A Dissertation

On

Reliability Analysis Using Soft Computing Techniques

By

SANGEETA

Roll No. 2k14/PhD/CO/02

Under the Joint Supervision of

Dr. Kapil Sharma
Professor & Head,
Department of Information Technology
Delhi Technological University

Dr. Manju Bala
Assistant Professor,
Department of Computer Science
IP College of Women, Delhi University

Submitted in fulfillment of the requirements of the degree of

Doctor of Philosophy to the

Department of Computer Science and Engineering

Delhi Technological University

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi 110042, 2019

DECLARATION

I, Sangeeta, Ph.D. student (Roll No. 2k14/PhD/CO/02), hereby declare that the thesis entitled "Reliability Analysis using Soft Computing Techniques" which is being submitted for the award of the degree of Doctor of Philosophy in Computer Science & Engineering, is a record of bonafide research work carried out by me in the Department of Computer Science & Engineering, Delhi Technological University. I further declare that this work is based on original research and has not been submitted to any university or institution for any degree or diploma.

Date: _____

Place: New Delhi

Sangeeta
2k14/PhD/CO/02
Department of Computer Science & Engineering
Delhi Technological University (DTU)
New Delhi -110042

CERTIFICATE

DELHI TECHNOLOGICAL UNIVERSITY

(Govt. of National Capital Territory of Delhi)

BAWANA ROAD, DELHI - 110042

Date: _____

This is to certify that the work embodied in the thesis titled "RELIABILITY ANALYSIS USING SOFT COMPUTING TECHNIQUES" has been completed by Sangeeta under our supervision and guidance towards fulfillment of the requirements for the degree of Doctor of Philosophy of Delhi Technological University, Delhi. This work is based on original research and has not been submitted in full or in part for any other diploma or degree of any university.

Dr. Kapil Sharma
Professor & Head,
Department of Information Technology
Delhi Technological University

Dr. Manju Bala
Assistant Professor,
Department of Computer Science
IP College of Women, Delhi University

Acknowledgement

It gives me an immense pleasure while acknowledging the contribution of persons those helped me to achieve my research goal. The journey was started with the motivational support of my supervisors, Professor Kapil Sharma and Dr. Manju Bala.

First of all, I would like to thank Dr. Kapil Sharma, Head of Department, Information Technology, DTU, Delhi, for providing me the healthy research environment and academic facilities at DTU campus. Without his support and motivation it was never possible for me to pursue research goals. He has always been an inspirational source for me due to his motivational and supportive nature. He always guided me as a parent in every phase of my research work. I want to thank him again for sparing his valuable time from the very busy schedule while he was working as an OSD at DTU. I could never forget those early mornings and weekends during which i used to visit at his home for discussion of research paper and thesis writing.

I am very thankful for encouragement and support that he gave to me throughout this work.

I also thanks to Dr. Manju Bala, Assistant Professor, IP College of Women, Delhi University, for her moral support and providing me valuable guidance. Her academic and professional intelligence, strong commitment and dedication always inspired me to complete my research goals. She always provided me a healthy and helpful atmosphere for learning. She provided me the right directions to pursue my research with confidence in all meetings with her. This research journey has been an enriching experience of working under her guidance. She used to share her experiences about various challenges that she had faced during her research work and how she came out with the solutions in a successful manner, which motivated me. I am also thankful for the delicious breakfast that she used to offer me while visiting at her home.

I am also thankful to Professor Rajni Jindal, Head of CSE department, DTU for providing me an opportunity to pursue my research work at DTU. She had been a source of constant help and inspiration throughout the tenure of my research.

Further, I thanks to Professor Yogesh Singh, Vice Chancellor, Delhi Technological University for creating research environment and availability of various research facilities at DTU. He has been role model for me right from the beginning of my research journey at DTU. I am also thankful to all DRC Committee members for their time to time valuable suggestions and modifications.

A very special thanks to my dearest husband Mr. Sitender for his constant help and support while editing, proof reading my research work and suggesting valuable changes time to time. He has always been with me in different roles as a friend, as a teacher and as a parent and motivated me whenever I stuck during the research work. Without his cooperation it was never possible for me to be in a position where I am today.

I also pay my sincere regards to my parents and parents in- laws for their unconditional love, financial as well as moral support that helped me to concentrate on my research work without worrying about family responsibilities. At last i am also thankful to my friends for making me happy whenever I feel sad. I really enjoyed a lot with them during this research journey.

Sangeeta

2k14/PhD/CO/02
Department of Computer Science & Engineering
Delhi Technological University (DTU)
New Delhi -110042

List of Publications

- Sangeeta, Kapil Sharma and Manju Bala, "A Quantitative Testing Effort Estimate for Reliability Assessment of Multi Release OSS Systems" Journal of Computational and Theoretical Nano science, March 2019. (Accepted, Scopus Indexed)
- Sangeeta, Kapil Sharma and Manju Bala, "A New Optimization Algorithm for Software Reliability Model Parameter Estimation" Journal of Advanced Research in Dynamical and Control Systems, Dec 2018. (Published, Scopus Indexed)
- Sangeeta, Kapil Sharma and Manju Bala, "Reliability Analysis and Modeling of Green Computing Based Software Systems" Recent Patents in Computer Science, June 2019. (ISSN: 2213-2759) (Accepted, Scopus Indexed)
- Sangeeta, Kapil Sharma and Manju Bala, "An Ecological Space Based Hybrid Swarm-Evolutionary Algorithm for Software Reliability Model Parameter Estimation" International Journal of System Assurance Engineering and Management, Jan 2019. (ISSN :975-6809) (Under Review, Scopus Indexed)
- Sangeeta, Kapil Sharma and Manju Bala, "Exhausting Meta heuristic Nature inspired approaches for the parameter estimation analysis of software reliability growth models" International Conference on Industrial Applications on Control, Automation & Robotics 2018 (IACAR 2017, Jakarta Indonesia) available at Advanced Science and Technology letter/144/2017-18. (Published)
- Sangeeta, Kapil Sharma and Manju Bala Published "Quality issues with big data analytics" in IEEE sponsored conference entitled Computing for Sustainable Global development (India-com- March 2016). (Published)
- Sangeeta, Kapil Sharma and Manju Bala Published "A statistical View of software Reliability and modeling" in IEEE sponsored conference entitled Computing for Sustainable Global development (India-com-March 2015). (Published)

Abstract

With the growing advances in the digital world, software development demand from industries is growing at an exponential rate. Due to enormous demand and lack of time and budget, software companies are not able to develop fault-free software. Latest tools and techniques have been applied for the development of defect-free software, but still, it is impossible for the software developers to develop defect-free software practically. Software must go through exhaustive testing and debugging, which requires time and money to enhance the reliability. The occurrence of fault is inevitable in the current demand of software. There should have some means to avoid software failures so that devastating losses whether related to life or any other field could be evaded. According to IEEE standard 729 [1] , reliability is the most significant quality aspect of the software. If we could measure the reliability of software under development, better we can predict whether the software would be operational in the future or not. Reliability estimation process must be precise to provide information to the manager like what should be the release time of the software and amount of man-hour consumption etc. while developing any software. Software reliability models are only the ways in order to simulate software reliability estimation curve to predict the reliability of the system under study. Numerous reliability estimation models for software have been developed, and all are working on specific applications, specific environments, datasets and assumptions made by them. Initially, a systematic survey of software reliability models is done that shows how a new model is evolved from other models on the basis of their assumptions and attributes.

Among the available research in software reliability model development [2] [3], It is found that, all developed models are basically made for reliability estimation of software's developed under traditional waterfall software development life cycle process. As estimation of software reliability is primarily reliant on the process of software development. Software's developed using latest software development life cycle approach can capture and implement all the user requirements within time and budget. This work focuses on development of new software reliability estimation model that incorporates iterative software development life cycle process by replacing earlier waterfall based development process. It assumes imperfect debugging during each of iteration. All the latest iterative software development life cycle processes may be used to predict software reliability by applying proposed software reliability estimation model. Existing failure rate models cannot be applied to the current software development methodology. The proposed model takes care of complexity and paradigm shift of iterative based software development process by introducing modulation factor.

Further, Keeping in mind multi-release policies of open source software, this thesis work also proposed another model that introduces a new testing effort factor based fault content function. This factor is showing the change in fault content function with the amount of testing effort in each version of open source software development. Altogether it is reflecting complete testing effort functionality added or upgraded in each version of the software. Effort factor changes its value according to the effort coefficient which takes its value from zero to one by assuming that complexity value added in each version increases from lower to higher. Effort factor has a change in its value, depending on whether it is a minor or a major release.

