People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Larbi Tbessi University – Tebessa
Faculty of Exact Sciences,Sciences of Nature and Life
Department: Mathematics and Computer Science

End of study dissertation
For MASTER graduation

Domaine : Mathematics and Computer Science
Study : Computer Science
Option : Information Systems

Theme

Reliability Prediction Approach for Multi-Agent Systems

Presented By :
Gueddouche Roumaissa

Before The Jury :

Mrs S.Bourouguaa	MCB	Larbi Tbessi University	President
Mr F.Hamidane	MAA	Larbi Tbessi University	Examiner
Mr Y.Menassel	MAA	Larbi Tbessi University	Supervisor

Defence date: Juin 2019

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

The greatest gift that life has given me is my family, on the occasion of my graduation.

To my dear parents

No dedication can express my respect, my eternal love and my consideration for the sacrifices you have made for my education and my well-being. I thank you for all the support and love you have given me since my childhood and I hope that your blessing always accompanies me. May this modest work be the fulfillment of your wishes so formulated, the fruit of your innumerable sacrifices, although I will never discharge you enough.

They are the best guiders, I hope this work can repay them and I promise them that this is the first step only, hoping of other accomplishments in the future inshallah.

To my friends

In memory of our sincere and deep friendship and pleasant moments we spent together. Please find in this work the expression of my deepest respect and my most sincere affection.

GUEDOUCHE ROMAISSA

Thanks

***I**t is with great pleasure that we reserve these few lines as a sign of gratitude and profound gratitude to all those who, from near or far, have contributed to the realization and the outcome of this work.*

***F**irst, we thank **Allah** Almighty, allowed us to carry out this work, and guided us to the path of knowledge.*

***T**hen, we sincerely thank Mr. **MENASSEL Yahia**, Assistant Professor in the Mathematics and Computer Science Department of the Faculty of Natural Sciences and Life Sciences at the University of Tébessa, for his supervision, his assistance, his support, availability and valuable advice.*

***O**ur warm thanks go to Mrs. **BOUROUGUAA Salima**, PhD and Mr. **HAMIDANE Fathi**, Assistant Professor, in the Mathematics and Computer Science Department of the Faculty of Natural Sciences and Sciences of Nature and Life at the University of Tebessa, for the honor that they granted us by agreeing to judge our work.*

***F**inally, we warmly thank our teachers for the quality of the teaching they have been kind enough to give us during our studies to provide us with efficient training.*

***T**hank you to all.*

Abstract

The software's are created to implement the needs of the customer in reality, and this raises the need to measure the applicability and accuracy of the software. Software reliability engineering is based on models and measurements that quantify the software reliability.

In our thesis, we have proposed a reliability prediction method of Multi agent systems integrating metrics depending on the agent's behavior and MAS characteristics.

For the development of our approach we used machine learning regression algorithm that is based on finding the correlation between the proposed metrics in order to predict the agent's reliability.

Keywords: SRE, Reliability, MAS, Machine Learning.

Résumé

Le logiciel est créé pour répondre aux besoins réels du client, ce qui impose de mesurer l'applicabilité et la précision du logiciel. La fiabilité du logiciel est basée sur des modèles et des mesures pour quantifie la fiabilité du logiciel.

Dans notre mémoire, nous avons proposé une approche de prévision de la fiabilité des systèmes multi-agents (SMA) intégrant des métriques dépendant du comportement de l'agent et des caractéristiques des SMA.

Pour le développement de de notre approche, nous avons utilisé l'algorithme de régression d'apprentissage automatique, reposant sur la recherche de la corrélation entre les métriques proposées afin de prédire la fiabilité des agents.

Keywords: SRE, fiabilité, SMA, apprentissage automatique.

ملخص

يتم إنشاء البرنامج لتلبية الاحتياجات الحقيقية للعميل (client)، والذي يفرض قياس مدى قابلية تطبيقها ودقة البرنامج. تعتمد هندسة موثوقية ودقة البرامج على النماذج والقياسات التي تحدد دقة هذه البرامج.

في مذكرتنا، اقترحنا طريقة تنبؤ بدقة الأنظمة متعددة العملاء تدمج المقاييس اعتمادًا على سلوك العميل وخصائص الأنظمة متعددة العملاء (Multi-agent Systems).

لتطوير الطريقة، استخدمنا خوارزمية الانحدار للتعلم الآلي، التي تستند إلى إيجاد العلاقة بين المقاييس المقترحة من أجل التنبؤ بموثوقية ودقة العملاء.

كلمات مفتاحية: هندسة موثوقية ودقة البرامج، التعلم الآلي، الأنظمة متعددة العملاء، الموثوقية.

Table of contents

List of figures

List of tables

List of abbreviations

SR: Software Reliability.

SRE: Software Reliability Engineering.

SRGM: Software Reliability Growth Model.

MTTF: Mean Time To Failure.

MTTR: Mean Time To Repair.

MTBF: Mean Time Between Failure.

ROCOF: Rate of OCcurrence Of Failure.

POFOD: Probability Of Failure On Demand.

SOFTREL: SOFTware RELiability process simulator.

CASRE: Computer Aided Software Reliability Estimation tool.

MEADEP: MEAsure and DEPendability.

SREPT: Software Reliability Estimation and Prediction Tool.

SMERFS: Statistical Modeling and Estimation of Reliability Functions for Software.

MAS: Multi Agent System

DAI: Distributed artificial intelligence

ML: Machine Learning

General Introduction

"In nature and human imagination, anything is possible."

John Archibald Wheeler

Introduction

Nowadays, software has appeared in all industrial sectors whether in air traffic control, nuclear reactors, aircraft and hospital patient monitoring systems and even in our homes and the daily routine.

Software reliability is a major attribute in software quality together with functionality, usability, performance, serviceability, capability, installability, maintainability and documentation, it is directly attached to software defects and failures, in an Inverse way more defect leads to less software reliability. High software reliability has become hard to obtain especially with the huge growth in software size and complexity, as instance the large next-generation aircraft will have over one million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines [50].

The Software reliability engineering attention is to develop engineering techniques to quantitatively evaluate Software reliability. In order to forecast reliability, failure data were integrated in reliability models like jelinsky and Moranda model, Weibull and Gamma model...; and also, multiple metrics were used like the number of failures in a time period, and time between failures.

Many software reliability testing tools were developed like SMERFS, CASRE, MEADEP, SOFTREL, SREPT that were able to estimate and predict software reliability.

Problematic

Although reliability forecasting still now based on reliability models but the most of this model are based on assumptions that are not realistic like the assumptions that faults are independent of each other and that correction of a fault never introduces new faults. Based on the different reliability testing tools, which their aim was to estimate or to predict reliability or both in the same time, the tools based on both prediction and estimation have shown better results in reliability measuring.

Because the exhaustive study in the SRGM, we tried to focus on the prediction part. Our concern, is predicting multi-agent systems reliability, more precisely agent's reliability because reliable agents mean reliable MAS, taking into account the complexity of the agents' behavior.

1. On what basis can we predict agent reliability?
2. What approach can we apply?

Manuscript Organization

Our manuscript comprised of three chapters, including this general introduction ending with general conclusion.

In the **first chapter** we have presented software reliability engineering defining the reliability and the different reliability forecasting models and metrics with their classifications.

The **second chapter** introduces reliability test tools, describing these tools and how they execute, and comparison of the tools based on criterions, concluding with a ranking of this tools.

The third chapter contains an initiation to the multi agent systems and the agent as an entity.

Fourth chapter defines in the first part an approach to predict the reliability of agents based on metrics that derived from the agents behaviors, applying the regression algorithm in machine learning to predict reliability; in the second part we have presented the coding environment and then explain each step in the code and the obtained results.

We conclude this work by presenting perspectives in this area of research.

Chapter 1
Software Reliability Engineering

"The good thing about science is that it's true
whether or not you believe in it."
Neil deGrasse Tyson

Introduction

The technological revolution is in her peak, with the rapid pace that software and hardware are managed to pass in the last decades.
We are reaching the point where perfection is more and more demanded in the current and future developments, especially on the software level because of the huge advance that hardware has reached. That has opened the way to developing software's with reliable performance, better quality especially minus cost and less production time.
In this chapter, we will present some of the software crisis that had big impact, and have mentioned the factors that jeopardize the reliability. Then we enter the software reliability engineering that includes the reliability testing measurements and models.

1. Software crises

Due to the increased demand for computer programs in all industry areas [1], the complexity of the programs developed has increased; the quality of the software became the main challenge facing programmers and developers.
The world has seen many software crises like:
The THERAC 25 crisis in 1986 which is a radiotherapy system for cancer tumor treatments caused by the inability of the system to call off a treatment in case of error diagnosis entered by mistake and an error message improperly displayed [2].
The PATRIOT disaster On February 25, 1991 during the Golf War, an error of 0.000000095 second in precision in every 10th of a second, an enemy missile skipped the patriot defenses leading to killing 28 innocents [3].
Telephone outage happened in 1991, after modifying three lines of code in a signaling program containing millions of lines of code, local telephone systems in California and along the east coast came to a halt [4].

Ariane 5, On 4 June 1996, after the Ariane 4 rocket successful launch, Ariane 5's inaugural flight flew away, while the control software's design flaws were revealed by the faster horizontal drift speed of the new rocket [5].

All these tragedies were the results of different causes like:

- Maintenance costs are almost as important as its development cost.
- Passing the delivery deadline.
- Software inefficiency.
- The poor quality of the software.
- Certain requirements were unapplied.
- The software was never delivered.

2. Software Reliability

Software is defined as a collection of computer programs, procedures, rules and data. Software features are classified into six main characteristics and 27 sub-characteristics [6].

ISO/IEC 9126 defines this characteristic and sub-characteristic as:

- **Functionality**: Suitability, accuracy, interoperability, security.
- **Reliability**: Maturity, fault tolerance, recoverability.
- **Usability**: Understandability, learnability, operability, attractiveness.
- **Efficiency**: Time behavior, resource utilization.
- **Maintainability**: Analyzability, changeability, stability, testability.
- **Portability**: Adaptability, installability, replaceability, coexistence.

According to ANSI, "Software Reliability is defined as the probability of failure-free software operations for a specified of time in a specified environment". Software reliability engineering is based on a key characteristic which is software reliability.

IEEE defines Reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time".

Software Reliability Engineering (SRE) is so defined as the quantitative study of the operational behavior of software systems with compliance with user requirements for reliability.

SRE was integrated like standard or current best practice of more than 50 organizations in their projects and software reports, including AT & T, Lucent, IBM, NASA, Microsoft and many more in Europe, Asia and the North America [7] but comparing it to the number of the software producers it is small amount.

2.1. Factors affecting software Reliability

The software development process consists of five phases: analysis, design, coding, testing, and operation. In each phase, there is factors can affect the software reliability and eventually software quality [8].

Factors such as subsystem configuration, operational profile, working languages, and applications categories, etc., must be taken into account and incorporated in the estimation of software reliability.

In Schneberger (1997), the authors listed the main causes of software errors according to the project managers. These reasons can be classified in the following eight main categories:

- Modification / addition / definition of a requirement.
- Programmer or team member experience, turnover.
- Design / scope / complexity changes.
- Coding and test phase problems.
- New technology / language / tools.
- Management experience.
- Upper management influence/bidding and time constrains.
- Data available for use in metrics and models.

Potential factors that can influence the reliability of each component or the application system are listed in the following table:

Factor	**Factors affecting reliability**
1	Inadequate test.
2	Operations errors.
3	Lack of a consistent quality assurance process.
4	Management change issues.
5	Lower quality source code.
6	Different operating conditions - high levels of use and overload.
7	Hardware failure - hard disks, network equipment, servers, power sources, memory, CPU.
8	Interactions with external services or applications.
9	Random Events - Security Failures.
10	Problems related to the operational environment.

Table 1.1. Factors affecting software reliability [8].

2.2. Software reliability techniques

Software reliability techniques have known several times that we will try to mention [9]:

2.2.1. Fault lifecycle techniques

It’s actually can be divided into four (04) principal techniques

- **Prevention of errors**: avoid, by construction, a fault occurrence.
- **Elimination of defects**: detect, by verification and validation, the existence of faults and eliminate them.
- **Fault tolerance**: provide, by redundancy, a service comply with the specification despite defects having occurred or occurring.
- **Failure/Outage forecasting**: to estimate, by evaluation, the presence of faults and occurrences and consequences of failures.

These techniques were like defense barriers to limit the cost and time of construction all respecting the user requirements, but fault prevention technique was not able to prove herself.

2.2.1.1. The bathtub curve for Software Reliability

In the bathtub curve we will observe the different stages of the failure rate in time:

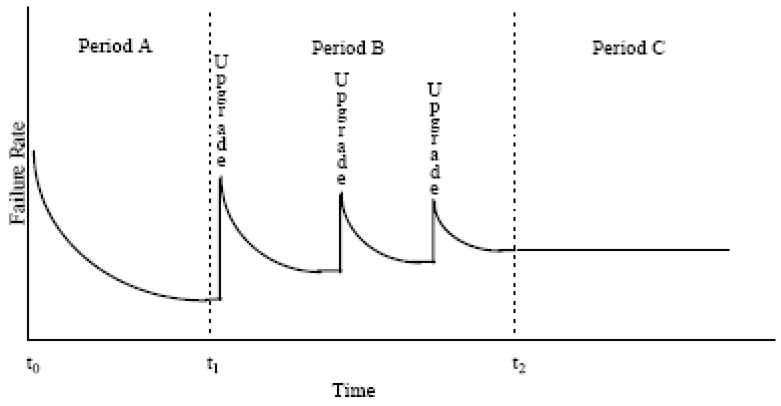


Figure 1.1. Bathtub curve for software reliability [9].

In the last phase, software does not have an increasing failure rate, in this phase, software is approaching obsolescence; there are no motivation for any upgrades or changes to the software. Therefore, the failure rate will not change. In the useful-life phase, software will experience a drastic increase in failure rate each time an upgrade is made. The failure rate levels off gradually, partly because of the defects found and fixed after the upgrade.

2.2.2. Software reliability models and measurement

Forecasting failures were the goal of SR modeling on both axes' estimation and prediction. Estimation which is measuring the current state, and prediction as assessment of the future state of the reliability of a software system.

SR model specifies the form of a random process that describes the behavior of software failures with respect to the time. Based on three main reliability modeling approaches:

- The error seeding and tagging approach.
- The data domain approach.
- The time domain approach, which is considered to be the most popular one.

The goal of time domain SR modeling is to create curve fitting of observed time-based failure data by a pre-specified model formula, parameterized with statistical techniques (such as the Least Square or Maximum Likelihood methods).

The model supply's an estimate of the existing reliability or predictability of future reliability by extrapolation techniques.

SR models are generally based on a number of common assumptions, as follows.

1. The operating environment identical to that of the test, the measurement environment is where the reliability model has been set.

2. In case of default, the fault that causes the failure is immediately deleted.

3. The removal of defects does not cause new ones.

4. The number of defects inherent in the software and the way these faults manifest themselves to cause failures follow, at least statistically, some mathematical formulas.

As the number of defects (as well as the failure rate) of the software system reduces as tests progress, resulting in increased reliability, these models are often referred to as software reliability growth models (SRGM) [9].

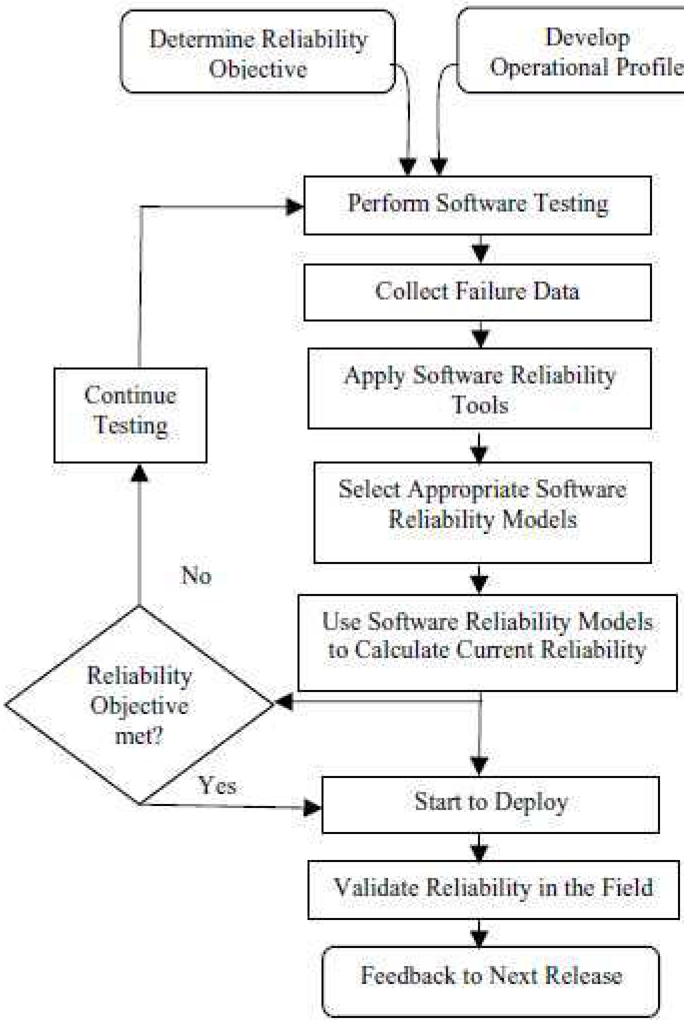

Determine Reliability Objective
Develop Operational Profile
Perform Software Testing
Collect Failure Data
Apply Software Reliability Tools
Select Appropriate Software Reliability Models
Use Software Reliability Models to Calculate Current Reliability
Continue Testing
No
Reliability Objective met?
Yes
Start to Deploy
Validate Reliability in the Field
Feedback to Next Release

Figure 2.1. Software Reliability Engineering Process Overview [9].

This figure shows an SRE framework in current practice First, a reliability goal is determined quantitatively from the customer's point of view to maximize customer satisfaction, and customer use is defined by developing an operational profile.

The software is then tested according to the profile, failure data collected and reliability monitoring during tests to determine the release time of the product. This activity can be repeated until some reliability level has been reached.

The operational profile is a set of disjointed alternatives operating scenarios of the system and their probabilities of occurrence.

For better comprehension of the two axes' we will see the main differences between the two:

Issues	**Prediction Models**	**Estimation Models**
Data Reference	Uses historical data.	Uses data from the current software development effort.
When used in	Usually made prior to development or	Usually made later in life cycle

development cycle	test phases; can be used as early as concept phase.	(after some data have been collected); not typically used in concept or development phases.
Time frame	Predict reliability at some future time.	Estimate reliability at either present or some future time.

Table 1.2. Difference between software reliability prediction models and software reliability estimation models [10].

2.3. Basic Reliability Metrics

The choice of the applied metrics depends primarily on the domain of execution and the requirements of the user and this metrics are the way to quantify software reliability [1].

- MEAN TIME TO FAILURE (MTTF): MTTF is defined as the time interval between successive failures. An MTTF of 300 means that a failure can be waited every 300 units of time. Time units are totally depending on the system and it can even be specified in the number of transactions. MTTF is relevant for systems with long transactions.
- MEAN TIME TO REPAIR (MTTR): Once the failure has occurred, it is sometimes necessary to repair the error. MTTR measures the average time needed to track errors causing failure and repairing them.
- MEAN TIME BETWEEN FAILURE (MTBF): The combination of the MTTF and MTTR metrics is the MTBF metric. MTBF = MTTF + MTTR, an MTBF of 300 indicates that once the failure has occurred, the next failure should only occur after 300 hours. In this case, the time measurements are in real time and not the execution time as in MTTF.
- RATE OF OCCURRENCE OF FAILURE (ROCOF): ROCOF is the frequency of occurrence with which unexpected behavior is likely to occur in a time interval. A ROCOF of 0.02 means that two failures are likely to occur on 100 no operational time unit. It's also called failure intensity metric.
- PROBABILITY OF FAILURE ON DEMAND (POFOD): POFOD is the probability that the system will fail when a Service request is made. A POFOD of 0.1 means that one out ten service requests may result in failure. POFOD is an important measure for critical safety systems.
- AVAILABILITY (AVAILABLE): An availability of 0.995 means that every 1000 units of time, the system will probably be available for 995 of them. The percentage of time that a system is available for use, taking into account forecasts and unforeseen downtime, and it takes into account the repair time and the system restart time. If a system is down on average four hours on 100 hours of operation, its availability is 96%.

2.4. Software Reliability Growth Models

These models refer to models that attempt to predict software reliability from test data. They show a relationship between error detection data and known mathematical functions, such as logarithmic or exponential functions.

The model that describes error detection in software reliability is called the software reliability growth model [11].

There are many models we will try to mention some of them:

2.4.1. Non Homogeneous Poisson Process Model

General software reliability models follow the NHPP as follows:

$$pr\{N(t) = n\} = \frac{[m(t)]}{n!}e^{-m(t)},\ \text{n} = 0,1,2\ldots.(1)$$

Where m(t) is mean value function, which is expected number of failures detected by testing time t. It can be written as: $\text{m(t)} = \int_0^t \lambda(s)ds.$ ....(2)

Where λ(s) is intensity function of failure.

Most of NHPP SRGM is expressed using the differential equation as:

$$\frac{dm(t)}{dt} = b(t)[a(t) - m(t)]\ldots\ldots\ldots\ldots\ldots..(3)$$

Through solution of Equation (3), that make to find unique m(t) using a(t) and b(t). Also, this process can be applied to assume for software testing.

2.4.2. Model Jelinski-Moranda

Introduced for the first time in 1972, it is a continuous time-independently distributed inter failure time and independent and identical error behavior model.

2.4.3. Goel-Okumoto Model

It has been proposed by Goel and Okumoto, and is one of NHPP's most popular models in the field of Software Reliability Modeling and is also known as the Exponential NHPP Model.

2.4.4. Generalized NHPP model of Goel

This is the generalization of the Goel-Okutmoto model, and is proposed by Goel to determine the situation in which the software failure intensity increases slightly at the beginning and then decreases.

2.4.5. Inflected S-shaped model inflicted

This model is proposed by ohba and is based on the concept that software reliability growth becomes an S if the defects of a program are mutually dependent and some defects are not detectable before others deleted and that this model solves a technical problem in Goel-Okumoto model.

2.4.6. Logistic Growth Curve Model

This model is designed to predict the economic growth of the population and could also be applied to the growth of software reliability. The logistic growth curve model is one and has an S-shaped curve.

2.4.7. Musa-Okumoto model

In this model, a property is incorporated, which is explained by Musa-Okumoto. They observed that the reduction in the failure rate resulting from repairs resulting from early failures is often greater because they tend to occur more often than once.

2.4.8. Yamada Delayed S-Shaped Model

It is the model with the modification of the inhomogeneous possession process which makes it possible to obtain an S-shaped curve for the cumulative number of detected failures, so that the failure rate initially increases and then fades.

Model Name	Mean Value Function	Intensity Value Function
Jelinski-Moranda Model	m(t) =n (1 − exp – ɸt).	λ i= (n −k) μ.
Goel-Okumoto Model	m(t) =a 1 − exp − bt, α > 0, b> 0 a= expected total number of fault. b= fault detection rate.	λ (t) =ab ∗exp – bt. a> 0, b> 0
Generalized Goel NHPP Model	m(t) = a(1-exp([-btc]) , a> 0, b> 0, c> 0 a= expected total number of faults. b,c =reflect quality of testing.	λ (t) =abctc −1 exp−btc. α> 0,b > 0,c > 0
Inflected S-Shaped Model	m(t) =a ∗(1 −exp [−bt] /1 + ψ (r) ∗exp –bt). ψ (r) = 1 –r /r a > 0,b > 0,r > 0 a= expected total number of faults. r = rate of detectable fault. b =fault detection rate.	λ (t) =(ab exp [−bt](1 +ßt) / (1 +ß ∗exp [−bt]) 2. a> 0,b > 0, ß > 0.
Logistic Growth Curve Model	m(t) =α /1 +k ∗ exp–[bt] α> 0, b> 0, k> 0 a = expected total number of faults. k,b = estimated by fitting the failure data.	λ(t) = ab exp− bt 1 + k∗exp −bt 2 a> 0,b > 0,k > 0
Musa-Okumoto Model	m(t) = a∗ ln (1 +bt). a > 0, b> 0 a = expected total number of faults. b = fault detection rate	λ(t) =ab (1 +bt). a > 0, b> 0
Gompertz Growth Curve Model	m(t)=akbt a>0,0<b<0,0<k<1 a= expected total number of faults. b= estimated using regression analysis.	λ(t) = abln(k)kexp[-bt] exp [-bt] a>0,0<b<0,0<k<1
Yamada Delayed S-Shaped Model	m(t)= a(1-(1+bt) * exp[-bt]), a>0,b>0 a = expected total number of fault to be detected b = fault detection rate	λ(t)= ab2t*exp[-bt], a>0,b>0
Yamada exponential	m(t)=a*(1-exp[-rα(1-exp[ßt])]). a>0,b>0,α>0, ß>0 a = total number of fault to be detected. α = fault introduction rate r, ß =constants.	λ(t)=ara(exp[-rα(1-exp[-ßt])])*exp[-ßt]. a>0,b>0,α>0, ß>0
Yamada Imperfect Debugging Model	m(t)=a*b*(exp[αt]-exp[-bt]/α +b) a>0,b>0,α>0	λ(t)=a*b*(α*exp[αt]+b*exp[-bt]/α +b). a>0,b>0,α >0

	a = total number of fault to be detected.
	b =fault detection rate.
	α = fault introduction rate.	

Yamada Raleigh	m(t)=a(1-exp[-rα (1-exp[-ßt2/2])]) a>0,r>0,α >0, ß>0 a= total number of fault to be detected. α = fault introduction rate. r, ß =constants.	λ(t)=araßt(exp[-rα(1-exp[-ßt2/2])]) *exp [-ßt2/2]. a>0,r>0,α >0, ß>0
Modified Duane Model	m(t)=a{1-(b/b+t)c)} a>0,b>0,c>0 a = total number of fault to be detected.	λ(t)=acbc (b+t)-(1+c). a>0,b>0,c>0

Weibull-Type Testing-Effort Function Model	m(t)=a(1-exp[-ba(1-exp{-ßtɣ])])
	a,b,α, ß, ɣ >0
	a=total number of fault to be detected
	b = fault detection rate
	α =Total number of test effort
	ß = scale parameter.
	ɣ = shape parameter.	

Table 1.3. Mean Value and Intensity of Various Models [11].

3. Classification Based on Failure History

The existing SWRMs are classified into four main classes on the basis of failure history [12].

- Time between Failure Models (TBF Models).
- Fault Count Models (FC Models).
- Fault Seeding Models (FS Models).
- Input domain-based Models (IDB Models).

3.1. TBF models: In this class of models; process under consideration is the time between failures. He assumed that the elapsed time between the (i-1) and the i-th faults is a random variable.

The estimates of these parameters are obtained from the observed values of TBF and the TOS parameter is obtained from the adjusted models.

3.2. FC Models: The random variable of interest is the number of failures (failures) occurring during specified time intervals, called FC models. It is assumed that the number of failures follows a known stochastic process. The time is used whether it is calendar or can be a CPU time.

3.3. FS models: In this model, we tested and observed the number of seeded and native faults counted. The MLE and combinatorial method makes it possible to obtain an estimate of the defect content of the program before seeding before seeding, and then from the value of the parameter SWR is calculated.

3.4. BDI models: In this model approach, a set of test cases is generated from the entry covering the operational profile of the input. The input domain is partitioned into a set of equivalent classes.

