules
M	Number of modules in the i-th version
MCM	Multiple correction modules
NM	System growth (number of new modules)
OCM	Number of old corrected modules
OS/360	Operation system 360 (a product of IBM company)

1. INTRODUCTION

Software programs typically contain errors. In the most general way a program error can be defined as a situation when a function, included into the program's specification, is not actually performed by the program. Software reliability can be defined (also generally) as the ability of software to perform predefined functions under predefined conditions, and using predefined hardware resources. The extent of software reliability can be estimated using some of software metrics. Software reliability models are aimed to predict the values of these metrics, and thus give a possibility to evaluate reliability on various stages of software testing. For example, if a significant number of errors has already been detected and corrected, it may cause the impression, that the testing process is nearly over, i.e. just very few errors are left in the software. However, it may be far from being true, and reliability models may help to clarify this situation.

The purpose of this work is to provide a survey of time and stochastic software reliability models and investigate some of the models in practice. The paper presents two groups of models: the first group - risk function based models and the second group - error "seeding" and tagging and input data structure based models. Models of the first group have both time and stochastic elements, and models of the second group are purely stochastic.

In the practical part only models of the first group will be investigated, because models of the second group are much more difficult to test. They cannot be tested on generated data, but require real software projects. Models of the first group, which will be tested, are Jelinski-Moranda model, the first geometrical model and

the simple exponential model. They were selected as typical representatives of this group. Testing will be done on two kinds of data – real data, taken from literature sources (published results of software testing), and also generated data. The latter will be used not only because of the lack of real data, but it also because it allows to investigate the influence of input data distribution on the models' functionality. Another aspect, investigated in the practical part, will be sensitivity of various software reliability models to the changes in the input data size (and content, correspondingly, because if the size is changed, the content does not remain the same).

In section 2 the software metrics, which can be used to evaluate and analyze software reliability, are listed and described. Section 3 provides a survey of the first group models, based on the risk function estimation, and section 4 – of the second group modes, based on error "seeding" and tagging, and also on input space structure. In section 5 results, obtained in the practical part, are presented and discussed. Section 6 contains the conclusion of this document.

2. BASIC SOFTWARE RELIABILITY METRICS

In this section basic concepts of software reliability are listed and defined, and these concepts will be used further in the paper. So the concepts are: software error; number of errors, remaining in the program (i.e. errors, passed to the user); probability of software faultless work; error detection intensity (or risk function); a run of the program; a fault of the program.

It is not so easy to give a strict definition of software error, because this definition, in fact, is a function of the program itself, because it depends of the program's functionality, expected by the user. For this reason instead of a strict definition only the indications will be listed, which help to identify software errors:

- occurrence of a wrong operand or operator during programming;
- incorrespondence of the functions, performed by software, to its specification, as well as an error in the specification, requiring some corrections to the software;
- calculation errors (e.g. overflowing etc.);
- corrections to software, improving its user interface;

This list can be considered open, because it can be continued by developers as they get more and more experience in reliability improvement. It is incorrect to consider as an error, for example, creation of codes, replacing a program part, which is missing just temporarily, or program recompilation, caused by corrections in other modules.

The number of errors, remaining in the program, is the potential number of errors, which can be detected on later stages of the life cycle, after corrections, made on the current life cycle stage. This number of errors in the program (later denoted by B), is one of the most important software reliability metrics.

Let P(t) be the probability of the fact, that no error will occur on [0,t] interval. Then probability of at least one fault during this period is Q(t) = 1 – P(t), and the density of fault probability can be presented as

$$q(t) = dQ/dt = -dP(t)/dt.$$

Let us consider risk function R(t), as conditional probability density of software fault at time moment t, under condition, that there were no faults before this moment:

$$R(t) = \lim_{\Delta t \to 0} [Q(t, t+\Delta t)]/\Delta t P(t) = -[1/P(t)]dP(t)/dt. \qquad (2.1)$$

The risk function has dimension [1/time], and is very useful for basic distributions' classification. Distributions with increasing risk function correspond to the situations, when staticstical reliability characteristics get worse with time. And vise versa, distributions with decreasing risk function correspond to the opposite situation, when reliability is improving with time, as a result of error detection and correction process.

It is clear from equation (2.1), that $dP(t)/P(t) = -R(t)dt$, and, consequently,

$$\ln P(t) = -\int_0^t R(t)dt,$$

or

$$P(t) = \exp\left\{-\int_0^t R(t)dt\right\}. \qquad (2.2)$$

Equation (2.2) is one of the most important in reliability theory. It will be shown further, that various ways of risk function's behaviour in time yield various possibilities for building software reliability models. Error detection intensity (risk function), together with software faultless work probability and the number of

errors, remaining in the program, are the most important softwre reliability indicators.

A program run is a set of actions, including: inputting of one of the possible combinations E_i of the input data space E ($E_i \in E$); execution of the program, which ends either with obtaining a result $F(E_i)$ or with a fault.

For some of the sets E_i of the input data, its output result's ($F'(E_i)$) deviation from the required output $F(E_i)$ lies within an acceptable limit Δ_i, i.e. the following inequation holds:

$$\left|F'(E_i) - F(E_i)\right| \le \Delta_i, \qquad (2.3)$$

and for all the other E_i, forming a subset $E_i \subset E$, the program execution does not provide an acceptable result, i.e.

$$\left|F'(E_i) - F(E_i)\right| > \Delta_i, \qquad (2.4)$$

Cases, described by inequality (2.4), are also called program faults.

Let us consider a binary variable φ_i:

$$\varphi_i = \begin{cases} 0, \text{if (2.3) holds;} \\ 1, \text{otherwise.} \end{cases}$$

Then the statistical estimation of software fault probability during m runs will be:

$$\hat{Q}(m) = \frac{1}{m} \sum_{i=1}^{m} \varphi_i. \qquad (2.5)$$

Let us denote as δ acceptable $\hat{Q}$ estimation error of fault probability Q. Then the required number of program runs m must be proportional to the value of $(Q\delta^2)^{-1}$, where Q is the given software fault probability. It means, that if, for example, the relative error of estimation (2.5) is required to be $\delta = 10\% = 0.1$, and the required

(desired) value of $Q = 10^{-3}$, then the number of independent runs m should be not less, than $m \approx 10^3 \cdot 10^2 = 10^5$, which is, of course, not so easy to realize in practice. A solution to this problem can be applying the procedure of sequential Wald analysis (its examples are given in section 4.2).

And finally one more reliability metric, which will be used in this paper – average time of software working before a fault occurrence:

$$t_{av} = \int_0^\infty P(t)dt.$$

3. RISK FUNCTION BASED MODELS

3.1. Jelinski-Moranda model.

This is one of the first and simplest models of classical type, which was a basis for further development in this direction. The model was used in rather important and noticeable software projects, such as Apollo program (some of its modules) [2]. Jelinski-Moranda model is based on the following assumptions:

1. The intensity of error detection R(t) is proportional to the current number of errors in the program, i.e. initial number of errors minus number of already detected errors.
2. All errors occur with equal probability, and are independent on each other.
3. All errors are considered as equally serious.
4. Time, remaining until the next program fault, is distributed exponentially.
5. Software working environment is close to its real working environment.
6. Error correction is done without making any new errors.
7. R(t) = const between any two adjacent moments of error detection.

According to these assumptions, the risk function can be represented as:

$$R = K\left[B - (i-1)\right].$$

In this formula t is a random moment between (i-1)-th and i-th error detection; K is an unknown scaling coefficient; B is initial (also unknown) number of errors, remaining in the software.

Thus, if during a time interval of length t (i-1) errors were detected, it means, that B-(i-1) errors still remain undetected in the software.

Assuming, that

$$X_i = t_i - t_{i-1}, i = \overline{1,n}$$

and using assumption 7, and also equation (1.2), we can conclude, that all X_i have exponential distribution:

$$P(X_i) = \exp\{-K[B-(i-1)]X_i\}$$

and fault probability density equals, correspondingly,

$$q(X_i) = K[B-(i-1)]\exp\{-K[B-(i-1)]X_i\}$$

Then likelihood function (according to assumption 2) is

$$L(X_1,..,X_n) = \prod_{i=1}^{n} q(X_i) \qquad (3.1)$$

or, in terms of likelihood function logarithm, we have:

$$\ln L(X_1,..,X_n) = \sum_{i=1}^{n} [\ln(K(B-i+1) - K(B-i+1)X_i)] \qquad (3.2)$$

Likelihood function's maximum can be found using the following conditions:

$$\frac{\partial \ln L}{\partial B} = \sum_{i=1}^{n} \left[\frac{1}{B-i+1} - KX_i \right] = 0. \qquad (3.3)$$

$$\frac{\partial \ln L}{\partial K} = \sum_{i=1}^{n} \left[\frac{1}{K} - (B-i+1)X_i \right] = 0; \qquad (3.4)$$

From equation (3.3) we can get K maximum likelihood estimation:

$$K = \frac{n}{\sum_{i=1}^{n} (\hat{B}-1+i)X_i} = \frac{n}{(\hat{B}+1)\sum_{i=1}^{n} X_i - \sum_{i=1}^{n} iX_i}. \qquad (3.5)$$

Substituting equation (3.5) into (3.4), we find a non-linear equation for calculation of $\hat{B}$ – maximum likelihood estimation for B:

$$\sum_{i=1}^{n} \frac{1}{\hat{B} - i + 1} = \frac{n \sum_{i=1}^{n} X_i}{(\hat{B} + 1) \sum_{i=1}^{n} X_i - \sum_{i=1}^{n} iX_i} \tag{3.6}$$

Authors of papers [3,4] recommend to solve equation (3.6) using numerical methods, for example Newton-Rafson method. It is possible to simplify this equation before looking for a solution, if we write it as:

$$f_n(\hat{B} + 1) = g_n(\hat{B} + 1, A), \tag{3.7}$$

where

$$f_n(m) = \sum_{i=1}^{n} \frac{1}{m - i}; \quad g_n(m, A) = \frac{n}{m - A}; \quad m = \hat{B} + 1; \quad A = \frac{\sum_{i=1}^{n} iX_i}{\sum_{i=1}^{n} X_i}$$

Since only integer $\hat{B}$ values really make sense, functions from equation (3.7) can be considered only for integer arguments. Moreover, $m \geq n + 1$, because n errors are already detected.

Thus an estimation of B can be obtained by calculating of initial values of functions $f_n(m)$ and $g_n(m)$ for m=n+1, n+2…, and analyzing the difference $f_n(m)$-$g_n(m)$. Both f and g functions are monotonically decreasing on this m value range. This is obvious for $g_n(m)$, and $f_n(m)$ can be easily calculated by a recursive equation:

$$f_n(m+1) = f_n(m) - \left(\frac{1}{m - n} - \frac{1}{m} \right).$$

Since both right and left part of equation (3.7) are similarly monotonic, it causes a problem of unique solution existence, and solution existence in general. In paper [5] it is shown, that a finite solution $\hat{B}$ in the area of $\hat{B} \geq n$ exists if, and only if holds

$$\frac{\sum_{i=1}^{n}(i-1)X_i}{\sum_{i=1}^{n}(i-1)} > \frac{\sum_{i=1}^{n}X_i}{n} \tag{3.8}$$

Otherwise the only maximum likelihood estimation will be $\hat{B} = \infty$. Condition (3.8) can be rewritten in a more convenient way:

$$A > (n+1)/2, \tag{3.9}$$

where A is the same as in equation (3.7). It is important to notice, that A is an integral characteristic of n observations of software errors, and represents (in statistical sense) the set of intervals X_i between errors.

Another problem of equation (3.7) solution finding is related to instability of the estimation, because of possible multimodality of the likelihood function. If A is large enough, then the obtained estimation approaches n – the number of errors, detected by the current time. This makes an optimistic impression, that testing process is nearly completed, whereas the real B may be much larger than n. In paper [6] a solution to this problem is offered.

Let us consider an example of Jelinski-Moranda model usage, applying it to experimental data, obtained in software testing process, described in paper [7]. During 250 days 26 errors were detected; intervals between error detection are presented in table 3.1. For the given data we have n=26 and $\sum_{i=1}^{26} X_i = 250, \sum_{i=1}^{26} iX_i = 4258, A = 17.032$. Condition (3.9) holds, and thus maximum likelihood equation has a unique solution. Table 3.2 presents initial values of functions, comprising (3.7), for argument range $m \geq n+1$.

The best suitable solution of (3.7) is m=32 (semi bold line in the table gives minimum absolute value of the difference, which must be as close to zero as possible) , i.e. $\hat{B}$ = m-1=31. From equation (3.5) we have $\hat{K}$ = 0.007.