For precise estimation of open source software system reliability, there is a need to have a parameter estimation method that could provide optimum parameter values of models. Classical methods [4] of parameter estimation are based on number of constraints. An alternative to these classical mathematical optimization methods is nature-inspired optimization algorithms for solution of the non-differential, non-linear and multi-modal problems [5]–[7]. The research work move ahead and also focuses on the development of new hybrid swarm evolutionary algorithm for software reliability model parameter estimation. A new algorithm based on the concept of ecological space [8] , method of differential evolution and intelligent behaviour of artificial bee colony for optimizing the parameter values [9][10]. The exploration capability in ABC algorithm has been improved by introducing the concept of ecological space. Ecological space is one of the important factors for evolution and reflects the expansion of individual bee in search space. DE technique provides the diversity of bee's population and faster convergence. The proposed algorithm has been tested with four standard failure datasets. Proficiency of proposed algorithm is compared with other well-known algorithms. Proposed algorithm is very much effective in a field of software reliability estimation and would be a competitive one among meta-heuristic optimization algorithms. Finally the thesis is concluded with the perspective of future work.

Contents

ACKNOWLEDGEMENT ..... **IV**

LIST OF PUBLICATIONS ..... **VI**

ABSTRACT ..... **VII**

LIST OF FIGURES ..... **XII**

LIST OF TABLES ..... **XIII**

CHAPTER 1 INTRODUCTION **17**

1.1 OVERVIEW OF THE STUDY 17

1.2 IMPACT OF SOFTWARE RELIABILITY ANALYSIS ON HUMAN SOCIAL LIFE 19

1.3 IMPORTANCE OF SOFTWARE RELIABILITY MODELING AND ANALYSIS 20

1.4 RESEARCH OBJECTIVES 21

1.5 OVERVIEW OF THE WORK DONE 22

1.6 CONTRIBUTION OF THE RESEARCH WORK 23

1.7 ORGANIZATION OF THESIS WORK 25

CHAPTER 2 LITERATURE SURVEY **27**

2.1 INTRODUCTION 27

2.2 SOFTWARE RELIABILITY MODELS 27

2.3 CLASSIFICATION OF SOFTWARE RELIABILITY MODELS 28

2.3.1 Early Prediction Models *28*

2.3.2 Architecture Based Models *28*

2.3.3 Hybrid Black Box Models *29*

2.3.4 Hybrid White Box Models *29*

2.3.5 Input Domain Based Models *30*

2.3.6 Software Reliability Growth Models *30*

2.3.7 Failure Rate Behaviour Based Models *31*

2.3.8 NHPP Behaviour Based Models *33*

2.4 MODEL EVOLUTION 38

2.4.1 Attributes used to Categorize Software Reliability Growth Models *39*

2.5 PARAMETER ESTIMATION TECHNIQUE 42

2.5.1 Meta-heuristic Algorithms in Various Application Domains *43*

2.5.2 Meta-heuristic Algorithms in Ground of Software Reliability *46*

2.6 PERFORMANCE MEASUREMENT METHODS 48

2.7 INFERENCE ON THE BASIS OF LITERATURE SURVEY 49

CHAPTER 3 ITERATIVE SOFTWARE FAILURE RATE MODEL 52

3.1 INTRODUCTION 52

3.2 ITERATIVE SOFTWARE DEVELOPMENT LIFE CYCLE PROCESS 54

3.2.1 Reliability in Iterative Software Development Environment 54

3.3 PROPOSED MODEL 55

3.3.1 Analytical Software Failure Rate Model for SDLC 55

3.3.2 Proposed Model Assumptions 55

3.3.3 Model Formulation 56

3.3.4 Parameter Estimation 57

3.4 APPLICATION DATASETS USED FOR EXPERIMENTATION 58

3.5 EXPERIMENTAL SETUP 58

3.6 RESULT ANALYSIS 61

3.6.1 CASE 1: Eclipse Software Dataset 61

3.6.2 CASE 2: JDT Dataset 63

3.7 CONCLUSION 70

CHAPTER 4 MODELING AND ANALYSIS OF OPEN SOURCE SOFTWARE SYSTEM RELIABILITY 71

4.1 INTRODUCTION 71

4.2 MODELING PROCEDURE FOR OSS SYSTEMS 74

4.3 PROPOSED NHPP MODEL 76

4.3.1 Non Homogenous Poisson Process based model for OSS System Development 76

4.3.2 Proposed Model Incorporating Effort Based Fault Content Function 76

4.3.3 Model Formulation 77

4.4 RESULTS AND DISCUSSION 79

4.4.1 Example 1: Firefox Dataset Analysis 79

4.4.2 Example 2: Result analysis for Genome failure data-sets 82

4.5 CONCLUSION 86

CHAPTER 5 PARAMETER ESTIMATION ALGORITHM 87

5.1 INTRODUCTION 87

5.2 GENERAL STUDY OF META-HEURISTIC ALGORITHMS 89

5.2.1 Artificial Bee Colony 89

5.2.2 Particle Swarm Optimization 90

5.2.3 Differential Evolution Algorithm 91

5.2.4 Hybrid Particle Swarm Optimization and Gravitational Search Algorithm 92

5.3 PROPOSED ALGORITHM 93

5.3.1 Mathematical Formulation for Artificial Bee Colony Algorithm 94

5.3.2 Mathematical Differential Evolution Algorithm for Global Solution of a Problem 95
5.3.3 Hybridization of Proposed Swarm-Evolutionary Algorithm 96
5.3.4 Framework of the proposed algorithm 98
5.4 EXPERIMENTAL SETUP 101
5.5 RESULTS AND DISCUSSION 102
5.5.1 Result analysis 102
5.6 CONCLUSIONS 115
CHAPTER 6 FAILURE DATASETS 116
6.1 INTRODUCTION 116
6.2 DATASETS 117
CHAPTER 7 RESULTS AND SCOPE FOR FUTURE RESEARCH 152
7.1 INTRODUCTION 152
7.2 MAJOR FINDINGS 152
7.3 FUTURE SCOPE 156

List of Figures

FIG. 1.1 GENERAL PROCESS OF SOFTWARE RELIABILITY MODELING AND ANALYSIS 18
FIG. 2.1 CLASSIFICATION OF SOFTWARE RELIABILITY MODELS BASED ON ITERATIVE APPROACH 30
FIG. 2.2 HISTORY OF SOFTWARE RELIABILITY GROWTH MODELS 31
FIG. 2.3 CATEGORIZATION OF SRGMS BASED ON WELL-DEFINED CRITERIA 40
FIG. 2.4 EVOLUTION OF SOFTWARE RELIABILITY GROWTH MODELS 42
FIG. 3.1 PLOT OF MODULATION PARAMETER VERSUS ITERATION FOR DS1 60
FIG. 3.2 PLOT OF MODULATION PARAMETER VERSUS ITERATIONS FOR DS2 61
FIG. 3.3 PLOT OF RELIABILITY VERSUS NUMBER OF FAULTS 62
FIG. 3.4 PLOT OF FAULT INTENSITY IN VARIOUS ITERATIONS 62
FIG. 3.5 PLOT OF RELIABILITY VERSUS ITERATIONS 64
FIG. 3.6 PLOT OF FAILURE INTENSITY VERSUS ITERATIONS 64
FIG. 3.7 PLOT OF RELIABILITY VERSUS FAULTS IN VARIOUS ITERATIONS 69
FIG. 3.8 PLOT OF FAILURE INTENSITY IN SEVERAL ITERATIONS 69
FIG. 4.1 SOFTWARE RELIABILITY MODELING PROCESS FOR OSS SYSTEMS 75
FIG. 4.2 ESTIMATED NUMBER OF FAULTS USING FIREFOX 3.0 81
FIG. 4.3 ESTIMATED NUMBER OF FAULTS USING FIREFOX 3.5 82
FIG. 4.4 ESTIMATED NUMBER OF FAULTS USING FIREFOX 3.6 82
FIG. 4.5ESTIMATED NUMBER OF FAULTS USING GENOME 2.0 85
FIG. 4.6ESTIMATED NUMBER OF FAULTS USING GENOME 2.2 85
FIG. 4.7 ESTIMATED NUMBER OF FAULTS USING GENOME 2.3 86
FIG. 5.1 ARTIFICIAL BEE COLONY ALGORITHM 90
FIG. 5.2 PARTICLE SWARM OPTIMIZATION ALGORITHM 91
FIG. 5.3 DIFFERENTIAL EVOLUTION ALGORITHM 92
FIG. 5.4 HYBRID PSOGSA ALGORITHM 92
FIG. 5.5 2D VIEW OF HYBRID DE ASSISTED ABC ALGORITHM 99
FIG. 5.6 ECOLOGICAL SPACE BASED HYBRID SWARM-EVOLUTIONARY (DE ASSISTED ABC) ALGORITHM 100
FIG. 5.7 DE ASSISTED ABC ALGORITHM 101
FIG. 5.8 ESTIMATED NUMBER OF ERRORS AT TIME T USING GO MODEL AND DS5 105
FIG. 5.9 ESTIMATED NUMBER OF ERRORS AT TIME T USING INFLECTION S SHAPED MODEL AND DS5 105
FIG. 5.10 ESTIMATED NUMBER OF ERRORS AT TIME T USING PTZ MODEL AND DS6 106
FIG. 5.11 ESTIMATED NUMBER OF FAULTS USING GO MODEL AND DS7 106
FIG. 5.12 ESTIMATED NUMBER OF ERRORS USING PTZ MODEL AND DS8 107