Time Between Failure (TBF) Models	1. It's an independent times between failure.
	2. Each fault has equal probability.
	3. After each occurrence fault are removed.
	4. At the time of correction new faults are introduce.
	5. Ex. J-M De-Eutrophication, Schnick and Wolverton, Goel and Okumoto Imperfect Debugging, Littlewood-Verall Bayesian Models
Fault Count (FC) Models	1. Fault or failure in specified time interval.
	2. Testing during intervals is reasonably homogenous.
	3. Numbers of fault detected during non-overlapping intervals are independent of each other.
	4. Estimate software reliability mean time by fault count.
	5. Ex. Generalized Poisson Model, Goel-Okumoto NHPP Model, IBM Binomial and Poisson Models, Logarithmic Poisson Execution Time Model, Musa Okumoto
Fault Seeding (FS) Models	1. A known number of faults are “seed”.
	2. Seeded faults are randomly distributed in the program.
	3. A Program has unknown number of indigenous faults.
	4. Indigenous and seeded faults have equal probabilities of being detect.
	5. Ex. Lipow model, Mills seeding model, Basin model
Input Domain Based (IDB) Models	1. Test cases are generated from the input covering.
	2. Estimate software reliability by failure observed in test cases.
	3. Random testing is used.
	4. Input domain can be partitioned into equivalent classes.
	5. Input profile distribution is known.
	6. Ex. Bastani Model, Nelson Model, Ramamoorthy

Table 1.4. Overview of Models based on Failure History [12].

Conclusion

In this chapter we have viewed or better say entered to the software reliability engineering with the main definitions of the domain, without a doubt the reliability measuring has become an important factor in software development industry.

In the next chapter we will detail the reliability metrics and growth models, ending with tools comparison.

Chapter 2

Reliability Testing Tools: State of art

"This world is grand and there lies an ocean of undiscovered findings."

Isaac Newton

Introduction

Forecasting failures were the goal of SR modeling on both axes' estimation and prediction. Estimation which is measuring the current state, and prediction as assessment of the future state of the reliability of a software system.

Software reliability is measured using measurements, models and leading to reliability test tools.

In this chapter we will make a comparison between the most known and used software reliability tools including CASRE, SMERFS, SOFTREL, MEADEP and SREPT for each tool we will details the different modules of it and the way it executes, and counting their advantages and dis advantages; ending with comparison table and tools ranking.

1. Reliability estimation tools

1.1. CASRE - A Computer-Aided Software Reliability Estimation Tool

CASRE as Computer Aided Software Reliability Estimation, it is an extension of the tool SMERFS. It is a user-friendly reliability estimation tool, where the user can manipulate different options like selecting a set of failure data or executing a model through a menu.

After the execution the results are presented as failure intensities or inter-failure times in a graphical form (plots) or in tabular form, the user can also use this window of the results to detect cumulative number of failures and reliability growth curve.

The models combination grants better predictive reliability estimation, CASRE provides this option [11]. With the ability to determine user custom combination and add it as a configuration to the tool. To help the users CASRE applies different techniques to determine the applicability of a model to a set of failure data [13].

1.1.1. The functionalities that CASRE provides:

- *Data modification:* includes data editing, smoothing (altering), and data transformation (logarithmic, exponential, or linear).

- *Failure data analysis* (statistics of the failure data).
- *Modeling and measurements:* allow users to execute several models on the failure data.
- *Display of results:* provides the user graphical display models [13].

For further details on this tool we start by giving a global view on its main functionalities in the Figure below:

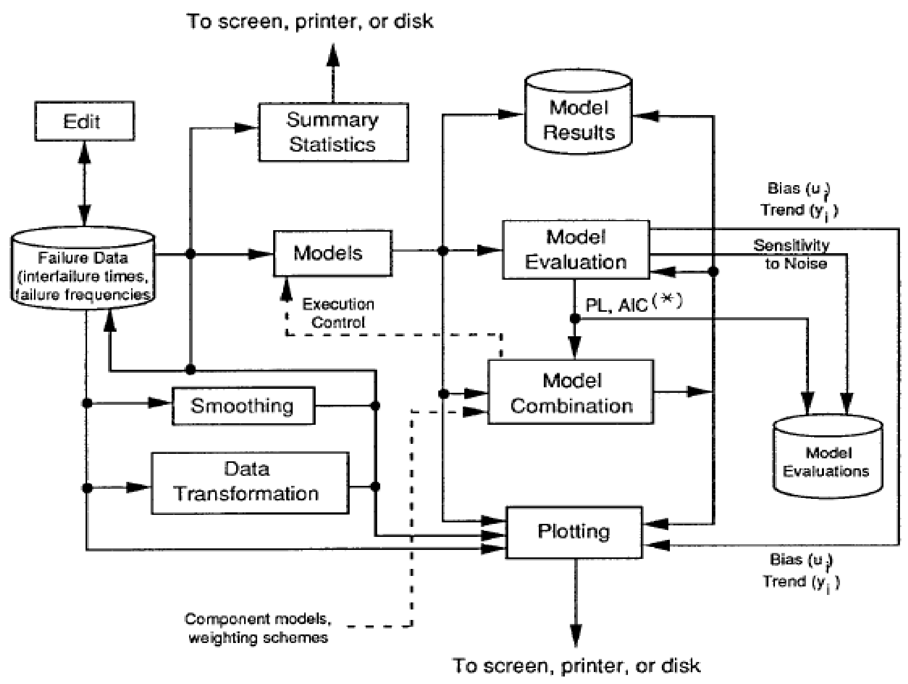


Figure 2.1. CASRE architecture [13].

Data modification

- *Editing:* Casre allows the users to custom or create the failure data history. with easy to use interface with the ability of choosing time between failures or test interval lengths manually, with the possibility to choose preferred editor
- *Smoothing:* To handle the noisy data, it uses the next smoothing techniques: Sliding rectangular window, hann window, polynomial fit, and specific cubic-polynomial fits (e.g. B-Spline, Bezier Curve).
- *Data Transformation:* Is very important part it concerns the result display, we need to select performing operations like logarithmic, exponential, or linear transformations of the failure data for better or more understandable results, as example we have the following operations:
  - log(a * x(i)) + b); x(i),
  - exp(a * x(i) + b),
  - x(i) ** a,
  - x(i) + a,
  - x(i) * a.
  - user-specified transformation.

Failure data analysis

The "Summary Statistics" allows users to display the failure data summary statistics, including the mean and median of the failure data, 25% and 75% hinge points, skewness, and kurtosis.

Modeling and measurement

It has two modeling functions. The "Models" block executes single software reliability models on a set of failure data. The "Model Combination" block, allows users to execute several models on the failure data and combine the results of those models.

The block labeled "Model Evaluation" allows users to determine the applicability of a model to a set of failure data:

- *Single Model Execution:* with the following models:
  (1) Bayesian Jelinski-Moranda Model (BJM) [14], [15].
  (2) Brooks and Motley Model (BM) [16].
  (3) Duane Model (DU) [17], [18].
  (4) Geometric Model (GM) [16].
  (5) Goel-Okumoto (GO) [19].
  (6) Jelinski-Moranda (JM) [20], [21].
  (7) Keiller-Littlewood Model (KL) [22], [23].
  (8) Littlewood Model (LM) [24].
  (9) Littlewood non-homogeneous Poisson Process.
  (10) Littlewood-Verrall (LV) [25].
  (11) Musa-Okumoto (MO) [26].
  (12) Generalized Poisson Model (PM) [16].
  (13) Schneidewind's Model (SM) [27].
  (14) Yamada Delayed S-Shape Model (YM) [28].

CASRE allows users to choose the parameter estimation method (maximum likelihood, least squares, or method of moments), Model outputs include:

- Current estimates of failure rate/ inter-failure time.
- Current estimates of reliability.
- Model parameter values, including high and low parameter values for a user-selectable confidence estimated bound.
- Current values of the pdf.
- The probability integral transform uī.
- The normalized logarithmic transform of **uī,** yī.

- *Combination Models:* the users can set their own models combination results according to several combination schemes. The resulting combination models could be further used as the component models to form another combination model.
- *Model Evaluation:* help users decide which model or combination models could be applied to a specific failure data set by using statistical methods:
  - Computation of prequential likelihood **(PL)** function (the "Accuracy" criterion).
  - Determination of the probability integral transform ***ui ,*** (plotted **as** the u-plot - the "Bias" criterion).
  - Computation of **yi** to produce the y-plot (the "Trend" criterion).
  - Noisiness of model predictions (the "Noise" criterion).

Display of results

CASRE graphically displays model results in the following forms:

- Inter-failure time /failure frequencies, actual and estimated.
- Cumulative failures, actual and estimated.
- Reliability growth, actual and estimated.

Both current and estimated quantities are available on the same plot. Users are able to control the range of data to be plotted as well as the usual cosmetic aspects of the plot (e.g. X and Y scaling, titles), multiple plots could be simultaneously displayed.

It allows users to save them on a file or to be printed, and also includes the ability of saving the used data to produce a plot to a file that can be imported by a spreadsheet, a DBMS, or a statistics package for further analyses.

CASRE On-screen

The Main Steps to Use CASRE are [29]:

Step 1. Create a set of failure data.

Step 2. Start CASRE.

Step 3. Open a set of failure data.

Step 4. Change the failure data.

Step 5. Apply filters and smoothing operations to the data.

Step 6. Apply trend tests to the failure data to determine whether or not software reliability models should be applied.

Step 7. Apply models to the failure data.

Step 8. View the model outputs.

Step 9. Print failure data and model results.

Step 10. Save failure data and model results to disk.

Step 1: Create a set of failure data

The Failure data files has a specific format, we can create it using word processor or text editor. We have two kinds of failure data: Time between failures and Failure count, with different formats. Two types of inputs:

• Time between failures:

```
Seconds
4       9       1
5       6       7
6       50      2
```

The first line represents the time units for the data file (seconds, minutes, hours, days weeks, months, years). In the case of second unit the subsequent lines are:

- The first column is the current failure number.
- The second column represents the time that has passed since the last failure was observed.
- The values in the second column are measured in the time units given in the first line of the file.
- The third column indicates the severity of the failure on a scale of 1 to 9.

- Failure count:

Minutes

3	4	10	4
4	8	20	3
5	1	30	7

The first line represents the time units for the data file.

In the case of second unit the subsequent lines are:

- The first column gives a sequential test interval number.

- The second column specifies the number of failures that were observed during a given test interval.

- The third column gives the length of the test interval. Test interval lengths do not have to be equal.

- The fourth column indicates the severity of the failure on a scale of 1 to 9.

Step 2: Install and run CASRE

The CASRE main window should then appear as shown

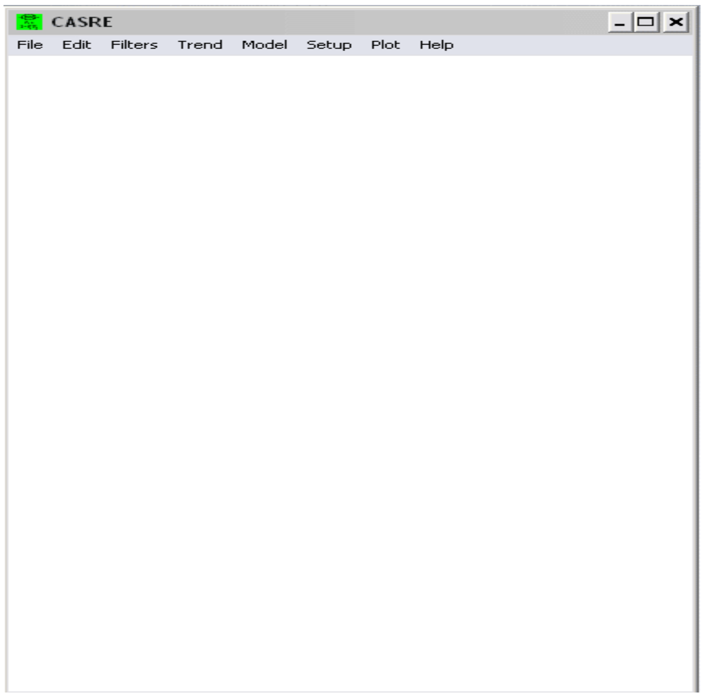


Figure 2.2. CASRE Main Menu.

Step 3: Opening a Data File

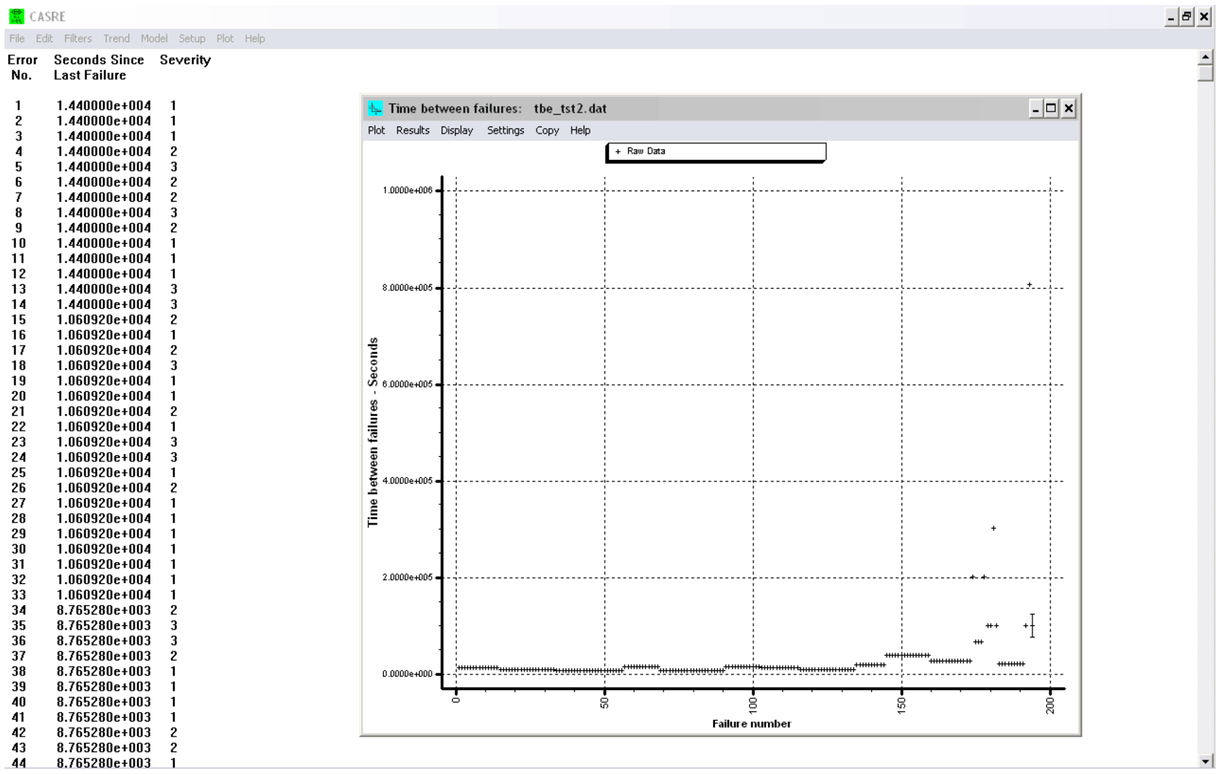
- Go to File -> Open

Browse to the directory where the file is located and select it to open it.

When a failure data file is opened, the text of the file is shown in the main window, while a plot of the data is shown in the graphic display window (see figure 2.3).

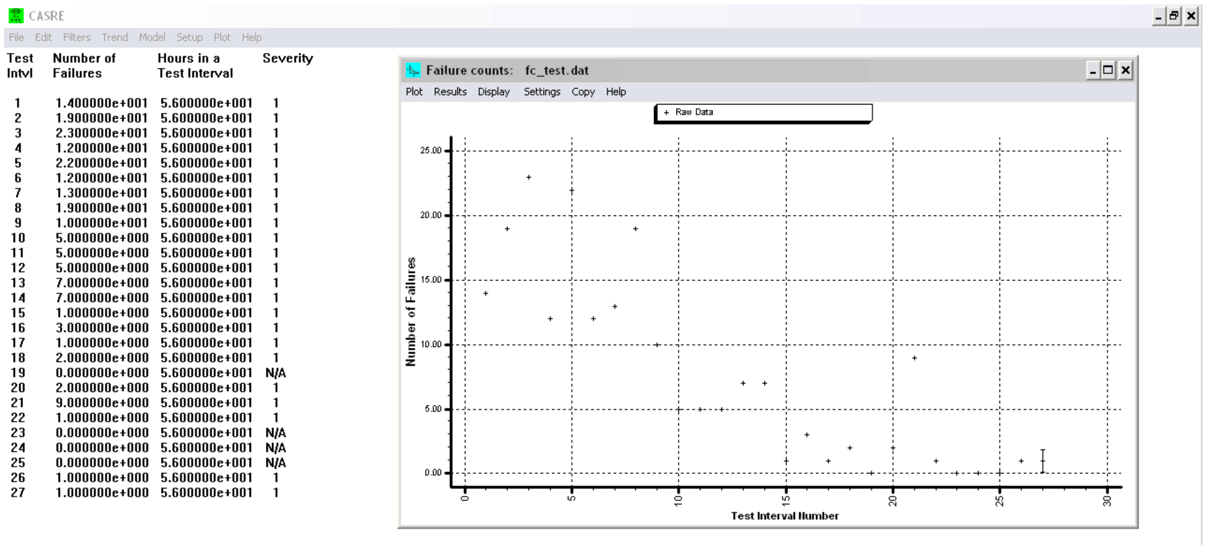
Step 4: Change the failure data

We can change from a data type from one to another using external application (see figure 2.4).


CASRE
File Edit Filters Trend Model Setup Plot Help
Time between failures: tbe_tst2.dat
Plot Results Display Settings Copy Help
+ Raw Data
Time between failures - Seconds
Failure number

```
Error  Seconds Since   Severity
No.    Last Failure

 1     1.440000e+004   1
 2     1.440000e+004   1
 3     1.440000e+004   1
 4     1.440000e+004   2
 5     1.440000e+004   3
 6     1.440000e+004   2
 7     1.440000e+004   2
 8     1.440000e+004   3
 9     1.440000e+004   2
10     1.440000e+004   1
11     1.440000e+004   1
12     1.440000e+004   1
13     1.440000e+004   3
14     1.440000e+004   3
15     1.060920e+004   2
16     1.060920e+004   1
17     1.060920e+004   2
18     1.060920e+004   3
19     1.060920e+004   1
20     1.060920e+004   1
21     1.060920e+004   2
22     1.060920e+004   1
23     1.060920e+004   3
24     1.060920e+004   3
25     1.060920e+004   1
26     1.060920e+004   2
27     1.060920e+004   1
28     1.060920e+004   1
29     1.060920e+004   1
30     1.060920e+004   1
31     1.060920e+004   1
32     1.060920e+004   1
33     1.060920e+004   1
34     8.765280e+003   2
35     8.765280e+003   3
36     8.765280e+003   3
37     8.765280e+003   2
38     8.765280e+003   1
39     8.765280e+003   1
40     8.765280e+003   1
41     8.765280e+003   1
42     8.765280e+003   2
43     8.765280e+003   2
44     8.765280e+003   1
```

Figure 2.3. Shows a data file.


CASRE
File Edit Filters Trend Model Setup Plot Help
Failure counts: fc_test.dat
Plot Results Display Settings Copy Help
+ Raw Data
Number of Failures
Test Interval Number

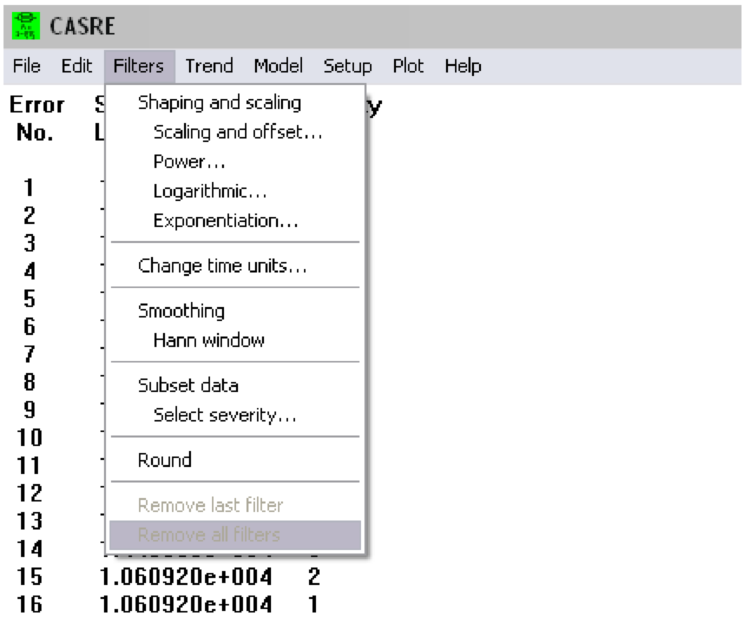
```
Test  Number of       Hours in a      Severity
Intvl Failures        Test Interval

 1    1.400000e+001   5.600000e+001   1
 2    1.900000e+001   5.600000e+001   1
 3    2.300000e+001   5.600000e+001   1
 4    1.200000e+001   5.600000e+001   1
 5    2.200000e+001   5.600000e+001   1
 6    1.200000e+001   5.600000e+001   1
 7    1.300000e+001   5.600000e+001   1
 8    1.900000e+001   5.600000e+001   1
 9    1.000000e+001   5.600000e+001   1
10    5.000000e+000   5.600000e+001   1
11    5.000000e+000   5.600000e+001   1
12    5.000000e+000   5.600000e+001   1
13    7.000000e+000   5.600000e+001   1
14    7.000000e+000   5.600000e+001   1
15    1.000000e+000   5.600000e+001   1
16    3.000000e+000   5.600000e+001   1
17    1.000000e+000   5.600000e+001   1
18    2.000000e+000   5.600000e+001   1
19    0.000000e+000   5.600000e+001   N/A
20    2.000000e+000   5.600000e+001   1
21    9.000000e+000   5.600000e+001   1
22    1.000000e+000   5.600000e+001   1
23    0.000000e+000   5.600000e+001   N/A
24    0.000000e+000   5.600000e+001   N/A
25    0.000000e+000   5.600000e+001   N/A
26    1.000000e+000   5.600000e+001   1
27    1.000000e+000   5.600000e+001   1
```

Figure 2.4. Example on data file with failure count.

Step 5: Filters and smoothing operations

As shown in figure 2.5:

- Shaping and scaling filters for changing the shape of the failure data curve.
- A filter for changing the time units for a failure data set. For example we can change between data form seconds to minutes.
- A Hann window for removing noise from the failure data.
- The capability of selecting a subset of the failure data based on severity classification.
- A filter for rounding the failure data to the nearest whole number.

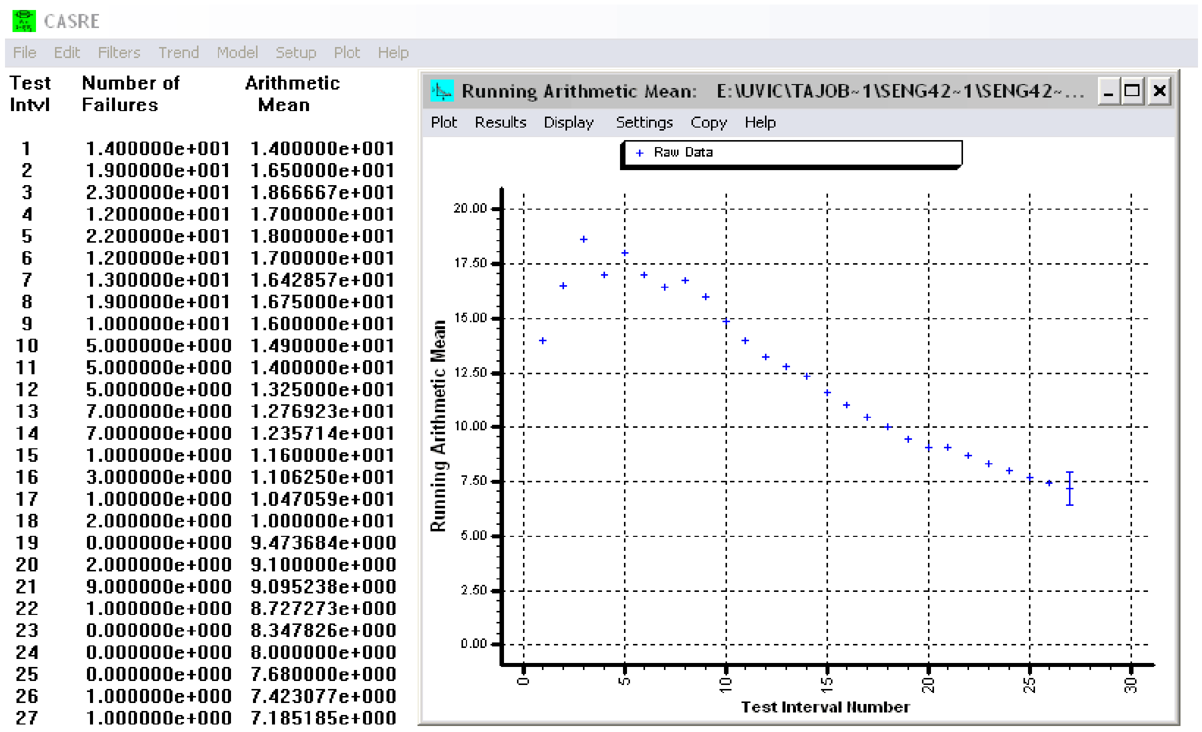


Error No.		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15	1.060920e+004	2
16	1.060920e+004	1

Figure 2.5. Shows Filters Sub Menu.

Step 6: Apply trend tests.

- Determine whether a set of failure data exhibits reliability growth.
- Running Arithmetic Average of Time Between Failures/Failure Counts.
- Laplace Test


CASRE
File Edit Filters Trend Model Setup Plot Help
Running Arithmetic Mean: E:\UVIC\TAJOB~1\SENG42~1\SENG42~...
Plot Results Display Settings Copy Help
+ Raw Data
Running Arithmetic Mean
Test Interval Number
20.00
17.50
15.00
12.50
10.00
7.50
5.00
2.50
0.00
0
5
10
15
20
25
30

Test Intvl	Number of Failures	Arithmetic Mean
1	1.400000e+001	1.400000e+001
2	1.900000e+001	1.650000e+001
3	2.300000e+001	1.866667e+001
4	1.200000e+001	1.700000e+001
5	2.200000e+001	1.800000e+001
6	1.200000e+001	1.700000e+001
7	1.300000e+001	1.642857e+001
8	1.900000e+001	1.675000e+001
9	1.000000e+001	1.600000e+001
10	5.000000e+000	1.490000e+001
11	5.000000e+000	1.400000e+001
12	5.000000e+000	1.325000e+001
13	7.000000e+000	1.276923e+001
14	7.000000e+000	1.235714e+001
15	1.000000e+000	1.160000e+001
16	3.000000e+000	1.106250e+001
17	1.000000e+000	1.047059e+001
18	2.000000e+000	1.000000e+001
19	0.000000e+000	9.473684e+000
20	2.000000e+000	9.100000e+000
21	9.000000e+000	9.095238e+000
22	1.000000e+000	8.727273e+000
23	0.000000e+000	8.347826e+000
24	0.000000e+000	8.000000e+000
25	0.000000e+000	7.680000e+000
26	1.000000e+000	7.423077e+000
27	1.000000e+000	7.185185e+000

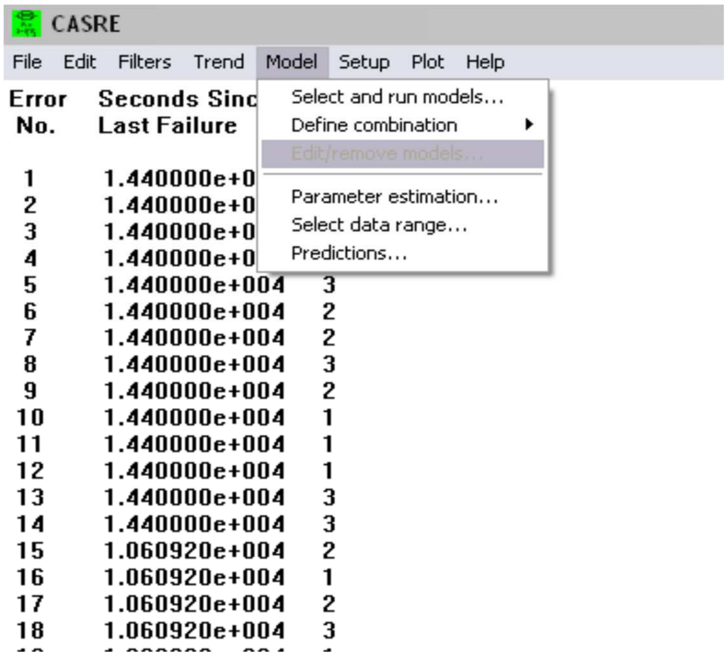
Figure 2.6. The results of applying the running arithmetic average.

