

# **SEMINAR WORK „SOFTWARE RELIABILITY MODELS“**

Jindra Bruderová  
jindra.bruderova@uhk.cz  
March 4, 2008

<b>JELINSKI – MORANDA MODEL</b> .....	3
Introduction .....	3
Software reliability .....	3
Software Reliability Models types .....	4
Jelinski – Moranda model .....	5
Jelinski – Moranda model assumptions .....	5
Failure intensity function .....	6
Estimation of model parameters.....	7
<b>PRACTICAL PART</b> .....	9
CASRE .....	9
CASRE benefits .....	9
Input .....	10
Output.....	10
Conclusion.....	10
<b>OTHERS MODELS</b> .....	17
Software Reliability Growth Models .....	17
Musa’s Basic Model.....	17
Musa/Okumoto Logarithmic Poisson Model .....	17
Comparison of these models .....	18
Failure count models .....	18
The Goel-Okumoto model .....	18
Error sendings models .....	18
Mills error sending model .....	18
References .....	20

# JELINSKI – MORANDA MODEL

## Introduction

Since time when computers took part in our daily life software and hardware developers have to be very careful of system reliability and safety. A lot of people are not aware of how many so common things are managed by computers. If we assume for example system for medical monitoring or system in the airport the break down is not thinkable. And therefore computer reliability became one of the most important part of each system. In this way we have to consider hardware reliability but reliability of software as well. In the next text I will focus on software reliability.

## Software reliability

*Reliability of software is possibility of no failure during a given operating time in a specified environment.* It is one of the most important software attribute. But real world programme without failure is not achievable. Commonly we can buy software which is still developing and failures are repaired in operation. Great example of this approach is operating system MS Windows which you can buy and from time to time have to download pack of repairings for improving your system.

Reliability can be adressed by two ways: models and tools. Models predict reliability of a system based upon failure data whereas tools implement reliability models using software.

Models have been developed to measure, estimate and predict the reliability of computer software. It is great way how to ensure that product reliability meet the consumers needs, predict and manage costs connected with the product. Anyway more economical is to solve the problems in the phase of development then later and just models should guarantee us the cheaper solution.

At the end of software development life cycle newly creating software should be tested. Testing is complex progress. It is based on simulation and thus the test provides us data

to estimate the reliability. The major goal of a software testing process is to find and fix as many defects as possible and release product with a reasonable reliability. We never can catch out all errors but we want to catch the majority of them.

Steps of software reliability test:

1. Simulation of usage in order to specify the usage
2. Statistical testing based on the specified usage
3. Modeling the test result to estimate current level of reliability
4. Prediction, using reliability modeling

### **Software Reliability Models types**

The models that are used in the software engineering can be divided into four classes that represent what they focus on.

Time between failures models

These types of models will be used if we need to see how the reliability changes over time. From these models we can see whether the failure intensity increases (reliability drops and the time between failures decrease) or decreases (reliability grows and the time between failures increases). Example of this kind of model is Jelinski – Moranda Model.

Failure count models

Models are based on the number of failures that occur in each time interval.

Failure seeding models

By seeding errors to a document and then let the document undergo testing of some kind it is possible to calculate how many „real“ errors that exist.

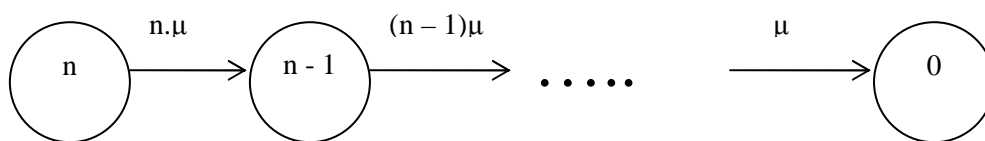
Input domain-based

By finding all unique paths through the program and then executing each and every one it is possible to guarantee that everything is tested.

## Jelinski – Moranda model

It was introduced in 1972 for discovery and removal of faults in computer software. It is a software product test-time error detection model. The big advantage of this model is possibility to estimate total testing time which is necessary for achievement of reliable system.

The main idea: „When the total number of remaining faults decreases as the errors are eliminated, the program should be able to run longer before a new failure occurs.“ Otherwise the failure intensity should decrease as time goes by.