List of Tables

TABLE 2.1 FAILURE RATE BEHAVIOUR BASED MODELS 32
TABLE 2.2 NHPP BASED MODELS 33
TABLE 2.3 PERFORMANCE MEASUREMENT METRICS[13] 48
TABLE 3.1 SUMMARY OF FAILURE RATE BASED SOFTWARE RELIABILITY MODELS FOR COMPARISON 60
TABLE 3.2GOODNESS-OF-FIT ES/TIMATED USING DS1 (ECLIPSE SOFTWARE FAILURE DATASET) 65
TABLE 3.3GOODNESS-OF-FIT ESTIMATED USING DS2 (JDT SOFTWARE FAILURE DATASET) 67
TABLE 4.1 RESULT ANALYSIS USING FIREFOX 3.0 80
TABLE 4.2 RESULT ANALYSIS USING FIREFOX 3.5 80
TABLE 4.3 RESULT ANALYSIS USING FIREFOX 3.6 80
TABLE 4.4 RESULT ANALYSIS USING GENOME 2.0 83
TABLE 4.5 RESULT ANALYSIS USING GENOME 2.2 83
TABLE 4.6 RESULT ANALYSIS USING GENOME 2.3 84
TABLE 5.1 STATISTICAL RESULTS OF PARAMETER ESTIMATION FOR GO MODEL 107
TABLE 5.2 STATISTICAL RESULTS OF MEAN SQUARED ERRORS FOR GO MODEL 107
TABLE 5.3 STATISTICAL RESULTS OF SUM OF SQUARED ERRORS FOR GO MODEL 108
TABLE 5.4 STATISTICAL RESULTS OF ELAPSED TIME IN SECONDS FOR GO MODEL 108
TABLE 5.5 STATISTICAL RESULT FOR PARAMETER ESTIMATION USING INFLECTION S SHAPED MODEL 108
TABLE 5.6 STATISTICAL RESULT FOR MEAN SQUARED ERRORS USING INFLECTION S SHAPED MODEL 109
TABLE 5.7 STATISTICAL RESULT FOR SSE USING INFLECTION S SHAPED MODEL 109
TABLE 5.8 STATISTICAL RESULT FOR ELAPSED TIME IN SECONDS USING INFLECTION S-SHAPED MODEL 109
TABLE 5.9 STATISTICAL RESULT FOR PARAMETER ESTIMATION USING PTZ MODEL 110
TABLE 5.10 STATISTICAL RESULT FOR MSE USING PTZ MODEL 110
TABLE 5.11 STATISTICAL RESULT FOR SSE USING PTZ MODEL 110
TABLE 5.12 STATISTICAL RESULT FOR ELAPSED TIME IN SECONDS USING PTZ MODEL 111
TABLE 5.13 STATISTICAL RESULT FOR PARAMETER ESTIMATION USING GO MODEL 111
TABLE 5.14 STATISTICAL RESULTS FOR MSE USING GO MODEL 111
TABLE 5.15 STATISTICAL RESULT FOR SSE USING GO MODEL 112
TABLE 5.16 STATISTICAL RESULT FOR ELAPSED TIME IN SECONDS BY VARIOUS ALGORITHMS USING GO MODEL 112
TABLE 5.17 STATISTICAL RESULT FOR PARAMETER ESTIMATION USING INFLECTION S SHAPED MODEL 112
TABLE 5.18 STATISTICAL RESULT FOR MSE USING INFLECTION S SHAPED MODEL 113
TABLE 5.19 STATISTICAL RESULT FOR SSE USING INFLECTION S SHAPED MODEL 113
TABLE 5.20 STATISTICAL RESULT FOR ELAPSED TIME IN SECONDS USING INFLECTION S SHAPED MODEL 113
TABLE 5.21 STATISTICAL RESULT FOR PARAMETER ESTIMATION USING PTZ MODEL 114
TABLE 5.22 STATISTICAL RESULT FOR MSE USING PTZ MODEL 114

TABLE 5.23 STATISTICAL RESULT FOR SSE USING PTZ MODEL.....114
TABLE 5.24 STATISTICAL RESULT OF ELAPSED TIME USING VARIOUS ALGORITHMS FOR PTZ MODEL.....115
TABLE 6.1 FAILURE DATASETS USED FOR IMPLEMENTATION.....117
TABLE 6.2 ECLIPSE DATASET (DS1).....120
TABLE 6.3 JDT DATASET (DS2).....134
TABLE 6.4 FIREFOX FAILURE DATASET (DS3).....136
TABLE 6.5 GENOME FAILURE DATASET(DS4).....141
TABLE 6.6 FAILURE DATASET(DS5).....143
TABLE 6.7 FAILURE DATASET (DS6).....144
TABLE 6.8 FAILURE DATASET (DS7).....146
TABLE 6.9 FAILURE DATASET (DS8).....147

Abbreviations

SRM	Software Reliability Models
SRGM	Software Reliability Growth Models
SDLC	Software Development Life Cycle
JM	Jelinski- Moranda
GO	Goel-Okumotto
GOI	Goel-Okumotto Imperfect Debugging
SW	Shick-Wolverton
MSW	Modified Shick-Wolverton
PTZ	Pham-Teng-Zhang
MLE	Maximum Likelihood Estimation
LSE	Least Square Estimation
LLF	Log Likelihood Function
FDR	Fault Detection Rate
FIR	Fault Introduction Rate
FRE	Fault Removal Efficiency
MVF	Mean Value Function
SSE	Sum of Squared Errors
MSE	Mean Square Error
MAE	Mean Absolute Error
MEOP	Mean Error of Prediction
AE	Accuracy of Estimation
TS	Theil's Statistics
PRR	Predictive Ratio Risk
NHPP	Non Homogenous Poisson Process
OSS	Open Source Software

DE	Differential Evolution
GA	Genetic Algorithm
GP	Genetic Programming
PSO	Particle Swarm Optimization
GSA	Gravitational Search Algorithm
ABC	Artificial Bee Colony
ACO	Ant Colony Optimization

Chapter 1 INTRODUCTION

This chapter familiarizes implications of software reliability in today's digital world. Motivation in software reliability model development and their parameter estimation methods are discussed. The aim of the research work is highlighted. Chapter wise organization of thesis work is presented at the end of this chapter.

1.1 Overview of the Study

Today's cyber world is heavily reliant on software and software-controlled applications. Software is considered as the most critical component in the growing digital world, it is playing a key role in every aspect of life whether related to digital identity, digital currency, gadgets, defense, medical, business enterprises, transportation, home security systems or daily money transactions and so on. Software has not only improved the living standards and makes human life more comfortable. But unpredictable software failures can severely affect proper functioning of the whole system [11]. It is necessary to eliminate all the latent problems in software as early as possible.

Reliability of software is considered as the most crucial quality attributes. Although at most care has been taken while developing a software system using latest tools and techniques but, still it is not feasible to develop defect free software practically. Organizations are trying to ensure highest reliability of the software being developed but ensuring the same is very difficult due to the increasing software size, budget constraints, time constraints and shortage of skilled man-power.

When the software is deployed, then only feedback of the customers, there complaints, compliments and outages are the ways to reflect software reliability. But by then, reliability estimation is too late. Perquisite of the software developers is to know whether developed software is reliable before they are dispatched to customers. In the competitive arcade of software development it is necessary for software industries to ensure reliability of their software to satisfy their customers and to make an outlook in the global market.

How to enrich software reliability and reduce its cost to a satisfactory level is a major concern of today's software development industries. There are four main methodologies to increase the reliability of software[3]. These approaches are primarily related to the prevention of faults, removal of faults, tolerance and forecasting of faults. Prevention of faults and their removal approach is in hands of software developers and testing team members. To remove faults, one should have skilled developers and tester's in-order to make software more reliable. Fault tolerance approach makes the system robust by hiding faults in place of removing existing faults. It involves the methods like recovery block, N-version software, self-checking software, rollback method and

design by diversity method, which involves functionally equivalent to independently developed components to tolerate faults. Fault forecasting approach is considered as the major approach and implemented using statistical modelling of software reliability methods. Flow chart in Fig. 1.1 depicts the way to develop a software reliability estimation model [12].