Average time $\hat{X}_{n+1}$ (time remaining until (n+1)-th error detection) is inverted estimated intensity for the previous error:

$$X_{n+1} = \frac{1}{\hat{z}(t_n)} = \frac{1}{\hat{K}(\hat{B} - n)}.$$

In this example, $X_{est\,27} = 29\ days$, and total time remaining until testing will be concluded, is $t_k = \sum_{i=27}^{31} \hat{X}_i = \frac{1}{\hat{K}} \sum_{i=1}^{5} \frac{1}{i} = 326\ days$. Although the obtained estimation of B is a bit overoptimistic (paper [7] contains information, that 8 more errors were detected on software testing and exploiting stages), detection of the first five errors took totally 290 days, which is rather near t_k value, predicted by the model.

Table 3.1. Intervals between error detection cases.

i	Xi	i	Xi	i	Xi	i	Xi
1	9	8	8	15	4	21	11
2	12	9	5	16	1	22	33
3	11	10	7	17	3	23	7
4	4	11	1	18	3	24	91
5	7	12	6	19	6	25	2
6	2	13	1	20	1	26	1
7	5	14	9				

Table 3.2. Function values

m	$f_{26}(m)$	$g_{26}(m, A)$	$f_{26}(m) - g_{26}(m, A)$
27	3.854	2.608	1.246
28	2.891	2.371	0.520
29	2.427	2.172	0.255
30	2.128	2.005	0.123
31	1.912	1.861	0.051
32	1.744	1.737	0.007
33	1.608	1.628	-0.020
34	1.496	1.532	-0.036

3.2 Simple exponential model.

The main difference of this model from Jelinski-Moranda model, discussed in previous section, is in not using assumption 7, and thus allowing the risk function not to be constant between error detection moments any more, so that it can change. Let N(t) be the number of errors, detected by moment t, and let the risk function be proportional to the number of errors, remaining in our software after moment t.

$$R(t) = K(B - N(t))$$

Let us differentiate both parts of this equation with respect to time:

$$\frac{\partial R(t)}{\partial t} = -K \frac{\partial N(t)}{\partial t}.$$

Taking into account, that $\partial N(t)/\partial t$ is R(t) (number of errors, detected per every time unit), we obtain a differential equation for R(t)

$$\frac{\partial R(t)}{\partial t} + KR(t) = 0. \tag{3.10}$$

If we consider initial conditions N(0)=0, R(0)=KB, then the solution for (3.10) will be

$$R(t) = KB\exp\{-Kt\} \tag{3.11}$$

Let us use the following designation: a = ln (KB); b = - K. Using these designations, equation (3.11) can be presented in form

$$R(t) = \exp\{a + bt\}.$$

Taking logarithm of the both parts of this equation, and using discrete t values, we obtain a set of equations:

$$\ln R(t_i) = a + bt_i;\ i = \overline{1,n}. \tag{3.12}$$

Equation set (3.12) can be presented in matrix form

$$AX = C,$$

where

$$A = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \Lambda & \Lambda \\ 1 & t_n \end{bmatrix};\quad X = \begin{bmatrix} a \\ b \end{bmatrix};$$

$$C = \begin{bmatrix} \ln R(t_1) \\ \ln R(t_2) \\ \Lambda \\ \ln R(t_1) \end{bmatrix}.$$

According to the least square method, we can transfer these equations to the normal form:

$$A^T AX = A^T C, \tag{3.13}$$

where T means matrix transposing.

The solution to (3.13) will is

$$X = \left[A^T A\right]^{-1} A^T C,$$

or, in more detail,

$$b = -\hat{K} = \frac{\sum_{i=1}^{n} t_i \ln R(t_i) - \frac{1}{n}\left(\sum_{i=1}^{n} \ln R(t_i)\right)\sum_{i=1}^{n} t_i}{\sum_{i=1}^{n} t_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} t_i\right)^2} \tag{3.14}$$

$$a = \frac{1}{n}\sum_{i=1}^{n} \ln R(t_i) - \left(\frac{1}{n}\sum_{i=1}^{n} t_i\right) b; \tag{3.15}$$

$$\hat{B} = \frac{\exp(a)}{\hat{K}} \tag{3.16}$$

Let us consider an example of using the simple exponential model. This example is based on a fragment of a dairy, containing information about changes made to really developed software, and some of these changes were classified as errors, according to the definition given in this paper. Volume of this software in Assembler code lines is 32 K [8]. Time unit, used in this experiment, was 1 month. During testing it was calculated, how many errors were detected for each of 20 month interval (R(ti)), included into the general debugging stage (table 3.3). Table 3.3 shows, that for the first month intensity was 53 errors per month, for the second month 74 errors per month etc. After calculation using equations (3.14), (3.15) and (3.16) the following estimations for B and K values were found: $\hat{B} = 713, \quad \hat{K} = 0.097.$

Thus, the estimated risk function for this example is

$$\hat{R}(t_i) = 69.1 \exp\{-0.097 t_i\}.$$

Based on this risk function estimation, a forecast was done for the next four months of debugging process (numbers 21,22,23,24 in table 3.4). At the same time, debugging and testing processes went on, and the next intensity values R(ti)

were obtained. As it can be seen from table 3.4, the forecast was confirmed well enough by experimental data.

Using (3.11) it is possible to determine time, needed to decrease error occurrence to one per month:

$$t^{(1)} = \ln\left(\hat{K}\hat{B}\right)/\hat{K} = a/\hat{K}.$$

For our example we have $t^{(1)} \approx 43.7$ (months from debugging start). Thus, we can conclude, that the model is working, in spite of some roughness and simplicity, and can be applied successfully in software development.

Table 3.3 Risk function estimation for each of 20 months.

t_i	$R(t_i)$	t_i	$R(t_i)$	t_i	$R(t_i)$	t_i	$R(t_i)$
1	53	6	50	11	24	16	34
2	74	7	14	12	36	17	23
3	38	8	43	13	46	18	11
4	21	9	69	14	11	19	1
5	27	10	48	15	28	20	10

Table 3.4 Estimations and observations for the next 4 months.

t_i	$\hat{R}(t_i)$	$R(t_i)$
21	9	4
22	8	9
23	7	11
24	7	9
Sum	31	33

3.3. Shick-Wolverton model.

This model, described in work [9], is based on the assumption, that risk function is proportional not only to the number of errors in software, but also to testing time duration. It is also assumed, that the longer software is tested, the more chances are to detect next errors, because some parts of the software are "cleaned", and it makes further testing process easier.

The model is based on the following assumptions:

1. All errors occur with equal probability, and are independent on each other.
2. All errors are considered to be equally serious
3. Software working environment is close to its real working environment
4. Error correction is done without making any new errors.

This model's risk function is

$$R(t) = K[B - (i-1)]X_i.$$

In this equation Xi is testing time from moment t_{i-1} of (i-1)-th error detection, to t_i - current moment.

Probability, that the software will have no faults on Xi interval, is:

$$P(X_i) = \exp\left\{-K[B-(i-1)]\frac{X_i^2}{2}\right\},$$

which yields fault probability density

$$q(X_i) = K[B-(i-1)]X_i \exp\left\{-K[B-(i-1)]\frac{X_i^2}{2}\right\}$$

Likelihood function for Xi is defined by equation (3.1). Differentiating its parts' logarithm with respect to K and B, we will obtain the following maximum likelihood conditions:

$$\frac{\partial \ln L}{\partial K} = \frac{n}{K} - \sum_{i=!}^{n} \frac{[B-i+1]X_i^2}{2} = 0;$$

$$\frac{\partial \ln L}{\partial B} = \sum_{i=1}^{n} \frac{1}{[B-i+1]} - K \sum_{i=1}^{n} \frac{X_i^2}{2} = 0.$$

From these equations we obtain K and B estimations:

$$\hat{K} = \frac{n}{\sum_{i=1}^{n} \left[\hat{B}-i+1\right]X_i^2}; \tag{3.17}$$

$$\sum_{i=1}^{n} \frac{1}{\left[\hat{B}-i+1\right]} = \hat{K} \frac{\sum_{i=1}^{n} X_i^2}{2};$$

$$t_{av} = \sqrt{\frac{\pi}{2\hat{K}\left(\hat{B}-n\right)}}.$$

For large values of n dispersions and covariances of estimations can be found using following equations:

$$\mathrm{var}\left[\hat{K}\right] = \frac{\sum_{i=1}^{n} \frac{1}{\left[\hat{B}-i+1\right]^2}}{D_1};$$

$$\mathrm{var}\left[\hat{B}\right] = \frac{n}{\hat{K}^2 D_1};$$

$$\mathrm{cov}\left[\hat{B},\hat{K}\right] = \frac{-\sum_{i=1}^{n} X_i^2}{D_1},$$

where

$$D_1 = \frac{\sum_{i=1}^{n} \frac{n}{\left[\hat{B} - i + 1\right]^2}}{\hat{K}^2 - \sum_{i=1}^{n} \frac{X_i^2}{2}}.$$

Now let us apply this model to the example, used for Jelinski-Moranda model (section 3.1). After (3.17) transform, we will have an equation, similar to equation (3.7):

$$f_n(m) = g_n(m, A'), \tag{3.18}$$

where

$$f_n(m) = \sum_{i=1}^{n} \frac{1}{m-i}; \quad g_n(m, A') = \frac{n}{m - A'}; \quad m = \hat{B} + 1; \quad A' = \frac{\sum_{i=1}^{n} iX_i^2}{\sum_{i=1}^{n} X_i^2}.$$

Equation (3.18) differs from (3.7) only by the formula for calculation of integral characteristic A'. Thus, solution analysis method, described in section 3.1, is also applicable for (3.18). For n=26 from table 3.1 we have $\sum_{i=1}^{26} X_i^2 = 10314$; $\sum_{i=1}^{26} iX_i^2 = 231828$; $A' = 22{,}477$. Condition (3.9) holds, so we can expect existence of a unique decision. Table 3.5 contains initial values of functions from (3.18) for $m \geq n+1$. It is easy to see, that the absolute value of $f_n(m) - g_n(m, A')$ is monotonically increasing, and does not have a minimum value for finite values of n. For this reason Shick-Wolverton model can be considered as unsuitable for the given experimental data. Besides this conclusion, it is also possible to add an upper limit A_{max} to (3.9) condition, situated, obviously, between A and A'. Calculations, done by a computer, have shown, that with value Amax = 20.25 (for n=26) $f_n(m) - g_n(m, A')$ does not change the sign.

Table 3.5. Initial values of functions from equation (3.18) .

m	$f_{26}(m)$	$g_{26}(m,A)$	$f_{26}(m)-g_{26}(m,A)$
27	3.854	5.748	-1.894
28	2.891	4.708	-1.817
29	2.427	3.986	-1.559
30	2.128	3.456	-1.328
60	0.574	0.693	-0.119

3.4. Lipov models.

These models, described in works [10,11], are a generalization of Jelinski-Moranda and Shick-Wolverton models. Opposed to these two models, Lipov models allow more than one error within one testing interval, and also correction of not all of the errors, detected in this interval is allowed. The first Lipov model (Jelinski-Moranda model generalization) is based on the following assumptions:

1. All errors occur with equal probability, and are independent on each other.
2. All errors are considered to be equally serious.
3. Error detection intensity is the same on the entire testing interval.
4. Software working environment is close to its real working environment.
5. On the i-th testing interval f_i errors are detected, but only n_i of them are corrected.

The last, fifth assumption makes this model quite different from previously considered models. Thus, risk function can be represented by the equation:

$$R(t) = K\left[B - F_{i-1}\right], \quad t_{i-1} \le t \le t_i,$$

where $F_{i-1} = \sum_{j=1}^{i-1} n_j$ - total number of errors, corrected by moment t_{i-1}, and t_i is the time of the i-th testing interval end (measured in a usual way or by the processor timer). Another difference from Jelinski-Moranda model is that t_i intervals are fixed, and not random.

Assuming, that the number of faults (errors, detected in the software) is a random value with Poisson distribution, we have the following equation for the likelihood function:

$$L(f_1, \mathrm{K}\ , f_M) = \prod_{i=1}^{m} \frac{\{K[B - F_{i-1}]X_i\}^{f_i} \exp\{-K[B - F_{i-1}]X_i\}}{f_i!}$$

Considering, just like in previous models, partial derivatives of lnL, and assigning them zero values, we can obtain a set of equations to find K and b maximum likelihood estimations. These estimations are:

$$K = \frac{F_M}{T_M\left[\hat{B}+1\right] - C_M};$$

$$\frac{F_M}{\hat{B}+1-\frac{C_M}{T_M}} = \sum_{i=1}^{M} \frac{f_i}{\hat{B} - F_{i-1}},$$

where $F_M = \sum_{i=1}^{M} f_i$ - total number of errors, detected within M testing intervals;

$C_M = \sum_{i=1}^{M} (F_{i-1}+1)X_i$; T_M - total duration of testing,

$$T_M = \sum_{i=1}^{M} X_i . \qquad (3.19)$$

M.Lipov gives the following expressions for variance and covariance of the found estimations:

$$\operatorname{var}\{\hat{K}\} = \frac{\sum_{i=1}^{M} \frac{f_i}{\left(\hat{B} - F_{i-1}\right)^2}}{D}; \tag{3.20}$$

$$\operatorname{var}\{\hat{B}\} = \frac{F_M}{\hat{K}^2 D}; \tag{3.21}$$

$$\operatorname{cov}(B,K) = \frac{\sum_{i=1}^{M} X_i}{D}; \tag{3.22}$$

$$D = \frac{F_M}{\hat{K}^2}\left(\sum_{i=1}^{M} \frac{f_i}{\left(\hat{B} - F_{i-1}\right)^2} \right) - \left(\sum_{i=1}^{M} X_i \right)^2. \tag{3.23}$$

If $f_i = 1 \quad \left(i = \overline{1,M}\right)$, i.e. on the given time interval only one error is detected and $F_{i-1} = i - 1$ (all detected errors have been corrected), then obtained estimations and their variance coincide with once of Jelinski-Moranda model.