The graph shows a decreasing in the failures count.

Step 7, 8: Apply Reliability Model and View Results

Let us look at model results and model evaluation statistics for the following scenario (see figures: 2.7, …, 2.11).

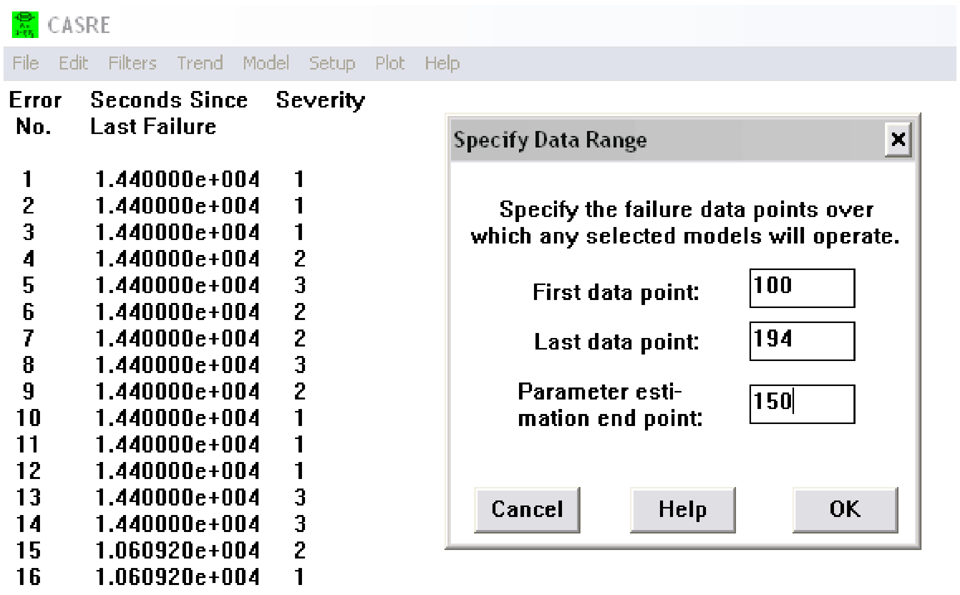
- Failure data: Same as that used before (time between failures data).
- Data Range: We've selected point's 100-194 as the interval to which to apply the data.
- The Parameter Estimation End Point is 150.
- No of future failures is 20.
- Models selected: We'll be looking at the results of the Musa Basic, Musa-Okumoto, Linear-LV, and Quadratic-LV models.



```
Error   Seconds Sinc
 No.    Last Failure

  1      1.440000e+0
  2      1.440000e+0
  3      1.440000e+0
  4      1.440000e+0
  5      1.440000e+004   3
  6      1.440000e+004   2
  7      1.440000e+004   2
  8      1.440000e+004   3
  9      1.440000e+004   2
 10      1.440000e+004   1
 11      1.440000e+004   1
 12      1.440000e+004   1
 13      1.440000e+004   3
 14      1.440000e+004   3
 15      1.060920e+004   2
 16      1.060920e+004   1
 17      1.060920e+004   2
 18      1.060920e+004   3
```

Figure 2.7. Model Sub Menu.

The first move is to click Model from the Main Menu.



```
Error   Seconds Since   Severity
 No.    Last Failure

  1     1.440000e+004   1
  2     1.440000e+004   1
  3     1.440000e+004   1
  4     1.440000e+004   2
  5     1.440000e+004   3
  6     1.440000e+004   2
  7     1.440000e+004   2
  8     1.440000e+004   3
  9     1.440000e+004   2
 10     1.440000e+004   1
 11     1.440000e+004   1
 12     1.440000e+004   1
 13     1.440000e+004   3
 14     1.440000e+004   3
 15     1.060920e+004   2
 16     1.060920e+004   1
```

Specify Data Range

Specify the failure data points over which any selected models will operate.

First data point: 100

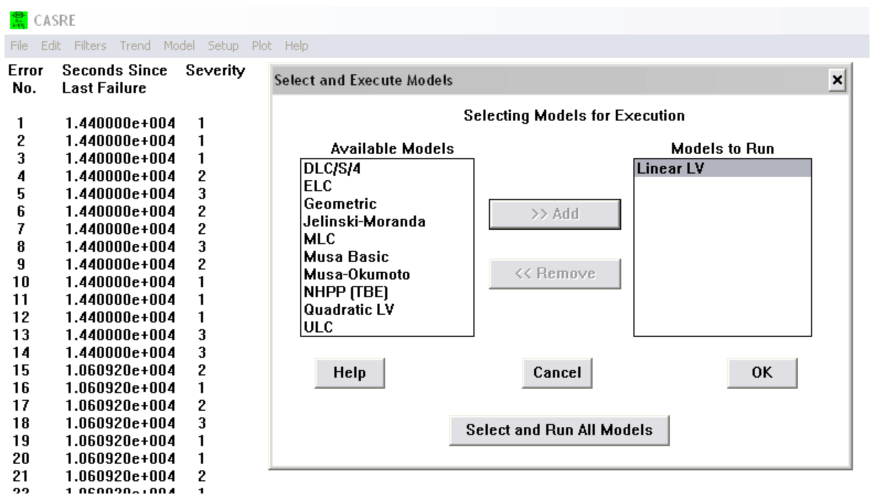
Last data point: 194

Parameter estimation end point: 150

Figure 2.8. Select Data Range.

Model -> Select and run Models…

Then a new dialog will show up (see the next figure).


CASRE
File Edit Filters Trend Model Setup Plot Help
Select and Execute Models
Selecting Models for Execution
>> Add
<< Remove
Help
Cancel
OK
Select and Run All Models

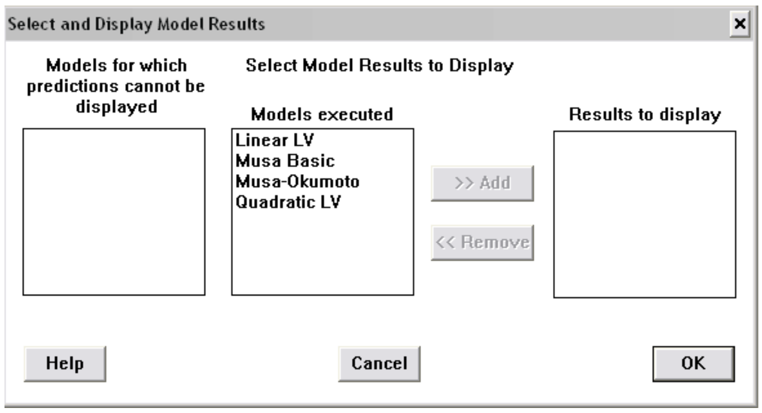
Error No.	Seconds Since Last Failure	Severity
1	1.440000e+004	1
2	1.440000e+004	1
3	1.440000e+004	1
4	1.440000e+004	2
5	1.440000e+004	3
6	1.440000e+004	2
7	1.440000e+004	2
8	1.440000e+004	3
9	1.440000e+004	2
10	1.440000e+004	1
11	1.440000e+004	1
12	1.440000e+004	1
13	1.440000e+004	3
14	1.440000e+004	3
15	1.060920e+004	2
16	1.060920e+004	1
17	1.060920e+004	2
18	1.060920e+004	3
19	1.060920e+004	1
20	1.060920e+004	1
21	1.060920e+004	2
22	1.060920e+004	1

Available Models: DLC/S/4, ELC, Geometric, Jelinski-Moranda, MLC, Musa Basic, Musa-Okumoto, NHPP (TBE), Quadratic LV, ULC

Models to Run: Linear LV

Figure 2.9. Shows the list of models.

From the Graph Window select -> "Select model results"


Select and Display Model Results
Models for which predictions cannot be displayed
Select Model Results to Display
Models executed
Results to display
>> Add
<< Remove
Help
Cancel
OK

Models executed: Linear LV, Musa Basic, Musa-Okumoto, Quadratic LV

Figure 2.10. Shows the select and display model results window.

Then the result will be displayed, in figure bellow

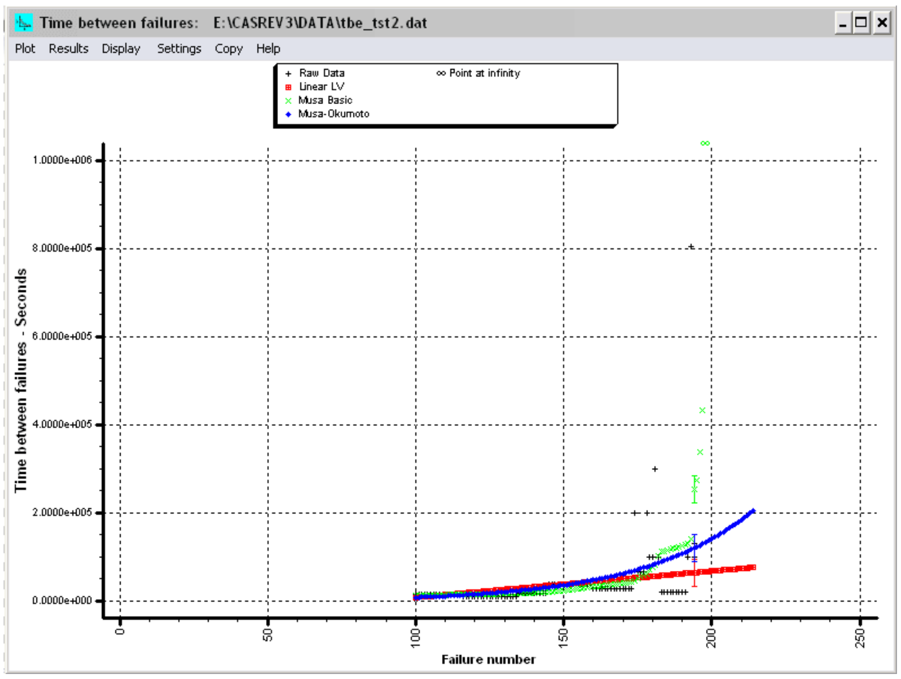


Figure 2.11. Shows the Results.

It shows the results of the Musa Basic, Musa-Okumoto, Linear-LV, and Quadratic-LV models selected in the previous steps.

1.1.2. Advantages of CASRE

CASRE have several advantages that we will mention [30]:

- *Increased time of execution:* Fast execution of tasks especially for the ones with diagramming and associated specifications, and with rate of improvement from 355 to more than 200%.
- *Increased Accuracy:* Due to the debugging and error checking, that's key component in early bug detection and removal.
- *Reduced Lifetime Maintenance:* because of the overall quality of the systems and documentation. The efforts and costs associated with maintenance are reduced , and thanks to CASRE's reengineering tools, it makes this process more efficient, less time consuming and less expensive by updating to latest version .
- *Documentation: A* lot of documents produced during the construction life cycle for better comprehension end explanation of the tool.
- *Facility of the use:* CASRE easy to use and understand by the users, and it leads to less training time and better acceptance of the tool.

1.1.3. Disadvantages of CASRE

- Necessitate significant work in the analysis phase to extract customer needs.
- Difficult to customize.
- Require training for maintenance personnel.
- Does not support cohabitation with other systems.

2. Reliability estimation and prediction tools

2.1. SOFTREL - The software reliability process simulator

SOFTREL mechanism is more based on incorporating the users and computers in all development phases to study the reliability of the life cycle and the effects between different phases, it also admit that testing requires the preparation and utilization of test cases, and that repairs must follow identification and isolation.

Configured to simulate processes having constant event rates per causal unit, It is considered as framework for experimentation, data generation for comparison with actual collected project data, The input to SoftRel is one file that define the dt time slice, and a model that shows the data structure contains about 70 traits of the software project and its reliability process, and a list of activity, schedule, and resource allocations.

Also, internally, the set of status monitors at any given time are stored in a data structure called facts, which records the overall clock time, the time and resources consumed by each activity (42 measures in total), and a snapshot of 48 measures of project status. The output is one file contains the facts series of each dt interval of time [31]. SoftRel simulates two kinds of failure events, defects in specification documents and faults in code.

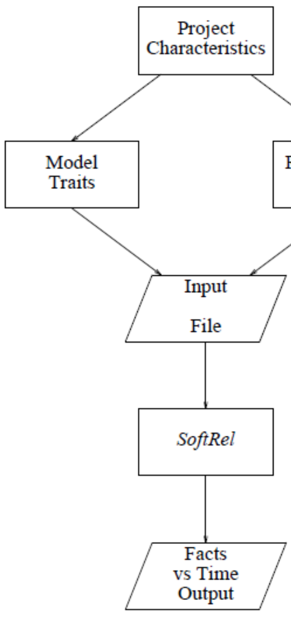


Figure 2.12. SOFTREL execution context [31].

2.1.1. The Major Components of the Simulator

To equal the goal values given in the model, SoftRel is initialized by setting sizes of items for construction, integration, and inspection, but the model values are considered only approximate. We have 14 major components in the simulator:

a. Document Construction: both documentation and there integration, it is approximated to be piece-Poisson, with constant average rates per working day sited in the model, not surpass the target values.

Defects are injected with a constant probability per unit of documentation. At each injection of a default, the document risk rises according to the fault detection characteristic.

b. Document Integration: Based on document reusability by applying small changes like: deletion of undesirable parts, addition of new material, the defects are created as a result of each sub-activity.

Documentation is integrated per average regular paste per day and the defects are injected with a regular probability per unit of documentation. The danger increases each defect according to the fault detection assumed characteristic.

c. Document Inspection: has a similar type to documents construction, it is a goal-limited piecewise-Poisson approximation Documents inspected at mean regular paste per workday.

Inspected units are dived to new documents and reused documents in proportion to the relative amounts of documentation in these two categories.

The defect discovery rate is proportional to the current collected document hazard and the inspection efficiency.

d. Document Correction: The staff level set the rate of the defect correction, and it may lid to add new defects the actual corrections are made according to the efficiency of the correction of defects, not to exceed the actual number of defects discovered, and can change the alter the documents total.

e. Code Construction: With same steps like document building however, the average rate at which mistakes are created is performed by the usual fault the density that is the consequence of coding, and by the density of not discovered defects in documentation and by the amount of missing documentation.

Each fault injected increases the risk of code. But while document defects are only discovered by inspection, code defects are detected both by inspection and by test, and different rates.

f. Code Integration: Has the same structure as document integration, the difference that code units replace document units and coding rates replace documentation rates. Each fault increases the code hazard.

g. Code Inspection: Reflects the document inspection process, but the number of faults discovered will not passe the total number of as-yet undiscovered faults. The rate of fault discovery is proportional to the current accumulated fault hazard and the inspection efficiency. The discovered faults may not yet have been removed at the time of discovery, the number of newly discovered faults is assumed to be in proportion to the number of as-yet undiscovered faults.

h. Code Correction: Apply the same algorithm given for document correction, translated to code units. Fault risque is reduced in the correction of a fault, and can increase in case of new faults are added by the correction process. Documentation alterations are produced at assumed regular mean pace per attempted correction.

i. Test preparation: Produces a number of test cases in each dt, in proportion to the test preparation rate, which is a constant average number of tests per workday.

j. Testing: The testing activity simulation consists of two parts:

If a test crushes in effect, the crush times indicator decrements and the time and effort increment. If a crush is not in effect, failures occur at the modeled rate; the number observed is computed as a binomial process that is regulated by the probability of observation.

The failure rate function returns a value relative to the current risque level. The function consumes computer resources and test cases, the latter at a mean constant rate.

k. Fault Identification: The total number of failures analyzed does not pass the number of failures observed. Failures are analyzed at a mean workday pace, because of the failures still remaining in the system.

The identification of faults is limited in number. The isolation process is regulated by the fraction of faults remaining undiscovered, the adequacy of the analysis process, and the probability of faithful isolation.

l. Fault Repair: The attempted repairs number cannot pass the number of faults identified by inspections and testing, plus faults corrected after inspection, and the identified for rework by validation and retesting.

Of those attempted, a select number will really be repaired, while the rest will wrongly be reported as repaired. Repairs are assumed here to be made on faults identified for rework first.

A select number of new faults may be generated by the attempt, and code units may be changed.

m. Validation of repairs: Validation of repair attempts occurs at a steady pace assumed to be average per workday.
The number of defective repairs detected is a selected number determined by the probability that the validation will recognize an unrepaired failure, if any, and the probability unrepaired defects are among those repair attempts are validated.
n. Retesting: There is an average constant number of retests per workday and consumes computer resources at the planned rate per day, without any new test cases generation, because the original test cases are assumed available for regression.

As example we will run FRESTIMATE TOOL to show the process execution [32], with the following steps:
Step1: open file: We can open an existing file.

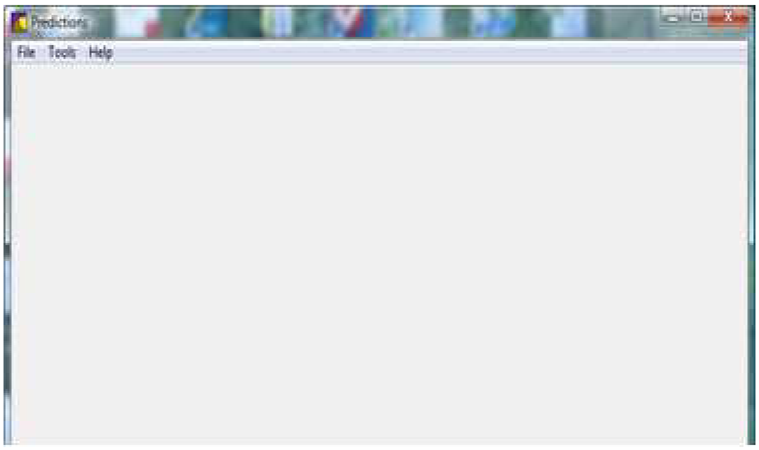


Figure 2.13. Main menu.

After we create a new project (see figure 2.14) another panel will display (figure 2.15), in this panel we need to select a model for predicting defects, after it another panel will open for more inputs.

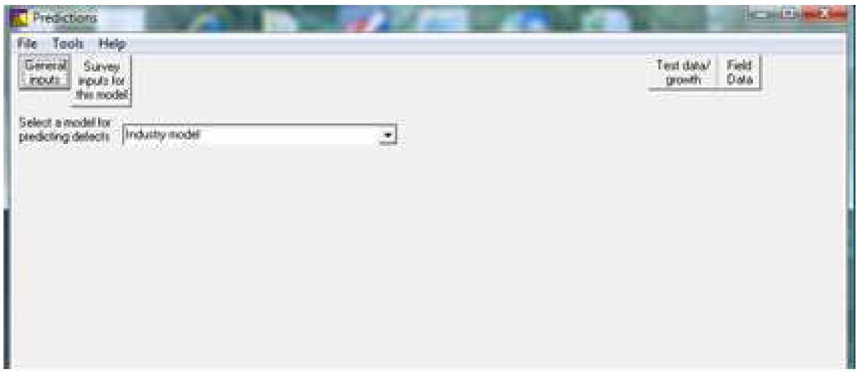
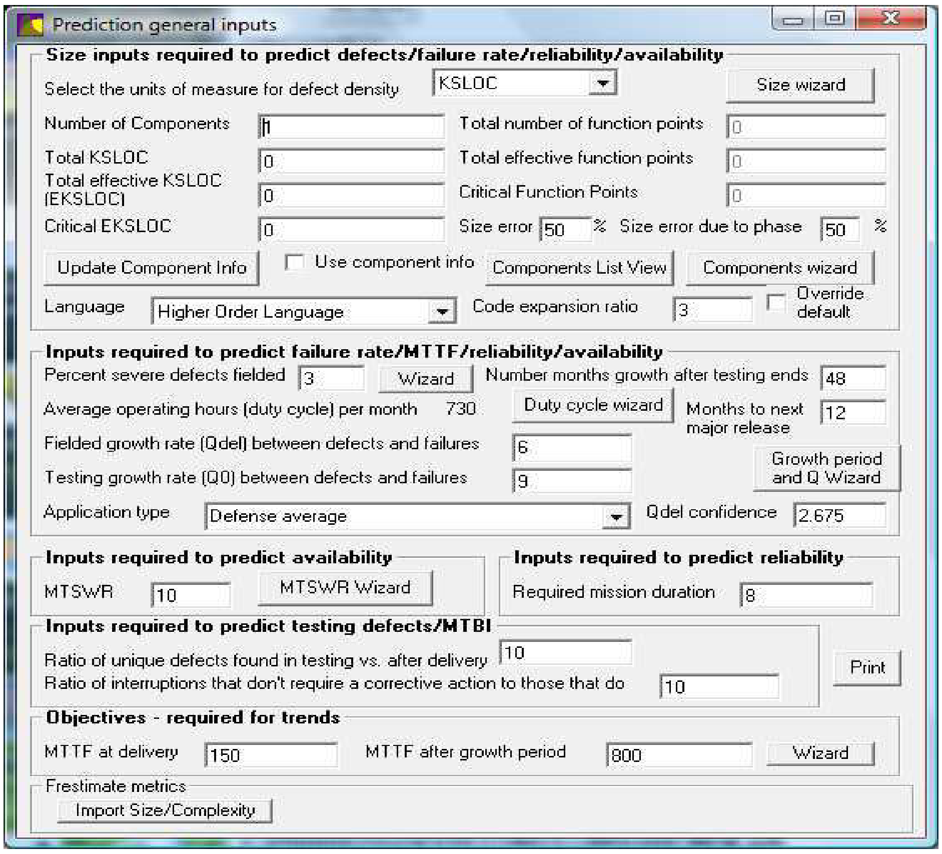


Figure 2.14. New project panel.

Step 2: *Enter General inputs and size.*
When starting a new prediction, you will need to enter a size prediction to see any results [7]. The other inputs have default values which should be reviewed and modified. There are wizards to help you enter these inputs.



Size inputs required to predict defects/failure rate/reliability/availability

Select the units of measure for defect density: KSLOC

Number of Components: 1
Total number of function points: 0
Total KSLOC: 0
Total effective function points: 0
Total effective KSLOC (EKSLOC): 0
Critical Function Points: 0
Critical EKSLOC: 0
Size error: 50 %
Size error due to phase: 50 %
Use component info: ☐
Language: Higher Order Language
Code expansion ratio: 3
Override default: ☐

Inputs required to predict failure rate/MTTF/reliability/availability

Percent severe defects fielded: 3
Number months growth after testing ends: 48
Average operating hours (duty cycle) per month: 730
Months to next major release: 12
Fielded growth rate (Qdel) between defects and failures: 6
Testing growth rate (Q0) between defects and failures: 9
Application type: Defense average
Qdel confidence: 2.675

Inputs required to predict availability

MTSWR: 10

Inputs required to predict reliability

Required mission duration: 8

Inputs required to predict testing defects/MTBI

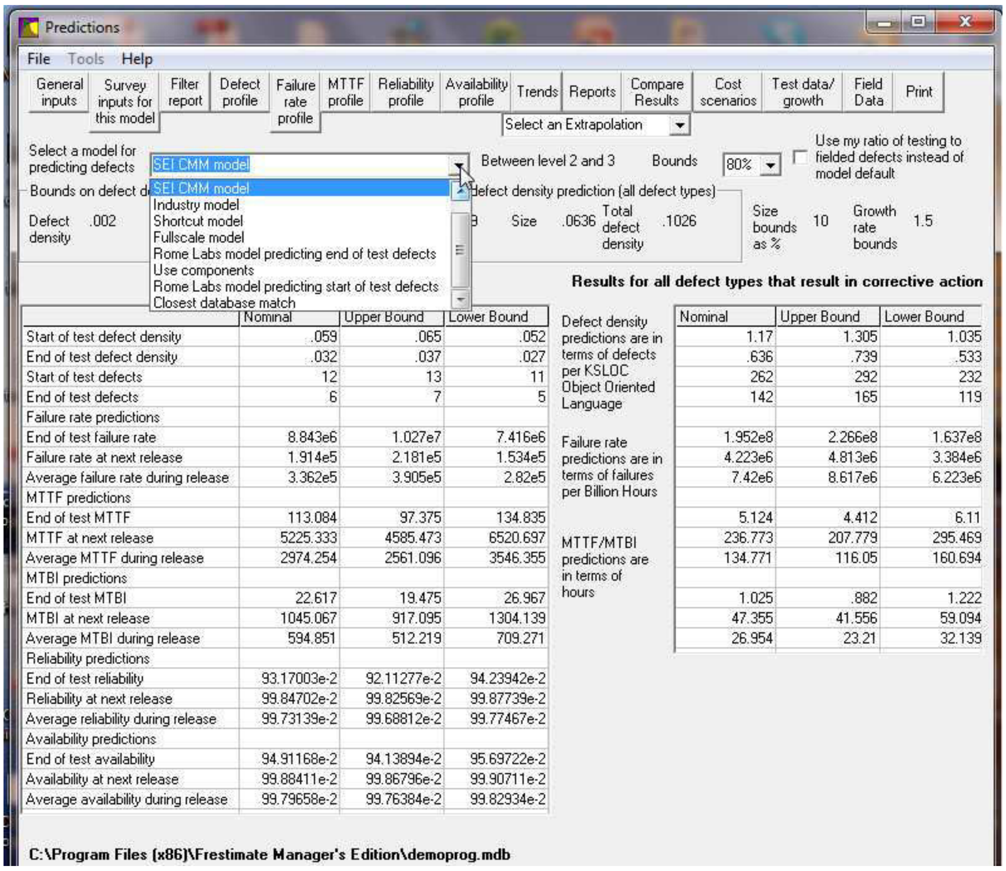
Ratio of unique defects found in testing vs. after delivery: 10
Ratio of interruptions that don't require a corrective action to those that do: 10

Objectives - required for trends

MTTF at delivery: 150
MTTF after growth period: 800

Frestimate metrics

Figure 2.15. Prediction general inputs panel.