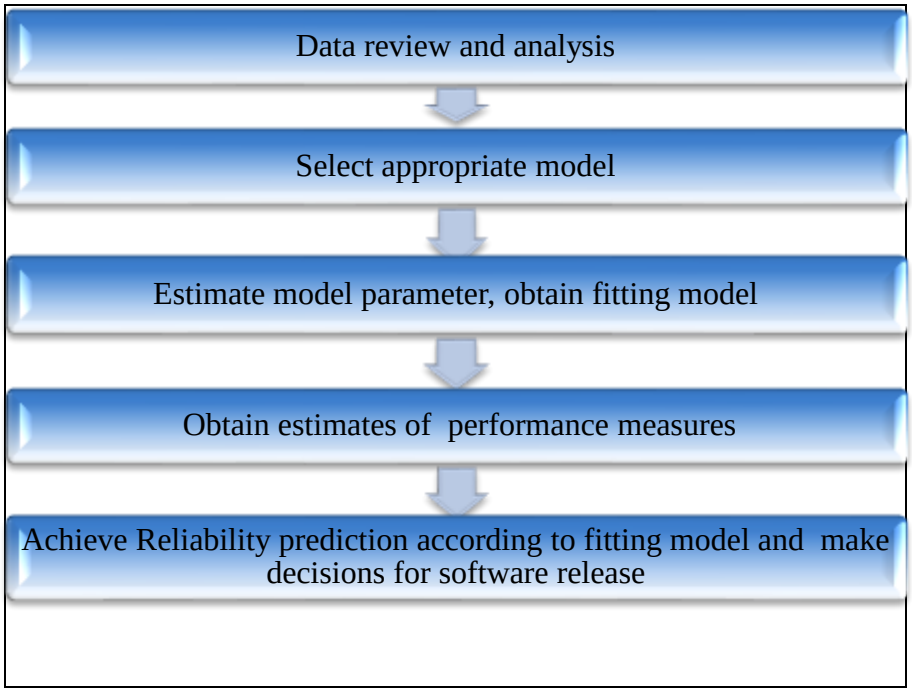

Fig. 1.1 General Process of Software Reliability Modeling and Analysis

For the software reliability models to be valuable, they must incorporate some values in decision making process. These models must help in determining the amount of effort, the time, the money invested, and must help in making decisions regarding when a piece of software can be deployed with confidence in its reliability. As a matter of economy to produce reliable software, it is essential to measure and control reliability of software while development. To do this, number of software reliability models is being developed. But in modeling software reliability, we often encounter number of problems that majorly includes the given below:

1. How to develop a model that would precisely estimate reliability of a specific software system which has been developed under certain software development methodology?
2. What are the ways to better estimate parameter values of developed software reliability models?
3. How to collect trustworthy software failure datasets to make precise measurement of future software reliability?

From last four decades, reliability community has developed many of software reliability estimation models in several domains but these models are specific to an application. One of the

models that have been developed for a specific application cannot be applied to precisely estimate reliability of the software developed under different environment and conditions [13]. With the growth of software industries in software development, this is always required to develop a specific application based reliability estimation model for software's that have been developed using latest tools, techniques and methodologies. Selection of software reliability model's parameter estimation method is a major problem in predicting the exact software reliability. There are number of existing methods for software reliability model parameter estimation, but these methods are not giving satisfactory results when applied on non-linear problem solution. There is a need to examine methods that can precisely estimate parameter values of the newly developed software reliability estimation model. Theses model parameters employ a great deal of influence on the accuracy of reliability estimation models.

Software reliability models are found to be the only methods that are in significant use in industries and academia for future software reliability estimation[14][15][16]. There is an emergent need in developing exact methods for software reliability estimation during development. Prediction and optimization of models are primarily dependent on parameters of models. Nonlinear nature of models makes it difficult to statistically analyse and estimate the parameters values. It is very difficult and challenging job to optimize software reliability model parameters to prove their proficiency.

Further, model development and parameter estimation procedures necessitate a sufficient amount of failure data to catch precise and trustworthy estimate of software reliability. Companies are unwilling to release their programs failure data due to competitive demand in the software market and fear of declining their image. Matter of software reliability estimation becomes worse due to lack of availability of software failure data from software companies. As a result, new models are formulated and validated against the datasets which are either available in the literature or which have previously appeared elsewhere.

Numerous software reliability estimation models have been developed by the researchers in last years. It is found that the model which is overall a best choice is not always good for a particular dataset. Reliability prediction made by the model may be less accurate than desired even when appropriate model is used. For this reason a great deal of research is needed in making precise reliability growth of the software developed under latest tools and techniques.

1.2 Impact of Software Reliability Analysis on Human Social Life

Booming software industries has a major impact on humanity. One may not have noticed that analog parts of appliances like washing machines, TVs and watches are being replaced by CPUs and software. Software controlled systems in various household appliances offers a

competitive cost, compact design, flexible handling with enriched features. Today's society is making use of these embedded software based appliances by trusting that "software never breaks". But there are number of misfortunes instigated by software that proves this myth to be a wrong. There is a need to make an estimate of software reliability to ensure dependability of the used software.

As more and more software is entering into embedded systems, there is a need to ensure that they wouldn't embed any further tragedies. If this is not deliberated cautiously, reliability of software may become reliability bottleneck of the complete system. To make precise estimate of software reliability, in literature there are number of reliability estimation models. These models have been developed up to a point that significant result could be attained just by the application of appropriate model to a problem. Yet now, no single model has been developed that is universal to all the situations. There is a need to make assumptions and perceptions to simplify the particular set of problems.

1.3 Importance of Software Reliability Modeling and Analysis

Among the models available only few are useful others are found to be wrong. The research papers are having no more than 31% experimental researches, where only about 13% are purely experimental [17]. This low number is due to two reasons: (1) public experimental data sets in reliability estimation are very limited; (2) producing software reliability data through experiments typically require long time clusters. Also, the number of research works in "Reliability Assessment" has clearly increased since 2002 and especially during the last five years. But still there is a big requirement to precisely estimate the reliability of software through software reliability growth models and various other techniques. As computer software's are used in recent day today applications, it shows that major focus of researchers has moved to precise software reliability growth estimation.

Further, there are number of examples of software failures by which authors get motivated towards a research in software reliability growth estimation methods for the software that are being developed. Some of the major software failures in history are:

1. Because of malfunctioning and early warning system of Soviet nuclear system, World War III narrowly averted and it was reported that the United State had terrified an attack on his country. But after some time soviet air defense officer named Stanislav Petrov detect this as a false alarm.
2. In June 1982, flaws in the software led to massive blast along part of a pipeline and it caused the biggest non-nuclear explosion in a planet's history.

3. In March 1986, a Mexican Airline gets worn-out to a mountain due to the reason that its software was not able to appropriately find out a mountain position.
4. Because of flaws in the computer software, patients of St. Mary's Mercy Medical Center gets killed on paper in 2003 and it costs the lives of about 8500 patients.
5. In February 2014, routine software upgrade malfunctioning in Suncorp bank caused the vanishing of money from customer's bank accounts.
6. The company Nissan because of a software malfunction, recalled vehicles for Airbag software malfunction in 2015.
7. In 2015, About 30,000 Swiss HSBC Bank Accounts Leaked to Media in 2015.
8. In August 2019, more than 100 US flights get cancelled and more than 200 get delayed due to major IT software failure.
All these circumstances and events have made it deceptive that one must regulate software reliability before planting them into operation.

1.4 Research Objectives

SRGMs are found to be the most significant in industries and academia. No existing model is well applicable in all domains to estimate software reliability. Each model has been developed with specific assumptions and datasets[13]. These specific models are not found to provide exact estimate of software reliability in other environmental conditions. Every model either needs enhancements or modifications to provide better estimate of reliability for other specific application environment. Further, traditional methods of parameter estimation do not provide satisfactory results within reasonable amount of time. Thus, new methods for software reliability model parameter estimation need to be adopted in the field of software reliability estimation. Hence the research objectives are extracted as below:
Developing a software reliability estimation model using some meta-heuristic algorithms (soft-computing technique) that will be applicable for measurement of reliability of software's that has been developed using latest tools and techniques. Developed model will be capable in implementing all the user requirements within time and budget for specific datasets**.** Therefore, the research goal can be seen as consisting of following sub-goals:

1. To, study various software reliability models and various optimization techniques involving meta-heuristic techniques**.** The first goal is to study existing software reliability estimation models from various domains. Also to study various traditional as well as latest methods of software reliability model parameter estimation.

2. a). To propose software reliability estimation model that will predict the reliability of system software. The objective of this goal is to enhance existing traditional approach based software reliability models in to a new model that can capture the behaviour of latest software development environment and can precisely estimate the reliability of open source software as well as closed source software systems.

   b). To implement the proposed model. Here, the goal is to validate the proposed model on some real world software reliability analysis problems.

3. To analyze the optimization capability of existing evolutionary algorithms and selecting the algorithm that will best suit in optimizing the reliability model parameters. There is a need to find the best method among the existing methods of parameter estimation in reliability estimation so that exact reliability could be estimated by the statistical models. A new variant of existing meta-heuristic can be proposed which can perform efficient reliability analysis for various software failure datasets.

1.5 Overview of the Work Done

Thesis work is dedicated to the development of some methods for reliability analysis of software's that has been developed using latest tools and techniques. Proposed work makes analysis of software reliability growth modeling and leverages the strength of meta-heuristic algorithms for parameter estimation of SRGMs. Moreover proposed software reliability models are validated on the real world failure datasets of open source software as well as close source software datasets. Subsequent work is carried out in order to achieve each objective.

1.a) In order to achieve first objective various existing software reliability models were studied. A tree structure is developed to show development history of models over a wide range from 1972 to 2019. These existing models are surveyed on the basis of assumptions used by these models. All surveyed models are categorized on the basis of some attribute based criteria that each model uses in its mathematical formulation. These categories facilitated authors in making evolution diagram of models from previous models. Further, using specific assumption of a model and attribute based criteria all models are evaluated on the basis of evolution of models from existing models. This evolution shows how one model is enhancing the features of already existing models and provides a way by which researchers can identify the need to propose a new model by seeing the limitations in the previous models.

b) To achieve the objective of studying various optimization techniques mainly meta-heuristic nature inspired algorithms are studied. We studied the methods of parameter estimation over a wide range from 1985 to 2019 and found the limitations in these methods.