The second Lipov model (Shick-Wolverton model generalization) is based on the following assumption. Error detection rate is proportional to the current number of errors in the software and total time, spent on its testing, including also "average" searching time for the error, detected in the current testing interval. Considering this, the risk function can be represented by expression:

$$R(t) = K\left(B - F_{i-1}\right)\left(\sum_{j=1}^{i-1} X_j + \frac{X_i}{2} \right), \quad t_{i-1} < t < t_i, \tag{3.24}$$

where F_{i-1} is total number of errors, corrected by t_{i-1} moment. Equation (3.24) differs from the first Lipov model by the second factor - $\left(\sum_{j=1}^{i-1} X_j + \frac{X_i}{2} \right)$, reflecting testing interval change.

Estimations, done similarly to previous ones, by maximum likelihood method, yield equations:

$$\left.\begin{aligned} \hat{K} &= \frac{F_M}{T_M^{'}\left[\hat{B}+1\right]-C_M^{'}}; \\ \frac{F_M}{\hat{B}+1-\frac{C_M^{'}}{T_M^{'}}} &= \sum_{i=1}^{M}\frac{f_i}{\hat{B}-F_{i-1}}, \end{aligned}\right\} \qquad (3.25)$$

where

$$C_M^{'} = \sum_{i=1}^{M}\left(F_{i-1}+1\right)X_i\left(\sum_{j=1}^{i-1}X_j+\frac{X_i}{2}\right);$$

$$T_M^{'} = \sum_{i=1}^{M}X_i\left(\sum_{j=1}^{i-1}X_j+\frac{X_i}{2}\right).$$

Variance of estimations $\hat{K}$ and $\hat{B}$ can be described by expressions (3.20) – (3.23), if in these expressions Xi will be replaced by $X_i\left(\sum_{j=1}^{i-1}X_j+\frac{X_i}{2}\right)$.

Let us consider as an example data from section 3.1 (Jelinski-Moranda model). Integral characteristic for the second Lipov model is

$$A^{''} = \frac{\sum_{i=1}^{n} iX_i\left(\sum_{j=1}^{i-1}X_j+\frac{X_i}{2}\right)}{\sum_{i=1}^{n}X_i\left(\sum_{j=1}^{i-1}X_j+\frac{X_i}{2}\right)},$$

and it equals A'' = 21.419 for n=26, which is greater, than A_{max}. Just like in the case of Shick-Wolverton model, there is no sensible solution, and the second Lipov model is inapplicable to the given data.

3.5. Geometrical models.

In this section three geometrical models will be discussed. The first and the third models were proposed by P.B. Moranda [12] (the first one is a modification of Jelinski-Moranda model). The second model, proposed by M. Lipov [11] , extends the first one.

In the first model it is assumed, that the initial number of errors in the program B is not a fixed value (not limited), and moreover, not all errors occur with equal probability. It is also assumed, that the longer software has been debugged, the more difficult it becomes to detect errors in this software, and thus software will never be absolutely free of errors. The basic assumptions of this model are as follows.

1. Total number of errors is unlimited;
2. Errors do not have equal probability;
3. Error detection process does not depend on errors;
4. Software working environment is close to its real working environment.;
5. Error detection intensity forms a geometrical progression, but between error detection cases the intensity does not change.

Based on these assumptions, risk function can be described by the following equation:

$$R(t) = DK^{i-1},$$

where t is the time interval between (i-1)-th and I-th error detection. Initial value of this function is R(0) = D, and the risk function is decreasing at the rate of geometrical progression (0<K<1) with error detection process. Changing rate of R(t) is proportional to inverted value of constant K:

$$\frac{DK^{i-1} - DK^{i-2}}{DK^{i} - DK^{i-1}} = \frac{1}{K} > 1,$$

which leads to decreasing of R(t) changing step size with error detection progress. Thus, later errors are more difficult to detect and they have less influence to error flow decreasing, than previous ones. If again we let $X_i = t_i - t_{i-1}$ (time interval between (i-1)-th and i-th error detection), then, in accordance with second and third assumptions, Xi are exponentially distributed with distribution density

$$q(X_i) = DK^{i-1} \exp\{- DK^{i-1} X_i\}.$$

Likelihood function for Xi is defined by expression (3.1), and its logarithm is

$$\ln L(X_1, \mathrm{K}\ , X_n) = n \ln D + \sum_{i=1}^{n} (i-1) \ln K - D \sum_{i=1}^{n} K^{i-1} X_i.$$

Maximum likelihood estimations for K and D can be found from the following equations:

$$\frac{\partial \ln L}{\partial D} = \frac{n}{D} - \sum_{i=1}^{n} K^{i-1} X_i = 0;$$

$$\frac{\partial \ln L}{\partial K} = \sum_{i=1}^{n} \frac{(i-1)}{K} - D \sum_{i=1}^{n} (i-1) K^{i-2} X_i = 0.$$

Solving these equations, we get:

$$\hat{D} = \frac{\hat{K} n}{\sum_{i=1}^{n} \hat{K}^{i} X_i}; \quad \frac{\sum_{i=1}^{n} i \hat{K}^{i} X_i}{\sum_{i=1}^{n} \hat{K}^{i} X_i} = \frac{n+1}{2}.$$

Average time from detection of n-th error until detection of (n+1)-th error can be estimates as follows.

$$\hat{t}_{av(n+1)} = 1 / \left(D \hat{K}^{n}\right).$$

This model does not allow to find, how many errors remain in the software, but it is possible to find its “purity level”. “Purity level” is a relation

$$r_{software} = \frac{R(t_0) - R(t_n)}{R(t_0)} = \frac{D - DK^n}{D} = 1 - K^n. \quad (3.26)$$

Maximum likelihood estimation of this value is $\hat{r}_{software} = 1 - \hat{K}^n$.

Now let us consider the second geometrical model (Lipov model modification). Its author aimed to weaken the assumption that the number of errors in the software is unlimited. Here the risk function is presented by the following expression:

$$R(t) = DK^{n_{i-1}}; \quad t_{i-1} \le t \le t_i,$$

where D and K (0<K<1) are defined similarly to the previous case, n_{i-1} is the total number of errors, detected on all testing intervals. Maximum likelihood estimations are the following:

$$\hat{D} = \frac{n}{\sum_{i=1}^{m} \hat{K}_{i-1}^{n} X_i};$$

$$\frac{1}{\hat{K}} \sum_{i=1}^{m} n_{i-1} = \hat{D} \sum_{i=1}^{m} n_{i-1} \hat{K}^{n_{i-1}-1} X_i,$$

where m is the number of testing intervals with lengths Xi ($i = \overline{1,m}$), and $n = \sum_{i=1}^{m} n_i$ is the total number of detected errors. It is noteworthy, that this model transforms into previously described model, if $n_{i-1} = i - 1$.

The probability of faultless working for our software is defined by the following expression:

$$\hat{P}(t) = \exp\left\{-\hat{D}\left(\hat{K}^{n_m}\right)t\right\}, \quad t_m < t,$$

and the average time from (m-1)-th to m-th fault is defined by expression:

$$t_{av(m)} = \frac{1}{\hat{D}\hat{K}^{n_m}}.$$

And finally the last, third geometrical model, proposed for the case, when software error notifications come periodically. Here, just like in previous model, only the total number of errors, detected during each testing interval, is required. However, opposed to the previous geometrical models, this model uses an assumption, that all testing intervals have the same length, measured by one day, one month etc.

This model is applicable, when the interval length is small compared to the entire testing period. The model is based on the following assumptions:

1. The number of errors in software is unlimited;
2. Error detection is an independent process;
3. Error detection is equally probable for all errors;
4. Software working environment is close to its real working environment;
5. The number of errors f_i, detected during the i-th testing interval, has Poisson distribution with parameter $D\phi^{i-1}$, where D is the initial error detection rate, and ϕ is a scaling constant coefficient $(0 < \phi < 1)$;
6. Every detected error is either corrected, or not taken into account any more.

From assumption 5 it is clear, that error detection rate is changing as a geometrical progression, i.e.

$$R(t) = D\Phi^{i-1} \tag{3.27}$$

for $t_{i-1} < t < t_i$, i.e. for i-th testing interval (interval length is fixed here).

Likelihood function for the number of detected errors f_i looks like:

$$L(f_1,\mathrm{K}\ ,f_m)=\prod_{i=1}^{m}\frac{\left(D\Phi^{i-1}\right)^{f_i}\exp\left\{-D\Phi^{i-1}\right\}}{f_i!}$$

and, consequently,

$$\ln L=\sum_{i=1}^{m}f_i\ln D+\sum_{i=1}^{m}f_i(i-1)\ln\Phi-\sum_{i=1}^{m}\ln(f_i!)-\sum_{i=1}^{m}D\Phi^{i-1}.$$

Then the equation, from which maximum likelihood estimations can be found, is:

$$\frac{\partial\ln L}{\partial D}=\frac{\sum_{i=1}^{m}f_i}{D}-\sum_{i=1}^{m}\Phi^{i-1}=0;$$

$$\frac{\partial\ln L}{\partial\Phi}=\frac{\sum_{i=1}^{m}f_i(i-1)}{\Phi}-D\sum_{i=1}^{m}(i-1)\Phi^{i-2}=0.$$

From these equations we get the following expressions for the estimations:

$$\hat{D}=\left(\sum_{i=1}^{m}f_i\right)\Big/\left(\sum_{i=1}^{m}\hat{\Phi}^{i-1}\right);\tag{3.28}$$

$$\frac{\sum_{i=1}^{m}f_i}{\sum_{i=1}^{m}f_i(i-1)}=\frac{\left(1-\hat{\Phi}^m\right)\left(1-\hat{\Phi}\right)}{\hat{\Phi}+(m-1)\hat{\Phi}^{m+1}-m\hat{\Phi}^m}.\tag{3.29}$$

Variance of these estimations can be approximately evaluated as follows:

$$\mathrm{var}\left|\hat{D}\right|=\frac{\hat{D}}{\Delta\hat{\Phi}^3}\left[\sum_{i=0}^{m-1}i\hat{\Phi}^{i+1}+\sum_{i=1}^{m}(i-1)(i-2)\hat{\Phi}^i\right];$$

$$\mathrm{var}\left\{\hat{\Phi}\right\}=\sum_{i=1}^{m}\hat{\Phi}^{i-1}\Big/\Delta\hat{D};$$

$$\mathrm{cov}\left(\hat{\Phi},\hat{D}\right)=-\left(\sum_{i=0}^{m-1}i\Phi^{i-1}\right)\Big/\Delta;$$

$$\Delta = \frac{\sum_{i=1}^{m} \hat{\Phi}^{i-1}}{\hat{\Phi}^3} \left[\sum_{i=0}^{m-1} i\hat{\Phi}^{i+1} + \sum_{i=1}^{m} (i-1)(i-2)\hat{\Phi}^{i} \right] - \left(\sum_{i=0}^{m-1} i\hat{\Phi}^{i-1} \right)^2.$$

As an example let us consider application of the third geometrical model to the data from section 3.2. We have m=20, $f_i \equiv R(t_i)$ in table 3.3. Expression (3.29) can be rewritten as

$$\frac{m\Phi^{m+1} - (m+1)\Phi^m + 1}{(1-\Phi^m)(1-\Phi)} = \frac{\sum_{i=1}^{m} if_i}{\sum_{i=1}^{m} f_i} = A, \qquad (3.30)$$

where A is an integral characteristic for the statistical data of the example. Looking for the solution of the polynomials in the left part of the equation (3.30) for $0 < \Phi < 1$, it can be found, that the only solution is $\hat{\Phi} = 0.9392$. From (3.28) also D estimation can be found: $\hat{D} = 56.224$. Purity level (3.26) is $r_{software} = 1 - \hat{\Phi}^2 = 0.118$.

Using (3.27), it is possible to calculate the moment, when average error occurrence intensity reaches one error per month rate:

$$t^{(1)} = 1 - \ln \hat{D} / \ln \hat{\Phi} \approx 65.2,$$

i.e. after 65 months of testing. Compared to the conclusions of the example in section 3.2, this estimation is rather pessimistic.