Select a model for predicting defects: SEI CMM model
- SEI CMM model
- Industry model
- Shortcut model
- Fullscale model
- Rome Labs model predicting end of test defects
- Use components
- Rome Labs model predicting start of test defects
- Closest database match

Between level 2 and 3   Bounds: 80%   ☐ Use my ratio of testing to fielded defects instead of model default

Bounds on defect d... | ...defect density prediction (all defect types)

Defect density: .002   ...3   Size: .0636   Total defect density: .1026   Size bounds as %: 10   Growth rate bounds: 1.5

Results for all defect types that result in corrective action

	Nominal	Upper Bound	Lower Bound	Nominal	Upper Bound	Lower Bound
Start of test defect density	.059	.065	.052	1.17	1.305	1.035
End of test defect density	.032	.037	.027	.636	.739	.533
Start of test defects	12	13	11	262	292	232
End of test defects	6	7	5	142	165	119
Failure rate predictions						
End of test failure rate	8.843e6	1.027e7	7.416e6	1.952e8	2.266e8	1.637e8
Failure rate at next release	1.914e5	2.181e5	1.534e5	4.223e6	4.813e6	3.384e6
Average failure rate during release	3.362e5	3.905e5	2.82e5	7.42e6	8.617e6	6.223e6
MTTF predictions						
End of test MTTF	113.084	97.375	134.835	5.124	4.412	6.11
MTTF at next release	5225.333	4585.473	6520.697	236.773	207.779	295.469
Average MTTF during release	2974.254	2561.096	3546.355	134.771	116.05	160.694
MTBI predictions						
End of test MTBI	22.617	19.475	26.967	1.025	.882	1.222
MTBI at next release	1045.067	917.095	1304.139	47.355	41.556	59.094
Average MTBI during release	594.851	512.219	709.271	26.954	23.21	32.139
Reliability predictions						
End of test reliability	93.17003e-2	92.11277e-2	94.23942e-2			
Reliability at next release	99.84702e-2	99.82569e-2	99.87739e-2			
Average reliability during release	99.73139e-2	99.68812e-2	99.77467e-2			
Availability predictions						
End of test availability	94.91168e-2	94.13894e-2	95.69722e-2			
Availability at next release	99.88411e-2	99.86796e-2	99.90711e-2			
Average availability during release	99.79658e-2	99.76384e-2	99.82934e-2			

Defect density predictions are in terms of defects per KSLOC Object Oriented Language

Failure rate predictions are in terms of failures per Billion Hours

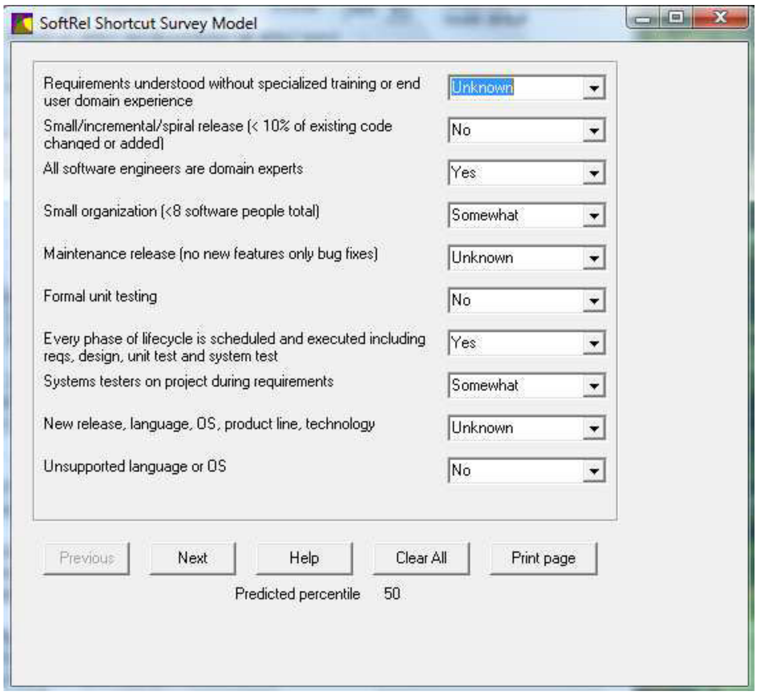
MTTF/MTBI predictions are in terms of hours

C:\Program Files (x86)\Frestimate Manager's Edition\demoprog.mdb

Figure 2.16. Tool option.

In the Tool option (figure 2.16) we select the model of predicting defects, and then select the "Survey Inputs for this Model", we will be then directed to the survey for the selected model, with three kind of surveys "Full scale model", "Full scale model B", "Full scale model C".

ALL prediction surveys were developed by a research organization that collected and organized lots of defect data from many real projects.


SoftRel Shortcut Survey Model

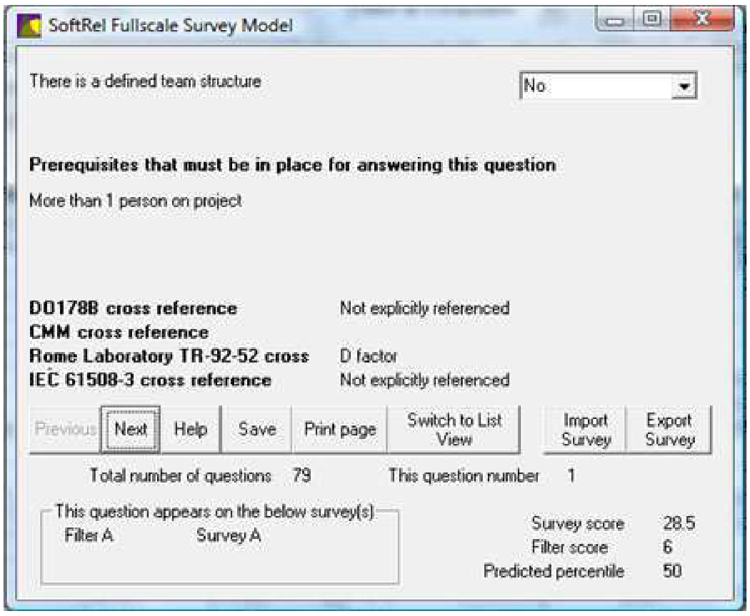
Requirements understood without specialized training or end user domain experience	Unknown
Small/incremental/spiral release (< 10% of existing code changed or added)	No
All software engineers are domain experts	Yes
Small organization (<8 software people total)	Somewhat
Maintenance release (no new features only bug fixes)	Unknown
Formal unit testing	No
Every phase of lifecycle is scheduled and executed including reqs, design, unit test and system test	Yes
Systems testers on project during requirements	Somewhat
New release, language, OS, product line, technology	Unknown
Unsupported language or OS	No

Predicted percentile 50

Figure 2.17. Survey Model.

The defect density is predicted by how many of each you check yes. The prediction formula can be viewed by pressing the Help button.

This is one page of the Full-scale model survey (see figure 2.18). Some surveys have one question, some have a few questions and some have many questions.


SoftRel Fullscale Survey Model

There is a defined team structure: No

Prerequisites that must be in place for answering this question

More than 1 person on project

DO178B cross reference	Not explicitly referenced
CMM cross reference	
Rome Laboratory TR-92-52 cross	D factor
IEC 61508-3 cross reference	Not explicitly referenced

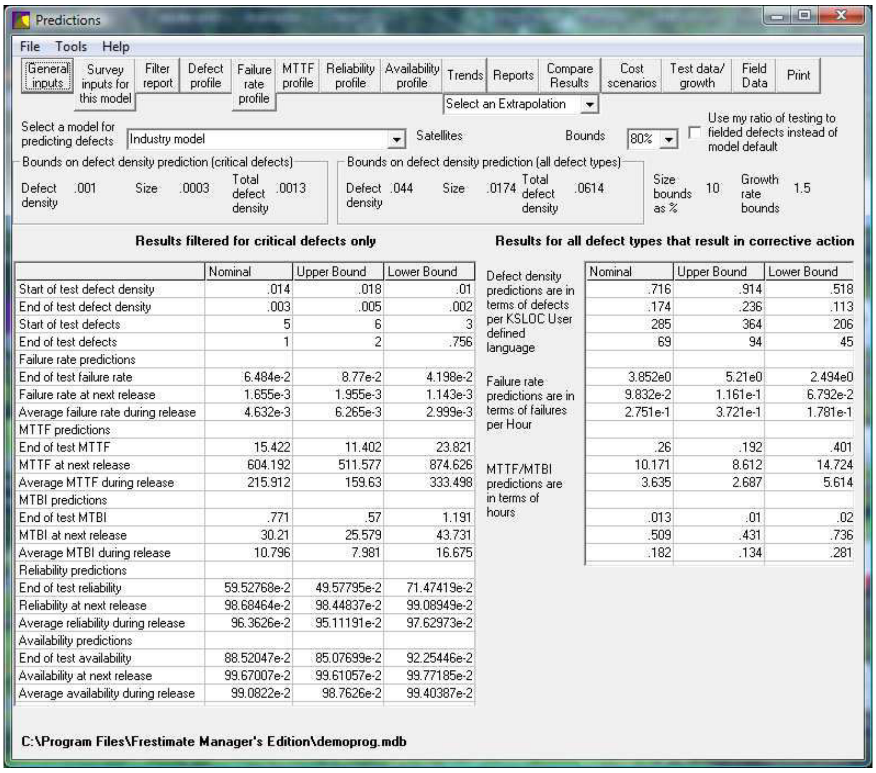
Total number of questions 79 This question number 1

This question appears on the below survey(s)
Filter A Survey A

Survey score	28.5
Filter score	6
Predicted percentile	50

Figure 2.18. Full-scale Survey Model.

Step 3. View results, profiles, trends.



Select a model for predicting defects: Industry model

Satellites

Bounds: 80%

Use my ratio of testing to fielded defects instead of model default

Bounds on defect density prediction (critical defects): Defect density .001 — Size .0003 — Total defect density .0013

Bounds on defect density prediction (all defect types): Defect density .044 — Size .0174 — Total defect density .0614

Size bounds as %: 10 — Growth rate bounds: 1.5

Results filtered for critical defects only

	Nominal	Upper Bound	Lower Bound
Start of test defect density	.014	.018	.01
End of test defect density	.003	.005	.002
Start of test defects	5	6	3
End of test defects	1	2	.756
Failure rate predictions			
End of test failure rate	6.484e-2	8.77e-2	4.198e-2
Failure rate at next release	1.655e-3	1.955e-3	1.143e-3
Average failure rate during release	4.632e-3	6.265e-3	2.999e-3
MTTF predictions			
End of test MTTF	15.422	11.402	23.821
MTTF at next release	604.192	511.577	874.626
Average MTTF during release	215.912	159.63	333.498
MTBI predictions			
End of test MTBI	.771	.57	1.191
MTBI at next release	30.21	25.579	43.731
Average MTBI during release	10.796	7.981	16.675
Reliability predictions			
End of test reliability	59.52768e-2	49.57795e-2	71.47419e-2
Reliability at next release	98.68464e-2	98.44837e-2	99.08949e-2
Average reliability during release	96.3626e-2	95.11191e-2	97.62973e-2
Availability predictions			
End of test availability	88.52047e-2	85.07699e-2	92.25446e-2
Availability at next release	99.67007e-2	99.61057e-2	99.77185e-2
Average availability during release	99.0822e-2	98.7626e-2	99.40387e-2

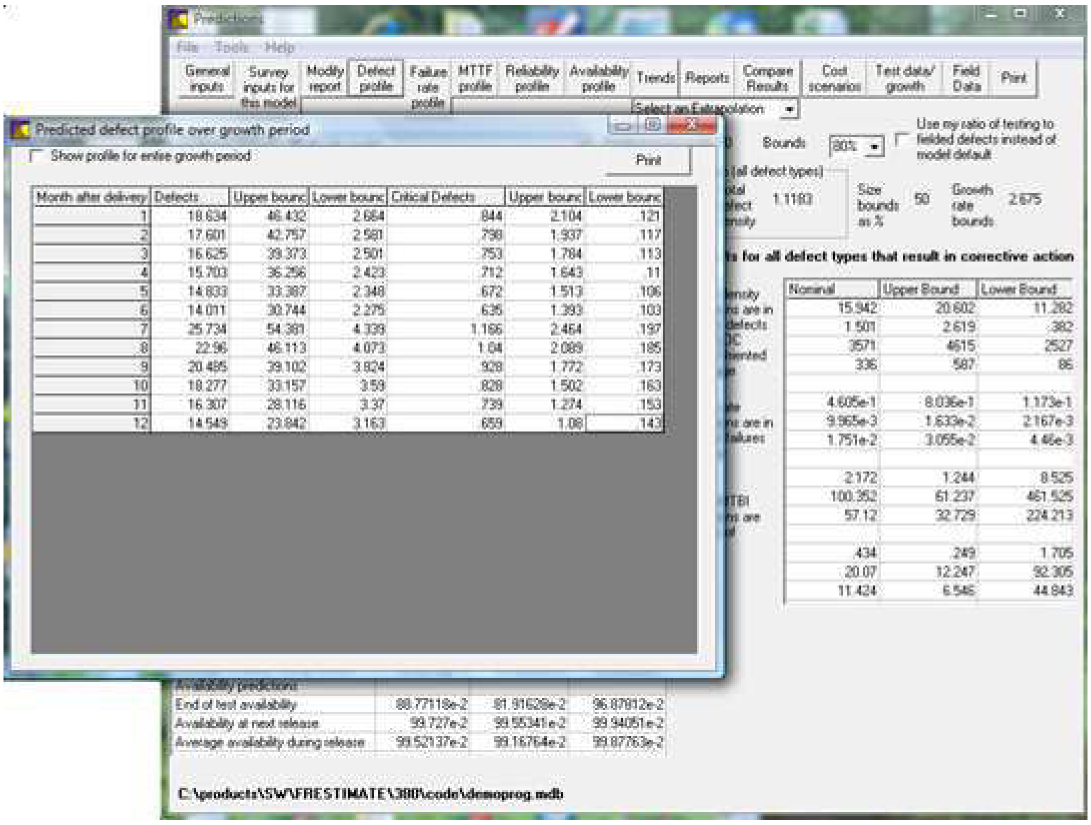
Results for all defect types that result in corrective action

	Nominal	Upper Bound	Lower Bound
Defect density predictions are in terms of defects per KSLOC User defined language	.716	.914	.518
	.174	.236	.113
	285	364	206
	69	94	45
Failure rate predictions are in terms of failures per Hour	3.852e0	5.21e0	2.494e0
	9.832e-2	1.161e-1	6.792e-2
	2.751e-1	3.721e-1	1.781e-1
MTTF/MTBI predictions are in terms of hours	.26	.192	.401
	10.171	8.612	14.724
	3.635	2.687	5.614
	.013	.01	.02
	.509	.431	.736
	.182	.134	.281

C:\Program Files\Frestimate Manager's Edition\demoprog.mdb

Figure 2.19. Prediction panel.

The results are filtered by criticality, we can save the exact page using ‘print’ option, or we can also save this as a report ‘.txt’ using the ‘REPORTS’ option.



Show profile for entire growth period

Month after delivery	Defects	Upper bound	Lower bound	Critical Defects	Upper bound	Lower bound
1	18.634	46.432	2.684	.844	2.104	.121
2	17.601	42.757	2.581	.798	1.937	.117
3	16.625	39.373	2.501	.753	1.784	.113
4	15.703	36.256	2.423	.712	1.643	.11
5	14.833	33.387	2.348	.672	1.513	.106
6	14.011	30.744	2.275	.635	1.393	.103
7	25.734	54.381	4.339	1.166	2.464	.197
8	22.96	46.113	4.073	1.04	2.089	.185
9	20.485	39.102	3.824	.928	1.772	.173
10	18.277	33.157	3.59	.828	1.502	.163
11	16.307	28.116	3.37	.739	1.274	.153
12	14.543	23.842	3.163	.659	1.08	.143

Bounds: 80%

Use my ratio of testing to fielded defects instead of model default

(all defect types) Total defect density 1.1183 — Size bounds as % 50 — Growth rate bounds 2.675

for all defect types that result in corrective action

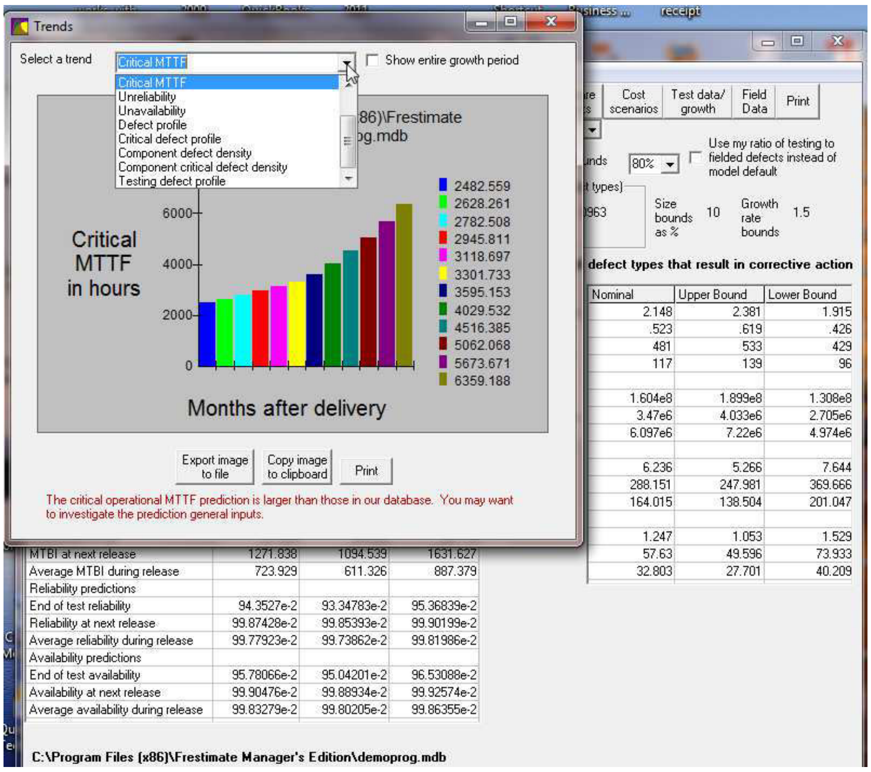
Nominal	Upper Bound	Lower Bound
15.942	20.602	11.282
1.501	2.619	.382
3571	4615	2527
336	587	86
4.605e-1	8.036e-1	1.173e-1
9.965e-3	1.633e-2	2.167e-3
1.751e-2	3.055e-2	4.46e-3
2.172	1.244	8.525
100.352	61.237	461.525
57.12	32.729	224.213
.434	.249	1.705
20.07	12.247	92.305
11.424	6.546	44.843

Availability predictions			
End of test availability	88.77118e-2	81.91628e-2	96.87812e-2
Availability at next release	99.727e-2	99.55341e-2	99.94051e-2
Average availability during release	99.52137e-2	99.16764e-2	99.87763e-2

C:\products\SW\FRESTIMATE\380\code\demoprog.mdb

Figure 2.20. View profiles.

A profile is a metric with respect to some particular point in time.



The critical operational MTTF prediction is larger than those in our database. You may want to investigate the prediction general inputs.

defect types that result in corrective action

Nominal	Upper Bound	Lower Bound
2.148	2.381	1.915
.523	.619	.426
481	533	429
117	139	96
1.604e8	1.899e8	1.308e8
3.47e6	4.033e6	2.705e6
6.097e6	7.22e6	4.974e6
6.236	5.266	7.644
288.151	247.981	369.666
164.015	138.504	201.047
1.247	1.053	1.529
57.63	49.596	73.933
32.803	27.701	40.209

MTBI at next release	1271.838	1094.539	1631.627
Average MTBI during release	723.929	611.326	887.379
Reliability predictions			
End of test reliability	94.3527e-2	93.34783e-2	95.36839e-2
Reliability at next release	99.87428e-2	99.85393e-2	99.90199e-2
Average reliability during release	99.77923e-2	99.73862e-2	99.81986e-2
Availability predictions			
End of test availability	95.78066e-2	95.04201e-2	96.53088e-2
Availability at next release	99.90476e-2	99.88934e-2	99.92574e-2
Average availability during release	99.83279e-2	99.80205e-2	99.86355e-2

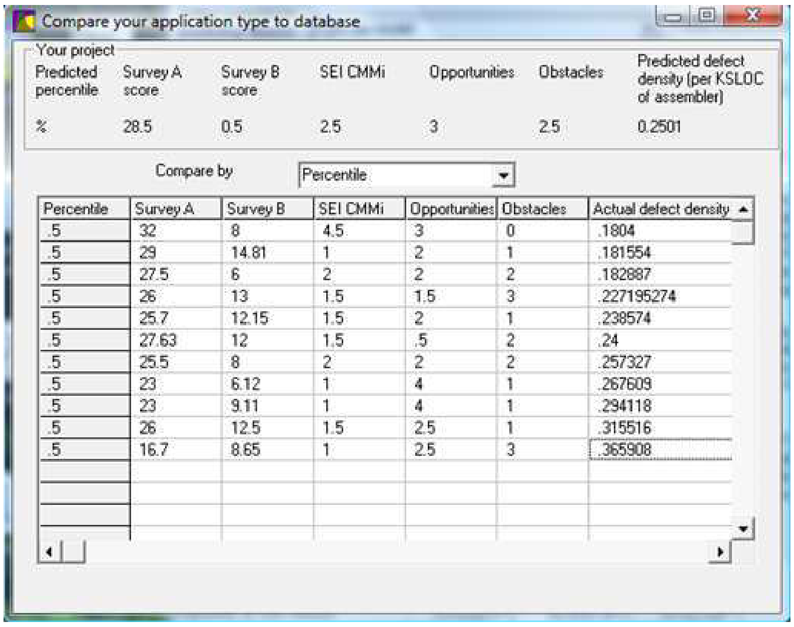
C:\Program Files (x86)\Frestimate Manager's Edition\demoprog.mdb

Figure 2.21. Trends panel.

Press the Trends button. Select any one of the trends from the list. The trends are graphical representations of the profiles and results. You can save them as a bitmap or copy to clipboard or print; we can select or even reduce the result metrics, by using the 'FILTER REPORT' option.

Step 4: Compare the results to others in our DB.

If we select the 'Compare RESULTS' option in the previous panel, it allows us to compare results of different projects in the same domain.



Your project

Predicted percentile	Survey A score	Survey B score	SEI CMMi	Opportunities	Obstacles	Predicted defect density (per KSLOC of assembler)
%	28.5	0.5	2.5	3	2.5	0.2501

Percentile	Survey A	Survey B	SEI CMMi	Opportunities	Obstacles	Actual defect density
.5	32	8	4.5	3	0	.1804
.5	29	14.81	1	2	1	.181554
.5	27.5	6	2	2	2	.182887
.5	26	13	1.5	1.5	3	.227195274
.5	25.7	12.15	1.5	2	1	.238574
.5	27.63	12	1.5	.5	2	.24
.5	25.5	8	2	2	2	.257327
.5	23	6.12	1	4	1	.267609
.5	23	9.11	1	4	1	.294118
.5	26	12.5	1.5	2.5	1	.315516
.5	16.7	8.65	1	2.5	3	.365908

Figure 2.22. Results comparison.

It allows us to conclude decision and get a better view, and see similar projects.

2.1.2. SOFTREL advantages

- Designed for real word systems.
- Test of reliability of each activity.
- Project documentation.
- Provides practical feedback for the users.
- Provides higher reliability due to overlapping tests.

2.1.3. SOFTREL disadvantages

- Time and resources consuming.
- Excessive calculations.
- Necessitate experts' helps of the domain.

2.2. MEADEP - MEAsure and DEPendability

MEADEP is a failure data-based dependability analysis and modeling tool of critical systems, Dependability measures created, have two sources either directly obtained from data, such as failure rate and event distribution, or evaluated by combined use of failure data and dependability models.

It consists of four software modules: a data preprocessor for converting data from different formats to MEADEP format, a data analyzer for graphical data presentation and parameter estimation, a graphical modeling interface for building block diagrams (including the exponential block, Weibull block, and k-out-of-n block) and Markov reward chains, and a model solution module for availability/reliability calculations with graphical parametric analysis.

The result of MEADEP consists of results obtained from data and results evaluated from models [33].

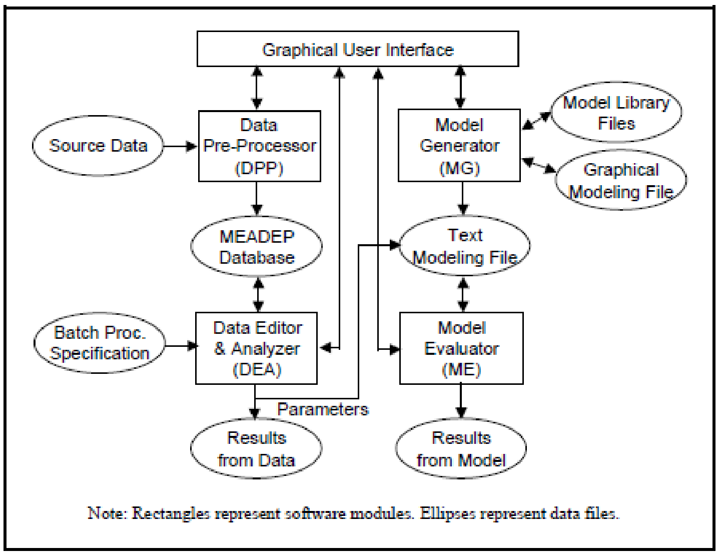


Figure 2.23. Layout of MEADEP [33].

Figure 2.23 illustrate the components of MEADEP. All modules are integrated with the graphical user Interface (GUI).

- The data preprocessor module (DPP) interacts with the user to convert source data to MEADEP internal data. Source data can be manually generated structured reports, usually in database format or computer-generated event logs.
- The Data Editor and Analyzer (DEA) module is used to edit internal data. Data and perform a statistical analysis of the data. The parameter values estimated from the data in this module can be inserted in the text modeling file generated by another module, the model generator (MG) [33].
- The MG module provides a graphical user interface that allows the user to draw model diagrams and then generate text modeling from the diagrams file that contains the appropriate model specifications for the solution. Template diagrams can be imported from library files containing predefined templates to save development time.
- The Model Evaluator (ME) module produces results based on: the specifications and parameters of the model in the text modeling file.

2.2.1. MEADEP modules

a. The DEA module: Works on data converted by the DPP module and performs statistical analysis. It has three categories of functions: data editing, graphical analysis, and parameter estimation.

In addition to individual parameter estimates, it can generate multiple parameters and confidence intervals by processing a query file.

The module can also plot, over a histogram, five different analytical probability distribution functions determined by the sample mean and sample variance: exponential, gamma, Weibull, normal and lognormal. Meanwhile, the estimated parameters for these functions as well as the results of the Chi-Square and Kolmogorov-Smirnov goodness-of-fit tests [34] are provided on the screen.

b. The MG module: Is a graphical "drag and drop" interface for constructing dependability models. A model is developed hierarchically, from the top level to the bottom level, forming a tree-structure. Each node in the tree is diagram of serial or parallel reliability blocks, a diagram of weighted blocks, or a diagram of Markov reward chain.

The user can navigate from one diagram to another to build models.

c. The ME module: Has two principal functions: edit the text modeling file (publisher) and evaluate the model (evaluator). The editor allows the user to review templates and settings, and then see immediately the effects of the revisions on the results.