2.a) To achieve second objective we propose an iterative failure rate behaviour based software reliability model that will predict the reliability of systems in each successive iterations of software development. This proposed models removed the limitations of traditional models that has been developed using traditional waterfall software development methodology. The proposed model works significantly well in iterative software development environment to estimate software reliability. Further, to extend the objective of software reliability model development, second objective has been extended to propose a new NHPP behaviour based model for multiple releases of OSS systems. These models are found to be in wide spread use in software industries for reliability estimation. Proposed failure rate model is validated on 12 iterations of Eclipse project failure datasets and 6 iterations if JDT project failure datasets. Similarly efficiency of proposed NHPP model is tested using 3 versions of Firefox failure datasets and three versions of Genome failure datasets. These models are well proving their efficiency in the used failure dataset environments.

2.b) Implementation and analysis of the proposed models is done using real world failure dataset of Eclipse, JDT, Firefox and Genome projects. Mat lab (R2015a) has been used to implement the proposed software reliability growth models. Results shows that proposed models are outperforming their counterparts in significantly estimating the system software reliability.

3. A new algorithm is proposed by using well-known artificial bee colony algorithm and differential evolution algorithm. ABC algorithm is having deprived exploration and may fall into local maxima due to lack of population diversity. DE algorithm is having unbalanced exploitation and exploration. Thus, this will be a good effort if limitations of these algorithms could be removed. Proposed algorithm is absorbing only the best features of these two algorithms. Proposed algorithm is hybridized using swarm based features of ABC algorithm and evolutionary features of DE algorithm. Proposed Swarm-Evolutionary hybrid algorithm is validated using well known NHPP based models and four real world datasets.

1.6 Contribution of the Research Work

Summary of the major contribution of our work is as follows:

- Thesis work is focused on the two famous groups of software reliability modeling. These are failure rate based models and NHPP group of models. In this work, two software reliability estimation models are proposed. Out of these two, one is belonging to the group of failure rate models and other is based on the assumptions of NHPP group of models.
- Earlier failure rate models are grounded on software systems that have been developed using traditional waterfall software development lifecycle process. Due to stability in implementing software using traditional approach, in this thesis work, a new Iterative

software reliability estimation model is proposed that considers behavior of recently used iterative software development life cycle process.

- Proposed iterative failure rate model assumes that in each of the iteration of software development, new functionality enhancement occurs due to addition and removal of bugs in each of the iteration. Due to added functionality in each of iteration, design modifications are made which causes change in requirements at each stage of software development. These changing needs in each of the iteration in-terms of defects, testing effort and functionality are reflected in the proposed model using modulation factor.
- In each of iteration new functionality is added or modified, existing bugs are fixed; some of them may remain unresolved and moves to the up-coming iterations. System development functionality gets improved in each of the iteration. This feature has been reflected using modulation parameter, which takes its values from 0 to 1 because functionality and user acceptance increases from lower to higher in each of the on-going iterations of software development. Using modulation parameter, modulation factor is quantified.
- Proposed iterative model has been validated for the reliability assessment of open source software.
- We found that failure rate decreases and reliability increases in each of the upcoming iteration of software development.
- Proposed iterative model supports software developers and end users in estimating software reliability at each iteration of software development and hence for each of the iterations in the evolution of a software.
- Further to incorporate modern software development environments and technologies, new NHPP model for reliability estimation of multiple versions of OSS systems has been developed.
- Proposed NHPP model incorporates a new testing effort factor based fault content function for integrating varying needs in each release of software development.
- Proposed model has been implemented and tested on various releases of Firefox and Genome project failure data set.
- We find out the shortcomings in software reliability model parameter estimation methods and proposed a new optimization algorithm based on the hybrid nature of swarm evolutionary algorithms.

1.7 Organization of Thesis Work

Rest of work in this thesis is structured as below:

Chapter 2 presents the survey of existing literature work in software reliability modeling. Literature work refers to various groups of software reliability models and parameter estimation techniques used in software reliability analysis. Depending on the survey, models are classified into various categories and an evolution diagram is presented to show the evolutionary history of software reliability models. This work emphasis necessity for development of a new model for estimation of software reliability in various software development environments. Further, for accurate reliability estimation there is a need to have an optimization method that can estimate model parameter values precisely. This further elevates the need of having a new parameter estimation method that can better adapt to latest software development methods in reliability analysis. Finally an inference from the literature is drawn to identify the requirement of the proposed work.

Chapter 3 presents a new failure rate model centered on iterative SDLC process. The proposed model has been derived from the general class of failure rate behavior-based models and exploits iterative behaviour of software development process. All latest iterative SDLC processes may be used to estimate software reliability by applying proposed model. The proposed model takes care of complexity and paradigm shift of iterative based software development process by introducing modulation factor. The accuracy of proposed model is summarized using SSE and MSE criteria by comparing with five existing well known failure rate behaviour based models. Proposed model is validated using 21 iterations of Eclipse and JDT project failure datasets.

Chapter 4 presents a new NHPP model for reliability estimation of multiple versions of OSS systems to incorporate modern software development environments and technologies. Proposed model incorporates a new testing effort factor for integrating varying needs in each release of software development. It comprises imperfect debugging with the possibility of fault introduction. Efficiency of the proposed NHPP model is analyzed by comparison with existing NHPP based models. Proposed model is validated on various releases of Firefox and Genome project failure data set.

Chapter 5 presents the methods of parameter estimation and a novel meta-heuristic algorithm for software reliability model parameter estimation. Proposed algorithm is a new hybrid algorithm that combines features of ABC and DE algorithms along with a new ecological space based factor used to reflect the ecological fitness of the individuals. The proposed algorithm is validated using well known mathematical functions of NHPP based models and real world datasets of Real Time Command and control Systems, US Navel Tactical Data Systems dataset and Tandem Computer Software failure datasets.

Chapter 6 presents the failure datasets that is used to implement the proposed work in this thesis.

Chapter 7 summarizes the results in the previous chapters of the thesis. In this chapter thesis conclusion is discussed along with future research possibilities.

References. This section details the used references in this thesis work.

Chapter 2 LITERATURE SURVEY

This chapter provides a systematic literature review in the field of software reliability assessment along with parameter estimation techniques. The chapter starts with the discussion of famous group of software reliability estimation models and their enhancements. It also presents the proposed classification scheme in the field of software reliability modeling.

2.1 Introduction

Software reliability models are the most powerful tool for assessing and predicting software reliability. In last four decades about 300 software reliability models has been developed [18][19][20][21][22]. All the developed models have specific environments, assumptions and applications [23][24][25]. To identify the concept of software reliability model development, extensive literature has been collected from various aspects of software reliability. Comprehensive literature in this section covers major books, review papers and research papers etc. to deal with all aspects of software reliability measurement. The collected literature is majorly research oriented and involves vital research papers that explains all perspectives to software reliability model development and majorly covers following categories to involve entire essential research articles to formulate a current research problem-

- Software Reliability Models
- Failure Rate Models
- NHPP Models
- Major attributes used to distinguish models
- Parameter estimation techniques

2.2 Software Reliability Models

In last few decades with the growing digital world, number of software reliability models have been developed under various categories for measurement and enhancement of software reliability[26]–[32]. An effort is made to reveal all the essential literature in an organized format. All the surveyed articles are very much helpful in present research work of software reliability model development. This thesis work majorly discusses all the influential articles in key categories of software reliability modeling. Authors have surveyed number of classifications of software reliability models [3], [11]. Survey of the key categories of software reliability modeling is done by the authors in this thesis. Different Classification schemes are given by number of researchers for software reliability modeling. For example, a classification given by Sharma [13] is based on the SDLC phases and depicts various software reliability models during each phase of software development. Major literature in this thesis work follows this SDLC based classification scheme

and focuses on the group of SRGMs that covers failure rate based models and NHPP based models.

2.3 Classification of Software Reliability Models

This section classifies software reliability models based on iterative SDLC process by extending the classification scheme given by Sharma et al. [13]. Iterative model is like a cyclic process that mainly focuses on initial very simple implementation that progressively gains complexity and wider set of features till final system is completed [33]. Most recently built iteration and its feedback from evaluation are used in the next iteration and accordingly refinements are made in future iterations. Each of iteration provides enhancements and at-least found to be better than the last. Iterative development adapts rapidly to ever-changing needs of projects and clients within lowest time and budget. Sharma et al. [13] has given classification of models based on waterfall SDLC process. In this work, keeping iterative software development practice in mind software reliability models is categorized.

Here classification of models based on iterative approach is given in Fig. 2.1. Models are grouped into five categories. Each model category is assigned to a specific phase in iterative software development process for reliability estimation. Every phase in iterative software development process is associated with specific requirements and future plans. It shows that each of the iteration is associated with a parameter that shows added functionality and user acceptance level in each of the iteration. Initially, system is assumed to be least reliable and as the number of iterations proceeds, system moves towards refinement and gains reliability.