3.6. Schneidewind model.

This model [13] includes the third geometrical model as a particular case. The basic approach in this model is that occurrence of later errors has more significant

influence on error prediction process. Let us assume, that there are m testing intervals, and let f_i errors be detected in the I-th interval. Then there are three possible approaches:

1. Use data about errors on all m intervals;
2. Kick out data about all errors, detected during first (s-1) intervals, and use only data of intervals from s-th to m-th;
3. Use summarized number of errors, detected from the first to (s-1)-th intervals, i.e. $F_{s-1} = \sum_{i=1}^{s-1} f_i$, and individual errors from s-th to m-th intervals.

It is proposed to use approach 1 in those cases, when data from all the intervals are necessary to have for future software state prediction. Approach 2 - when there are reasons to consider, that some significant change has happened in error detection process, and only the last m – (s-1) interval data are needed for prediction. And finally, approach 3 is a compromise between first two approaches.

The model is based on the following assumptions:

1. The number of errors on a testing interval does not depend on the number of errors on other testing intervals;
2. The number of detected errors is decreasing from one interval to another;
3. All testing intervals are of the same length;
4. Error detection rate is proportional to the number of errors, contained in the software at the current time moment.

Error detection process is supposed to be non-uniform Poisson process with exponentially decreasing error detection rate. Decreasing rate is given by the formula:

$$d_i = d \exp\{-\beta t\}$$

for i-th interval, where d > 0 and $\beta > 0$ are the model's constant characteristics.

Total number of errors is defined by the following expression:

$$D_i = \frac{\alpha}{\beta}\left[1 - \exp\{-\beta\, i\}\right],$$

and thus the number of errors in the I-th interval is

$$m_i = D_i - D_{i-1} = \frac{\alpha}{\beta}\left[\exp\left(-\beta(i-1)\right) - \exp\left(-\beta\, i\right)\right]. \tag{3.31}$$

Assuming, that Poisson's process takes place, we obtain likelihood function:

$$L(f_1, \mathrm{K}\ , f_m) = \frac{M_{s-1}^{F_{s-1}} \exp\{-M_{s-1}\}}{F_{s-1}!} \prod_{i=1}^{m} \frac{m_i^{f_i} \exp\{-m_i\}}{f_i!},$$

where M_{s-1} is the number of errors in the interval from 1 to (s-1), and $2 \le s \le m$ (s is an integer value).

Using the fact, that

$$M_{s-1} = \left(\alpha \big/ \beta\right)\left[1 - \exp\{-(s-1)\beta\}\right],$$

we obtain:

$$\hat{\alpha} = \frac{\left(\sum_{i=1}^{m} f_i\right)\hat{\beta}}{1 - \exp\{-\hat{\beta}\, m\}}; \quad \hat{\beta} = \ln(y);$$

Here y is the solution of the polynomial equation:

$$\frac{(s-1)F_{s-1}}{y^{s-1} - 1} + \frac{F_{s,m}}{y - 1} - \frac{mF_m}{y^m - 1} = G,$$

where

$$G = \sum_{i=0}^{m-s} (s + i - 1) f_{s+i};\ F_{s,m} = \sum_{i=1}^{m} f_i.$$

If s=1 and $F_{s,m} = F_m = \sum_{i=1}^{m} f_i$, then

$$F_m / (y - 1) - mF_m / \left(y^m - 1\right) = G.$$

In this case we have the following simplification for y=1:

$$Gy^{m+1} - (G + F_m)y^m + (mF_m - G)y + (G + F_m - mF_m) = 0.$$

There is a certain relation between this model and the third geometrical model. Really, if we assume, that

$$\alpha = D(-\ln\Phi)/(1-D); \quad \beta = -\ln\Phi$$

and substitute them into (3.31), we will obtain the third geometrical model. And vice versa, if we assume, that

$$D = (\alpha/\beta)[1 - \exp(-\beta)]; \quad \Phi = \exp(-\beta)$$

and substitute them into equation (3.27), then we will obtain Schneidewind model.

3.7. Model, based on Weibull distribution law.

Risk function for this model is presented as:

$$R(t) = (a/b)(t/b)^{a-1},$$

where a>0 and b>0 are the model constant characteristics, and $t \geq 0$ has the meaning of faultless work interval. If a>1, then error detection rate grows with time, and if a<1, then it decreases; if a=1 then risk function is constant. Fault distribution density (time until the first error occurrence) is described by Weibull distribution:

$$q(t) = (a/b)(t/b)^{a-1} \exp\left\{-(t/b)^{a-1}\right\},$$

and, correspondingly, for fault probability we have the Weibull distribution function:

$$Q(t) = \int_0^t q(x)dx = 1 - \exp\left\{-(t/b)^a\right\} \qquad (3.32)$$

Average time until fault occurrence in a general case can be expressed by the gamma-function:

$$t_{av} = \int_0^{\infty} P(t)dt = \frac{b}{a}\Gamma(1/a).$$

Work [36] contains the following method for evaluation of unknown constants a and b. Let n_i be the total number of errors on each testing interval, d_i $\left(i = \overline{1,k}\right)$ - the length of this testing interval, k – the total number of intervals, and M – the total number of errors, detected by the current moment - $M = \sum_{i=1}^{k} n_i$. Let us use the least square method to obtain a and b estimations.

Let us consider, that:

$$m = a;\ b_0 = -a\ln b;\ \hat{Q}(t_i) = \left(\sum_{j=1}^{i} n_j\right)(M+1),$$

where $\hat{Q}(t_i)$ is the normalized total number of errors, detected by moment t_i - the beginning of i-th testing interval (empirical distribution function).

Let

$$X_i = \ln\left(\sum_{j=1}^{i} d_i\right). \tag{3.33}$$

From expression (3.32) we have:

$$1/(1-Q(t)) = \exp\{(t/b)^a\}.$$

Taking logarithm of the both parts of this equation, we obtain:

$$\ln\left[\ln\left(\frac{1}{1-Q(t)}\right)\right] = a\ln t - a\ln b$$

or

$$Y = mX + b_{0,} \tag{3.34}$$

where

$$Y = \ln[\ln(1 - Q(t))], \quad X = \ln t .$$

Least square estimation of m and b_0 can be obtained from the discrete set of points:

$$X_1, Y_1; X_2, Y_2; \text{K } X_k, Y_k,$$

where

$$Y_i = \ln\left[\ln\left(\frac{1}{1 - \hat{Q}(t_i)} \right) \right], \tag{3.35}$$

and X_i is defined by equation (3.33).

Then the least square estimations for the parameters of the line, described by equation (3.34), are defined by the following dependencies:

$$\hat{m} = \frac{\sum_{i=1}^{k} (Y_i - \overline{Y})(X_i - \overline{X})}{\sum_{i=1}^{k} (X_i - \overline{X})^2};$$

$$\hat{b}_0 = \overline{Y} - \hat{m}\overline{X},$$

where

$$\overline{X} = \frac{1}{k} \sum_{i=1}^{k} X_i; \quad \overline{Y} = \frac{1}{k} \sum_{i=1}^{k} Y_i .$$

The estimations for a and b are, correspondingly:

$$\hat{a} = \hat{m}; \quad \hat{b} = \exp\{-\hat{b}_0 / \hat{m}\},$$

which yields the following expression for the probability of software faultless working:

$$\hat{P}(t) = \exp\left\{ -\left(\frac{t}{\hat{b}} \right)^{\hat{a}} \right\}; \tag{3.36}$$

$$t_{av} = \frac{\hat{b}}{\hat{a}}\Gamma\left(\frac{1}{\hat{a}}\right). \qquad (3.37)$$

Let us consider an example of application of this model to the data from section 3.1. We have all $n_i = 1$ and $d_i = X_i$ in table 3.1. The empirical distribution function looks like $F(t_i) = \frac{i}{n+1}$, where n=26.

Applying the least square method for a number of points, defined by expressions (3.34) and (3.35), we can obtain the estimations of Weibull distribution parameters: a = 1,458 and b = 109,001. Then from equation (3.36) we can define the probability of software faultless work after $t_{26} = 250$ days from the beginning of testing: $P(t_{26}) = 0{,}035$, and average time before fault occurrence (3.37) is $t_{av} = 98{,}84$ days.

3.8. Duan model.

This model [22] was proposed for reliability growth evaluation purposes. To estimate this growth the relation of error detection intensity to the total testing time is considered. Assumptions, on which this model is based, are the following:

1. Detection of all errors is equally probable, and all errors are considered to be equally serious;
2. Error occurrence is an independent process;
3. The total number of errors, detected by moment t, B(t), is distributed according to Poisson law with average value m(t), where $m(t) = \alpha\, t^{\beta}$.

Consequently,

$$\frac{m(t)}{t} = \frac{\alpha\, t^{\beta}}{t} = \frac{\text{exp}ected\ number\ of\ errors\ for\ period\ t}{total\ testing\ \ time\ period\ t}.$$

Taking logarithm of the both parts of this equation, we obtain:

$$Y = \ln\left(\frac{m(t)}{t}\right) = \ln\alpha + (\beta - 1)\ln t,$$

or

$$Y = a + bX,$$

where $a = \ln\alpha; \quad b = \beta - 1; \quad X = \ln t.$

Least square estimations for a and b look like:

$$\hat{a} = \overline{Y} - \hat{b}\overline{X};$$

$$\hat{b} = \frac{n\left(\sum_{i=1}^{n} Y_i X_i\right) - \left(\sum_{i=1}^{n} X_i\right)\left(\sum_{i=1}^{n} Y_i\right)}{n\sum_{i=1}^{n} X_i^2 - \left(\sum_{i=1}^{n} X_i\right)^2},$$

where $X_i = \ln(t_i); \quad Y_i = \ln\left(\frac{i}{t_i}\right).$

The only necessary thing to know for the realization of this model is the time moments of error occurrence - t_i.

4. ERROR "SEEDING" AND TAGGING AND INPUT DATA STRUCTURE BASED MODELS

Methods of software reliability estimation, based on error "seeding" and labeling models, will be considered on three model examples (Mills, Beisin and simple heuristic model). Methods, based on input data structure, are based on the model, proposed by Nelson, which was applied in practice only after its development based on application of sequential statistical Wald analysis. Also in this section will be considered two other models – reliability growth model and model of the IBM company.

4.1 Error "seeding" and tagging models.

Let us start considering this group of models with Mills model. According to the methodology, proposed by Mills [14], a program is initially "seeded" by a known number of certain errors - M. The basic assumption of this model is that the "seeded" errors are distributed in the same way, as natural errors of the software, and, consequently, the probability of detection of a "seeded" error is the same as for a natural error. After that the software testing process starts. Let (m+v) errors be detected during the testing process, where m – is the number of generated or "seeded" errors, and v – the number of natural errors. Then, according to the maximum likelihood method, the initial number of errors in the software can be estimated as follows: $B_M = (M/m)v$.

A serious disadvantage of this model is the assumption concerning similarity of distribution of generated and natural errors, which is impossible to verify, espe-

cially on later stages of software development, when many of simple errors (such, as syntax errors, for example) are already excluded, and only the most difficult for detection errors still remain in the software.

Now let us consider the Beisin model. This model is described in work [17]. Let a software product contain Nk instructions. From these Nk instructions n instructions are selected randomly, and an error is injected into each of these n instructions. After that r instructions are randomly selected for testing. If during testing m "seeded" and v natural errors have been detected, it means, that by the maximum likelihood method the initial number of errors can be estimated as follows:

$$B_{Bei\sin} = \left\rceil v\frac{N_k - n + 1}{r - m}\right\lceil ,$$

where by][is denoted the integer part of a number.

Procedures in this and in the previously described model have something in common with technology of animal labeling in a certain region, and releasing them on the same territory again. Later, based on the total number of caught animals, and the percent of labeled animals among them, it is possible to make conclusions about the population of this animal in the region.

When using this kind of procedure, tagging level (i.e. average number of tagged errors) has to exceed 20, to be sure that the obtained estimations are objective enough. These procedures can be applied at any stage after software coding is completed.

Now let us consider the simple heuristic model. This model was proposed by B. Rudner [15]. This model allows to get rid of the main disadvantage of the Mills

model. Here testing is realized in parallel way, by two independent groups of software developers.

Let m_1 and m_2 be the number of errors, detected by the first and the second groups, correspondingly, and $m_{1,2}$ will be the number of common errors, i.e. errors, detected by both groups, first and second. The initial number of errors, contained by software, can be estimated using hypergeometrical distribution and maximum likelihood method:

$$B_{Rudner} = \frac{m_1 m_2}{m_{1,2}}.$$

4.2 Nelson model. Applying sequential Wald analysis to reduce the number of program runs.

The model, proposed by E.Nelson [16], is based on taking into account of the program input data structure. In the easiest case (as it was shown in section 1.2), an input data set E_i, selected randomly from input space E, has a priori equal probability with other sets, contained by E, and this yields an estimation of fault probability, described by (2.5).