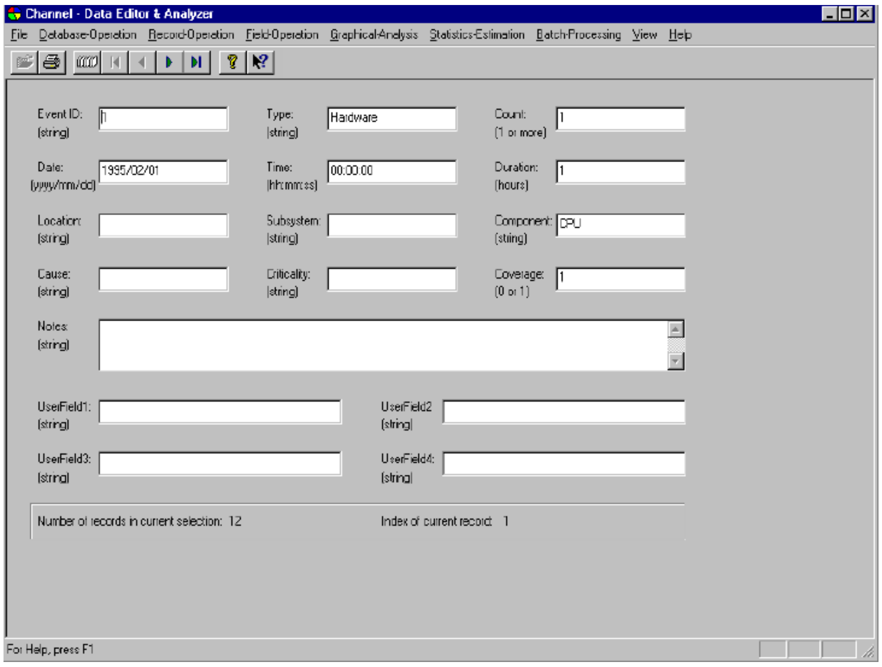
The evaluator provides regular results and parametric analysis. For the regular results, the modeling file is evaluated once and the results of all the diagrams (models) listed in the specification are generated.

In parametric analysis, the user specifies a loop and several sets of results are generated graphically for one or more diagrams.

2.2.2. MEADEP on screen

The MEADEP installation program will guide you through the process and at the end of the installation [34].

In the end of the installation you should find the following MEADEP modules: Data Editor & Analyzer, Data Pre-Processor, Model Evaluator, Model Generator, and User's Manual.


Channel - Data Editor & Analyzer
File Database-Operation Record-Operation Field-Operation Graphical-Analysis Statistics-Estimation Batch-Processing View Help
For Help, press F1

Event ID: (string)	1	Type: (string)	Hardware	Count: (1 or more)	1
Date: (yyyy/mm/dd)	1995/02/01	Time: (hh:mm:ss)	00:00:00	Duration: (hours)	1
Location: (string)		Subsystem: (string)		Component: (string)	CPU
Cause: (string)		Criticality: (string)		Coverage: (0 or 1)	1
Notes: (string)					
UserField1: (string)		UserField2 (string)			
UserField3: (string)		UserField4: (string)			

Number of records in current selection: 12 Index of current record: 1

Figure 2.24. Sample Database [34].

Click on the Data Editor & Analyzer (DEA) icon on the desktop, after the DEA main form appears, choose the "Open" command under the File menu and then follow the instructions below:

1. In the "Open" box, go to the C:\MEADEP\Example directory (unless you specify otherwise), pick the database file called "Plant.mdb" and then click the "Open" button.
2. In the "Choose Table Name" box, choose the "Channel" table and click "OK".

It contains an example database, contains information on component failures in an assumed digital safety system in a plant for a period of two years (1995/1/1 to 1996/12/31). This form contains 12 records that holds information's about the possible failures. We can draw this information's using the DEA graph capabilities:

- Select the "Event Pie Chart" option in the Graphical-Analysis menu.
- Under "Select a field by which to draw chart" choose "Component" and click"OK".
- Specify "4" for the number of top items and click "OK".

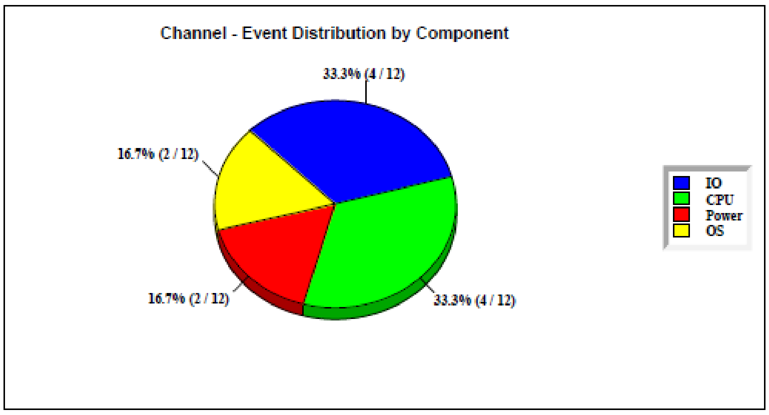


Figure 2.25. Example of Event Pie Chart [34].

We can also draw an MTBE Line Graph (Mean Time Between Events Line Graph).

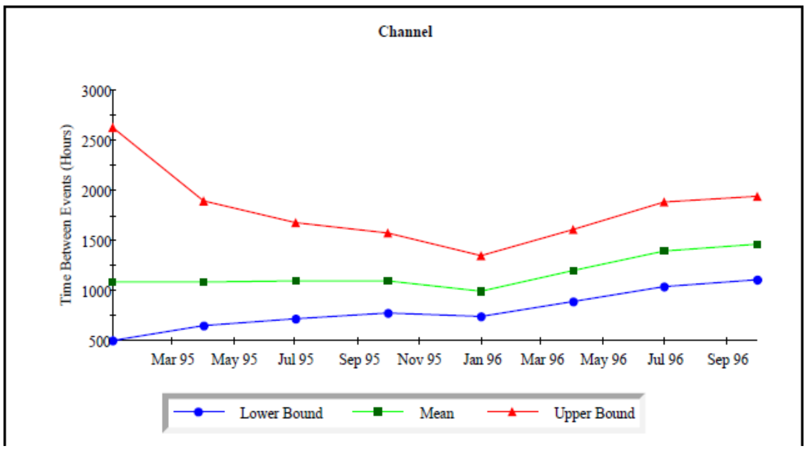


Figure 2.26. MTBE Line Graph [34].

1. Select the "MTBE Line Graph" option in the Graphical-Analysis menu.
2. Choose a value of 0.8 for the confidence level, also, enter the amount of time to be considered in the calculation.

For this example, let us pick a time interval from January 1, 1995 to December 31, 1996. Finally, enter "3 months" for the time between plotted points and click "OK".

2.2.3. MEADEP avantages

- Based on measurements dependability.
- Dedicated for critical systems.
- Ability to handle sensitive parameters.

2.2.4. MEADEP disadvantages

- For expert users.
- Require previous domain analysis.
- Executed on windows OS only.

2.3. SREPT - Software Reliability Estimation and Prediction Tool

SREPT developers knew the importance of keeping track of the software quality during the entire life cycle development of the software. It has several techniques adequate to each phase. All of it goes under a unified framework for software reliability estimation and prediction. SREPT combines the capabilities of the existing tools in a unified framework [35].

SREPT supports the following two approaches to software reliability prediction—the *black-box*-based and the *architecture*-based approaches.

2.3.1. Design and architecture of SREPT

a. Approaches based on a black box

Black box approaches treat the software as a whole regardless of its internal structure. The following measures can be obtained to help predict the black box:

- Software/process product metrics include number of lines of code, number of decisions, loops, the average length of the variable names and other static attributes of the code, or characteristics of the process.

- Test coverage is defined as the ratio of potential failure sites exerted by test cases divided by the total number of potential failure sites in the study [35].

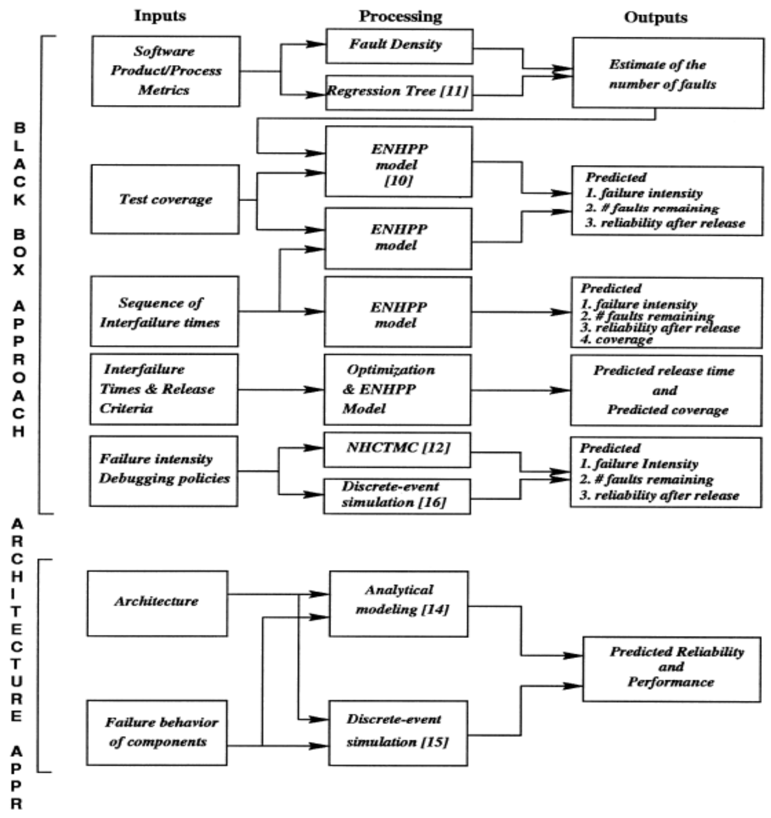

Inputs
Processing
Outputs
BLACK BOX APPROACH
ARCHITECTURE APPR
Software Product/Process Metrics
Fault Density
Regression Tree [11]
Estimate of the number of faults
Test coverage
ENHPP model [10]
ENHPP model
Predicted 1. failure intensity 2. # faults remaining 3. reliability after release
Sequence of Interfailure times
ENHPP model
Predicted 1. failure intensity 2. # faults remaining 3. reliability after release 4. coverage
Interfailure Times & Release Criteria
Optimization & ENHPP Model
Predicted release time and Predicted coverage
Failure intensity Debugging policies
NHCTMC [12]
Discrete-event simulation [16]
Predicted 1. failure Intensity 2. # faults remaining 3. reliability after release
Architecture
Analytical modeling [14]
Failure behavior of components
Discrete-event simulation [15]
Predicted Reliability and Performance

Figure 2.27. Architecture of SREPT [36].

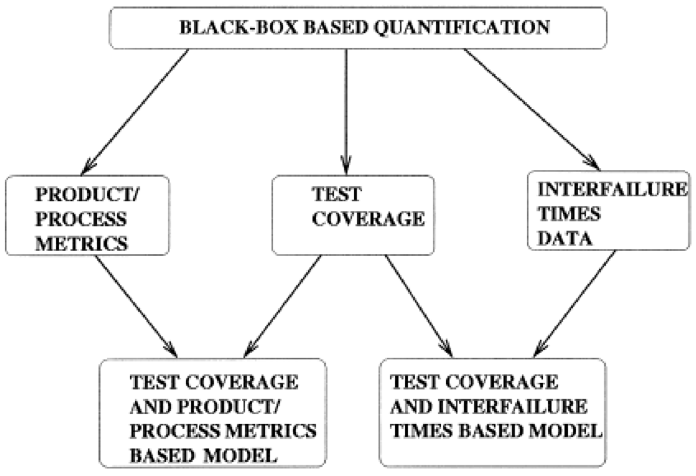

BLACK-BOX BASED QUANTIFICATION
PRODUCT/ PROCESS METRICS
TEST COVERAGE
INTERFAILURE TIMES DATA
TEST COVERAGE AND PRODUCT/ PROCESS METRICS BASED MODEL
TEST COVERAGE AND INTERFAILURE TIMES BASED MODEL

Figure 2.28. Black-box quantification [36].

- Inter-failure time data refers to the times observed between failures during software testing.

When product / process metrics are available, the total number of defects in the software can be estimated using the fault density approach or the regression tree model. In the fault density approach, experience from similar projects in the past is used to estimate the fault density (FD) of the software as: FD = total number of faults ÷ number of lines of code.

Now, if the number of lines of code in the current software is *N*L, the expected number of faults can be estimated as: FD = NLFD.

- The regression tree model is a goal-oriented statistical technique that attempts to predict the number of failures in a software module based on static complementarity metrics.

We use historical data sets to build the tree which is then used as a forecasting device for the current project. Inter-failure times data obtained from the test phase can be used to parameterize the ENHPP model (non-homogeneous enhanced Poisson process) to obtain estimates of the intensity of the failure, the number of remaining faults, the reliability after the publication and coverage of the software.

Coverage functions include Exponential, Weibull, S-shaped, Log-logistic [36].

b. Architecture Based Approach

Software reliability predicted using the internal software control structure. This assumes additional importance in assessing the reliability of modern software systems that are not monolithic entities, but are likely to consist of several modules distributed around the world.

SREPT can predict reliability based on:

- *Architecture of the Software:* specify how the different modules in the software interact, and are given by the inter-modular transition probabilities, or in a very broad sense, the operational profile of the software.

The architecture of the application can be modeled as a DTMC (Markov chain in discrete time), CTMC (Markov chain in continuous time), SMP (Semi-Markov process) or DAG (directed acyclic graph).

The state of the application at any time is given by the module running at that moment, and state transitions represent the transfer of control between the modules.

- *Failure Behavior:* This specifies the behavior in case of module failure and that interfaces between modules, in terms of probability of failure (or reliability) or failure rate (constant / time dependent). Transitions between modules may be either instantaneous or there may be an overhead in terms of time.

2.3.2. SREPT advantages

- User friendly.
- Graphics supporting.
- Estimation and prediction tool.

2.4. SMERFS- Statistical Modeling and Estimation of Reliability Functions for Software

The Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) is one of earliest tools that proved his efficiency, it incorporates eight different models, due their performance in comparative studies and for embedding collected data from homogeneous testing domains. IT incorporates eight different models, four using as input data the time between error occurrences and four using the number of detected errors per testing period.

The former includes: Littlewood and Verrall's model [37], Moranda's geometric model [38], John Musa's execution-time model [39], and an adaptation of Goal's non-homogeneous Poisson process (NHPP) model to time between-error data [39].

The latter models include: The generalized Poisson model [35], Goel's NHPP model [40], Brooks and MotIey's model [41], and Norman Schneidewind's model [42].

The process of determining the parameters of these models based on maximum likelihood and least squares.

2.4.1. The main functionalities that SMERFS provides [43, 44]:

- Data input.
- Data edit.
- Unit conversions.
- Data transformations.
- Data statistics.
- Plot(s) of the raw data.
- Model applicability analyses.
- Execution of the models.

The previous represents the SMERFS structure, which gives us more general idea on the different steps that can be established with this tool.

Data input

Smerfs offers two ways to enter the input data by a previously created ASCII file or the computer keyboard, with the ability to append new data on the currently executed; a different dataset cannot be analyzed only at the end of the program.

Edit module

In case of error during data input or discovered errors in the error vector values this module allows altering it instead of destroying the entire data file a various modification options can be performed like insert, combine, changing or deleting specified elements.

Unit conversions module

It concerns the time unit's transformation from unit to another, its useful in handling with history files, with a specified scale:

a. 60.0 for seconds to minutes.
b. 60.0 for minutes to hours.
c. 24.0 for hours to days.
d. 07.0 for days to weeks.
e. 04.0 for weeks to months.
f. 12.0 for months to years.

Transformation module

It allows the scaling of a software error data vector. With Five types of transformations are allowed as well as options for restoring the data vector in its unprocessed state and listing the data. The available transformation options are: LOG (A* X(I) + B), EXP(A * X(I) + B), *X* **A, X+A, X * A, restore the data, and list the current data.

Statistics module

This module generates summary statistics of the software error data, and its automatically generated without the user interferes.

From the obtained values we mention: MEDIAN OF THE DATA, LOWER & UPPER HINGES, MINIMUM AND MAXIMUM, NUMBER OF ENTRIES, AVERAGE OF THE DATA, *SKEWNES* & KURTOSIS…

Raw data module

'Raw data' expression is to distinguish real data and not predicted by the tool, this module generates plots of this data, which are the result of an internal tracer.

The internal plotter produces very coarse graphics to help the user during a quick interactive review of the data.

Model applicability module

SMERFS performs analyses to determine the models applicability on the failure data

All four analyses Accuracy (Prequential Likelihood), Bias, Noise, and Trend) have been implemented for the execution time models. Only the Accuracy has been implemented for the interval data models.

SMERFS on screen

We will try to demonstrate the main steps that we will apply:

Step 1: Running SMERFS.

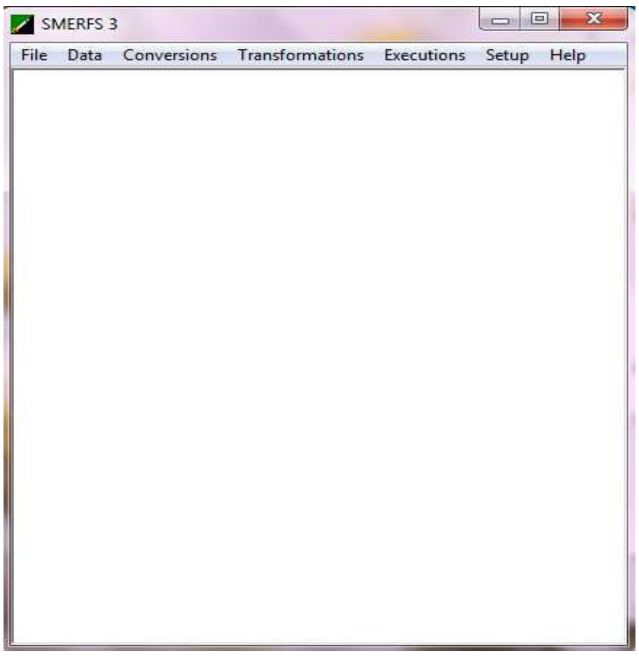


Figure 2.29. SMERFS home panel.

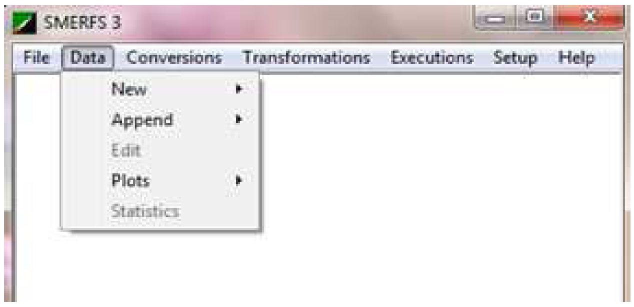


Figure 2.30. Data option.

The header toolbar contains all the options from data, transformation, execution …etc.

Step 2: Choosing the data set.

The first thing we need to enter a data set

- Select "Data" from the toolbar.
- Select "New".
- Select "File"

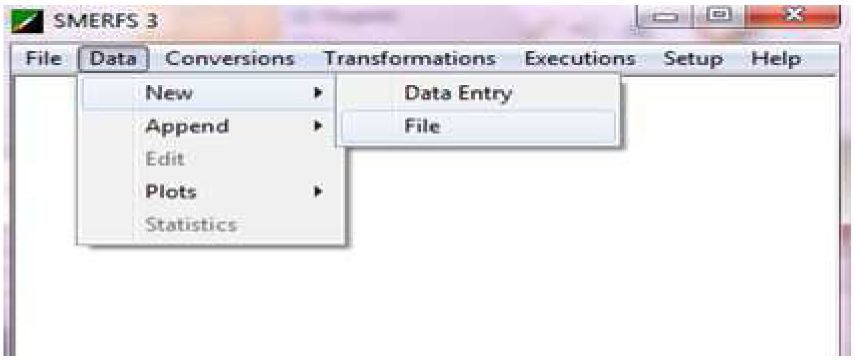


Figure 2.31. Data addition.

If we choose "Data Entry" instead of "File" and specify the data, Data Type and Error History manually, The SMERFS package contains data samples that we can execute or we can create our own data using NOTEPAD, Respecting the SMERFS data format which can be software, system or hardware time between error or interval counts as example we have:

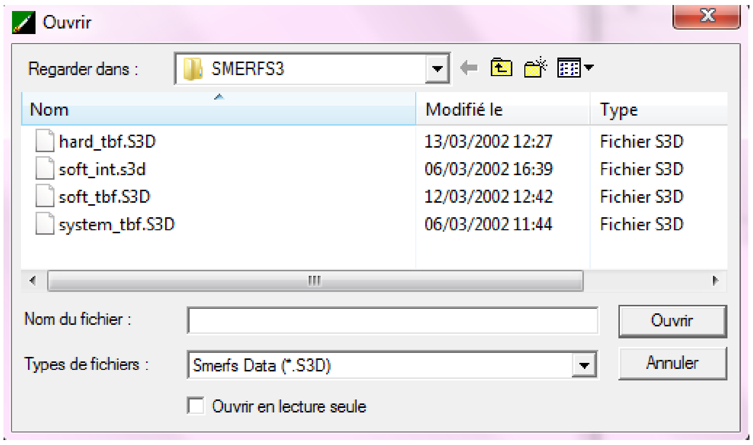
For Software "Time Between Errors" the format is:

Column 1	*Column 2*
"Elapsed Time"	"0" to denote a software failure
1.00000	0
1.50000	0
2.00000	0
2.50000	0

For Software "Interval Counts" the format is:

Column 1	*Column 2*
"Software Error Count"	"Interval length" usually normalized to 1 (1 day, week ,,,etc.)
20.0	1.0
18.0	1.0
45.0	1.0
62.0	1.0
63.0	1.0

In our case I will use a sample from the SMRF TOOL package; a window will show up. In this window we can choose the data files for the execution, another window will be opened (figures: 2.32 and 2.33). This window is to specify the data type executed with two options time between failure data or interval data counts. We will choose TBF.



Ouvrir

Regarder dans : SMERFS3

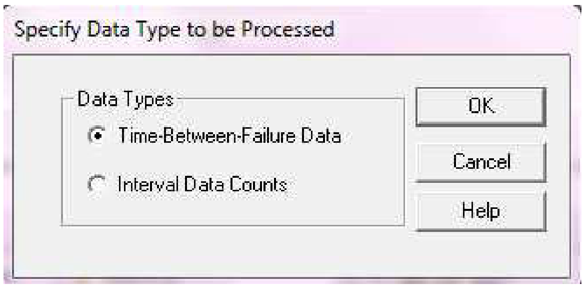
Nom	Modifié le	Type
hard_tbf.S3D	13/03/2002 12:27	Fichier S3D
soft_int.s3d	06/03/2002 16:39	Fichier S3D
soft_tbf.S3D	12/03/2002 12:42	Fichier S3D
system_tbf.S3D	06/03/2002 11:44	Fichier S3D

Nom du fichier :

Types de fichiers : Smerfs Data (*.S3D)

Ouvrir en lecture seule

Figure 2.32. File selection.



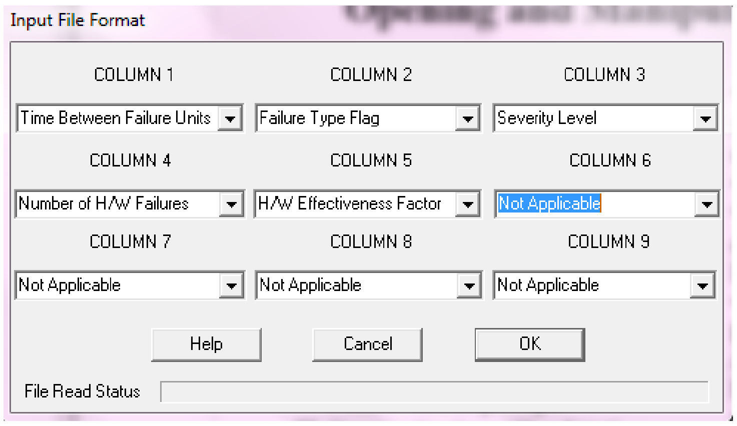
Specify Data Type to be Processed

Data Types

[✓ Time-Between-Failure Data]

Interval Data Counts

Figure 2.33. Data type.



Input File Format

COLUMN 1	COLUMN 2	COLUMN 3
Time Between Failure Units	Failure Type Flag	Severity Level
COLUMN 4	COLUMN 5	COLUMN 6
Number of H/W Failures	H/W Effectiveness Factor	Not Applicable
COLUMN 7	COLUMN 8	COLUMN 9
Not Applicable	Not Applicable	Not Applicable

File Read Status

Figure 2.34. Input file format.

In window shown in figure 2.34 we can indicate a parameter to each column. We click ok to proceed.

Step 3: Execution option.

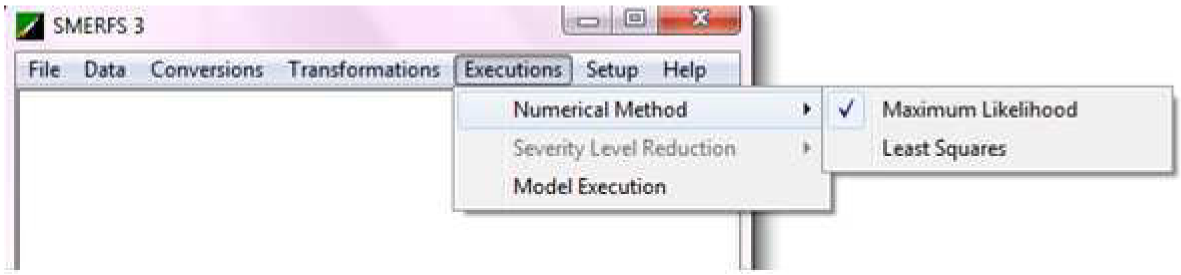
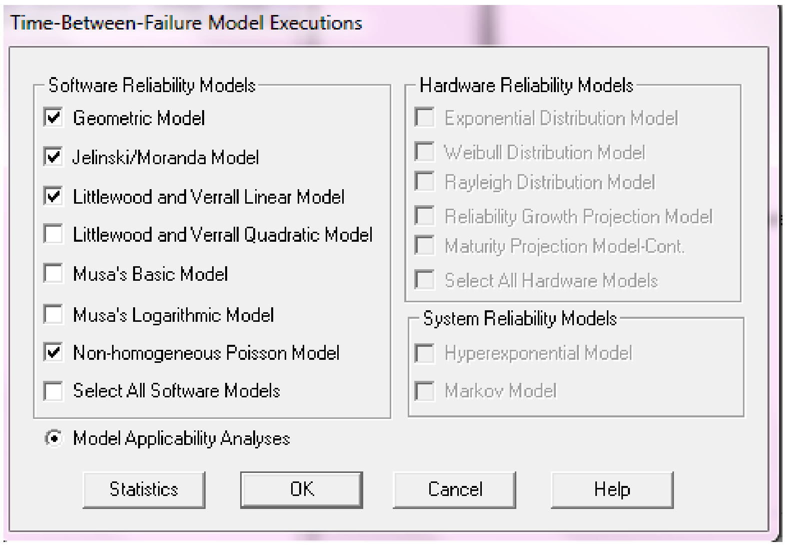


Figure 2.35. Execution panel.

In Execution we can choose the Numerical Method wanted Maximum Likelihood or Least Squares in choose the Maximum Likelihood method.
To choose the models we click: Execution, Numerical Method and Model Execution.


Time-Between-Failure Model Executions

Software Reliability Models
- [✓] Geometric Model
- [✓] Jelinski/Moranda Model
- [✓] Littlewood and Verrall Linear Model
- [ ] Littlewood and Verrall Quadratic Model
- [ ] Musa's Basic Model
- [ ] Musa's Logarithmic Model
- [✓] Non-homogeneous Poisson Model
- [ ] Select All Software Models

Hardware Reliability Models
- [ ] Exponential Distribution Model
- [ ] Weibull Distribution Model
- [ ] Rayleigh Distribution Model
- [ ] Reliability Growth Projection Model
- [ ] Maturity Projection Model-Cont.
- [ ] Select All Hardware Models

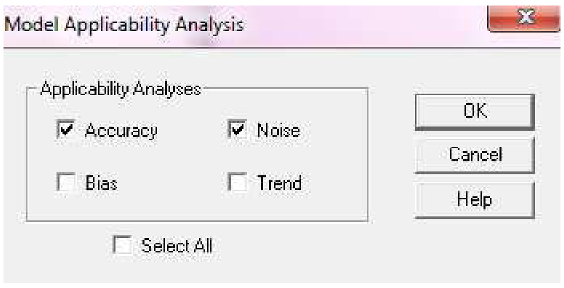
System Reliability Models
- [ ] Hyperexponential Model
- [ ] Markov Model

[✓ Model Applicability Analyses]

Statistics | OK | Cancel | Help

Figure 2.36. Models execution.