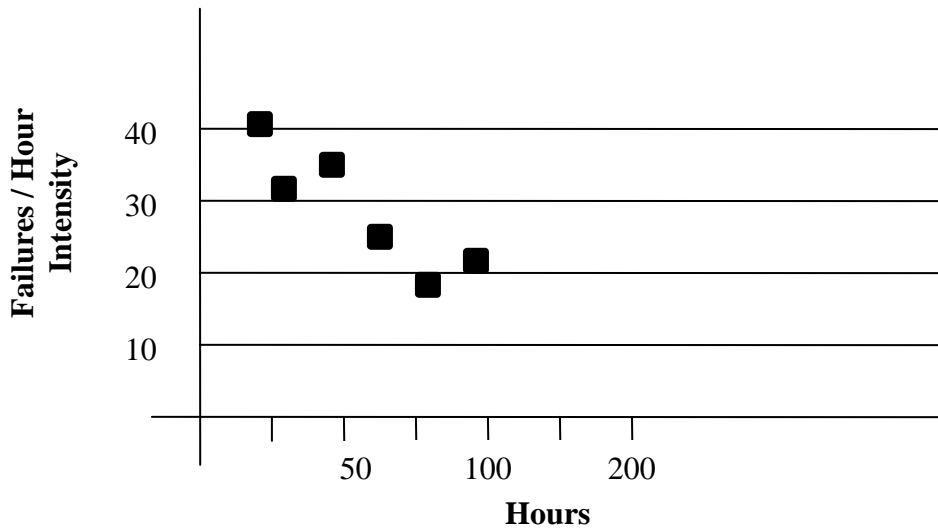


**Picture 1**

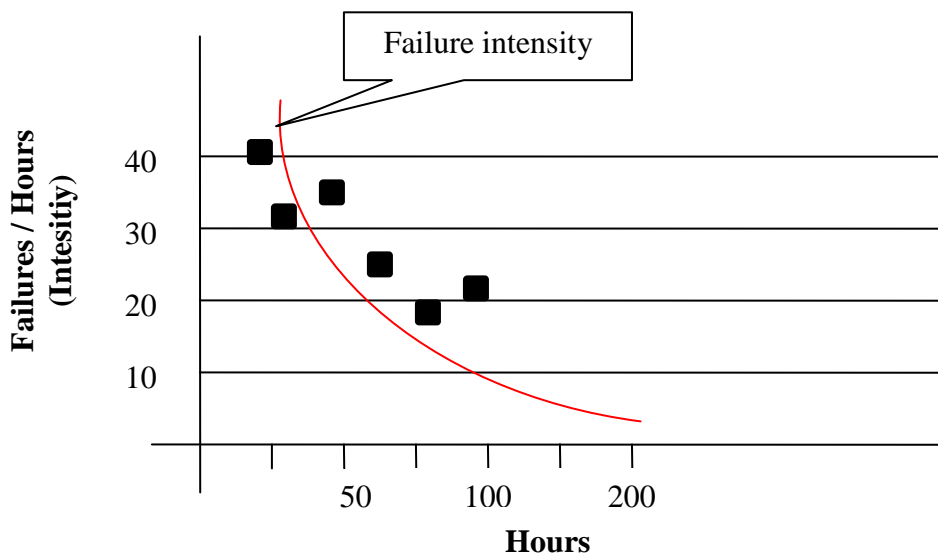
In the picture 1 we can easily see the main idea of Jelinski – Moranda model. In the beginning we have „n“ bugs and failure rate „μ“. Step by step we remove failures and in this way „n“ decreases. At the end of process we have no failure and our system is perfectly reliable. Of course this idea is just only Utopia. System without bugs does not exist. This is that what we would like to achieve.

### Jelinski – Moranda model assumptions

- The initial number of faults is constant but unknown. Faults are assumed before testing and probably these faults will cause errors during testing.
- The faults are independent of each other and equally hazardous. They have the same probability of emerging during the test.
- Faults are removed immediately in the time of emerging and any other new errors will not occur during the debug process.
- The failure rate is constant during a failure interval and it is proportional to number of faults remaining.



**Picture 2** The squares mean the measured failure intensity at different times. For each removal of a fault the failure intensity decreases. We try to read from the measured values how the failure intensity will drop and plot it.



**Picture 3** From this picture we can predict how the failure intensity will develop over the time.

### Failure intensity function

In the same way we can say hazard rate. This function drops by decreasing number of faults otherwise after  $k^{\text{th}}$  failure there are  $(N - k)$  faults left and the failure intensity decreases to  $\mu(N - k)$ .