If we also consider different probabilities p_i for each input data set, then estimation (2.5) will be transformed as follows.

$$\hat{Q}(m) = \sum_{i=1}^{m} p_i \varphi_i,$$

which allows to write the following expression for a successful run probability:

$$p = \sum_{i=1}^{m} p_i (1 - \varphi_i). \qquad (4.1)$$

Then, the probability of successful performing of m runs, with input data sets for each run selected independently, and in accordance with the distribution, defined by (4.1), will equal:

$$\hat{P}(m) = p^m = \left(\prod_{i=1}^{m} p_i (1 - \varphi_i) \right)^m . \qquad (4.2)$$

Model (4.2) allows to give the following definition to software reliability: reliability of software is the probability of m faultless runs of this software.

The next step in development of this model will be to admit, that in practice it is typically not valid that input data set selection is independent for each run. Exceptions of this rule may be only run sequences with gradually increasing of an input variable, or using a certain order of procedures (like in some real-time systems, for example).

Taking these circumstances into account, it is necessary to redefine the distribution (4.1) in terms of probabilities p_{ij} of selection E_i data set for j-th run from a certain run sequence. Then the fault probability for the j-th run is:

$$Q_j = \prod_{i=1}^{m} p_{ij} \varphi_i .$$

Correspondingly, the probability of faultless execution of m runs can be estimated by the following expression:

$$\hat{P}(m) = \prod_{j=1}^{m} (1 - Q_j).$$

This expression can be also presented in another form:

$$\hat{P}(m) = \exp\left\{ \sum_{j=1}^{m} \ln(1 - Q_i) \right\},$$

and if $Q_i << 1$, then $\hat{P}(m) = \exp\left\{ -\sum_{j=1}^{m} Q_j \right\}$.

Let us denote Δt_j - time of j-th software run execution, and $t_j = \sum_{k=1}^{j} \Delta t_k$ - summarized time, spent on execution of j runs, and let us use the following risk function expression:

$$R(t_j) = -\frac{\ln(1 - Q_j)}{\Delta t_j},$$

then we have:

$$\hat{P}(m) = \exp\left\{ -\sum_{j=1}^{m} R(t_j)\Delta t_j \right\}. \qquad (4.3)$$

If $\Delta t_j \xrightarrow{m \to \infty} 0$, then the sum is transforming into an integral, and equation (4.3) turns into the fundamental relation (1.2):

$$P(t) = \exp\left\{ -\int_0^t R(t)dt \right\}$$

of the main reliability rule.

Two categories of difficulties occur when applying Nelson model in practice. The first one is related to finding of p_{ij} - input data distribution parameters. The second one is related to the necessity of execution of a significant number of software runs (tens and hundreds of thousands) to obtain acceptable accuracy of the corresponding reliability estimations (fault or faultless work probability).

In practice p_{ij} is defined by splitting the whole input data space into subspaces, and definition of probabilities, that a selected data set Ei may belong to a particular subspace. Estimation of these probabilities is based on the evaluating of the

probabilities of occurrence of each set Ei in the real working environment of the software. After probabilities p_{ij} are found, a random selection of m input data sets in accordance with distribution p_{ij} is made, using an artificial random number generator. It is also important, that experiments (runs) should be held without interruption, and errors should not be corrected until the completion of all m runs. Now about the second difficulty of this model – a large number of required runs. This problem can be solved applying a well-known in statistics method, the sequential analysis, proposed by Wald [17]. Briefly the idea of this method is the following (applied to software reliability problem [18]). In the sequential analysis it is assumed, that if successful run probability P is close enough to p_0, then the risk of taking a wrong decision is sufficiently small. A wrong decision is understood here as either the decision to reject a working program, or the decision to accept an unreliable program. A programmer, testing the software, has to define the following variables before the start of experiments:

1. minimal acceptable probability of software faultless working: Pmin = p';
2. probability Pmax = p'', which makes him is nearly sure, that software will succeed in testing, and thus $p' < p_0 < p''$;
3. probability α of the first type error, i.e. probability of rejecting reliable software $(\alpha = \Pr\{P \geq p''\})$;
4. probability β of the second type error, i.e. probability of acceptance of unreliable software $(\beta = \Pr\{P \leq p'\})$.

The choice of α and β should be based on compromise considerations, because with decreasing these values the number of required runs is increased.

The idea of sequential analysis for H_0 hypothesis (software reliability $P = p_0$) is comprised of verification of two competing hypothesis: $H'(P = p')$ and

$H''(P = p'')$. Here faultless work probability is still understood as probability of m faultless runs' execution. Let us consider a binary variable φ_i :

$$\varphi_i = \begin{cases} 1, & \text{if (1.3) holds, i.e. the run is successful;} \\ 0, & \text{if (1.4) holds, i.e. a fault occured during the run.} \end{cases}$$

Then, based on m runs' results, we will obtain a set of ones and zeros: $\varphi_{1,}\varphi_2$,K φ_m. The probability of m faultless runs can be presented as the probability of obtaining a set $\varphi_{1,}\varphi_2$,K φ_m, in which d_m elements equal to zero:

$$P(m) = p^{m-d_m}(1-p)^{d_m}.$$

If hypothesis H' holds, then the probability of obtaining this kind of set P'(m) will be the following:

$$P'(m) = (p')^{m-d_m}(1-p')^{d_m}.$$

Similarly, if hypothesis H'' holds, then we have

$$P''(m) = (p'')^{m-d_m}(1-p'')^{d_m}.$$

Let us construct the likelihood relation:

$$\frac{P''(m)}{P'(m)} = \left(\frac{p''}{p'}\right)^{m-d_m}\left(\frac{1-p''}{1-p'}\right)^{d_m}.$$

Sequential analysis (i.e. testing process) should be continued until the following inequalities start to hold:

$$\frac{\beta}{1-\alpha} < \frac{P''(m)}{P'(m)} < \frac{1-\beta}{\alpha}. \tag{4.4}$$

If after m-th run we will have

$$\frac{P''(m)}{P'(m)} \geq \frac{1-\beta}{\alpha}, \tag{4.5}$$

then software can be considered unreliable.

However, if we have after m-th run, that

$$\frac{\beta}{1-\alpha} \le \frac{P^{"}(m)}{P^{'}(m)}, \tag{4.6}$$

then software can be accepted as reliable.

The inequalities (4.4), (4.5) and (4.6) can be rewritten in a more comprehensive, and thus more acceptable way. After taking logarithms and some other simple transformations, we can obtain new variables:

$$c_0 = \ln(1-p^{"}) - \ln(1-p^{'}) - \ln p^{"} + \ln p^{'};$$

$$\gamma = (\ln p^{'} - \ln p^{"})/c_0;$$

$$a_0 = (\ln\beta - \ln(1-\alpha))/c_0;$$

$$b_0 = (\ln(1-\beta) - \ln\alpha)/c_0;$$

in coordinates m, d_m we can build two lines $a_m = a_0 + m\gamma; \quad b_m = b_0 + m\gamma$.

Now for the number of unsatisfactory experiments (experiments, resulting in faults) we have relations $a_m < d_m < b_m$. If $d_m \ge b_m$, then software is unreliable, and if $d_m \le a_m$, then software can be accepted as reliable.

Thus it is necessary to perform the following steps:

- define values of $p^{'}, p^{"}, \alpha, \beta$ before the start of experiments;
- build lines a_m and b_m;
- during the testing place points m, d_m on the same 2D coordinate plane;
- if the current point m, d_m lies higher than b_m line, than the testing process is interrupted, and the software is considered to be unreliable (not satisfying the given reliability requirements).
- If the current point m, d_m lies lower, than a_m line, then testing process is interrupted, and software is accepted as reliable.

- If the current point m, d_m lies between the two lines, than the process has to be continued.

It is possible to evaluate the average number of runs $\overline{m'}$ provided that hypothesis H' is true, and the average number of runs $\overline{m''}$ provided that hypothesis H'' is true. In the first case we have:

$$\overline{m'} = [\beta \ln B + (1-\beta)\ln A] / \overline{\mu'},$$

where $A = (1-\beta)/\alpha$; $B = \beta/(1-\alpha)$; $\overline{\mu'}$ - mathematical expectation of random variable $\mu = \ln P''(m) - \ln P'(m)$, provided that H' hypothesis is true. It is possible to show, that $\overline{\mu'} = -\ln p'$.

In the second case we have:

$$\overline{m''} = [\alpha \ln A + (1-\alpha)\ln B] / \overline{\mu''},$$

where $\overline{\mu''}$ is the mathematical expectation of μ, provided that hypothesis H'' is true. In this case $\overline{\mu''} = -\ln p''$.

Let us consider an example of average run number estimating. Let $\alpha = 10^{-4}$, $\beta = 10^{-2}$; $p' = 0.98$, $p'' = 0.99$. Then we have:

$$A = \frac{1-10^{-2}}{10^{-4}} = 9999; \ln A = 9.21;$$

$$B = \frac{0.01}{0.9999} = 0.1; \ln B = -4.6;$$

$$\ln p' = -0.02; \ln p'' = -0.01;$$

$$\overline{m'} = \frac{10^{-2}(-4.6) + 0.99 \cdot 9.21}{0.02} = 449;$$

$$\overline{m''} = \frac{10^{-4} \cdot 9.21 + 0.9999(-4.6)}{-0.01} = 458.$$

If we exchange the values of α and β, and leave p' and p'' unchanged, then we will obtain the following result: $\overline{m'} = 228; \overline{m''} = 902$.

4.3 La Padula growth model.

The idea of the model, proposed by L. La Padula [19], is in constructing reliability curve by the least square method, according to the observed numbers of faults on various testing stages. In this model testing is arranged as series of N stages. Every stage is characterized by some correction or modification of the program.

On each stage $n_i \left(i = \overline{1, N}\right)$ tests are performed, s_i of which have been successful. The number of tests on a stage is not fixed in advance. After the completion of N-th stage (which is not known in advance by itself) the data is substituted into the growth curve equation:

$$P(i) = P(u) - A/i,$$

where P(i) is software reliability during the i-th testing stage.; P(u) is the limit of P(i) if $i \to \infty$; A is a growth parameter.

If A>0, then software reliability is increasing, otherwise it is decreasing. Let us use the least square method for P(u) and A evaluation. The following value has to be minimized:

$$s = \sum_{i=1}^{N} \left(P(i) - \frac{s_i}{n_i} \right)^2 = \sum_{i=1}^{N} \left(P(u) - \frac{A}{i} - \frac{s_i}{n_i} \right)^2.$$

Calculating partial derivatives of s with respect to P(u) and A, we obtain a two equations, from which we can get the least square estimations:

$$\hat{A} = \frac{N\left[\sum_{i=1}^{N} \frac{s_i}{n_i^2} - \left(\sum_{i=1}^{N} \frac{s_i}{n_i}\right)\left(\sum_{i=1}^{N} \frac{1}{i}\right)\right]}{\left(\sum_{i=1}^{N} \frac{1}{i}\right)^2 - N\left(\sum_{i=1}^{N} \frac{1}{i^2}\right)};$$

$$\hat{P}(u) = \frac{1}{N}\left[\hat{A}\sum_{i=1}^{N} \frac{1}{i} + \sum_{i=1}^{N} \frac{s_i}{n_i}\right].$$

Naturally, it is necessary to know the number of runs n_i, executed on each stage, and the number of successful runs on each stage - s_i. A testing stage is supposed to end and the next testing stage to begin when some changes are made in the software, related to error correction, or some program modifications, or to some other reasons.

4.4 Model , proposed by IBM company professionals.

During the exploitation of the current software version by the user, the developer typically performs active maintenance of this software, i.e. makes some amendments and bug fixes in this version without expecting the user's requirement for it. And this maintenance may include also creation of new functions for the software. From some moment, when the developer considers his tasks to be completed, passive maintenance starts, i.e. corrections are done only after the user's requests.

During software maintenance a considerable amount of new errors is added to every new version, together with amendments and modifications, which leads to corrections in the next version also. The developers of well known American company IBM tried to forecast such kind of corrections from version to version, based on a large number of experimental data, gathered during operational system OS/360 maintenance [3,4]. The model, proposed by IBM developers, is based on

observations of the software system development history, and the hypothesis of statistical stability of dependence between some parameters, characterizing various system versions. As the basic measurement unit of software complexity was selected a software module. Module creation rules were standardised.

The volume of the i-th version is presented by the number Mi of modules, included into this version. When releasing the i -th version, a parameter OCM_i is changed (the number of old corrected modules), and a parameter NM_i is added (the number of new modules), so that $M_i = M_{i-1} + NM_i$.

During the i-th version improving (the period of (i+1)-th version preparation) further correction of modules is happening. These corrected modules are divided into two groups: the first group is characterised by a parameter MCM_i – multiple correction modules (10 or more corrections per module), and the second group – by CM_i, modules with number of corrections less than 10. This classification is needed for simpler calculations, and also because of the fact, that small number of corrections is done to most of the modules.