In this window we select the models and we can use the model Applicability Analyses option, for accuracy comparison of models.


Model Applicability Analysis

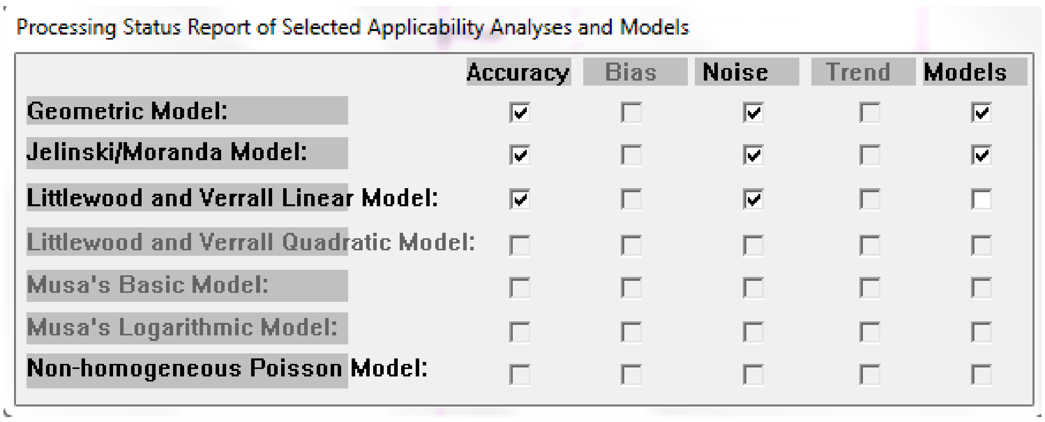
Applicability Analyses
- [✓] Accuracy
- [✓] Noise
- [ ] Bias
- [ ] Trend

[ ] Select All

OK | Cancel | Help

Figure 2.37. Model Applicability Analyses.

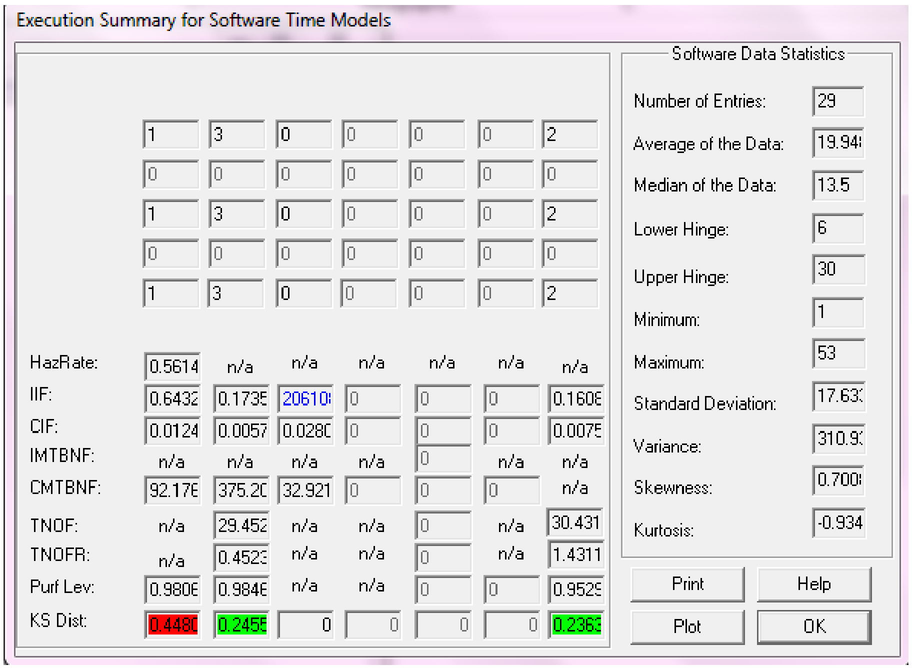
In this panel we can select the model Applicability Analyses methods.


Processing Status Report of Selected Applicability Analyses and Models

	Accuracy	Bias	Noise	Trend	Models
Geometric Model:	✓		✓		✓
Jelinski/Moranda Model:	✓		✓		✓
Littlewood and Verrall Linear Model:	✓		✓		
Littlewood and Verrall Quadratic Model:					
Musa's Basic Model:					
Musa's Logarithmic Model:					
Non-homogeneous Poisson Model:					

Figure 2.38. Analyses report.

Then the result of the Accuracy Analyses is presented in figure 2.38, the selected models are applicable.


Execution Summary for Software Time Models

	1	3	0	0	0	0	2
	0	0	0	0	0	0	0
	1	3	0	0	0	0	2
	0	0	0	0	0	0	0
	1	3	0	0	0	0	2
HazRate:	0.5614	n/a	n/a	n/a	n/a	n/a	n/a
IIF:	0.643	0.1735	20610	0	0	0	0.1608
CIF:	0.0124	0.0057	0.0280	0	0	0	0.0075
IMTBNF:	n/a	n/a	n/a	n/a	0	n/a	n/a
CMTBNF:	92.176	375.20	32.921	0	0	0	n/a
TNOF:	n/a	29.452	n/a	n/a	0	n/a	30.431
TNOFR:	n/a	0.4523	n/a	n/a	0	n/a	1.4311
Purf Lev:	0.9806	0.9846	n/a	n/a	0	0	0.9529
KS Dist:	🟥 0.4480	🟩 0.2455	0	0	0	0	🟩 0.2363

Software Data Statistics

Number of Entries:	29
Average of the Data:	19.94
Median of the Data:	13.5
Lower Hinge:	6
Upper Hinge:	30
Minimum:	1
Maximum:	53
Standard Deviation:	17.63
Variance:	310.9
Skewness:	0.700
Kurtosis:	-0.934

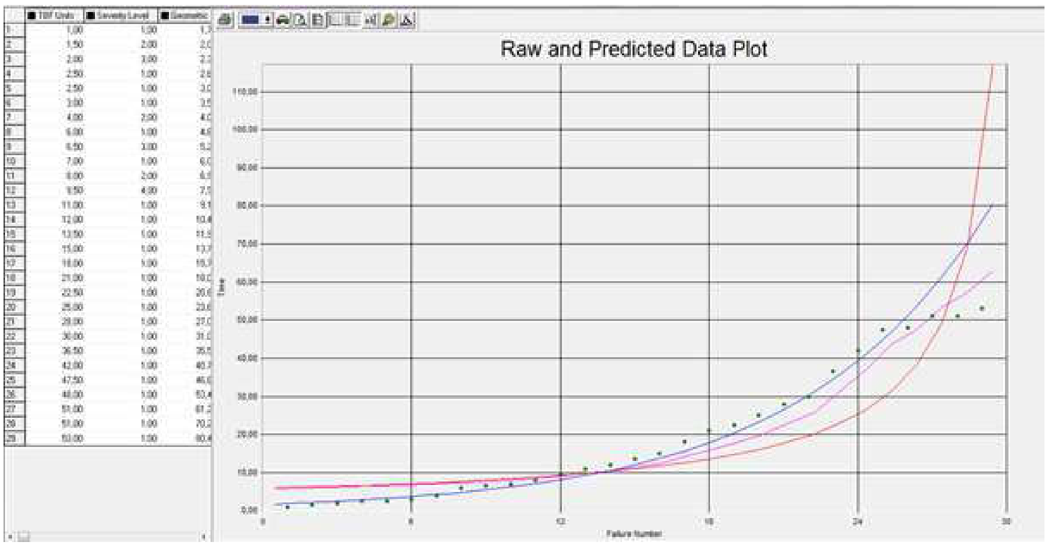
Figure 2.39. Execution Summary.

In the execution summary panel, the green values are successful fit for, and the red values not applicable models, we click on Plot to draw the graph.

In the right column we have the software Statistics with multiple values like:

- The median of the data: It represents the value such that 50 percent of the data have values below it; analogously, 50 percent of the values are greater.
- The minimum and maximum values are, respectively, the smallest and largest values in the data base; the number of entries simply shows the number of points in the data base.
- The lower and upper hinges represent a breakup of each of the two sections of the data, determined by the median, into two equal parts. They both are a measure of the spread of the data.

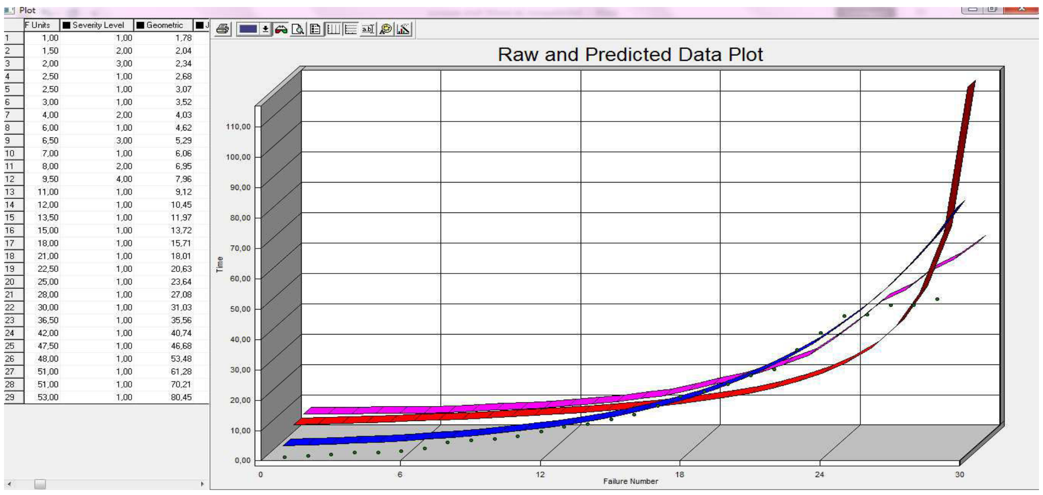
Step 4: The plot.



	TBF Units	Severity Level	Geometric
1	1.00	1.00	1.3
2	1.50	2.00	2.0
3	2.00	3.00	2.2
4	2.50	1.00	2.6
5	2.50	1.00	3.0
6	3.00	1.00	3.5
7	4.00	2.00	4.0
8	6.00	1.00	4.6
9	6.50	3.00	5.2
10	7.00	1.00	6.0
11	8.00	2.00	6.5
12	9.50	4.00	7.5
13	11.00	1.00	9.1
14	12.00	1.00	10.4
15	13.50	1.00	11.5
16	15.00	1.00	13.7
17	18.00	1.00	15.7
18	21.00	1.00	19.0
19	22.50	1.00	20.6
20	25.00	1.00	23.6
21	28.00	1.00	27.0
22	30.00	1.00	31.0
23	36.50	1.00	35.5
24	42.00	1.00	40.7
25	47.50	1.00	46.0
26	48.00	1.00	53.4
27	51.00	1.00	61.2
28	51.00	1.00	70.2
29	53.00	1.00	80.4

Figure 2.40. RAW and Predicted Data Plot.

In the left side we can scroll to see the models selected values and the TBF and Severity level in time. In the top of the panel we have few options like the 3D-glasses. And we can see other options like print and ZOOM option to know the exact coordination's of any point in the plot and other options.



	F Units	■ Severity Level	■ Geometric	■ J
1	1,00	1,00	1,78	
2	1,50	2,00	2,04	
3	2,00	3,00	2,34	
4	2,50	1,00	2,68	
5	2,50	1,00	3,07	
6	3,00	1,00	3,52	
7	4,00	2,00	4,03	
8	6,00	1,00	4,62	
9	6,50	3,00	5,29	
10	7,00	1,00	6,06	
11	8,00	2,00	6,95	
12	9,50	4,00	7,96	
13	11,00	1,00	9,12	
14	12,00	1,00	10,45	
15	13,50	1,00	11,97	
16	15,00	1,00	13,72	
17	18,00	1,00	15,71	
18	21,00	1,00	18,01	
19	22,50	1,00	20,63	
20	25,00	1,00	23,64	
21	28,00	1,00	27,08	
22	30,00	1,00	31,03	
23	36,50	1,00	35,56	
24	42,00	1,00	40,74	
25	47,50	1,00	46,68	
26	48,00	1,00	53,48	
27	51,00	1,00	61,28	
28	51,00	1,00	70,21	
29	53,00	1,00	80,45	

Figure 2.41. Plot on 3D.

2.4.2. Advantages of SMERFS

From the features of the SMERFS we mention [43]:

- *Maintainability:* Ensure it by complying with the structured programming standards in a Naval Surface Weapons Center (NSWC) publication, this document direct code generation toward top-down design.

  Additionally, the document contains scenarios with details author, purpose, description, Restrictions, local glossary, errors, associated subprograms, references, language declarations, and formats.
- *Complete Reliability Analysis Environment:* IT aims the completeness of the output, with the eight models, additional modules integrated like data input, data edit, transformations of the data, general summary statistics of the data, plots of the originally collected data, plots of the original and predicted values according to the fitted model, and a goodness-of-fit module to aid in determining the model adequacy.
- *Interactive in Nature:* Giving the user the experience of control was important factor, and this experience was guided by the choices that the user makes while the program execution and it was established by SMERFS interactive mode.

  Through various menus and questions, and the user inputs a response via the terminal keyboard. Free-format input of user responses was elected in order to reduce potential operational errors.
- *Error Detection Capability:* The program has a complete error detection code in place.

  This significant it was designed with the ability to emit information error message and still guided by the user, if it enters an illegal value response to a prompt or to the numeric procedure it can jeopardize the select of the proper model.

- *Machine Transportability:* Is designed to be compatible with all operating systems. It has been written in perfect accordance with FORTRAN V declarations approved by ANSI. This should minimize the transportability difficulties of the machine, but not eliminate all the problems. Therefore, SMERFS has been divided into two sections, a library and a "pilot".
  Currently, two of these drivers are available, one for a VAl 1 l / 780 with an operating system (OS) VAX / VMS 4.4 and the other for a 170f865 CDC with a NOS 2.2 OS. For all other system, the only driver modifications would be the Date and Time functions, possibly the graphs package, and the input/output instructions for special 'I/O' characteristics imposed by the operating system.

2.4.3. SMERFS Disadvantages

- complexity of the program construction
- More chances of the human error.
- Lack of specification due to the number of models embedded.

3. Comparison

The reliability tools comparison is a way of showing the point of strength and weakness of the compared tools and common points also. There are different articles who worked such comparisons with different aspects and criterions like [29][52]. In this part we will do a comparison between the tools viewed and detailed in the previous part.

3.1. Criterions

Based on different criterions that we will explain first:

- ***Language:*** A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks. Both tools CASRE and SMERFS uses FORTRAN language which is mostly used in numeric computation and scientific computing, SOFTREL created with C language an efficient procedural programming language, MEADEP coded with CV++ language which is Visual C++ a Microsoft implementation language of C++, SREPT coded with java which is class based object-oriented language.

- ***Operating system:*** The OS factor is very important because it reflects the tool transportability. MEADEP, SOFTREL, SREPT executed on WINDOWS environment, CASRE and SMERFS UNIX-WINDOWS environment.

- ***Number of models:*** This means the evaluation models and the reliability test models. CASRE has 14 models plus 2 evaluation models, SMERFS with 8 models plus 4 evaluation models, MEADEP based on Markov chain model, SREPT based on ENHPP model, SOFTREL based on piecewise-Poisson Markov processes with explicitly defined event rate functions.

- ***Models for Estimation/Prediction:*** Estimation is done with the failure data of the software but the prediction is done throw surveys or measurements evaluation, CASRE is an estimation tool, SOFTREL, MEADEP, SREPT, and SMERFS are hybrid tools.

- ***Phase of application:*** The first approach is assessment of the system in a later phase, typically, by test like CASRE and SMERFS the second approach modeling of a system in the design phase which used mostly in the reliability prediction during the system life cycle like MEADEP, SOFTREL and SREPT.
- ***Input type:*** Mostly there are three sources of input to apply reliability estimation or prediction. SMERFS incorporates eight different models, four using as input data the time between error occurrence's and four using the number of detected errors per testing period. CASRE input in the form of inter-failure times or failure frequencies. SREPT input consist of Complexity Metrics, Inter-failure Times & Release Criteria, Inter-failure times, Architecture, failure behavior of components. Input to SOFTREL consists of a single file that specifies the dt time slice, about 70 traits of the software project and its reliability process, and a list of activity, schedule, and resource allocations. Input to MEADEP are Data structured failure reports containing information on failure time, location, type, impact and other failure characteristics and reliability blocks.
- ***Input format:*** Can be an ASCII input data set (contains multiple fields: set of values consisting either of the time between discoveries of defects or the number of defects discovered per time period, or can be one of inter-failure times data only, inter-failure times and coverage data, or estimated faults and coverage data) like: CASRE, SMERFS, SOFTREL, SREPT. Or like MEADEP (which can be in a variety of formats such as ASCII Delimited Text, Access, dBASE, Paradox, etc.).
- ***Output type:*** All the tools output has similarity points and differences due to the model's nature applied in each model. CASRE output consists of inter-failure times/failure frequencies actual and estimated, cumulative failure actual and estimated, and reliability growth actual and estimated. For SREPT the output is estimation of the number of faults, failure intensity, fault remaining, reliability, estimated coverage. SOFTREL output is in form of facts data structure consists of products CPU, resources, fault, failure and outage values. MEADEP output is failure rate, recovery rate, and coverage, Time Between Events, Time to Recovery (TTR) distributions, event distribution, Mean Time Between Events distributions. Finally, SMERFS output consist of Total number of faults, Number of faults remaining, expected reliability for a specified time, Number of failures expected in a specified time.
- ***User friendly:*** We mean by it the ease flow of the tool use, by the available menus and their simplicity, which is a common option between all the tools.
- ***Graphics:*** This option is very important. MEADEP and SREPT both offer this option with multiple views of the results like histogram graphs, pie chart graphs and line graphs…etc.

3.2. Comparison table

In the next part we will summarize all the above criterions for each tool in a table:

Parametre	Tools	CASRE	SMERFS	SOFTREL	SREPT	MEADEP
Application Phase		TEST		Life Cycle		
Models Type		Estimation	Hybrid			
Language		FORTRAN	FORTRAN	C	JAVA	VC++
Operating System		UNIX/WIDOWS		WINDOXS		
Number of Models		16	14	2	1	1
Input Type	Failure data	✓	✓	✓	✓	✓
	Architecture	✗	✗	✗	✓	✗
	Parametre	✗	✗	✓	✓	✓
Input Format		ASCII TXT				✴
Output type :	Reliability	✓	✓	✓	✓	✓
	Total Failure	✓	✓	✓	✓	✗
	Remaining Failure	✓	✓	✓	✓	✗
Graphics :		✗ 3	✗ 3	✗ 2	✓ 0	✓ 2
		USER FRIENDLY				

Table 2.1. Reliability tools comparison.

✗ : no.

✓ : yes.

✴ : ASCII Delimited Text, Access, dBase, Paradox, etc.

Hybrid: Estimation and prediction tool.

3.3. Tools ranking

Tool	Ranking
SREPT	**1**
MEADEP	**2**
SOFTREL	**3**
SMERFS	**4**
CASRE	**5**

Table 2.2. Tools ranking.

This ranking is based on the previous tools study, and mainly on our criterions that we set in the comparison; as we see *the hybrid tools has better outcomes than the estimation tools*.

Conclusion

In this chapter, we detailed each tool going through their main functionalities with their interfaces, advantages and disadvantages.

We concluded with comparison between the tools, proposing parameters for comparison. As a conclusion, the reliability testing tools showed their performance in the industry, as obvious result the hybrid tools that are based on both estimation and reliability prediction gives better results due to the early detection and faults avoid in the design phase with the prediction models and then validation of this results with the estimation models in the test phase leading to reduction in time and cost consumed in the software construction.

Chapter 3
Multi Agent Systems

"It is strange that only extraordinary men make the discoveries
which later appear so easy and simple."
Georg C. Lichtenberg

Introduction

Multi-agent systems (MAS) have received considerable attention and have been developed in different disciplines to solve complex problems by subdividing them into smaller tasks.

In this chapter we have entered to the Distributed artificial intelligence (DAI) then to one of his subfields Multi Agent Systems concentrating on the agents as an entity.

1. Distributed Artificial Intelligence

Distributed artificial intelligence (DAI) is a subfield of Artificial Intelligence that is used to solve complex real-world problems [45].
It comprises three different areas. These are parallel AI, Distributed problem solving (DPS) and Multi-agent systems (MAS).

Multi-agent systems deal with the behavior of the computing entities available to solve a given problem. In a multi-agent system, each computing entity is referred to as an agent [46].

According to P.G. Balaji and D. Srinivasan 'MAS can be defined as a network of individual agents that share knowledge and communicate with each other in order to solve a problem that is beyond the scope of a single agent.'

2. Agent

According to Russell, 1997 an agent is an entity that perceives its environment and act on it. And according to IBM, Smart agents are software entities that perform operations in the place of a user or another program, with some kind of independence or of autonomy, and to do that they use some kind of knowledge or representation of goals or desires of the user.
The goal of each agent is to accomplish his task with conditions as a deadline. the agent first detects the environment settings. With this data, the agent can acquire knowledge about the environment.

It can use the knowledge of his neighbors. This knowledge as well as the history of previous actions undertaken and the objective are fed by an inference engine that decides the appropriate action to be taken by the agent [47].

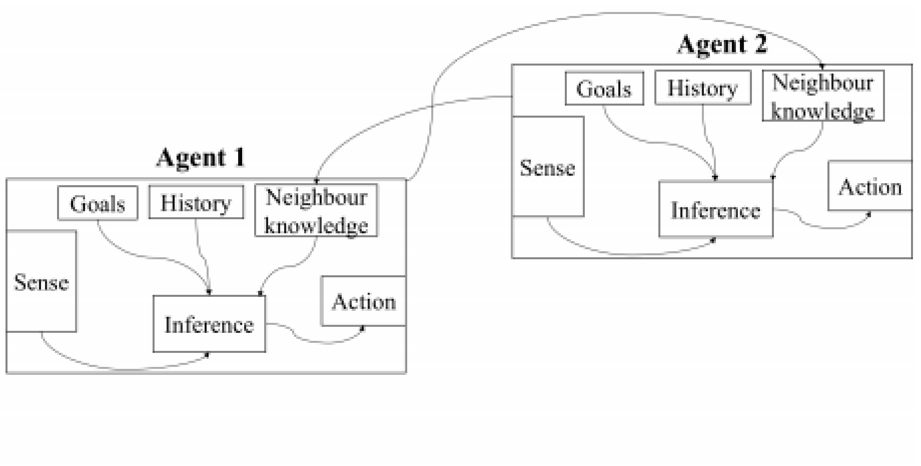


Figure 3.1 The structure of an agent [47].

From these previous definitions we conclude that agent is a smart entity that is driven by it goals with the ability to react and analyze his environment. through the features that it has like [46]:

- *Sociability:* This is the ability of change information between agents as form of a request.
- *Autonomy:* the ability to perform actions independently.
- *Proactivity:* based on its history, the detected parameters, and information from other agents to predict the possible future actions, as a result, agents take better actions to serve their goals.

An agent works alone is able to take actions (autonomy), the actual profit of the agents can only to be exploited when they work in collaboration with others agents. The result of this collaboration is multi-agent systems (MAS).

3. Multi-Agent Systems

The agents play different roles in MAS but all for serving a common goal that all share, it's more obvious with the MAS main features:

Features	**Categories**
Leadership	Leader-follow Leaderless
Decision function	Linear Non-linear
Heterogeneity	Heterogeneous Homogenize
Agreement parameters	First order Second order High order
Delay consideration	Time delay Without time delay
Topology	Static topology Dynamic topology
Data transmission frequency	Time triggered Event triggered

Mobility	Static agents Mobile agents

Table 3.1. MAS features [47].

3.1. MAS features

- ***Leadership:*** done by agent that plays the role of leader which, it defines the roles of agents, the MAS also can be leaderless in this case the agent's acts autonomously based on its own goals.
- ***Decision function:*** MAS is classified as follows: linear and nonlinear. In linear MAS, the decision of an agent is proportional to the detected parameters of the environment, In non-linear MAS, the decision of the agent is not proportional to the measurements detected because of the non-linearity of entry into the decision-making process.
- ***Heterogeneity:*** On the basis of the heterogeneity of MAS agents, we have: homogeneous and heterogeneous. A homogeneous MAS includes agents who all have the same features and functionality, while heterogeneous MAS include agents with various characteristics.
- ***Agreement parameters:*** In some applications of MAS, agents need to agree on particular parameters known as metrics. Based on the number of metrics, MAS are classified as first, second or higher order.
- ***Delay consideration:*** Agents can face multiple sources of delay to perform Tasks. MAS can be classified in two groups, namely with or without delay, most real-world applications experience significant delays.
- ***Topology:*** MAS topology can be static or dynamic. In a static topology, the position and relationships of an agent remain unchanged throughout the life of the agent. In dynamic MAS topology, an agent's position and relationships change as the agent moves, leaves, or joins. MAS, or establishes new communications.
- ***Data transmission frequency:*** Agents detect the environment and share the detected data with other agents either in a time or event triggered way, in the time triggered, it continuously sends the data to other agents. In MAS triggered by an event, when a particular event occurs, the agent sends the collected data to other agents.
- ***Mobility:*** Static or mobile agents, a static agent is always located in same position in the environment, while mobile agents can move in the environment, a mobile agent can be hosted by other agents and use their resources.

3.2. Classification of Multi Agent System

The classification of MAS is a difficult task as it can be done based on several different attributes such as Architecture, Learning, Communication, Coordination.

A general classification encompassing most of these features is shown in the figure bellow [46].

- **Internal Architecture:** multi-agent system, it may be classified as two types [46]:

1. Homogeneous structure

2. Heterogeneous structure

a) Homogeneous Structure: all agents forming the multi-agent system have the same internal architecture. Internal architecture refers to the Local Goals, Sensor Capabilities, Internal states, Inference Mechanism and Possible Actions. environment. There may be overlap in the sensor inputs received. In a typical distributed environment, overlap of sensory inputs is rarely present.

b) Heterogeneous Structure: the agents may differ in ability, structure and functionality, Based on the dynamics of the environment and the location of the particular agent.