Software reliability models are classified as below[13] :

2.3.1 Early Prediction Models

These models make use of characteristics of the requirement phase of software development to deduce information about the reliability prediction of software. These could predict the risk of software development very early before the projects actually starts. These models are very few and have a modest impact in software reliability measurement [34] [35].

2.3.2 Architecture Based Models

From the mid of 1990's architecture based models are getting an attention because of the increasing size and complexity of software [36][37]. The main emphasis of these models is to obtain an estimate of the component reliability on the basis of architecture of software. With the help of whole application architecture and system component reliability, these models can estimate sensitivity of application reliability. These models do not consider many aspects and features of modern software development techniques like concurrent execution of components. Various

approaches on which reliability estimation by architecture based models depends includes following steps.

Module Identification: Modules are considered as independent identity and can be designed, implemented and tested independently.

Architecture of the Software: Software architecture defines the way by which different modules interact to achieve specific application tasks. Information about the execution time of each module is defined by the software architecture, further all interaction among modules only occurs by the execution control transfer.

Failure Behaviour: Failure of the module is also associated to the software architecture. Failure of modules may happen either at the execution time of the module or at the control transfer from one module to the other module. Failure of the modules and interfaces between modules specifies the reliability of software.

Combining the Architecture with Failure Behaviour: There are three approaches that are mainly used to combine failure behaviour of software with the architecture of a system. Architecture based models are further categorised into three ways: state based models; Path based models and additive models. State based models epitomise the architecture of the software system using the control flow graphs. Path based models considers enumeration of several execution paths in an application. Additive models do not consider the architecture of the system to estimate reliability of the system. By using component's failure data, additive models estimates overall system reliability [38][39].

2.3.3 Hybrid Black Box Models

These SRGMs combines features of both the black box and input domain model. Group of input domain models involves internal structure of software in reliability estimation and are assumed as the group of white box models. Black box models only considers interaction of software with the system within which they are operated [40].

2.3.4 Hybrid White Box Models

These models contemplate how internal structure of the software is organised for software reliability estimation process. Hybrid white box models are combining the particular features of SRGMs with features from the white box models. These models are modeling software reliability prediction on the basis of architecture of the system [41].

2.3.5 Input Domain Based Models

This approach makes uses of the properties of the operational usage of the program that are in the form of test cases that gets executed properly. Assumptions related to specific software reliability models in this group are the outcomes from the input samples used in test cases that provides some information about the failure behavior of the software and the behavior of the program for other input values are also found to be close to the inputs used in test-cases[42].

2.3.6 Software Reliability Growth Models

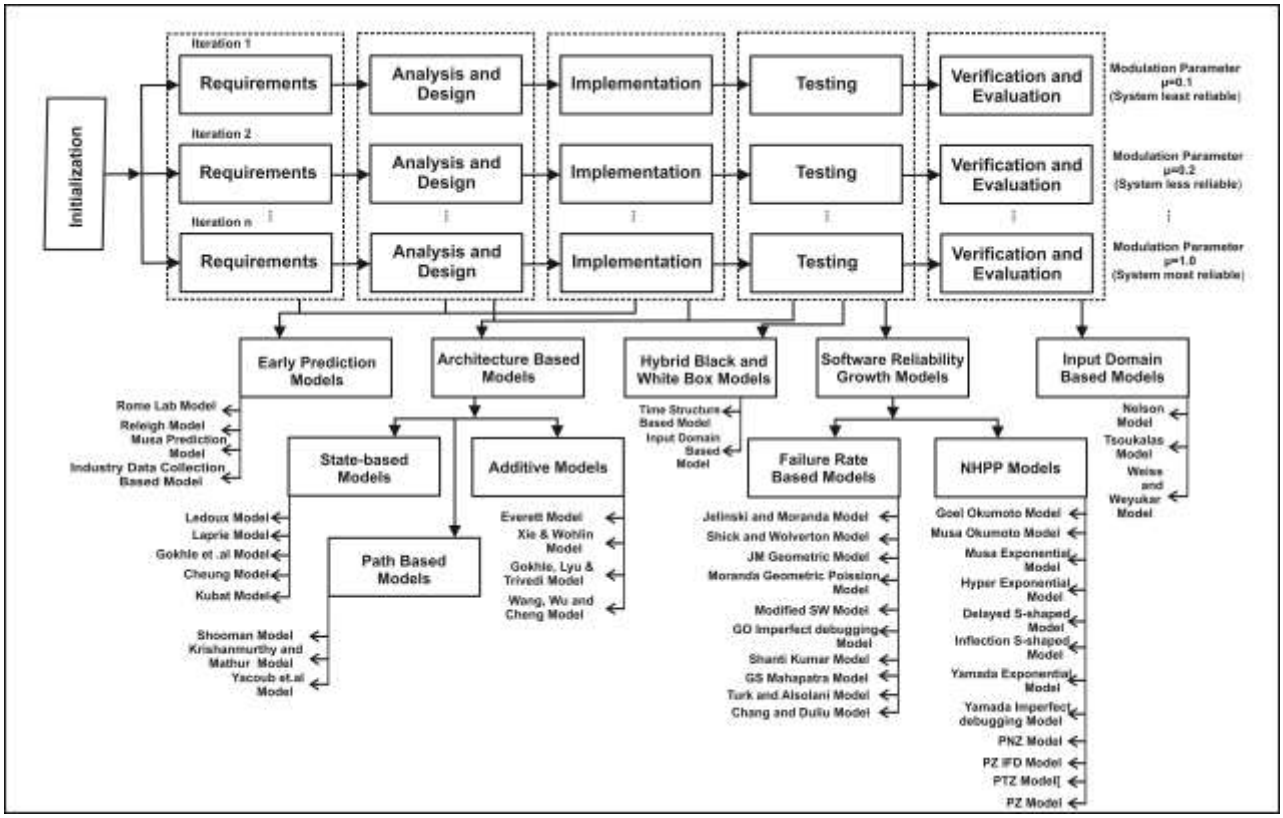

Fig. 2.1 Classification of Software Reliability Models Based on Iterative Approach

SRGMs are providing the best way to measure software reliability[43]–[45] [19], [46]–[48],[3], [49]–[51]. SRGMs are considered as the most successful class of software reliability models among various categories specified in the Fig. 2.1. These models are existed in one or other form through numerous publications and these publications in last decades are proving the success of this class of models. SRGM catches failure behaviour of software while doing testing and using this behaviour, they extrapolate the functioning of software during operation. These models depict the reliability of the software using failure data information and various trends that are observed in failure data. These models treat the software as monolithic entity that only interacts with the external environment due to which these models are also called as black box models. These models use the failure data to estimate model parameters.

The SRGMs are further classified into two most important categories of software reliability models. These are Failure rate models and Non–Homogenous Poisson process behaviour based models. History of software reliability growth models is shown in Fig. 2.2. A tree structure is made with all variation in software reliability modeling over a wide range from 1972 to 2019.