It is also noteworthy, that the CM group does not require any special debugging tools, whereas the MCM group may require some additional efforts during debugging.

During analysis of the maintenance history of IBM's OS/360 it was established, that there is a considerable correlation between parameters, characterizing the extent of changes and (correspondingly) error level (in CM and MCM groups), and parameters, characterizing complexity and volume of the next version (OCM, NM). Applied to OS/VS1 this statement looked as follows.

$$CM_i = 0.9NM_i + 0.15OCM_i. \tag{4.7}$$

$$MCM_i = 0.15NM_i + 0.06OCM_i. \quad (4.8)$$

If we suppose that terms "correction" and "error" are identical, than the model of evaluation of the total number of errors in software, proposed by IBM professionals and based on the given explanations, looks as follows:

$$B = COR_i = 23MCM_i + 2CM_i, \quad (4.9)$$

where COR_i is the total number of corrections, made in the modules (or, in other words, the total number of expected errors), and coefficients 23 and 2 are the average amount of corrections per module in MCM and CM groups, correspondingly.

The forecast is based on the planned number of corrections to old modules and added new modules (OCM_i, NM_i) for the realization of the new required functions of the software. If the number of actually done corrections is less than the predicted number of corrections, then there are probably still a lot of undetected errors in the software. The following conclusions can be made from the IBM's model:

- during passive maintenance stage ($CM_i = 0$, OCM_i is small), then the number of corrected modules and the number of corrections inside these modules are decreasing rapidly from one version to another;
- the number of expected errors in the next version may increase compared to the older version, if many enough of old modules have been changed (OCM), and/or many enough new modules have been added (NM).
- adding of new modules has a stronger effect on new errors' number increasing, than corrections, made to old modules; at the same time, if it is possible to create a new module instead of making corrections to a few older modules (5 or more), this leads to decreasing of the number of expected errors. In other words, on a certain maintenance stage it becomes no longer effective to modify old modules, and creation of a new module is required.

Let us consider an example, illustrating application of the reliability model, proposed by IBM, which is described by equations (4.7), (4.8) and (4.9). Work [19] contains data related to maintenance of 19 versions of the OS/360 system. By the time of 19-th version release, its volume reached 4800 modules, which was four times as much as the volume of the first version, released 4.3 years earlier. Table 4.1 contains the initial data of the last five versions.

To use the data from table 4.1, it is necessary to make some additional calculations. Knowing the size of the 19-th version, it is possible to calculate volumes Mi of the four previous versions using the first table row. This allows to evaluate the total number of modified modules ($NM_i + OCM_i$), using their part of the total volume (row 2 of in table 4.1) . For verification of the calculation correctness it is possible to multiply the data of the 3-rd and 4-th rows, or use the last row. This kind of "verification" gives a less accurate result, because of decreased number of decimals in the initial data. After finding the values of NM_i and OCM_i we are able to use the model, i.e. the equations (4.7), (4.8) and (4.9). The final results are summarized in table 4.2. Unfortunately, work [20] doesn't allow to make any conclusions about correspondence of scientific estimations and actual error amount, because of the company's security considerations. In work [21], however, it is mentioned, that every new version of OS/360 contains more than 1000 errors; it is also mentioned, that one of the last versions contained 11000 errors. This confirms indirectly the estimations, given in table 4.2, and it is an argument in favor of the reliability model, proposed by IBM.

T able 4.1. The initial data used for model application.

Parameter	Version number				
	15	16	17	18	19
System growth (number of new modules NM_i)	135	171	183	354	410
Changed modules (part of the total volume	0.33	0.43	0.48	0.50	0.55
Change rate (modules/day)	12.5	12.0	9.6	9.9	9.6
Duration of a version development (days)	96	137	201	221	275
OCM_i/CM_i relation	7.9	8.6	10.0	5.1	5.4

T able 4.2. The results of IBM's model application.

Parameter	Version number				
	15	16	17	18	19
Version volume M_i number of modules)	3682	3853	4036	4390	4800
Total number of corrections ($NM_i + OCM_i$)	1215	1657	1937	2195	2650
Number of new modules (NM_i)	135	171	183	354	410
Number of old changed modules (OCM_i)	1080	1486	1754	1841	2240
Number of corrected modules (CM_i)	284	377	428	595	705
Number of many times corrected modules (MCM_i)	85	115	133	164	196
Total number of corrected modules COR_i (the number of expected errors in the software)	2523	3399	3915	4962	5918

5. RESULTS

This section contains the testing results for some of the models, described in section 3. Experiments were done with three models: Jelinski-Moranda model, the first geometrical model, and also simple exponential model. The data, which was used in these tests, were of two kinds – real data, taken from literature sources, and generated data. In this situation it was not possible to obtain real data by experiments, because it would require long-time observations of significant software projects. Data from section 3.1 example was used to test Jelinski-Moranda model and the first geometrical model, and simple exponential model was tested using example from section 3.3. The rest of testing data was generated using three different distributions: exponential, normal and uniform. It was done with the purpose to investigate influence of distribution change on models' functionality. All the tests were done not only for a single data set, but also for its reduced forms. In the first experiment the whole data set was used, but later – 95, 90, 85, 80 etc. percents of this data set. The purpose of it was to investigate, how sensitive are the models to the input data set size. For most of the experiments reduction of the data set was regular – from 100 to 5 percents, with step 5. For Jelinski-Moranda model, however, it was not possible, because of convergence problems. This model gave a finite output not for all possible input sets, but just for some of them, so tests were done for those percents of initial size, with which the model converged.

In order to estimate the extent of models' sensibility to the data size change, correlation coefficient was calculated between percent of data used and the model's output. A correlation coefficient is a number between -1 and 1 which measures the degree to which two variables are linearly related. If there is perfect linear rela-

tionship with positive slope between the two variables, we have a correlation coefficient of 1; if there is positive correlation, whenever one variable has a high (low) value, so does the other. If there is a perfect linear relationship with negative slope between the two variables, we have a correlation coefficient of -1; if there is negative correlation, whenever one variable has a high (low) value, the other has a low (high) value. A correlation coefficient of 0 means that there is no linear relationship between the variables.

Output values of the models, for which correlation coefficients were calculated and diagrams built, were not the same for all three models, because there is some difference in their way of result's presentation form. For Jelinski-Moranda model the output values of predicted total number of remaining errors and time left until the next error detection were used. For the first geometrical model – average time until the next error detection and software purity level (this model is based on the assumption, that the total number of errors is unlimited, and thus it can't be used to predict its value). For the simple exponential model as the output values were used the total number of errors in the program and (the most sensible for this model) the number of errors, detected on the next testing interval.

Section 5.1 reviews results of the Jelinski-Moranda model testing, section 5.2 – the first geometrical model testing results, and section 5.3 -results for the simple exponential model. In section 5.4 the testing results are discussed

5.1 Testing results for Jelinski-Moranda model.

The Jelinski-Moranda model was tested on four different input data sets: the data set used in example from section 3.1 of 26 samples and three generated data sets

of 50 samples each. They were generated using exponential distribution (with mu = 0.5), normal distribution (with mu = 50 and sigma = 20), and uniform distribution with A = 0 and B = 100. Figures 5.1-5.4 depict the resulting diagrams, obtained from testing of data sets from section 3.1, exponentially , normally, and uniformly distributed data sets, correspondingly. Table 5.1 contains correlation coefficients, calculated for all the testing sets, between the percent of data size, used for testing, and the output values of the model (i.e. total number of errors and time left until the next error occurrence). A complete set of testing results for this model can be found in Appendix 1.

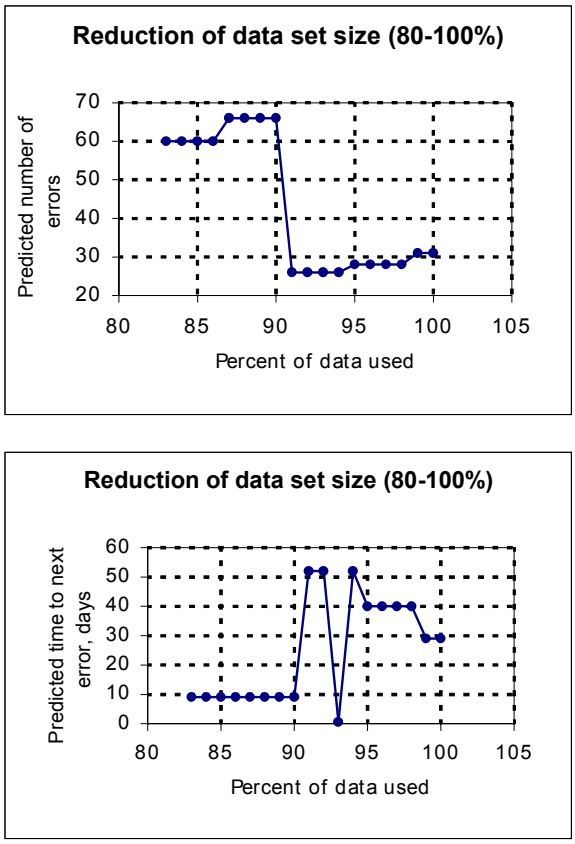
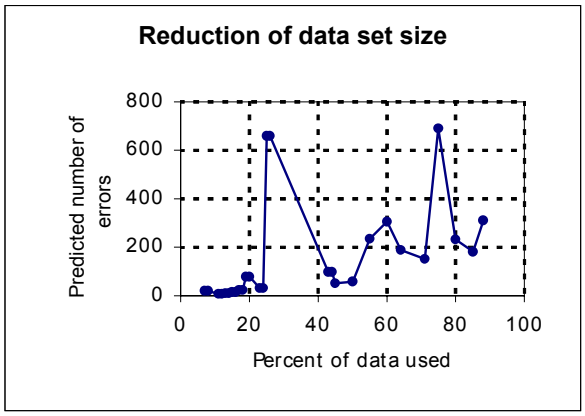


Figure 5.1. Test results for the dataset from section 3.1. Diagrams for predicted number of errors and time to the next error.



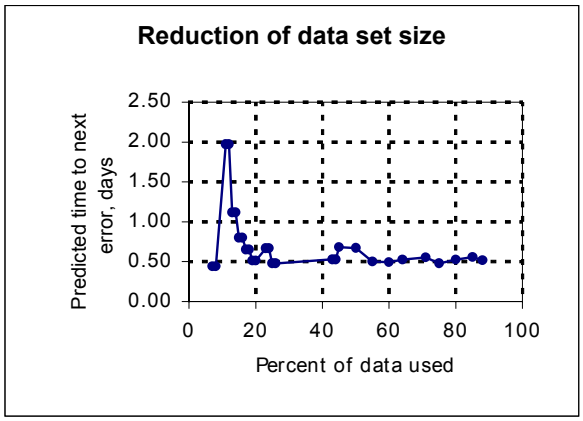


Figure 5.2. Test results for the exponentially distributed dataset. Diagrams for predicted number of errors and time to the next error.

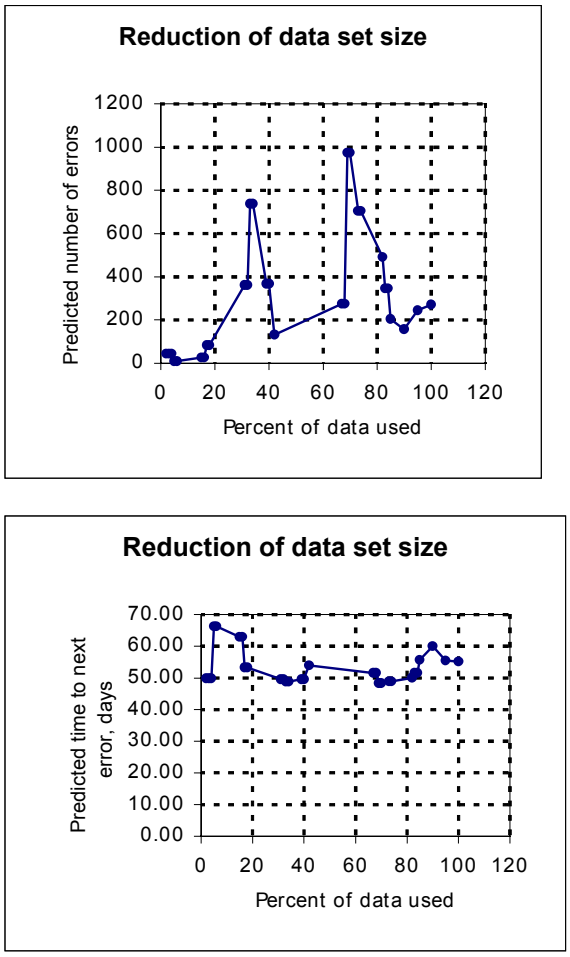


Figure 5.3. Test results for the normally distributed dataset. Diagrams for predicted number of errors and time to the next error.