$$\lambda_i = (N - k)\mu$$

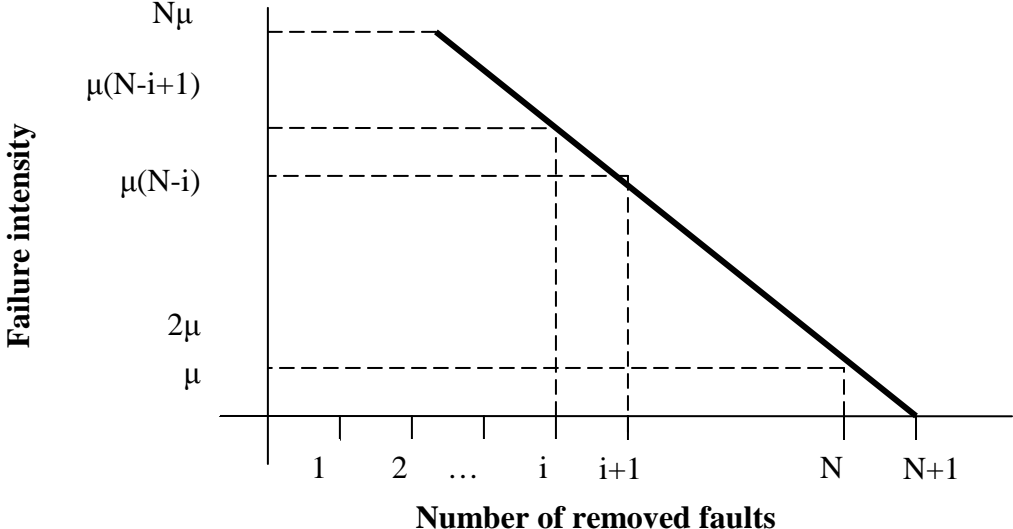
N... Finite unknown number of bugs in the software

$\mu$ ... Failure rate is directly proportional to the current fault content of the software

Denote by  $T_i$   $i = 1, 2, 3, \dots, N$  the time between the  $(i-1)^{th}$  and the  $i^{th}$  failures,  $T_i$  is thus the  $i^{th}$  failure-free time interval.  $T_i$  values are exponentially distributed random variables with parameter  $\lambda_i = (N - i + 1)\mu$ ,  $i = 1, 2, 3, \dots, N$ .

The distribution of  $T_i$  is given by  $P(T_i < t_i) = \mu(N - i + 1)\exp\{-\mu(N - i + 1)t_i\}$ ,  $i=1, 2, \dots, N$

The main property of the Jelinski – Moranda model is that the failure intensity is constant between the detection of two consecutive failures.



**Picture 4** The failure intensity versus Number of removed faults

On this picture we can see the same as on the previous pictures but here the failure intensity is confronted by number of removed faults. For each removal fault the intensity rate decreases by  $\mu$ .

**Estimation of model parameters**

We know at least 2 practical methods of parameters estimation for software reliability models. The first one is maximum-likelihood and the second one least square. For purposes of the Jelinski – Moranda Model I will focus on maximum-likelihood method. The maximum likelihood estimates the parameters by solving a set of equations. The base of this methods is the likelihood function. Let  $t_i$  is  $i^{th}$  failure-free time interval during the testing phase . It is time between the  $(i-1)^{th}$  and the  $i^{th}$  failure. The number of detected failure will be denote as „n“. Suppose that the failure data set  $t=\{t_1, t_2, t_3, \dots, t_n; n>0\}$  is given, the paramters  $N_0$  and  $\mu$  can be easily estimated by maximazing the likelihood function. This function of parameters  $N_0$  and  $\mu$  is given by:

$$L(t_1, t_2, \dots, t_n; N_0, \mu) = \prod_{i=1}^n \mu(N_0 - i + 1) \exp\{-\mu(N_0 - i + 1)t_i\}$$

The natural logarithm of the above likelihood function is

$$\ln L = \ln[\mu^n \{ \prod_{i=1}^n (N_0 - i + 1) \} \exp\{-\mu \sum_{i=1}^n (N_0 - i + 1)t_i\}]$$

By taking the partial derivatives of this log-likelihood function with respect to  $N_0$  and  $\mu$  respectively, and equating them to zero, we get the following likelihood equations

$$\frac{\partial \ln L}{\partial N_0} = \sum_{i=1}^n \frac{1}{N_0 - i + 1} - \sum_{i=1}^n \mu t_i = 0$$

$$\frac{\partial \ln L}{\partial \mu} = \frac{n}{\mu} - \sum_{i=1}^n (N_0 - i + 1)t_i = 0$$

Usually numerical procedures have to be used solve these two equations. However, the equation system can be simplified as follows. By solving  $\mu$  from the second equation above we get

$$\mu = n \left\{ \sum_{i=1}^n (N_0 - i + 1)t_i \right\}^{-1}$$

and by inserting this into the first equation, we obtain an equation independent of  $\mu$

$$\frac{1}{N_0} + \frac{1}{N_0 - 1} + \dots + \frac{1}{N_0 - n + 1} = \frac{n \sum_{i=1}^n t_i}{\sum_{i=1}^n (N_0 - i + 1)t_i}$$

The estimate of  $N_0$  can then be obtained by solving this equation. Inserting the estimated  $N_0$  into the expression of  $\mu$ , we may then get the maximum likelihood estimate of  $\mu$ .