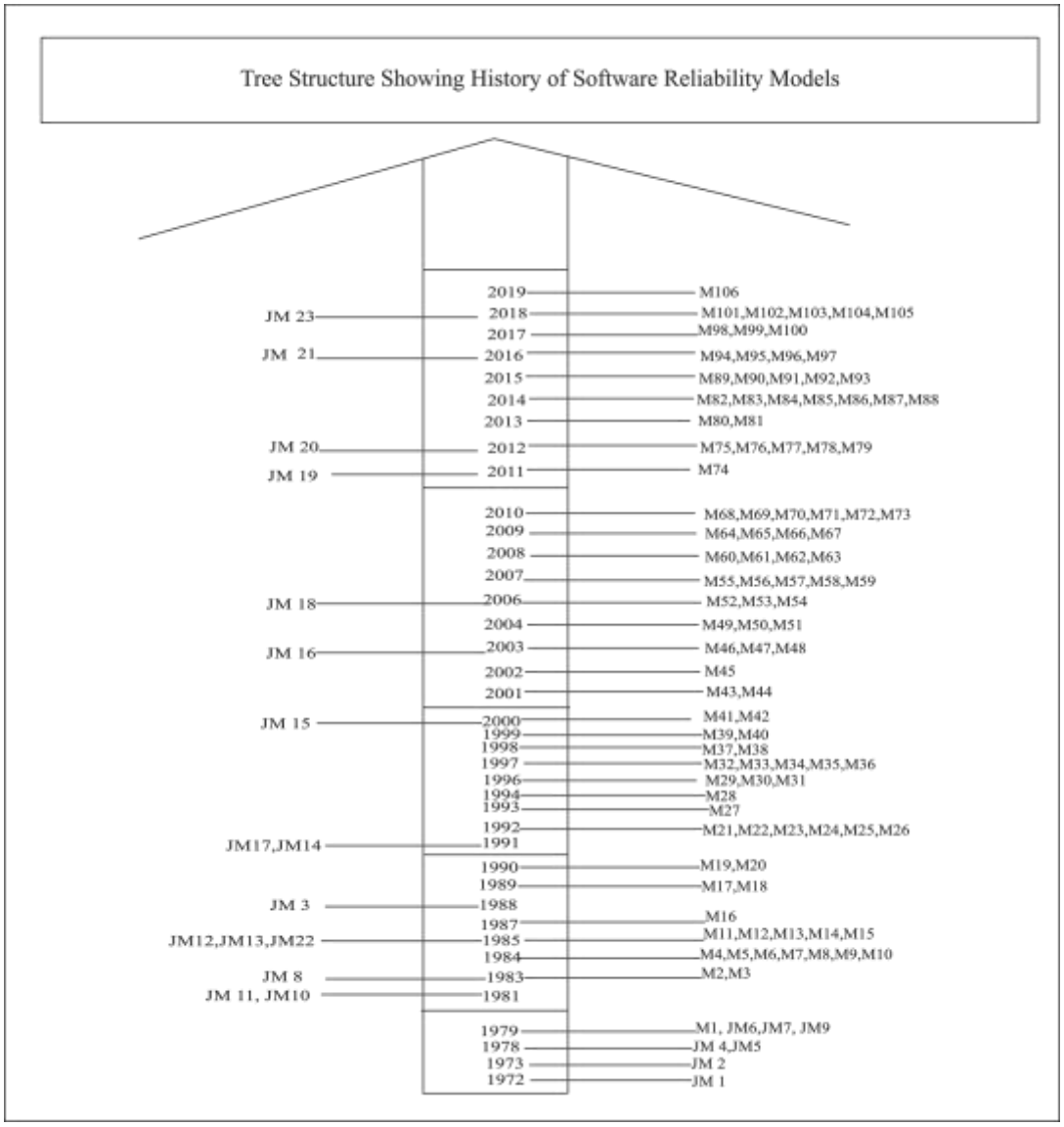

Fig. 2.2 History of Software Reliability Growth Models

2.3.7 Failure Rate Behaviour Based Models

Failure rate behaviour based models are basically used to analyze the program failure rate per fault and studies how the rate of failure changes at time of failure in an interval of time. JM model is the first software reliability estimation model [44]. This is a perfect debugging behaviour based model and assumes that initially there is fixed, constant and unknown number-of faults in the software. These models assume that the time between failures is independent and distributed exponentially. Specific work that includes subsequent work proposed by number of researchers in this group of models is discussed in Table 2.1.

Table 2.1 Failure Rate Behaviour Based Models

Sr. no.	Year	Assumptions
JM1[44]	1972	It is an earliest Markov process based Model, this assumes that total no. of initial faults in a software is always unknown and has a fixed value. This considers that time between failures are always independent random quantities and are distributed exponentially.
JM2[52]	1973	A Bayesian reliability growth model is presented here and assumes that program is complete to work for a continuous time periods between the failures. It also considers a repair rule for program developers at each failure. It does not consider the internal structure of the program.
JM3[53]	1988	This Model is fitting well into a general framework of Bayes problem, and assumes a Bayesian approach for inference by considering the conditions as an empirical Bayes-problem.
JM4[54]	1978	This model is evolved from the JM model; Hazard function is considered as proportional to the current number of total fault content and to the time elapsed since a last failure.
JM5[51]	1977	This model extends JM Model and SW Model; It allows possibility of more than one fault at each of the time interval.
JM6 [43]	1979	It follows Markov process like the JM Model. This characterizes the transition between the modules while execution as following the Markov property.
JM7[51]	1979	This is evolved using the JM and geometric model based and follows Poisson distribution based failure rate. It assumes that the number of faults occurring at intervals follows Poisson distribution with an intensity rate Dk^{i-1}.
JM8[55]	1998	It has extended the JM geometric model by describing the behaviour of software as having safe and unsafe states.
JM9[14]	1979	This model considers phenomenon of imperfect debugging for software development and testing.
JM 10[45]	1981	This model is developed by a variation of the JM model. And assumes that : a) Time separations between error detections, b) Number of errors per written instruction, c) Failure rate of software is considered as proportional to current error content in software.
JM11[56]	1981	This model is presented as a special case of JM and NHPP models.
JM 12[57]	1985	This is evolved by considering the Bayesian process in JM model and Mein-hold and Singpurwalla model. It also assumes that it is easy to calculate the distribution of undetected errors at the end of testing in-order to see the relative effects of uncertainty in number of errors and in the efficiency of fault detection.
JM 13[58]	1985	In this model alternative formulation of JM and Little-wood models are presented. Here a formulation in terms of failure rate rather than inter failure time is given.
JM14[59]	1991	It assumes that different faults may have different contributions to the failure rate. In addition, the structure of software is also incorporated in this approach.
JM15[51]	2000	This model is extended from the JM Model. Effect of environmental factors has been incorporated in the model development.
JM16[60]	2003	A Moranda de-eutrophication model is proposed by assuming time between failure as statistically independent exponential random variable with given failure rate.

JM17[61]	1991	It has been evolved from the assumptions of JM model and formulates the total expected costs of software with two different release policies.
JM18[62]	2006	This model generalizes the JM model by introduction of a negative-binomial prior distribution to represent the number of faults remaining and a Gamma distribution to represent the rate by which each fault is exposed.
JM19[63]	2011	This is the modification of the famous JM model and it is based on cloud theory.
JM20 [47]	2012	This model considers the imperfect debugging in fault removal. And considers that when a failure occurs then the detected fault is assumed to be removed with probability p and it is not removed perfectly with probability q. It also assumes that new fault may be generated with probability r.
JM21[49]	2016	Discussed that the analysis of reliability of software can be done at various phases during the development of engineering software. JM and SW SRGMs are two special cases of this general SRGM.
JM22[64]	1985	This model considers that the reliability of computer software can be comprehensively viewed by adopting a Bayesian point of view and provide an alternative motivation for a commonly used model, the JM model.
JM23[65]	2017	The objective Bayesian inference was proposed to estimate parameters of JM model. Gibbs sampling is utilized to obtain the Bayesian estimators, credible intervals and coverage probabilities of the parameters.

2.3.8 NHPP Behaviour Based Models

NHPP group of models offers a way for unfolding the software failure phenomenon analytically during testing [66], [3], [51]. These models are also providing a promising tool to estimate the software reliability. The main concern in these models is to estimate mean value function of cumulative number of failures that gets experienced up to a specified point in time. The models included in this group comprises number of research publications that has been extensively surveyed by the authors.

Table 2.2 discusses major work in the field of NHPP based software reliability modeling.

Table 2.2 NHPP Based Models

Sr. no.	Author	Assumptions
M1[66]	1979	1. The failure process is analyzed to develop a suitable MVF of NHPP based model. 2. Software system is subject to failures at random times with a previous analysis.
M2[30]	1983	This model assumes error detection phenomenon to be of S shaped type.
M3[67]	1983	1. It defines mean value function in terms of censoring and removing. Censoring occurs when the test is stopped at detection of a specified number of errors or specified time.

		2. It classifies the typos of software error in to four types of the observed data.
M4[48]	1984	It extends GO Model and continuous version of Moranda Geometric Model.
M5[51]	1984	Fault content is considered different than Yamada model 1 which is imperfect debugging behaviour based.
M6[51]	1984	1. Extends GO Model but assumes fault content to be time dependent and exponential function. 2. FDR is assumed to be a constant. 3. Assumes the possibility of fault introduction, when existing errors are removed and considers the probability of finding error, proportional to remaining errors.
M7[51]	1984	Extends Ohba Model by using different fault detection rate function then inflection s shaped model. FDR is assumed constant.
M8[18]	1984	Extends GO Model using an inflection factor.
M9[51]	1984	Extends GO Model but involves number of clusters of modules.
M10[68]	1985	The effects of factors related to uncertainty are analyzed to obtain uncertainty bounds.
M11[69]	1985	Extends GO Model and Hyper exponential model.
M12[70]	1985	This model considers the effect of random coefficient autoregressive to analyze the decay in reliability. Also incorporate reused and newly developed sub-systems.
M13[71]	1985	This model used two types of errors, those which are easy and difficult to find.
M14[3]	1985	It has Similarities to GO model and considers that there is relationship between Execution and Calender time.
M15[72]	1986	Introduces two type of performance measures to accurately model reliability of distributed systems, these are related to reliability of program and whole distributed system.
M16[73]	1986	NHPP model with testing effort using Weilbull distribution due to flexibility in TE expenditure is proposed.
M17[74]	1989	It considered the complexity of errors for mathematical modeling.
M18[75]	1996	It turns out that examination of accuracy of past predictions can be used to improve future predictions by a simple recalibration procedure.
M19[76]	1990	Time dependent behaviour of effort expenditure is described using Rayleigh and exponential curve.
M20[77]	1990	NHPP model With two type of errors.
M21[31]	1992	This model is dependent on the domain of testing.
M22[78]	1992	This assumes that using assumptions based on Schneidewind model several NHPP models can be derived.
M23[79]	1992	This illustrates usefulness of connectionist based models for prediction of software reliability growth.
M24[32]	1992	Assumes that new faults get introduced sometimes when existing faults get corrected and removed and assumes that FDR is dependent on remaining and fault introduced in the system.
M25[80]	1992	Assumes that detected fault can cause detection of remaining faults in system.
M26[81]	1992	The probability density estimation of the number of software failures in the event of clustering or clumping of failure is considered for development of discrete compound Poisson (CP) prediction model.
M27[82]	1993	SRGM with imperfect debugging is discussed here, it defines a variable that represents cumulative number of faults that gets corrected up-to a specific testing time.
M28[51]	1994	This is based on GO Model but considered two type of modules.
M29[83]	1996	This is an Extension of GO Model.
M30[84]	1994	Incorporates imperfect debugging phenomenon with multiple type of failure.