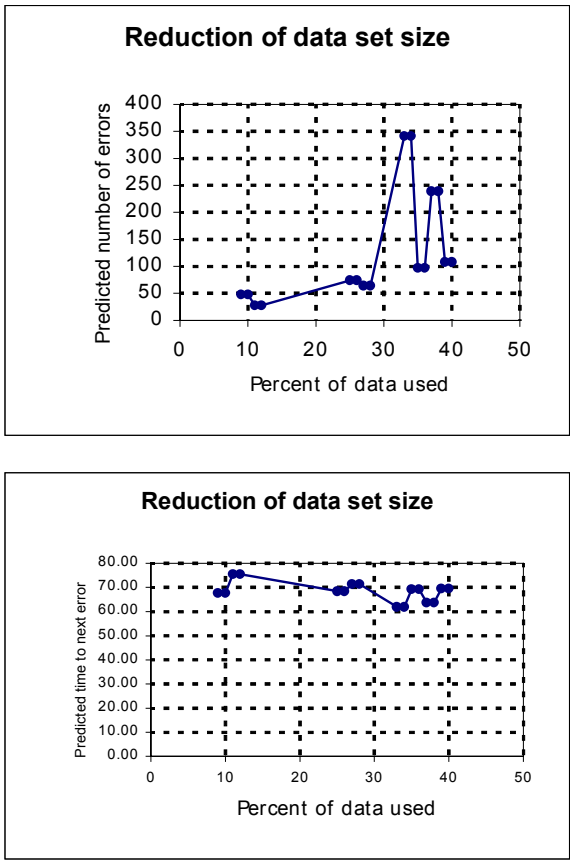


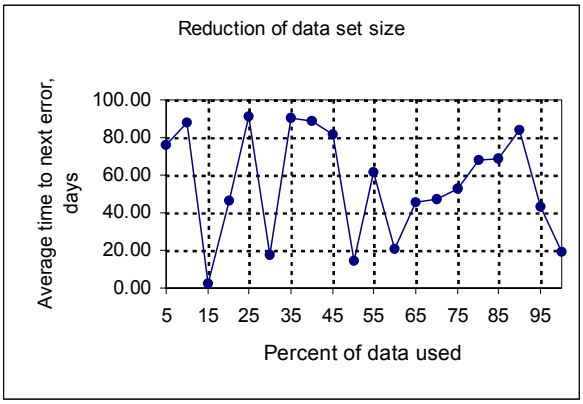
Figure 5.4. Test results for the uniformly distributed dataset. Diagrams for predicted number of errors and time to the next error.

Table 5.1. Correlation coefficients for Jelinski-Moranda model.

	Total number of errors, B	Time to next error, Xnext
Example from 3.1	-0.8	0.6
Exponential	0.4	-0.4
Normal	0.5	-0.2
Uniform	0.6	-0.5

5.2 Testing results for the first geometrical model.

The first geometrical model was tested on the same datasets, as the Jelinski – Moranda model. The only difference in testing was that the percent values of used data size was changed on regular basis – from 100 to 5 with step 5. It became possible for this model, because it has no convergence problem – the output can be obtained for any data set. Figures 5.5 – 5.8 depict the diagrams, obtained after testing the same four examples, as for Jelinski-Moranda model. Table 5.2 contains the correlation coefficients between the percent of data size used and the model's output values: average time to the next error occurrence and software purity level. A complete set of testing results for this model can be found in Appendix 2.



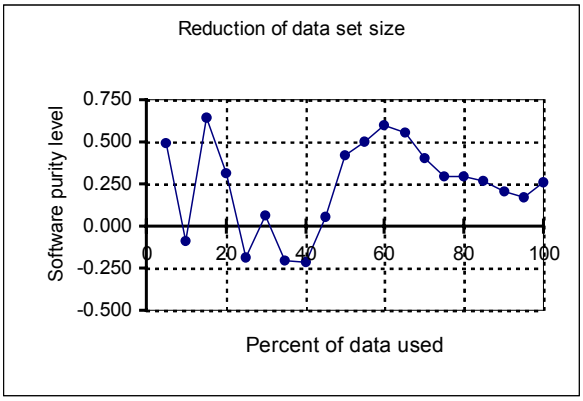
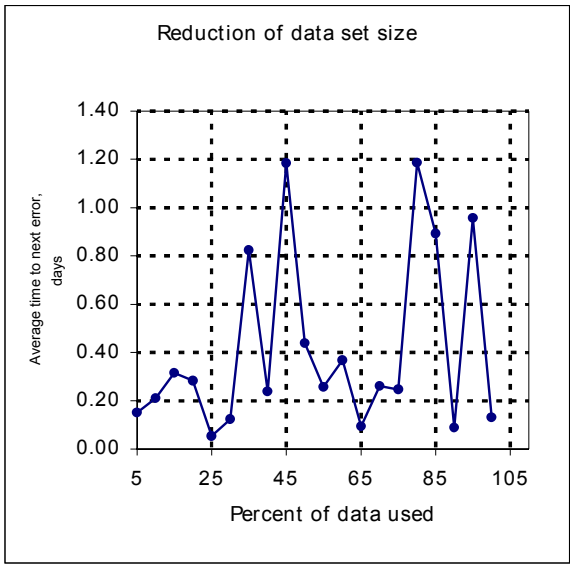


Figure 5.5. Test results for the dataset from section 3.1. Diagrams for predicted time to the next error occurrence and software purity level.



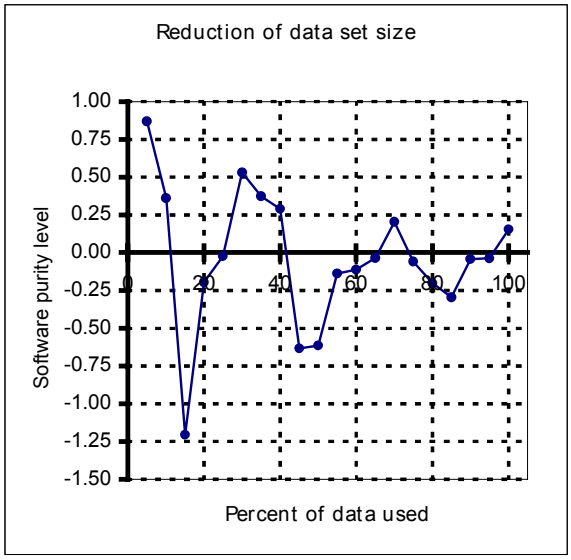


Figure 5.6. Test results for the exponentially distributed dataset. Diagrams for predicted time to the next error occurrence and software purity level.

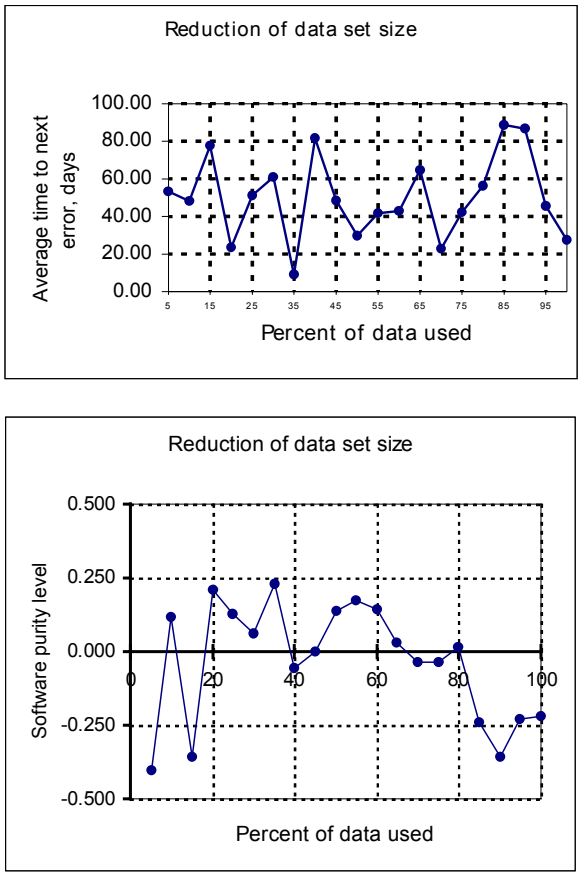


Figure 5.7. Test results for the normally distributed dataset. Diagrams for predicted time to the next error occurrence and software purity level.

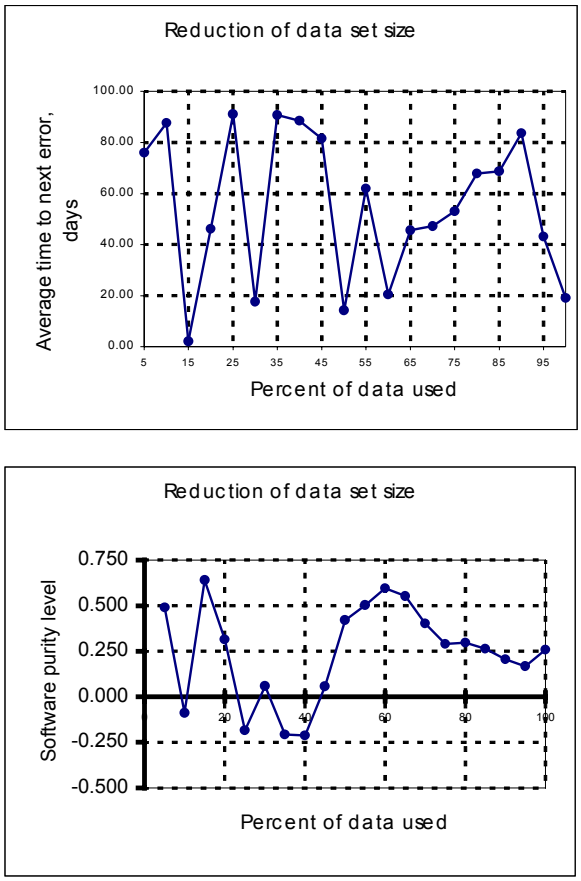


Figure 5.8. Test results for the normally distributed dataset. Diagrams for predicted time to the next error occurrence and software purity level.

Table 5.2. Correlation coefficients for the first geometrical model.

	Time to next error, t_{avr}	Software purity level, r
Example from 3.1	0.03	-0.55
Exponential	0.03	-0.12
Normal	0.07	-0.25
Uniform	-0.14	0.17

5.3 Testing results for the simple exponential model.

This model was also tested on four data sets, but the first example was not from section 3.1, but from section 3.3., however it is also a real-life example. The other three examples were also of three distributions (exponential, normal and uniform), but the sets were generated separately (thus, the datasets were not the same, as for the two previous models, only distribution was the same). They were generated using exponential distribution (with mu = 5), normal distribution (with mu = 50 and sigma = 20), and uniform distribution with A = 1 and B = 100. Figures 5.9-5.12 depict the testing results for the four datasets, and table 5.3 contains correlation coefficients for the predicted values of total number of errors and next value of the risk function (or probable next number of detected errors). A complete set of testing results for this model can be found in Appendix 3.

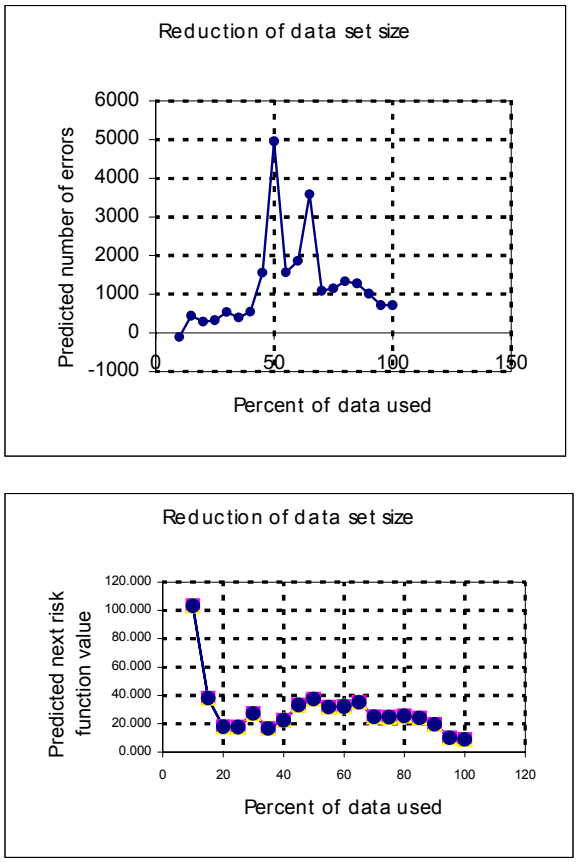


Figure 5.9. Test results for the dataset from section 3.3. Diagrams for the total number of errors and predicted time to the next error occurrence.