# PRACTICAL PART

## CASRE

For showing practical utilization of the Jelinski – Moranda model I decided to use the program „CASRE“. This program is free downloaded on the Internet on <https://www.openchannelfoundation.org>. CASRE is abbreviation of Computer Aided Software Realiability Estimation. It was developed as a software reliability measurement tool which is easy to use for nonspecialists in software reliability engineering. Two file formats of input are allowed:

- Time Between Failures (error #, time since last failure, error severity)
- Failure Counts (interval #, # errors in interval, interval length, error severity)

Program outputs are different graphs and calculations of different indicators. We can choose from diferrent models.:

### Time between failure models

- Geometric
- Jelinski-Moranda
- Littlewood-Verrall
- Musa-Basic
- Musa-Okumoto
- NHPP

### Failure Count models


- Generalized Poisson
- NHPP
- Schneidewind
- Shick-Wolverton
- Yamada S-shaped

## CASRE benefits

- The CASRE user interface is fairly user friendly. Mechanisms such as “parameter estimation”, make it easier to use the various reliability models.
- The ability to combine the results of several models is a powerful feature.
- Results are available in tabular and graphical formats.
- Can apply various filters and noise reduction functions to effect the shape of the input data.

## Input

The data from table 1 was used as an input to CASRE. The data has to be in an exact text

format  and the file has to have the extension „.dat“. For the purposes of the Jelinsky – Moranda model we use the input format „Time Between Failure“.

## Output

In the graph 1 the Cumulative Time Between Failures (CTBF) was plot from the data. Reliability is increasing if the Mean Time Between Failures (MTBF) is increasing as the total number of failures accumulates during the testing phase. In the graph 1 we can see that the slope of CTBF grows still more slowly as the total number of failures accumulates during testing. So that means that CTBF is decreasing as time goes by due to fact that the MTBF is increasing.

In the graph 2 we can see the failure intensity which is supposed to be decreasing.

In the graph 3 the Time Between Failure is plotted which was around 8 hours at the end of the testing period .

Finally the program offers us the overview of all estimate for the given model (in our case the Jelinski – Moranda model) in the table 2.

## Conclusion

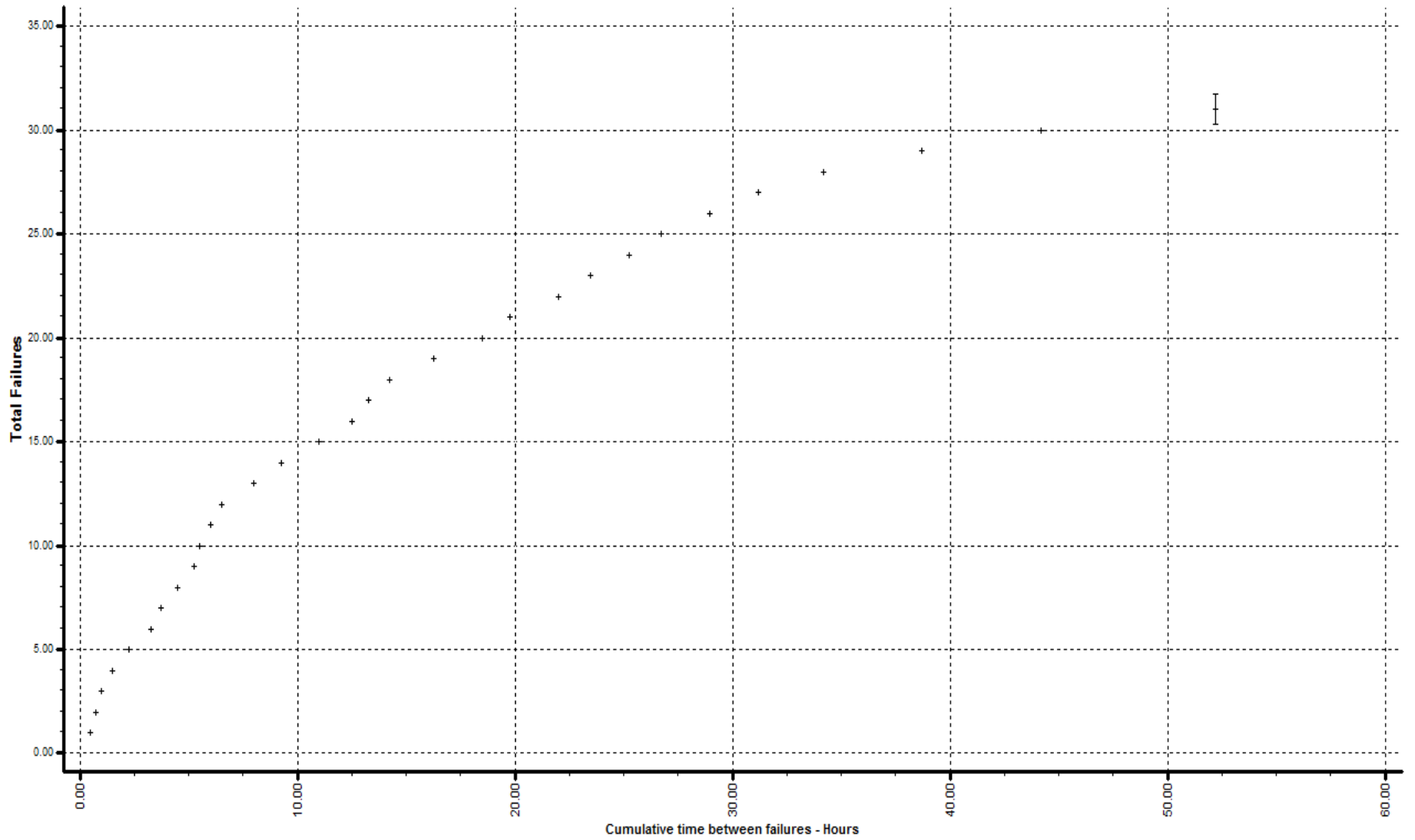
The software needs to be tested for approximately an additional 54 hours before its release to achieve the failure intensity of 0,084 faults/hour, which can be supposed to be satisfactory.