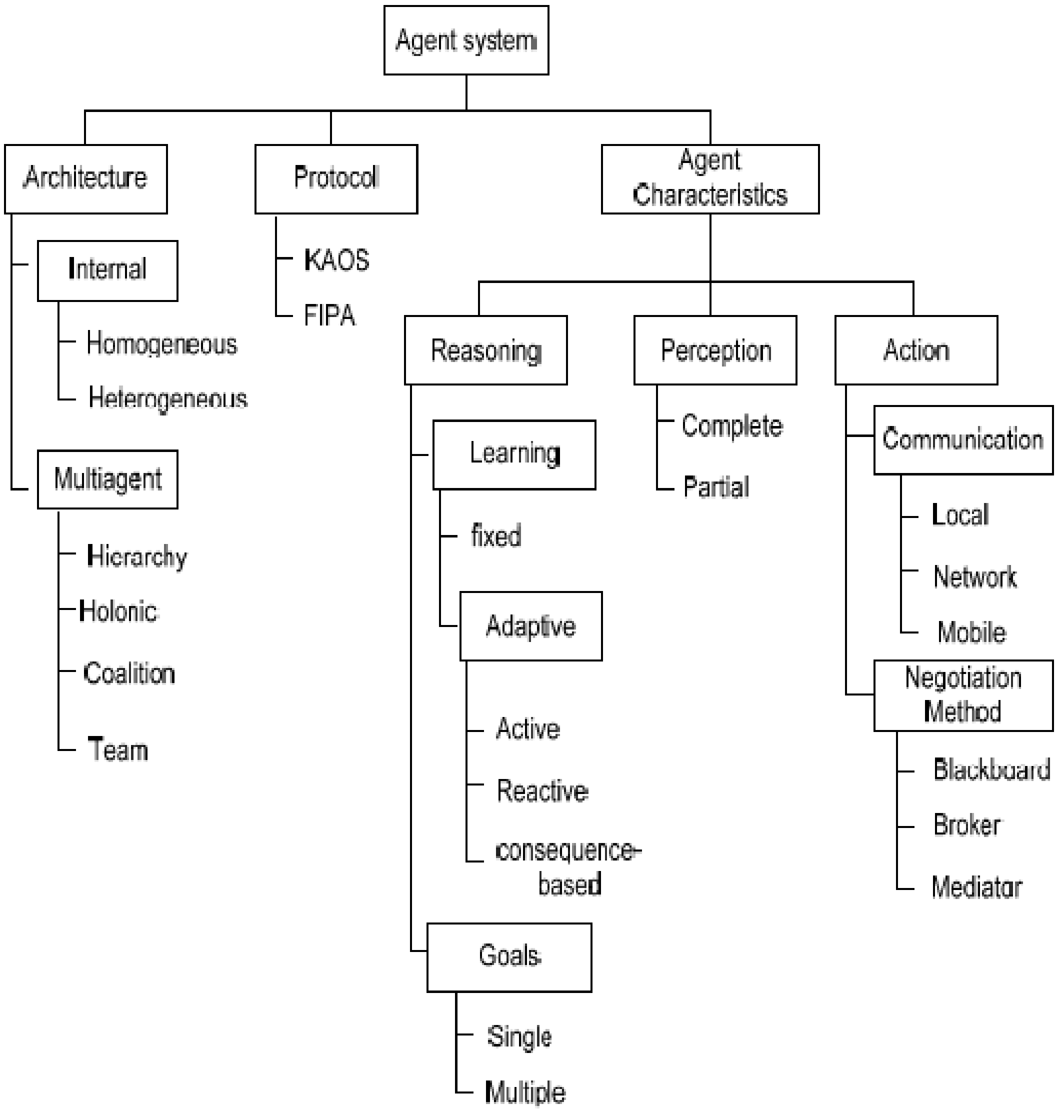


Figure 3.2 Classification of a multi agent system based on the use of different attributes [46].

3.3. MAS Application

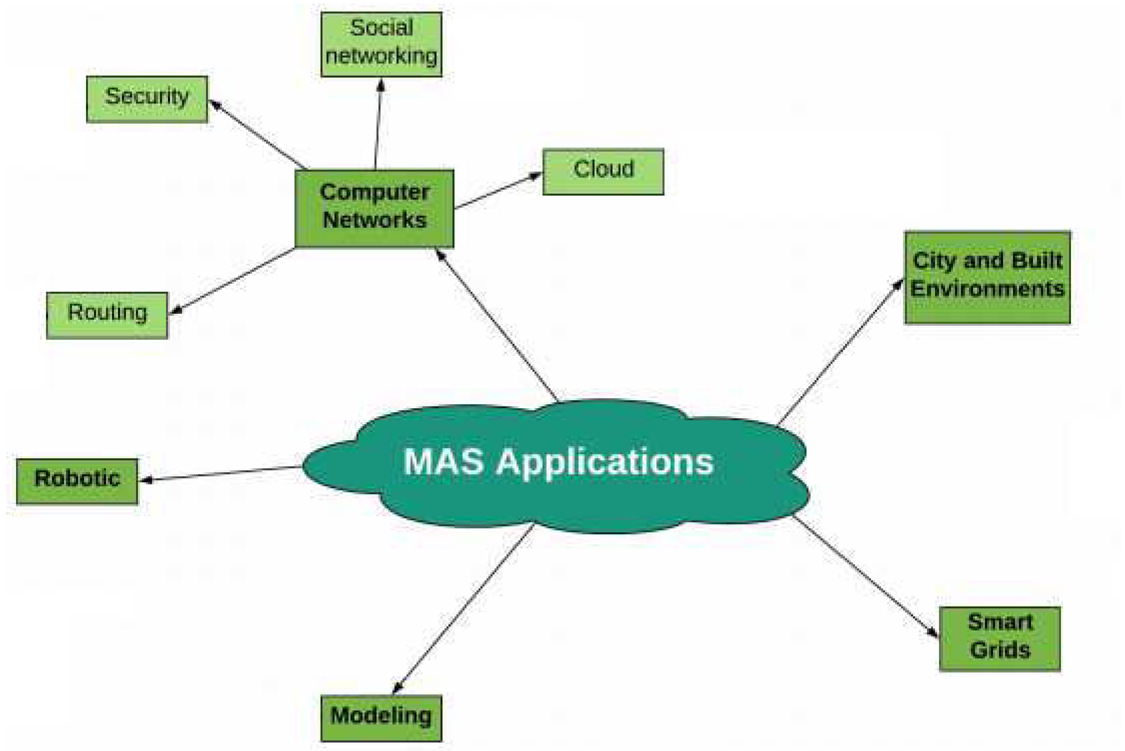


Figure 3.3 MAS application summary [47].

The MAS are impended in multiple fields that we can mention like [47]:

. **COMPUTER NETWORKS:** The complexity of computer networks increases dramatically because of the emergence of new technologies and proliferation devices connected to the Internet. Agents are widely used to overcome this complexity. Due to the wide range of MAS applications in networks

. **ROBOTICS AGENTS:** Cena et al. Argued that there are two main challenges in robotics, namely:
1- cooperation and coordination between the robots

2- plan their movement path.
The authors then proposed a method using material and software agents to overcome the challenges.

. **MODELING AGENTS FOR COMPLEX SYSTEMS:** Modeling complex dynamic systems is expensive and involves high processing overhead due to the demand for powerful modeling platforms and high complexity Flexibility, autonomy and scalability offered by agents makes agent-based modeling a low-cost, low-resource solution for modeling complex systems.

. **AGENTS IN THE CITY AND BUILDING ENVIRONMENTS:** The use of agents for the management of cities and buildings has received considerable attention from researchers. In a city, unorganized freight distribution increases the cost, pollution and congestion. Khayyat and Awasthi proposed an agent-based approach to this challenge using six agents, namely: RFIDG, retailer, supplier, carrier, network agents and the city.

. **AGENTS IN INTELLIGENT GRIDS:** Agents are used to address the multiple challenges of smart grids, including balancing generated and generated resources. The energy demanded, the negotiation between the energy consumer and the producer on the price of energy.

3.4. MAS Challenges

MAS challenges are typically application-specific [47]:

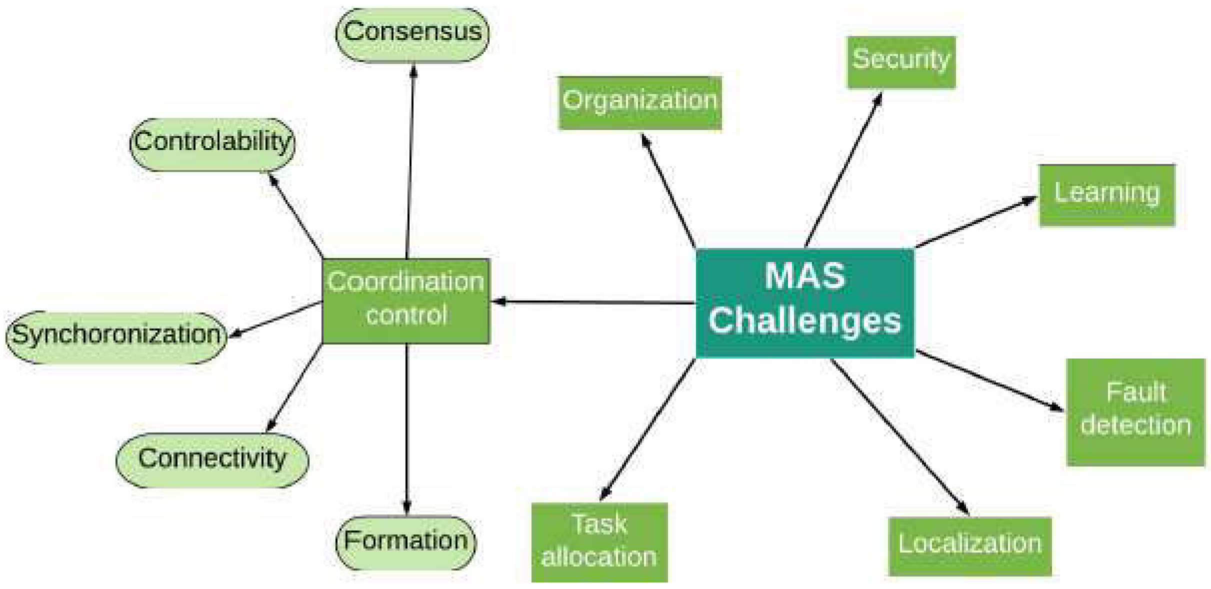


Figure 3.4 MAS challenges summary [47].

Even with the increasing applicability of MAS in multiple disciplines, important research indicates that the following challenges must be addressed:

. **coordination between agents:** The agents actions reflects on the environment and therefore the decision made by other agents. Coordination control refers to management agents to achieve collaboration their goals.

. **learning:** In MAS, each agent decides autonomously action to achieve its objective according to several measures. Agents can use machine learning algorithms to discover and predict changes to the environment, adapt to unexpected situations, and thus form multi-agent learning systems.

. **fault detection:** faulty agents can infect the agents in collaboration with him, that's why Detecting and isolating of defective agent is a fundamental task with. Current methods to detect and isolate defects (FDI) are mainly centralized, where a center to detect and then isolate the defective agents.

. **task allocation:** Task Allocation refers to assigning tasks to agents considering cost, time and communication costs and treatment overhead. The division of tasks can be centralized or decentralized.

. **Localization:** Remember that each agent has a limited view (only his neighbors) MAS topology. With this limited view, locate a particular agent, namely localization, can be difficult. An agent can be located based on: 1) having resources known as resource localization, (2) specific services, or (3) have a specific identity.

. **Agent organization:** Organization refers to how agent communications and the connections are defined.

. **security:** Security is very difficult in MAS because of decentralization, sociability and mobility.

Conclusion

in this chapter we entered to the Distributed artificial intelligence (DAI) then we focused on one of his subfields which is Multi Agent Systems and also focusing on the Agents as an autonomous Entity and his behavior, we have citate the MAS features, Applications and challenges.

In the next chapter we will show case our hypothesis on MAS reliability Prediction.

Chapter 4

Multi Agent System Reliability: Hypothesis & Experimentation

"*There is no law except the law that there is no law.*"

John Archibald Wheeler

Introduction

The need for reliable programs has become crucial with all the technological revolution we are experiencing. In this chapter, we want to manifest it in multi-agent systems, we propose our hypothesis. Finally, we finish with the experimentation.

1. Hypothesis

Due to the complex construction of the agent, we focus on how to predict agent reliability integrating agent behavior into the prediction process.

We created our own metrics based on the agent definition and the SMA definition, as well as on their different characteristics. The results of the forecast can be validated with a selected reliability growth model based on the agent failure data. To explain the approach, we introduce the following figure:

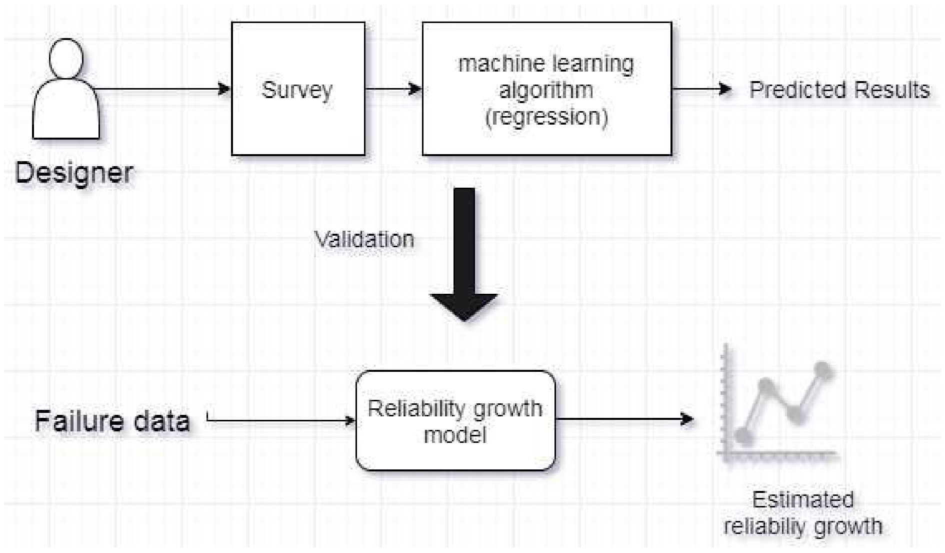


Figure 4.1. Proposed approach.

In our approach we are not depending on failure data only, we tried to add different measures to better judge reliability. We choose the *designer* to answer the survey, because the questions are more relevant to the coded behavior of the agent and his environment.

The *survey* contains, the metrics that we have proposed to measure the agent reliability as form of questions with predefined answers also include entering data on the agent code. Then we apply *supervised machine learning algorithm regression*, it predicts discrete outputs like our data, because we have multiple categories (variables) and we will work with multiple linear regression.

As a result, we got the predicted mean time between failures. We can validate the predicted result with the failure data that is processed with the reliability growth model giving an estimated reliability growth of the agent; the choice of the appropriate model is based on the obtained failure data from the agent to get the appropriate fit model.

1.1. MAS reliability metrics

The metrics are inspired from the agent behavior and MAS's environment, which can be used to measure the agent reliability; we tried to capture the metrics from the different definitions of the agent and MAS as bellow:

Definition 1: "An agent is an entity that perceives its environment and interacts with it "(Russell, 1997);

- ***The rate of sending/receiving messages:*** the agents are capable to interact with each other throw sending and receiving messages, which very important metric to evaluate the ability of communication of agents.
- ***Environment perception rate:*** the ability of an agent to perceive his environment means his ability to use the resources needed to achieve his goal.

Definition 2: "An agent is a computer system, located in an environment that acts autonomously to achieve the goals for which it was designed" (Wooldridge and Jennings, 1995).

- ***Agent statue:*** We mean by whether it works autonomously or collaborating with other agents.
- ***Goal accomplishment:*** the most important factor that we can with it measures the agent reliability.

Definition 3: "Intelligent agents are software entities that perform operations in place of a user or another program, with some kind of independence or autonomy, and to do that they use some kind of knowledge or representation of goals or desires of the user"(The IBM agent).

- ***Mobility:*** whether static or mobile agent that can percept multiple environments and react with other agents to serve his goal.
- ***MAS topology:*** can be static or dynamic, unlike static in dynamic topology, the agent moves, leaves, or joins. MAS, or establishes new communications.

Definition 4: "An agent is an autonomous entity, real or abstract, which is able to act on itself and on its environment, who, in a multi-agent universe, can communicate with other agents, and whose behavior is the consequence of his observations, knowledge and interactions with other agents"(Ferber, 1995).

- ***Proactive:*** The agent must exhibit proactive and opportunistic behavior, while being able to take the initiative at the right time.

With this different metrics we can predict the agent reliability in accomplishment of his goal.

2. Work environment

2.1. Google Colab

Our work environment is Google Colab or "the Colaboratory", which is a free cloud service hosted by Google to encourage research on machine learning and artificial intelligence for academics or experts [48].

2.1.1. Colab strength

- Python 2.7 and Python 3.6 support.
- Free GPU acceleration.
- Pre-installed libraries: All major Python libraries like Scikit-learn, Matplotlib and others are pre-installed and ready to be imported.
- Built on top of Jupyter Notebook.
- Allow Collaboration between developers to use and share Jupyter notebook among each other using google drive.
- Available documentation.
- Google Colab notebooks are stored on the drive.

2.2. Programming language: Python

Python is a high-level programming language widely used for versatile programming. Python is an excellent, object-oriented, interpreted and interactive programming language. It includes modules, classes, exceptions, high-level dynamic data types, and dynamic typing.

There are interfaces for many system calls and libraries, as well as for various windowing systems. Python can also be used as an extension language for applications written in other languages that require easy-to-use scripting or automation interfaces [49].

2.2.1. Python strength

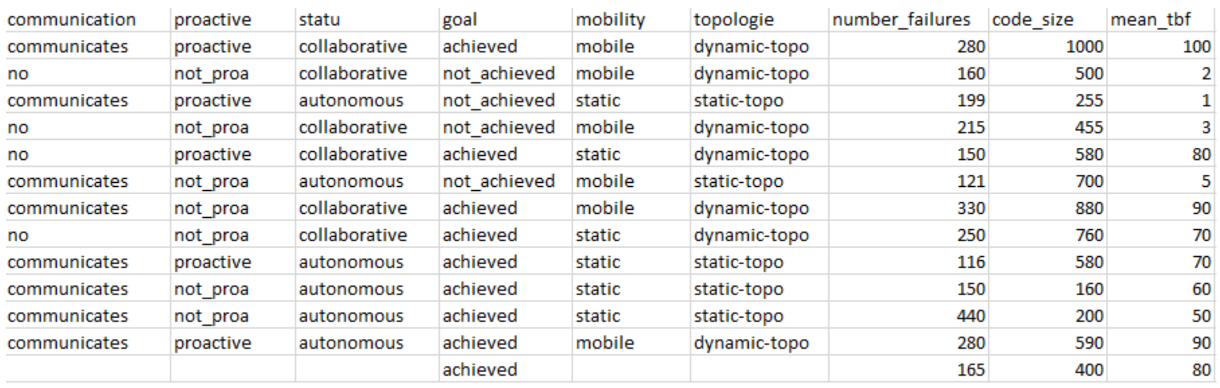
- It’s simple to learn.
- Open source programming language.
- Clear syntax.
- Capability of interacting with almost all the third-party languages and platforms.

3. Experimentation

In This section, we will explain the regression algorithm and our dataset. Concluding with the code of the experimentation and the obtain results.

3.1. Dataset

We have based our experimentation on theoretical data; to obtain the data for the experimentation we need a study case on the agent's behavior in MAS.



communication	proactive	statu	goal	mobility	topologie	number_failures	code_size	mean_tbf
communicates	proactive	collaborative	achieved	mobile	dynamic-topo	280	1000	100
no	not_proa	collaborative	not_achieved	mobile	dynamic-topo	160	500	2
communicates	proactive	autonomous	not_achieved	static	static-topo	199	255	1
no	not_proa	collaborative	not_achieved	mobile	dynamic-topo	215	455	3
no	proactive	collaborative	achieved	static	dynamic-topo	150	580	80
communicates	not_proa	autonomous	not_achieved	mobile	static-topo	121	700	5
communicates	not_proa	collaborative	achieved	mobile	dynamic-topo	330	880	90
no	not_proa	collaborative	achieved	static	dynamic-topo	250	760	70
communicates	proactive	autonomous	achieved	static	static-topo	116	580	70
communicates	not_proa	autonomous	achieved	static	static-topo	150	160	60
communicates	not_proa	autonomous	achieved	static	static-topo	440	200	50
communicates	proactive	autonomous	achieved	mobile	dynamic-topo	280	590	90
			achieved			165	400	80

Figure 4.2. Proposed data.

This table contains the data on the metrics previously explained in addition to number of failures detected in the code test and the code size, our target value to predict reliability is the mean time between failures which is an important indicator of the reliability.

Mean time between failures defined as [11]:

MTBF = MTTF + MTTR, an MTBF of 300 indicates that once the failure has occurred, the next failure should only occur after 300 hours; Where: MTTF is defined as the time interval between the successive failures, And MTTR measures the average time it takes to track the errors causing the failure & to fix them.

3.2. Regression algorithm

3.2.1. Multiple Linear Regression

We used the multiple linear regression algorithm because this algorithm consists of a target variable/result (or dependent variable) to predict from a given set of predictors (independent variables).

we can use it to find out which factor has the highest impact on the predicted output and how different variables relate to each other. That is known with the correlation.

Using these sets of variables, we generate a function that maps the inputs to the desired outputs [50].

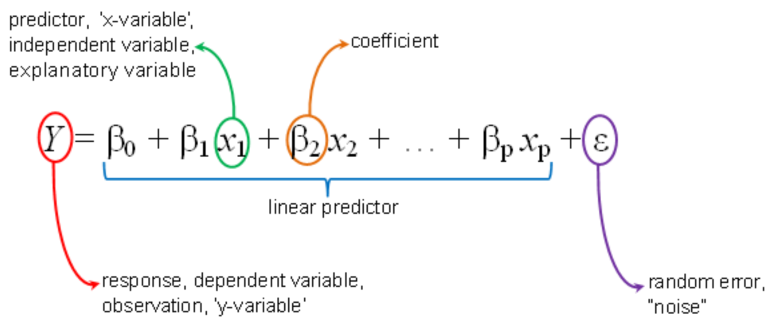

predictor, 'x-variable', independent variable, explanatory variable
coefficient
$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \varepsilon$
linear predictor
response, dependent variable, observation, 'y-variable'
random error, "noise"

Figure 4.3. Multiple Linear Regression [51].

To evaluate the performance of algorithm, three evaluation metrics are commonly used [51]:

1. Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

2. Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

3. Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

3.3. Code explanation

In this part we will go through all the steps of the code execution to the final result, with each code segment we will incorporate a comment to explain briefly how it works.

Step1: importing libraries

```
import pandas as pd
import numpy as np
from pandas import DataFrame
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
import seaborn as sns
from pandas.plotting import scatter_matrix
```

Comment: Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python and our used libraries are included in her stack like [58]:

- Pandas: Data structures and analysis.
- NumPy: Base n-dimensional array package.
- Matplotlib: Comprehensive 2D/3D plotting.

- IPython: Enhanced interactive console.

Step2: Uploading the dataset file.

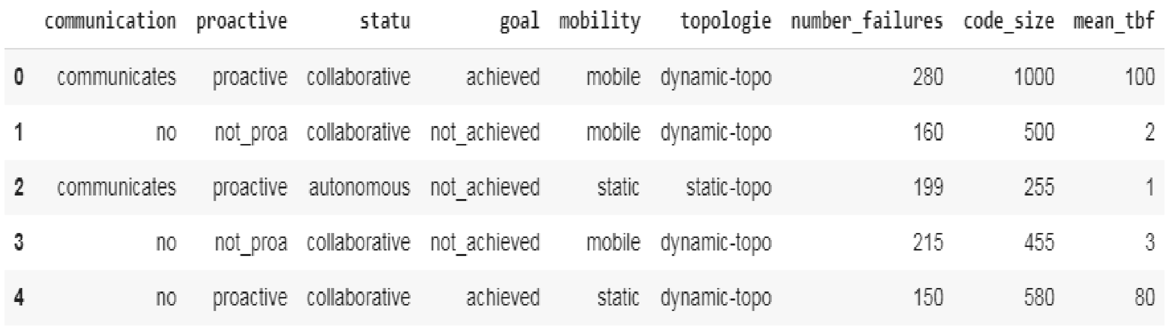
```
from google.colab import files
uploaded = files.upload()
```

Comment: files.upload (), allows you to upload from the device, which is very useful in case of constant changes in the dataset.

Step 3: reading the data.

```
dataset = pd.read_csv('dd.csv', delimiter=";")
dataset.head()
dataset.shape -> (13, 9)
```

Comment: After reading the data with panda csv reader, we poste our data with the head function it only displays the first columns, to know the data dimensions we use the shape function. Our data comprises 13 lines and 9 columns.

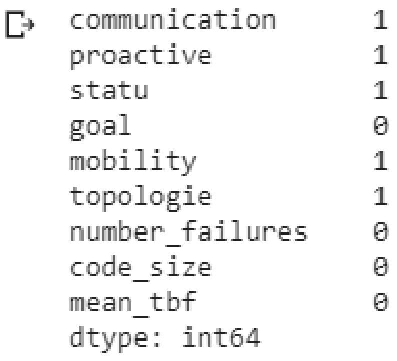


	communication	proactive	statu	goal	mobility	topologie	number_failures	code_size	mean_tbf
0	communicates	proactive	collaborative	achieved	mobile	dynamic-topo	280	1000	100
1	no	not_proa	collaborative	not_achieved	mobile	dynamic-topo	160	500	2
2	communicates	proactive	autonomous	not_achieved	static	static-topo	199	255	1
3	no	not_proa	collaborative	not_achieved	mobile	dynamic-topo	215	455	3
4	no	proactive	collaborative	achieved	static	dynamic-topo	150	580	80

Figure 4.4. The dataset.

Step 4: Identifying the Nan values and removing theme.

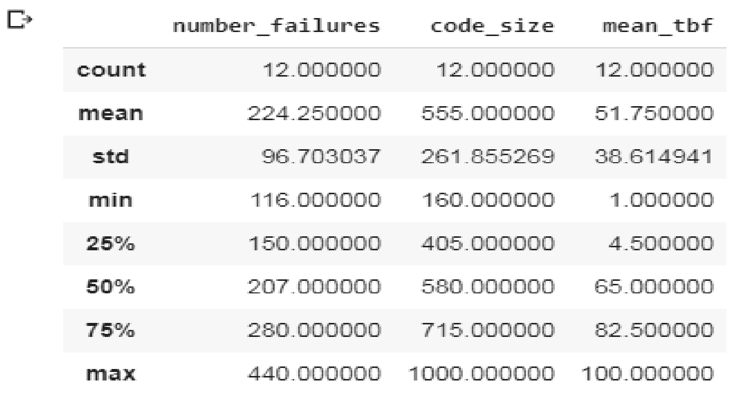
```
dataset.isna().sum()
```



```
communication      1
proactive          1
statu              1
goal               0
mobility           1
topologie          1
number_failures    0
code_size          0
mean_tbf           0
dtype: int64
```

Figure 4.5. Displaying the null values.

```
dataset.dropna(inplace=True)
dataset.describe()
```

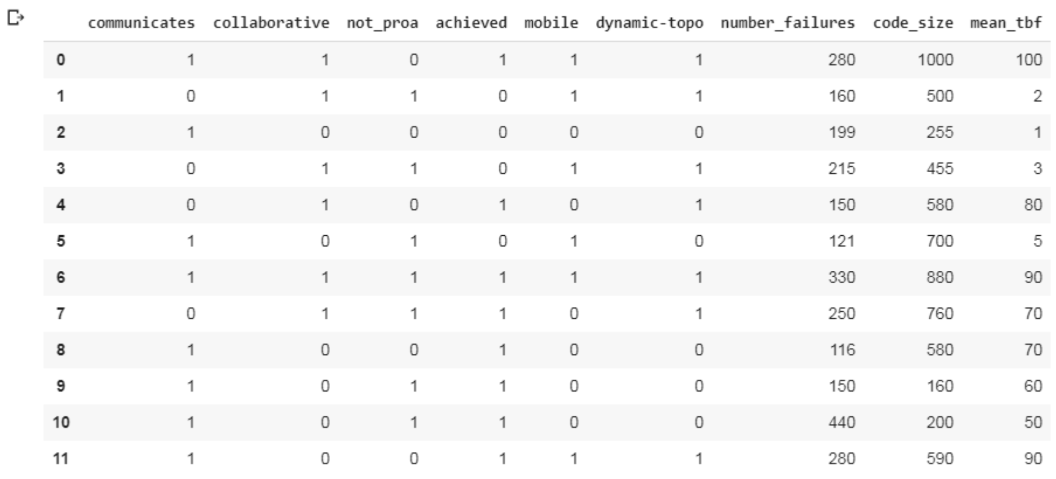


	number_failures	code_size	mean_tbf
count	12.000000	12.000000	12.000000
mean	224.250000	555.000000	51.750000
std	96.703037	261.855269	38.614941
min	116.000000	160.000000	1.000000
25%	150.000000	405.000000	4.500000
50%	207.000000	580.000000	65.000000
75%	280.000000	715.000000	82.500000
max	440.000000	1000.000000	100.000000

Figure 4.6. Data information.