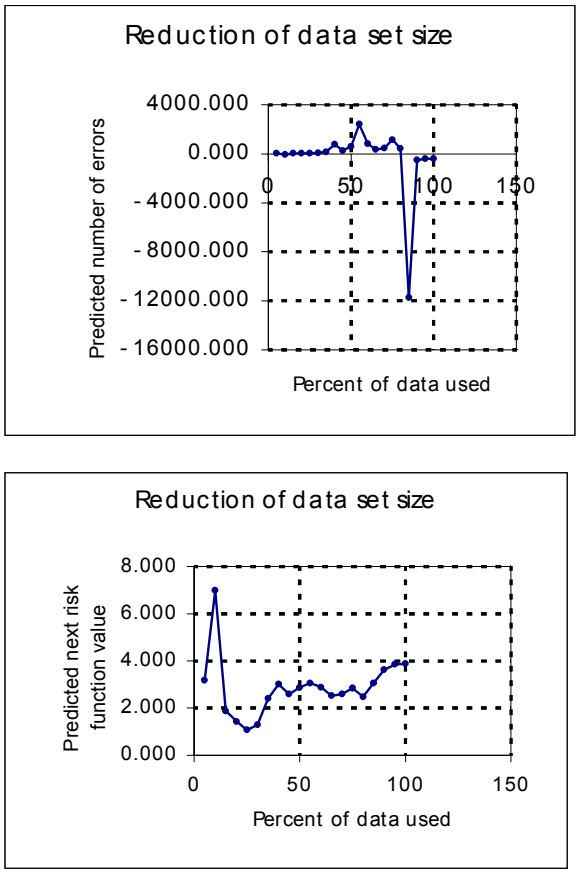


Figure 5.10. Test results for the dataset of exponential distribution. Diagrams for the total number of errors and predicted time to the next error occurrence.

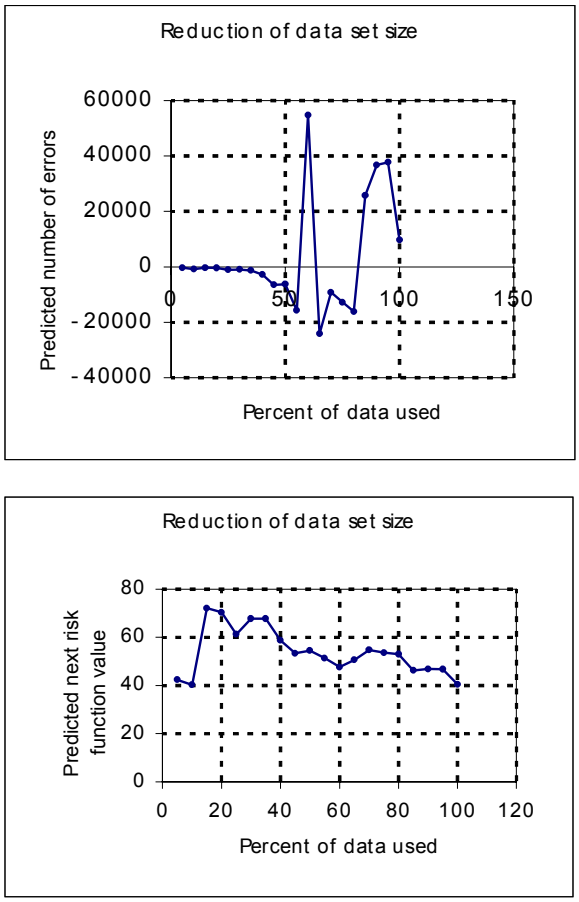
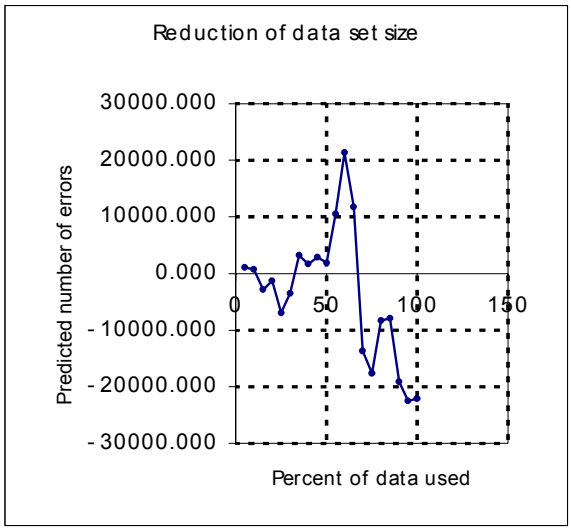


Figure 5.11. Test results for the dataset of normal distribution. Diagrams for the total number of errors and predicted time to the next error occurrence.



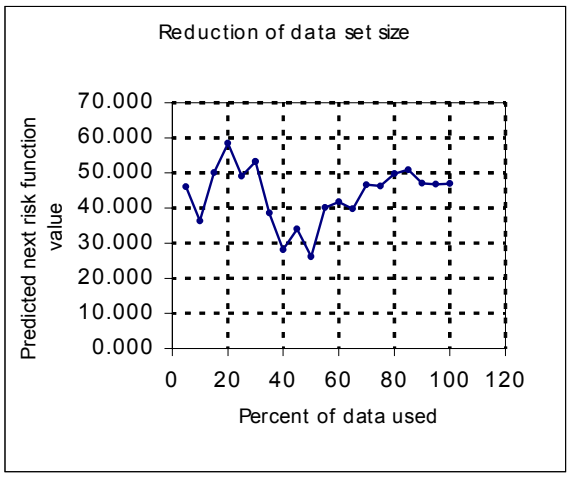


Figure 5.12. Test results for the dataset of uniform distribution. Diagrams for the total number of errors and predicted time to the next error occurrence.

Table 5.3. Correlation coefficients for the simple exponential model.

	Total number of errors, B	Next risk function value, Rnext
Example from 3.3	0.24	-0.49
Exponential	-0.25	0.13
Normal	0.33	-0.45
Uniform	-0.51	0.08

5.4 Discussion of the results.

For Jelinski-Moranda model it is possible to say, that the worst for its convergence is uniform distribution. It was almost impossible to make it work with this data set, and only for a few cases (a few values of the percent of used data size) the experiments succeeded. Exponential distribution was much better (which was expectable, because the model is built under the assumption of this distribution), and for normal distribution the result for the total number of errors was quite close to one for exponential case. Even the shape of the corresponding diagrams is nearly similar. The other two models are much less sensitive to data distribution, in the sense, that they don't have convergence problems at all. For any input data set they give some output. The examples, taken from "real life" (opposed to the generated ones) can be used to confirm that all of the three models are usable, because the results given by them are close to reality (see sections 3.1 and 3.3 for details about these examples). As to the reduction of data set size, the best results have been shown by the simple exponential model. The result, obtained by this model (R_{next} – the next predicted risk function value) is the most stable with respect to the size of input dataset, which can be easily seen from the diagrams

size of input dataset, which can be easily seen from the diagrams (figures 9-12). For Jelinski-Moranda model it seems, that the predicted time left until the next error occurrence is more stable to the data set size changes, than the total number of errors (figures 2-4). It can be well seen from diagrams in figures 2-4, but not in figure 5.1, because in the test case, depicted in figure 5.1 (example from section 3.1) the data size was initially small (26 samples), and after reduction of this data size the model did not converge in most of the cases. The first geometrical model seems to be the most sensitive to the data size change, because it gave the most unstable results (figures 5-8). This can be also concluded from the correlation coefficients (table 11). The coefficients for t_{avr} (average time until the next error) have quite small values, which means, that the result is changing nearly independently on the data size used, and thus even a small change in the initial data can yield an unpredictable influence on the predicted t_{avr}. Of course, sensitivity to the data size in this context means sensitivity to initial data in general, because if the data set is reduced, not only its size, but also its content is changed.

6. CONCLUSION

In this paper a survey of two software reliability model groups has been provided: risk-function based models, and models, based on error "seeding" and tagging and input space structure. Also practical work was done on investigation of the first group models, and its results were presented and discussed. The investigated models were Jelinski-Moranda model, the first geometrical model and the simple exponential model. The results were discussed from two viewpoints: how the models handle various input data distributions, and how sensitive they are to changes in the input data size. The most certain conclusions on the distribution problem were done about Jelinski-Moranda model – uniform distribution is definitely not suitable for this model, and exponential distribution seems to be the best for it. The simple exponential model turned out to be the most stable, i.e. not very sensitive to the changes in the input data size, and the first geometrical model – vice versa, the most unstable. For Jelinski-Moranda model the total number of errors turned out to be more sensitive to the data size changes, than the time, left until the next error detection.

It is impossible to say, which of the models is the best applicable for software reliability evaluation. In practice it is recommended to "try on" each of the preselected models to the particular problem (to the error flow), and choose the most adequate one. It is important to pay attention to the assumptions, on which the models are based, and their requirements to the input data. After that each of the suitable models can be applied to the problem. In order to be able to estimate, if the result was correct or not (to select the best suitable of the models), it is reasonable to apply the models not to all data available, but only to some part of it (about

75-80%), and the rest would serve as a test set, to verify the correctness of the results, given by the models. The model, which gives the best result, can be applied further for this problem.

REFERENCES

[1] Pressman, R.: Software Engineering: A Practitioner's Approach. New York: McGraw-Hill, Inc, 1997.

[2] Jelinski Z., Moranda P.B. Software reliability research // Statistical Computer Performance Evaluation/ W. Freibenger. – New York: Academic Press, 1972. P. 465-484.

[3] Липаев В. В. Качество программного обеспечения. - М. Финансы и статистика, 1983. – 263 с.

[4] Майерс Г. Надежность программного обеспечения. – М.: Мир, 1980. – 360 с.

[5] Littlewood B., Verall B. J. A Bayesian reliability growth model for Computer Software // Journal of the Royal Statistical Society. - Ser.C. – Vol. 22. – 1978. – N3. – P. 332 – 346.

[6] Forman E. H., Singpurwalla N. D. Optimal Time Intervals for Testing Hypotheses on Computer Software Error // IEEE Transactions on Reliability. – Vol R-28. – 1979. – N3. – P. 250 – 258.

[7] Goel A., Okumoto K. Time-Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures // IEEE Transactions on Reliability. – Vol. R-28, 1979. – N3. – P. 206 - 211

[8] Никандров А. В., Полонников Р. И. Анализ надежности программного обеспечения // Вопросы радиоэлектроники. – 1989. – N 3 (14). С. 88 –92.

[9] Schick G. J., Wolverton R. W. An Analysis of Computing Software Reliability Models // IEEE Transactions on Software Engineering. Vol. SE – 4, 1978. – N 2. – P. 104 – 120.

[10] Lipow M. Model of Software Reliability // Proceedings of the Winter Meeting of the Aerospace Division of the American Society of Mechanical Engineers. 1978. –78 – WA/Aero-18. – P. 1-11

[11] Lipow M. Some Variations of a Model of Software Time-to-Failure // TRW System Group. – August, 1974. – Correspondence ML-74-2260-1 – P. 9-21

[12] Moranda P.B. Event-Altered Rate Models for General Reliability Analysis // IEEE Transactions on Reliability. Vol. R-28, 1979. N5. P. 376-381.

[13] Schneidewind N.F. Analysis of Error Processes in Computer Software // Sigplan Not. Vol. 10, 1975. N6. P. 337 - 346

[14] Mills H. D. On the statistical Validation of Computer Programs // FSC-72-6015, IBM Federal System Division. -

[15] Rudner B. Seeding / Tagging Estimations of Software: Models and Estimates, RADC TR 77-15, Rome Air Development Center A036655, 1977.

[16] Nelson E. Estimating Software Reliability From Test Data // Microelectronics and Reliability. Vol. 17, 1978. – N1. – P. 67 – 73.

[17] Вальд А. Последовательный анализ. – М.: Физматгиз, 1960. – 3 28 с.

[18] Бочаров В. П. Последовательный анализ надежности программной продукции. // Программирование. – 1988. – N4. – С. 93 – 98.

[19] La Padula L. J. Engineering of Quality Software Systems, vol. V111. – Software Reliability Modeling and Measurement Techniques. // Mitre Corp., RADC TR 74 – 325. Rome Air Development Center, 1976.

[20] Леман М. М. Программы, жизненные циклы и законы эволюции ПО // Гр. Института инженеров по электротехнике и радиотехнике. – 1980. – Т. 68. – N9. – с. 26 – 45.

[21] Йодан Э. Структурное проектирование и конструирование программ. – М.: Мир, 1979. 416 с.

[22] Duan J. T. Learning Curve Approach to Reliability Monitoring // EEE Transactions on Aerospace. – Vol. 2, 1964. – P. 563 – 566.

[23] Fenton N. "Software Metrics: A Rigorous Approach", Chapman & Hall, 1991.

[24] Kan S. H. "Metrics and Models in Software Quality Engineering", Addison-Wesley Publishing Compagny, 1994.

[25] Schneidewind N. "Methodology for Validating Software Metrics", IEEE Transactions on Software Engineering, Vol. 18, no. 5, , May 1992. - pp. 410-442.