<b>Date</b>	<b>Time</b>	<b>Cumulative Execution Time</b>	<b>Time Between Failures</b>	<b>Fault Description</b>	<b>Fault Severity</b>
Sept 5, 2001	9:30 AM	0.50 hours	0.50 hours	GUI Warning Message Error	Low (1)
Sept 5, 2001	9:45 AM	0.75 hours	0.25 hours	Calculation Engine Error	High (9)
Sept 5, 2001	10:00 AM	1.00 hours	0.25 hours	Output Report Error	Low (1)
Sept 5, 2001	10:30 AM	1.50 hours	0.50 hours	Output Report Error	Low (1)
Sept 5, 2001	11:15 AM	2.25 hours	0.75 hours	GUI Schematic Diagram Error	Medium (5)
Sept 5, 2001	1:15 PM	3.25 hours	1.0 hours	GUI Fatal Error	High (9)
Sept 5, 2001	1:45 PM	3.75 hours	0.50 hours	Calculation Engine Error	High (9)
Sept 5, 2001	2:30 PM	4.50 hours	0.75 hours	GUI Warning Message Error	Low (1)
Sept 5, 2001	3:15 PM	5.25 hours	0.75 hours	Output Report Error	Medium (5)
Sept 5, 2001	3:30 PM	5.50 hours	0.25 hours	GUI Dialog Screen Error	Medium (5)
Sept 5, 2001	4:00 PM	6.00 hours	0.50 hours	GUI Dialog Screen Error	Low (1)
Sept 5, 2001	4:30 PM	6.50 hours	0.50 hours	Calculation Engine Error	High (9)
Sept 6, 2001	10:00 AM	8.00 hours	1.5 hours	Output Report Error	Low (1)
Sept 6, 2001	11:15 AM	9.25 hours	1.25 hours	GUI Warning Message Error	Low (1)
Sept 6, 2001	2:00 PM	11.00 hours	1.75 hours	Output Report Error	Low (1)
Sept 6, 2001	3:30 PM	12.50 hours	1.50 hours	Calculation Engine Error	High (9)
Sept 6, 2001	4:15 PM	13.25 hours	0.75 hours	Output Report Error	Low (1)
Sept 7, 2001	9:15 AM	14.25 hours	1.00 hours	Output Report Error	Low (1)
Sept 7, 2001	11:45 AM	16.25 hours	2.00 hours	GUI Warning Message Error	Low (1)
Sept 7, 2001	3:00 PM	18.50 hours	2.25 hours	Calculation Engine Error	High (9)

Sept 7, 2001	4:15 PM	19.75 hours	1.25 hours	GUI Dialog Screen Error	Medium (5)
Sept 17, 2001	10:00 AM	22.00 hours	2.25 hours	GUI Dialog Screen Error	Low (1)
Sept 17, 2001	11:30 AM	23.50 hours	1.50 hours	Calculation Engine Error	Medium (5)
Sept 17, 2001	2:15 PM	25.25 hours	1.75 hours	GUI Dialog Screen Error	Low (1)
Sept 17, 2001	3:45 PM	26.75 hours	1.50 hours	GUI Dialog Screen Error	Low (1)
Sept 18, 2001	9:15 AM	28.75 hours	2.00 hours	GUI Schematic Diagram Error	Low (1)
Sept 18, 2001	11:30 AM	31.00 hours	2.25 hours	GUI Dialog Screen Error	Low (1)
Sept 18, 2001	3:30 PM	34.00 hours	3.00 hours	Calculation Engine Error	Medium (5)
Sept 19, 2001	1:30 PM	38.50 hours	4.50 hours	Output Report Error	Low (1)
Sept 19, 2001	11:00 AM	44.00 hours	5.50 hours	GUI Dialog Screen Error	Low (1)
Sept 19, 2001	1:30 PM	52.00 hours	8.00 hours	GUI Dialog Screen Error	Low (1)