Step5: Data normalization.

```
d1=pd.get_dummies(dataset.communication)
dataset2=pd.concat([dataset,d1],axis='columns')
dataset2=dataset2.drop(['communication','no'],axis='columns')

d2=pd.get_dummies(dataset.statu)
dataset3=pd.concat([dataset2,d2],axis='columns')
dataset3=dataset3.drop(['statu','autonomous'],axis='columns')

d3=pd.get_dummies(dataset.proactive)
dataset4=pd.concat([dataset3,d3],axis='columns')
dataset4=dataset4.drop(['proactive'],axis='columns')

d4=pd.get_dummies(dataset.goal)
dataset5=pd.concat([dataset4,d4],axis='columns')
dataset5=dataset5.drop(['goal','not_achieved'],axis='columns')

d5=pd.get_dummies(dataset.mobility)
dataset6=pd.concat([dataset5,d5],axis='columns')
dataset6=dataset6.drop(['mobility','static'],axis='columns')

d6=pd.get_dummies(dataset.topologie)
dataset7=pd.concat([dataset6,d6],axis='columns')
dataset7=dataset7.drop(['topologie','static-topo'],axis='columns')

dataset7=dataset7.drop(['number_failures','code_size','mean_tbf'],axis='columns')
dataset7=pd.concat([dataset7,dataset.number_failures],axis='columns')
dataset7=pd.concat([dataset7,dataset.code_size],axis='columns')
dataset7=pd.concat([dataset7,dataset.mean_tbf],axis='columns')
dataset7
```



	communicates	collaborative	not_proa	achieved	mobile	dynamic-topo	number_failures	code_size	mean_tbf
0	1	1	0	1	1	1	280	1000	100
1	0	1	1	0	1	1	160	500	2
2	1	0	0	0	0	0	199	255	1
3	0	1	1	0	1	1	215	455	3
4	0	1	0	1	0	1	150	580	80
5	1	0	1	0	1	0	121	700	5
6	1	1	1	1	1	1	330	880	90
7	0	1	1	1	0	1	250	760	70
8	1	0	0	1	0	0	116	580	70
9	1	0	1	1	0	0	150	160	60
10	1	0	1	1	0	0	440	200	50
11	1	0	0	1	1	1	280	590	90

Figure 4.7. Normalized data.

Comment: We normalize data from string to numerical data, using the dummies function from panda, then we drop the main column and one of the dummies columns, after that we concatenate the selected column to the normalized data set.

Step6: Identifying the correlation between columns.

```
corr=dataset7.corr()
axes =pd.plotting.scatter_matrix(dataset7,alpha=0.2,figsize=(10,10) ,s=80)
corr = dataset7.corr().as_matrix()
plt.title('Correlation Between Features',x=-5,y=13,fontsize=25)
# to change fontsize
plt.xticks(fontsize =10,rotation =0)
plt.yticks(fontsize =10)
for ax in axes.ravel():
    ax.set_xlabel(ax.get_xlabel(),fontsize = 12, rotation = 60)
    ax.set_ylabel(ax.get_ylabel(),fontsize = 12, rotation = 60)
# put the correlation between each pair of variables on each graph
for i, j in zip(*np.triu_indices_from(axes, k=1)):
    axes[i,  j].annotate("%.3f"  %corr[i,  j],  (0.8,0.8),  xycoords="axes
fraction", ha="center", va="center")
```

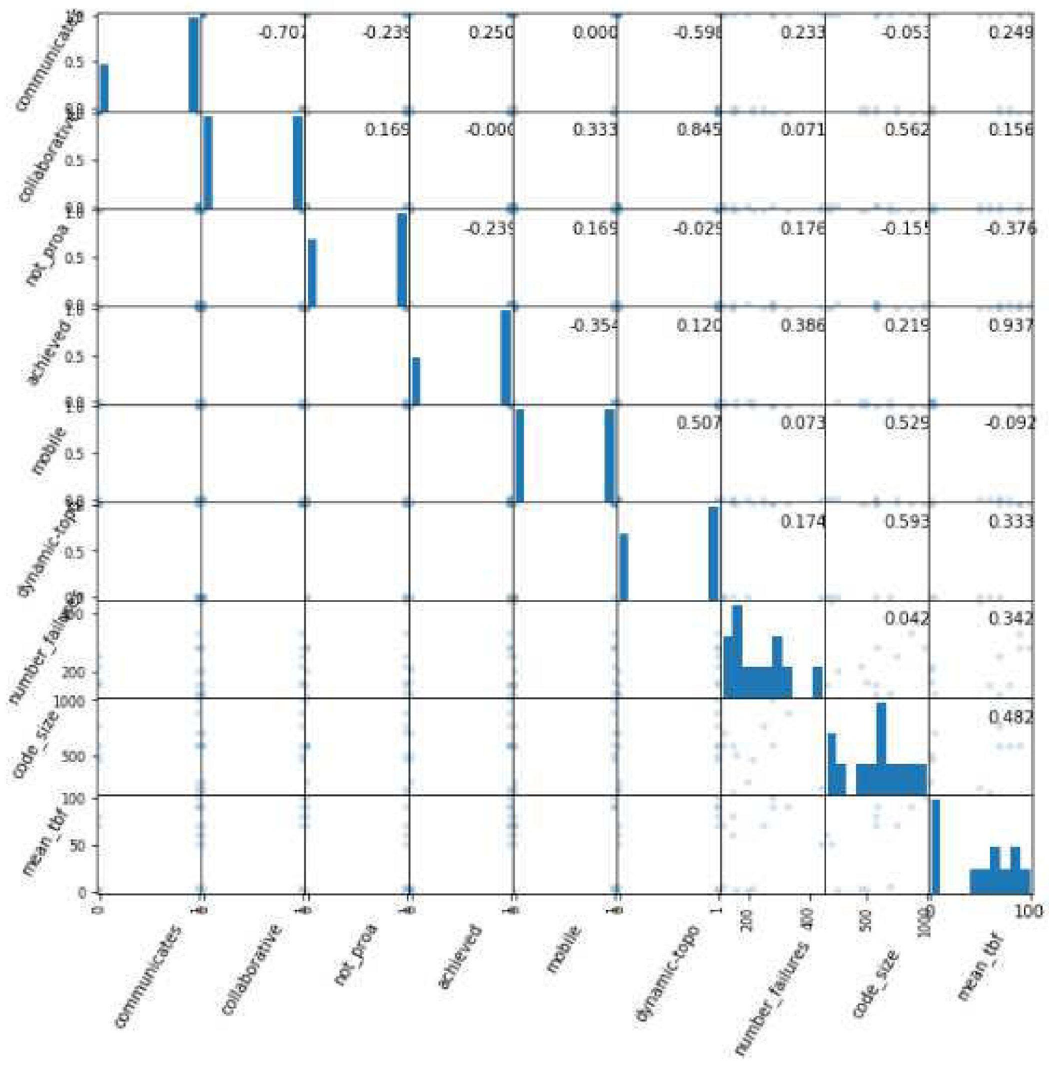


Figure 4.8. Scatter matrix correlation.

```
Plt.figure(figsize=(14,6))
sns.heatmap(corr,annot=True)
```

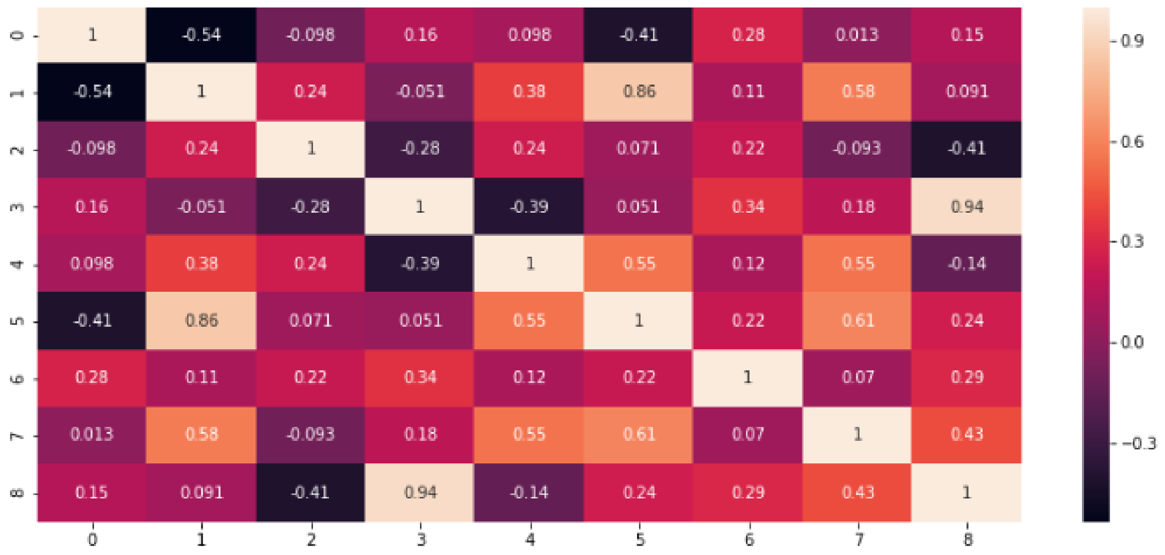
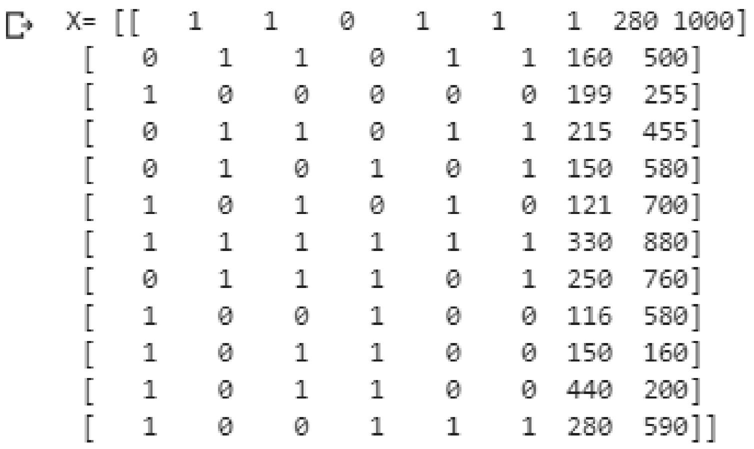


Figure 4.9. Heatmap correlation.

Step 7: Splitting the data to train set and test set.

```
X=dataset7[['communicates','collaborative','not_proa','achieved','mobile','
dynamic-topo','number_failures','code_size']].values
y = dataset7['mean_tbf'].values
print('X=',X)
```



```
X= [[   1    1    0    1    1    1  280 1000]
 [   0    1    1    0    1    1  160  500]
 [   1    0    0    0    0    0  199  255]
 [   0    1    1    0    1    1  215  455]
 [   0    1    0    1    0    1  150  580]
 [   1    0    1    0    1    0  121  700]
 [   1    1    1    1    1    1  330  880]
 [   0    1    1    1    0    1  250  760]
 [   1    0    0    1    0    0  116  580]
 [   1    0    1    1    0    0  150  160]
 [   1    0    1    1    0    0  440  200]
 [   1    0    0    1    1    1  280  590]]
```

Figure 4.10. The categories values.

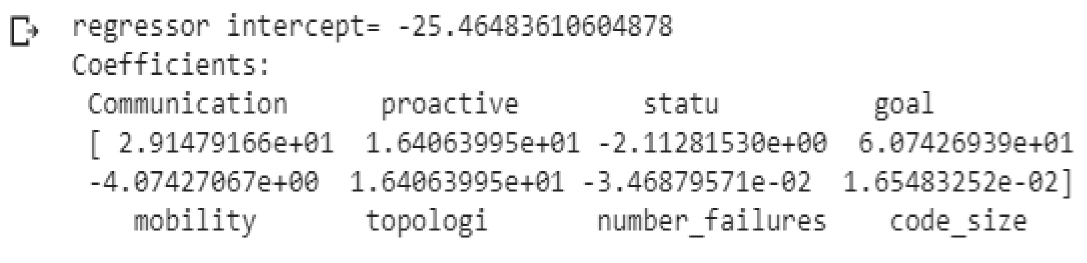
```
print('y=',y)
```

Y= [100 2 1 3 80 5 90 70 70 60 50 90]

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_sta
te=0)
```

Step 8: running the regression.

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
print('regressor intercept=',regressor.intercept_)
print('Coefficients: \n','Communication proactive statu goal\n',regressor.
coef_)
print(' mobility topologi number_failures code_size')
```



```
regressor intercept= -25.46483610604878
Coefficients:
 Communication      proactive        statu          goal
 [ 2.91479166e+01  1.64063995e+01 -2.11281530e+00  6.07426939e+01
 -4.07427067e+00  1.64063995e+01 -3.46879571e-02  1.65483252e-02]
    mobility        topologi       number_failures    code_size
```

Figure 4.11. Intercept and coefficient.

Step 9: Prediction execution.

```
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df1 = df.head(25)

df1.plot(kind='bar',figsize=(8,6))
plt.grid(which='major', linestyle='-', linewidth='0.3', color='coral')
plt.grid(which='minor', linestyle=':', linewidth='0.3', color='black')
plt.show()
```

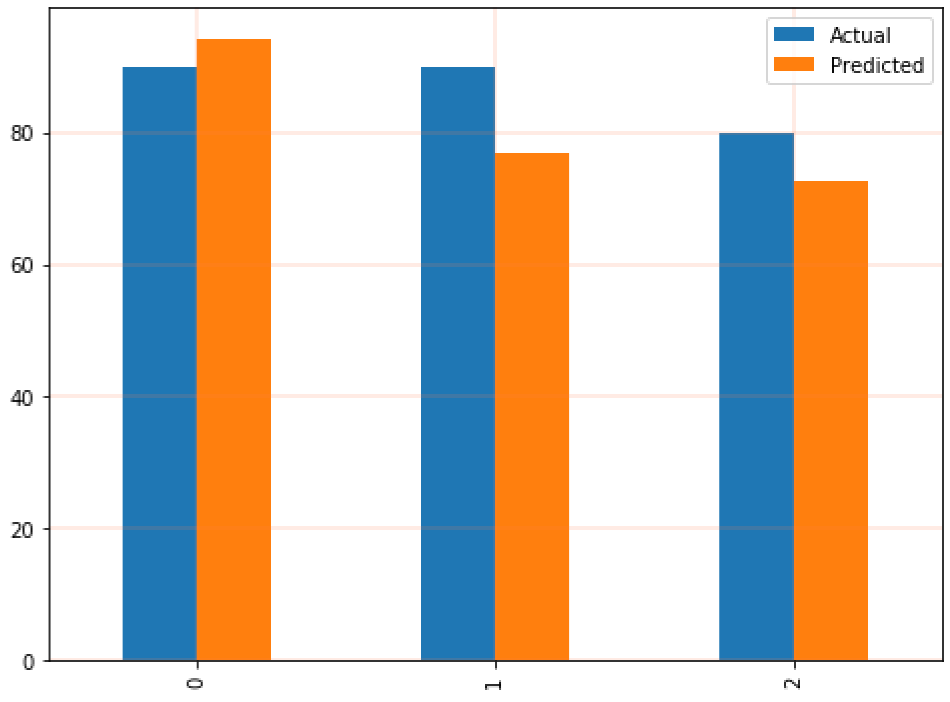
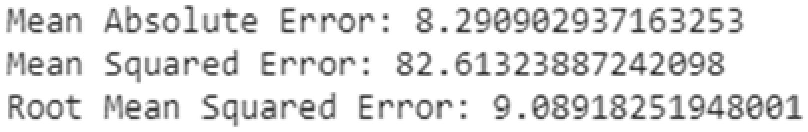


Figure 4.12. Comparison of the actual and predicted data.

Step 10: Regression evaluation.

```
Print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
Print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```



```
Mean Absolute Error: 8.290902937163253
Mean Squared Error: 82.61323887242098
Root Mean Squared Error: 9.08918251948001
```

Figure 4.13. Algorithm evaluation.

Comment: Our root mean squared error is 0.8% bigger than the mean absolute error. That means that our algorithm was not very accurate but can still make reasonably good predictions.

Conclusion

Investments in multi-agent systems have reached billions of dollars, demonstrating the need for reliable agents. That shows the importance of the early prediction of the MAS reliability, leading to more accurate results, less cost and less development time.

In this chapter, we accomplished the agent prediction using machine learning algorithm regression, integrating metrics inspired from the agent behavior and MAS environment.

Next, we will conclude our manuscript.

General Conclusion & Perspectives

"Your imagination is your preview of life's coming attractions."

Albert Einstein

Conclusion

Reliability measurement has been an active area of research for decades; software reliability modeling has attracted a lot of research attention in the estimation (measurement of the current state) as well as the prediction (assessment of the future state) of the reliability of a software system.

The time domain approach, proved its effectiveness by performing an adjustment of the curve of observed failure data as a function of time with a model formula and parameters. Models can then provide an estimate of existing resources reliability or predictability of future reliability by extrapolation techniques.

Software reliability tools that provides both estimation and prediction had more effective results, that reflects on the development process of the software's in term of time consumption and cost reduction and correctness level.

In our hypothesis we tried to focus on the reliability prediction of multi agent systems, introducing metrics based on the behavior of the agents and MAS characteristics like agent mobility, ability to communicate, MAS topology, number of failures detected and number of code lines …etc. In the experimentation we developed our idea using machine learning regression algorithm to determine the correlation between our proposed metrics and to predict the reliability of agents.

Perspective

As perspective, to lack of time we have built our experimentation on theoretical data, that is only obtained through a study case of long duration to analyze a MAS community and agent behavior as units to obtain and enhance our dataset; and even further we could validate the results obtained in the prediction by adopting an appropriate SRGM.

Bibliographical References

[1] Humphrey, W. S., “The Future of Software Engineering”, Watts New Column, News at SEI, vol. 4, no. 1, March, 2001.

[2] Israelski, E. W., Muto, W. H., “Human Factors Risk Management as a Way to Improve Medical Device Safety: A Case Study of the Therac 25 Radiation Therapy System”. The Joint Commission Journal on Quality and Safety, 30(12), 689–695, 2004.

[3] Marshall, E., “Fatal error: how Patriot overlooked a Scud.”, Science, vol. 255, no. 5050, p. 1347, Academic OneFile, 1992.

[4] Pauchant, T., Mitroff, I., and Lagadec, P., “Toward a systemic crisis management strategy: learning from the best examples in the US, Canada and France.”, Industrial Crisis Quarterly, Elsevier, 1991.

[5] Le Lann, G. (n.d.), “An analysis of the Ariane 5 flight 501 failure-a system engineering perspective.”, Proceedings International Conference and Workshop on Engineering of Computer-Based Systems, 1997.

[6] Ho-Won, J., Seung-Gweon, K., and Chang-Shin, C., “Measuring Software Product Quality: A Survey of ISO/IEC 9126.”, IEEE Software, 2004.

[7] Musa, J. D., “Software Reliability Engineering: More Reliable Software Faster and Cheaper.”, (2nd Edition), Author House, 2004.

[8] Zhang, X., Pham, H., “An analysis of factors affecting software reliability. Journal of Systems and Software, 50(1), 43–56, 2000.

[9] Michael, R. L., “Software Reliability Engineering: A Roadmap.”, Computer Science and Engineering Department the Chinese University of Hong Kong, Future of Software Engineering (FOSE'07)0-7695-2829-5/07 $20.00 © 2007 IEEE.

[10] Reliability Analysis Center, “Introduction to Software Reliability: A state of the Art Review.”, RAC, 1996, consulted on: http://rome.iitri.com/RAC/.

[11] Durga, P., Pallavi, “Software Reliability: Models.”, International Journal of Computer Applications (0975 – 8887), Volume 152 – No.9, October, 2016.

[12] Aman, J., Yukti, M., “Metrics and Models for Software Reliability: A Systematic Review.”, ITM University Gurgaon-12200 1, International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), IEEE, 2014.

[13] Rosenberg, L., Hammer, T., and Shaw, J., Unisys/NASA GSFC, “Software Metrics and Reliability.”, IOSR Journal of Computer Engineering (IOSR-JCE), 1998.

[14] Lyu, M. R., Nikora, A., “CASRE - A Computer-Aided Software Reliability Estimation Tool.”, Conference: Computer-Aided Software Engineering, IEEE, August, 1992.

[15] For, E.H., Singpurwalla, N.D., “An Empirical Stopping Rule for Debugging and Testing Computer Software.”, Journal American Statistics Association, vol. 72, pp. 750-757, December, 1977.

[16] Joe, H., Reid, N., “Estimating the Number of Faults in a System.”, Journal of the American Statistical Association, 80(389):222-226, March, 1985.

[17] Langberg, N., Singpunvalla, N.D., “A Unification of Some Software Reliability Models via the Bayesian Approach.”, Technical Report TM-66571, George Washington University, 1981.

[18] Duane, J. T., “Learning Curve Approach to Reliability Monitoring.”, IEEE Transactions on Aerospace, vol.AS-2, pp. 563-566, 1964.

[19] Goel, A., Okumoto, K., “Time-dependent error-detection rate model for software reliability and other performance measures.”, IEEE Trans. on Reliability, R-28(3): 206- 211 (1979).

[20] Sona, A., Guru, S.M., and Agam, P. T., “Jelinski – Moranda Model for Software Reliability Prediction and its G.A. based Optimised Simulation Trajectory.”, D. E. I. Dayalbagh, Agra, pp. 399-404, 2002.

[21] Jelinski, Moranda, P.B., “Software Reliability Research.”, In Statistical Computer Performance Evaluntion, ed. W. Freiberber, pp. 465484, Academic, New York, 1972.

[22] Shooman, M., “Operational Testing and Software Reliability During Program Development.”, In Proceedings 1973 IEEE Symposium on Computer Software Reliability, pp. 51-57, New York, April, 1973.

[23] Keiller, P.A., Littlewood, B., Miller, D. R., and Sofer, A., “Comparison of Software Computing.”, pp. 128-134, 1983.

[24] Littlewood, B., “Stochastic Reliability Growth. A Model for Fault-Removal in Computer Programs and Hardware Designs.”, IEEE Transactions on Reliability, vol. R-30, pp. 313-320, October, 1981.

[25] Littlewood, B., Verrall, J. L., “A Bayesian Reliability Growth Model for Computer Software.”, Journal Royal Statistics Society C, vol. 22, pp. 332-346, 1973.

[26] Musa, J.D., Okumoto, K., “A Logarithmic Poisson Execution Time Model for Software Reliability Measurement.”, In Proceedings Seventh International Conference on Software Engineering, pp. 230-238, Orlando, Florida, 1984.

[27] Schneidewind, N.F., “Analysis of Error Processes in Computer Software.”, In Proceedings International Conference on Reliable Software, pp. 331-346, Los Angeles, 1975.

[28] Yamada, S., Ohba, M., and Osak, S., “S-Shaped Reliability Growth Modeling for Software Error Detection.”, IEEE Transactions on Reliability, vol. R-32, pp. 475- 478, December, 1983.

[29] Manohar, S., “Software Reliability Testing Tools: An Overview and Comparison.”, International Journal of Advanced Trends in Computer Science and Engineering, ISSN: 2319-7242, Volume 5, Issue Page No. 18886-18891, November, 2016.

[30] University of Phoenix Technology and Mathematics, “Advantages and Limitations of CASE Tools.”, In Petruska Site, 15/02/2019.

[31] Tausworthe, R. C., Lyu, M. R., “A Generalized Software Reliability Process Simulation Technique and Tool.”, Published 1071-9458/94, IEEE, December, 1994.

[32] “Benefits of Frestimate Software.”, http://www.softrel.com/About_Frestimate.html.

[33] Dong, T., Myron, H., Jeffrey, M., and Jady, H., “MEADEP — A Dependability Evaluation Tool for Engineers.”, In IEEE Transactions on Reliability, 1998.

[34] User’s Manual, “A software Tool for System Dependability Measurement, Modeling and evaluation.”, consulted on: http://www.reliability-safety-software.com/downloads/download-documents/.

[35] Kishor, S., Trivedi, “SREPT: A Tool for Software Reliability Estimation and Prediction.”, Proceedings of the International Conference on Dependable Systems and Networks, (DSN’02) 0-7695-1597-5/02, IEEE, 2002.

[36] Srinivasan, R., Swapna, S. G., and Kishor, S. T., “SREPT: software reliability estimation and prediction tool.”, Elsevier Science B.V, 2000.

[37] Littlewood, B., Verrall, J. L., “A Bayesian reliability growth model for computer software.”, Royal Statistical Society, 22(3):332-346, January, 1973.

[38] Moranda, P., “Predictions of software reliability during debugging.”, Proc. Reliability and Maintainability Symp., Washington, D.C., p. 327, 28-30 January 1975.

[39] Musa, J., “A theory of software reliability and its applications.”, Software Engineering, SE-1(3):312-327, IEEE Trans. 1975.

[41] W. D. Brooks and R. W. Motley (1975), Analysis of Discrete Software Reliability modems. Rome Air Development Center Technical Report, RADC-TR-80-84, April 1980. [31] N. F. Schneidewind, Analysis of error processes in computer software, Sigslan Notices 10(6):337 346 (1975).

[42] Xie, M., Zhao, M., “The Schneidewind software reliability model revisited.”, Proceedings Third International Symposium on Software Reliability Engineering, Research Triangle Park, NC, USA, 7-10 October, 1992.

[43] William, H. F., Oliver, D. S., “A Tool for Statistical Modeling and Estimation of Reliability Functions for Software: SMERFS.”, The Journal of Systems and Software 8, 8(1):47-55, January, 1988.

[44] William, H. F., Oliver, D. S., “Statistical Modeling and Estimation of Reliability Functions for Software: SMERFS.”, User's guide, September, 1993.

[45] Maevsky, D., Maevskaya, E., Shapa, L., and Stetsyuk, D., “Program Tools for Estimation of the Green Software Reliability.”, ITM Web of Conferences 9, 03008, 2017.

[46] Balaji, P. G., Srinivasan, D., “An Introduction to Multi-Agent Systems.”, Studies in Computational Intelligence, 1–27, 2010.

[47] Dorri, A., Kanhere, S., and Jurdak, R., “Multi-Agent Systems: A survey.”, IEEE Access, DOI: 10.1109/ACCESS.2018.2831228., April, 2018.

[48] Lall, V., “Google Colab—The Beginner’s Guide.”, consulted on : https://medium.com/lean-in-women-in-tech-india/google-colab-the-beginners-guide-5ad3b417dfa, April 1, 2018.

[49] Content Strategist - Ivy Pro School, "Why Python is the preferred language for Machine Learning?", consulted on: https://ivyproschool.com/blog/2017/08/21/why-python-is-the-preferred-language-for-machine-learning/, August 21, 2017.

[50] Sunil, R., "Commonly used Machine Learning Algorithms (with Python and R Codes).",consulted on : https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/, September 9, 2017.

[51] Robinson, S., "A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library.", consulted on: https://stackabuse.com/linear-regression-in-python-with-scikit-learn/, February 06, 2018.

[52] Dmitry Maevsky*, Elena Maevskaya, Ludmila Shapa and Dmitry Stetsyuk," Program Tools for Estimation of the Green Software Reliability", ITM Web of Conferences **9**, 03008 (2017) AMCSE 2016.