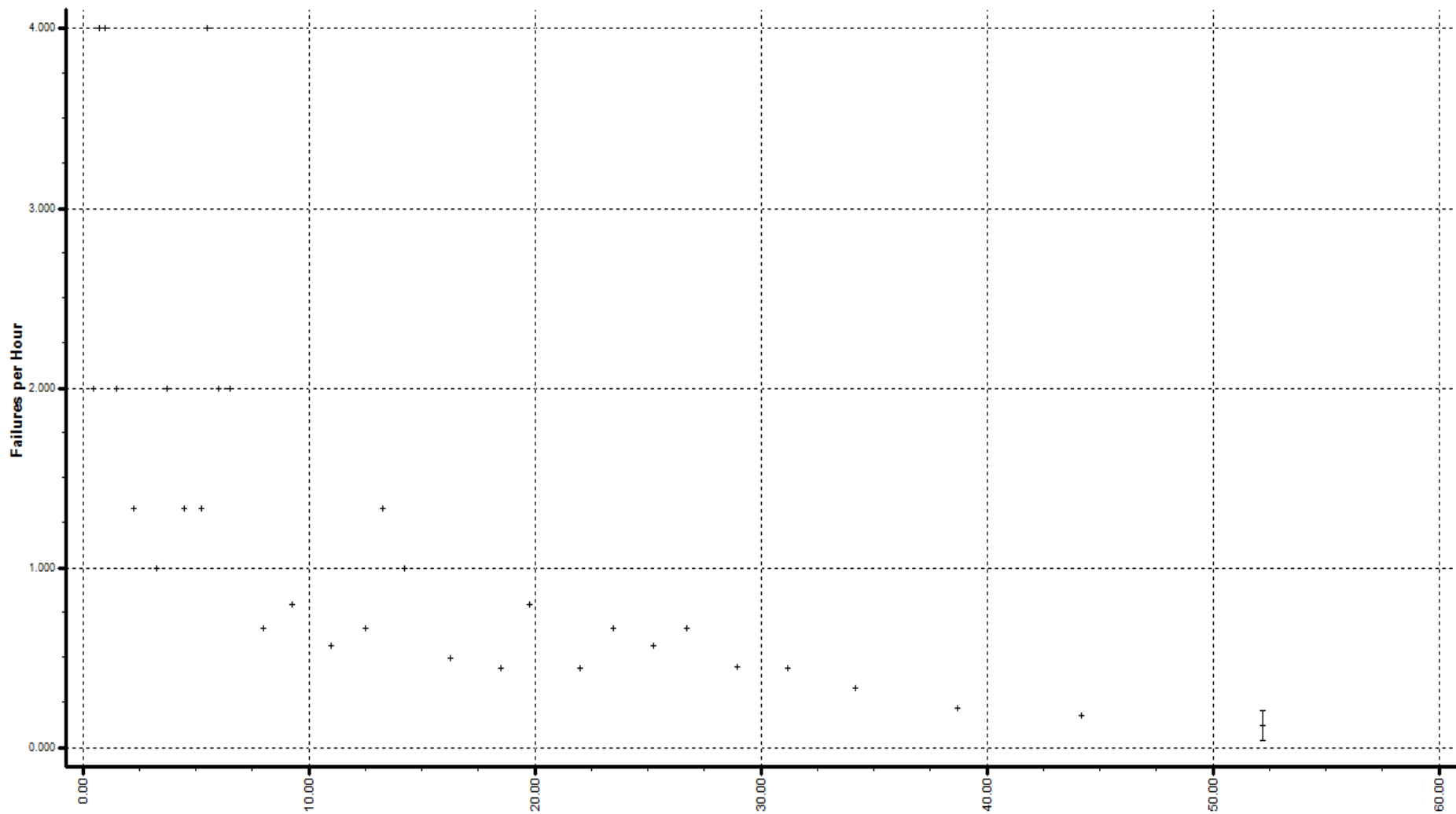
**Table 1**

+ Raw Data



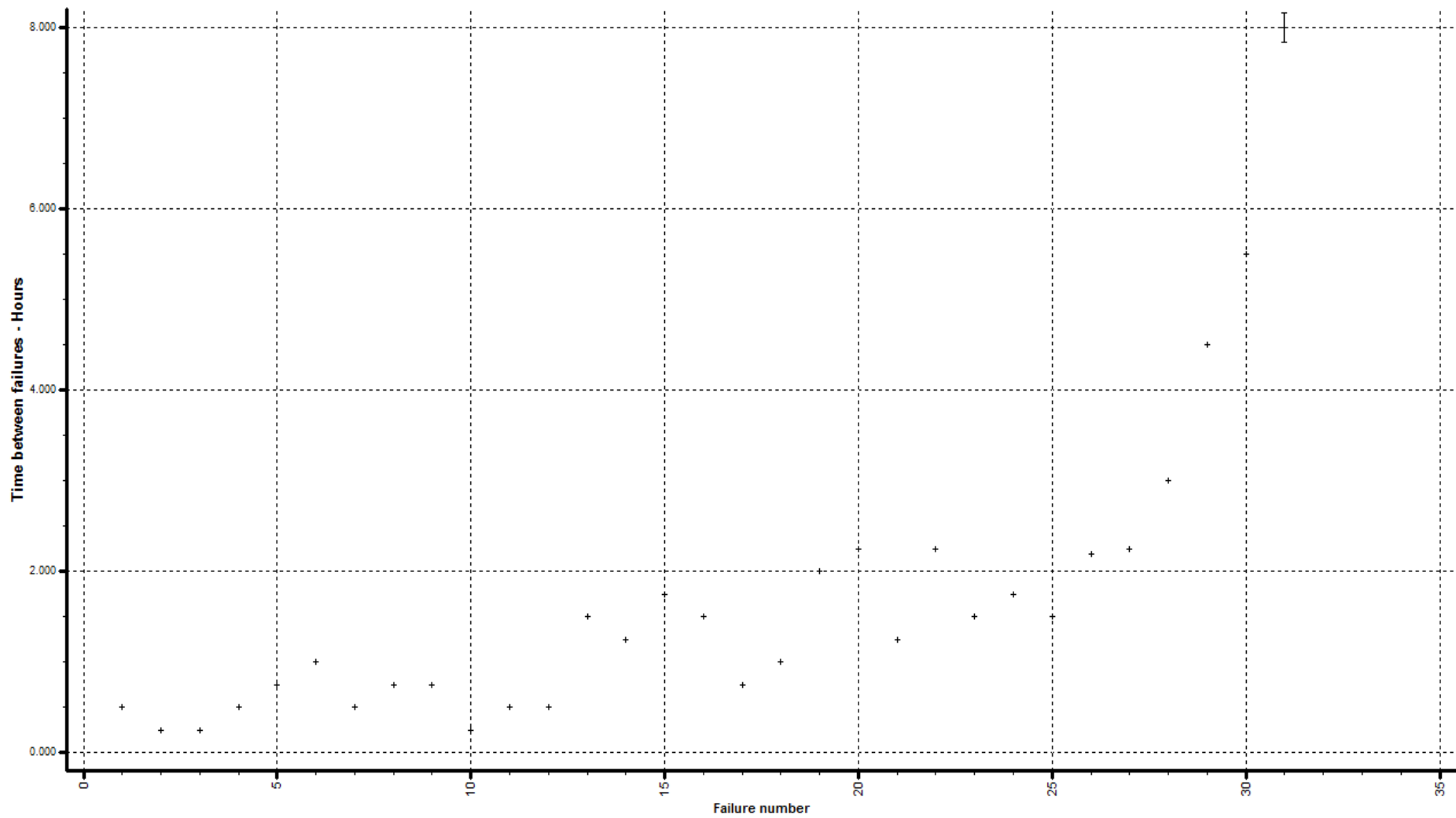
Graph 1 – Cumulative time between failures count

+ Raw Data



Graph 2 – Failure intensity

+ Raw Data



Graph 3 – Time Between Failures





# OTHERS MODELS

## Software Reliability Growth Models

These models rely on observation of failure occurrence and try to predict future failure behavior. Here are two examples of these models

- Musa's Basic Model
- Musa/Okumoto Logarithmic Model

General assumptions:

- Faults are independent and distributed with constant rate of encounter
- Execution time between failures is large compare to instruction execution time
- All failures are observed
- Fault causing failure is corrected immediately and reoccurrence of this failure is not counted

### Musa's Basic Model

Assumption: Decrement in failure intensity function is constant.

Consequence: Failure intensity is function of average number of failures experienced at every given point in time( =failure probability).

$$\lambda(\mu) = \lambda_0 \left[ 1 - \frac{\mu}{v_0} \right]$$

- $\lambda(\mu)$ : failure intensity
- $\lambda_0$ : initial failure intensity at start of execution
- $\mu$ : average total number of failures at a given point in time
- $v_0$ : total number of failures over infinite time

### Musa/Okumoto Logarithmic Poisson Model

This model uses a failure intensity decay parameter ( instead of the total failures expected, which is used in Musa's basic model). The decrement per failure for the logarithmic Poisson model is smaller each time a failure is experienced.

$$\lambda(\mu) = \lambda_0 e^{-\theta\mu}$$

$\theta$ : failure intensity decay parameter

## Comparison of these models

Basic model assumes that there is a 0 failure intensity while logarithmic model assumes convergence to 0 failure intensity.

Basic model assumes finite number of failures in the system while logarithmic model assumes infinite number.

Parameter estimations are usually obtained from system test observation and operational system by comparison with values from similar projects.

## Failure count models

Group of models that are based on the number of failures that occur in each time interval. One subgroup of these models is the NonHomogeneous Poisson Process models (NHPP), firstly proposed by Goel and Okumoto in 1979, which has formed the basis for the failure count models.

### The Goel-Okumoto model

Assumptions:

- The cumulative number of faults detected at time  $t$  follows Poisson distribution
- All faults are independent and have the same chance of being detected
- All detected faults are removed immediately and no new faults are introduced

The Goel-Okumoto model assumes that the failure process is modeled by an NHPP models with the mean value function  $\mu(t)$  given by

$$\mu(t) = a(1 - e^{-bt})$$

Where  $a > 0$  and  $b > 0$ ,  $a$  is the expected total number of faults and  $b$  is the “shape” factor. The failure intensity is  $\lambda(t) = abe^{-bt}$  which is equal to the derivative of  $\mu(t)$ .

## Error sendings models

### Mills error sending model

Mills propose a method to estimate the number of errors in a program by introducing „pseudo-errors“ into the program. From the debugging data, which consists of indigenous errors and induced errors, the unknown number of indigenous errors can be estimated. This model can be represented by a hypergeometric distribution.

The probability of  $k$  induced errors in  $r$  removed errors follows a hypergeometric distribution.

$$P(k; N + n_1; n_1, r) = \frac{\binom{n_1}{k} \binom{N}{r-k}}{\binom{N+n_1}{r}}$$

where

$N$  = number of indigenous errors

$n_1$  = number of induced errors

$r$  = number of errors removed during debugging

$k$  = number of induced errors in  $r$  removed errors

$r-k$  = number of indigenous errors in  $r$  removed errors.

Since  $n_1$ ,  $r$  and  $k$  are known, the maximum likelihood estimate of  $N$  can be shown as

$$N = \frac{n_1(r-k)}{k}$$

But this method was criticised for its inability to determine the type, location and difficulty level of the induced errors such that they would likely to be detected equally likely as the indigenous errors.

$$P(k; N + n_1; n_1, r) = \frac{\binom{n_1}{k} \binom{N-n_1}{r-k}}{\binom{N}{r}}$$

And the maximum likelihood estimate of  $N$  was modified to:

$$N = \frac{n_1 r}{k}$$

## References

- [1] Björn Andersson, Marie Persson: *Software reliability prediction* Master Thesis Software Engineering, Thesis no: MSE-2004-17, June 2004, document is available from [http://www.pdfdownload.org/pdf2html/pdf2html.php?url=http%3A%2F%2Fwww.bth.se%2Ffou%2Fcuppsats.nsf%2Fall%2Fbaac9c916708446cc1256eb7003eba23%2F%24file%2FSoftware Reliability Prediction.pdf&images=yes](http://www.pdfdownload.org/pdf2html/pdf2html.php?url=http%3A%2F%2Fwww.bth.se%2Ffou%2Fcuppsats.nsf%2Fall%2Fbaac9c916708446cc1256eb7003eba23%2F%24file%2FSoftware%20Reliability%20Prediction.pdf&images=yes)>
- [2] K.S. Trivedi: *Probability and Statistic with Reliability*, Publisher – John Wiley & Sons, document is available from [http://www.pdfdownload.org/pdf2html/pdf2html.php?url=http%3A%2F%2Fwww2.toki.or.id%2Fprobter%2Fdoc%2Fslide%2Fchap8\\_p5\\_s.pdf&images=yes](http://www.pdfdownload.org/pdf2html/pdf2html.php?url=http%3A%2F%2Fwww2.toki.or.id%2Fprobter%2Fdoc%2Fslide%2Fchap8_p5_s.pdf&images=yes)>
- [3] Michael J. Mangieri: *A Comparison of Two Software Reliability Engineering (SRE) Models: Jelinski-Moranda and Musa-Okumoto Logarithmic Poisson*, available from <http://members.tripod.com/mjmangieri/SE/csmn658.htm>>
- [4] M.C.J van Pul: *Simulations on the Jelinski – Moranda model of software reliability*, Department of operations research, Statistic and System theory, Report BS-R9122 September
- [5] Philip J. Boland: *Challenges in Software Reliability and Testing*, Department of Statistic National University of Ireland – Dublin
- [6] Raihan Al-Ekram: *Software Reliability Growth - Modeling and Prediction*, Dept. of Electrical and Computer Engineering, University of Waterloo
- [7] Dave Baker, Alan Goodbrand, David Menks: *CASRE Software Reliability Tool*, available from <http://www.ucalgary.ca/~dbmenks/seng/seng609.11/casre.html#toc>
- [8] Michael Grottko: *Software Reliability Model Study*
- [9] Jie Lian: *Software Reliability*, ECE-355 Tutorial
- [10] Hoang Pham, Michelle Pham: *Software Reliability Models for Critical Applications*, Idaho National Engineering Laboratory, Idaho 83415