M31[85]	1996	Proposed an enhanced NHPP model. It involves time varying test coverage function in its formation.
M32[51]	1997	Generalized form of the imperfect debugging fault detection model. 2.Assumes that the FDR is different among faults, new error can be introduced when error is removed.
M33[86]	1997	NHPP model is developed
M34[87]	1997	Incorporates a logistic testing-effort function in reliability assessment.
M35[88]	1997	1. Bayes inference for a NHPP with an S-shaped mean value function is proposed. 2. Two Gibbs sampling approaches are used to compute the Bayes estimates of faults remaining in the system.
M36[51]	1997	1. Fault content is assumed to be different function. 2. FDR is Inflection s-shaped curve and it is non-decreasing time dependent.
M37[89]	1998	Presents frameworks for development effort among software components to provide cost-effective system reliability goal. It uses the operational profile of the usage environment and considers the utilization matrix to link all the usage with the structure of the system.
M38[90]	1998	Proposed log-logistic growth model and can capture nature of increasing or decreasing failure occurrence rate/fault.
M39[22]	1999	It is a Generalized model based on NHPP that integrates imperfect debugging with learning phenomenon. It assumes that fault repair is always associated with a FIR due to imperfect debugging.
M40 [21]	1999	1) In this approaches for estimation of reliability are developed by indulging information from a similar project. 2) GO model is used by assuming the same value of the fault detection rate.
M41[2][51]	2000	Environmental factor effect has been added in the GO Model. Based on proportional hazard function, fault intensity rate consist of effect of the environmental factors.
M42[51] [22]	2000	It incorporates inflection s shaped factor and exponential fault introduction rate as in Yamada imperfect debugging model 1.
M43[91]	2001	This paper presents a technique by making use of time and code coverage measurements for prediction of failures in software operation.
M44[92]	2001	The model is formulated by a NHPP process and makes use of three kinds of testing-domain.
M45[93]	2002	Developed a Version programming based Model that considers FIR and FRE for Version based Programming by application of a general NHPP model into Version based system.
M46[19]	2003	1) Incorporates Fault removal probability. 2) Evolved from the GO exponential model. Fault removal efficiency model and incorporates fault removal efficiency (p), and β (t) i. e. the FIR.
M47 [51]	2003	1) Testing coverage has been used in order to define b (t) and it has been extended from GO Model. 2) Involves time based testing coverage functions and uses percentage of code coverage which has been examined during testing.
M48 [51]	2003	1) It is the specification of the generalized testing coverage and fault removal models. 2) FIR is considered as a linear time based function and incorporates testing coverage function in model development.
M49[94]	2004	New SRGM is developed by formulating the relationship between alternative testing-coverage evaluation function and the amount of detected faults.
M50[95]	2004	It proposed that different SRM can be developed just by the application of time dependent delay functions.

M51[96]	Pham /2005	1) Imperfect fault detection rate has been incorporated in the testing coverage model. 2) It Assumes imperfect FDR by combining the Fault introduction phenomenon. 3) Assumes constant rate of fault content function
M52[97]	2006	Random effect of field environment is captured .
M53[51]	2006	Effect of random field environment has been added.
M54[98]	2006	1) General framework for modeling software FD and FC process is proposed. 2) This has changed the assumption that faults are always fixed immediately on detection.
M55[99]	2007	Importance of multiple change point is incorporated into Weibull type of testing effort function.
M56[100]	2007	1) It is a Generalized discrete SRGM where time of fault occurrence follows discrete probability distribution. 2) It enables to assess reliability of software with effect of size of program.
M57[101]	2007	This incorporates, time dependencies between fault detection, fault correction and focused on parameter estimations of combined model.
M58[102]	2007	1) It is a model for the software systems developed for safety critical application under a specific testing environment. 2) Incorporates severity of errors.
M59[103]	2007	Extended NHPP model with different fault content function and FDR, A common parameter has been used to show the inter-relationship between a(t) and b(t).
M60[104]	2008	1) It investigates the problems related to imperfect debugging, in which fixing of one bug may create another. 2) Assumes that the debugging process can be described and modeled using finite and infinite server queuing system.
M61[105]	2008	It makes use of logistic function in order to describe growth of testing coverage. Fault detection is based on this test coverage function.
M62[106]	2008	Proposed model is a two-dimensional SRGM that consist of 2D time space to represent testing time and testing coverage.
M63[107]	2008	1) Assumes that testing effort is used in-order to make decision about the failure causes and to remove them. FRR is logistic function of time. 2) There are two type of faults-type 1 and type 2 depending severity levels.
M64[108]	2010	This model describes how one can incorporate Exponential-Weibull testing-effort function into inflection S-shaped SRGM based on NHPP.
M65[109]	2008	A general NHPP SRM is developed that considered quasi-renewal time-delay FR function and general mean value function is extracted using the method of steps.
M66[110]	2009	This model describes Software reliability process assuming two types of SRG factors; these are Testing Time and Testing Effort factors.
M67[111]	2009	A framework for testing effort dependent SRGM is proposed that incorporated imperfect debugging and phenomenon of error generation for different stages of fault removal process.
M68[112]	2010	Proposed method is a simple and reliable method to forecast levels of defect backlog in large software projects. This is a multivariate linear regression, expert estimations based predictions.
M69[113]	2010	Three tier client server system based on NHPP is proposed by partitioning failures and defects into three categories.
M70[114]	2010	This model is developed to capture the effect of faults generation in software due to additions at various points in time. It uniquely identifies left over faults in operational phase and during the testing phase of software development.
M71[115]	2010	To approximate reality much more closely, It incorporates concepts of testing compression factor and ratio of faults to failures in modeling software reliability.
M72[116]	2010	Defined scope of important factors in modeling reliability and also describe a

		novel approach in order to obtain a realistic estimate of system reliability.
M73[117]	2010	1) Dependent on time, behaviour of TE expenditure is described by a new modified Weibull distribution function. 2) It assumes that EDR to the amount of TE spent during testing phase is Proportional to current amount of fault content.
M74[118]	2011	Discussed how to incorporate variable fault removal function into modeling reliability of software.
M75[29]	2012	1) Reliability assessment based on discrete NHPP model is proposed. 2) A bootstrap method is applied to the regression analysis for statistical inference on software reliability assessment.
M76[25]	2012	1) Extended GO Model by considering b (t) as a time dependent function. 2) FDR is estimated by tester's ability of learning and by the number of remaining faults in software.
M77[119]	2012	It considered the joined effect of scheduled pressure and limited resources using a Cobb Douglas production function for software reliability growth.
M78[120]	2012	For model development it uses covariate information in a form of software metrics with neural network regression for estimation of failure rate in model development.
M79[121]	2012	Fuzzy time series approach is used to estimate TBF of software during phase of testing.
M80[122]	2013	1) Extended GO model with un certainty of fault detection rate in random field environment, 2) Fault detection rate follows v tub shaped curve in random field environment.
M81[123]	2013	1) Proposed SRGM failure time follows normal distribution. 2) Developed algorithm is based on an Expectation and Maximization algorithm.
M82[124]	2014	1) By setting different values for FDR, probability of fault introduction and removal can be extracted. 2) When p(t)=1 and r(t)=0, GO model is evolved, irrespective of TE.
M83[125]	2014	Represents FDP and FCP with the incorporation of TE function and method of imperfect debugging in model development.
M84[126]	2014	1) Assumes FDR function that is based on a Log-Log distribution. TC is assumed to be exposed to uncertainty of operating environment. 2) GO Model and testing coverage based model has been extended.
M85[127]	2017	To indicate defect density of software in early phases of SDLC, fuzzy logic and software metrics of early artefacts are used.
M86[124]	2014	1) Incomplete fault debugging and introduction of new faults are described. 2) Imperfect phenomenon based testing effort is developed.
M87[128]	2014	In this a combinational model is proposed using weighted arithmetic, geometric and harmonic combinations of models.
M88[129]	2014	Existing models are improved using historical project data.
M89[130]	2015	1) Assumes two types of effort functions to describe s shaped testing effort factor. 2) Delayed s shaped and inflection s shaped factors are used to implement testing effort function.
M90[131]	2015	A multi-objective and multi-stage software reliability growth based planning method in the early software development strategy is proposed.
M91[132]	2015	1) A method for estimating the reliability growth of complex continuous operating system is developed. 2) A framework via posterior distribution on system failure intensity is developed.
M92[133]	2015	1) GO and delayed s shaped model has been extended for testing failures in multi-release software. 2) Failure phenomenon of software is investigated by considering delay in failure time of the software.

M93[134]	2015	A bivariate software reliability growth model considering effect of uncertainty of change of failure-occurrence phenomenon at a change-point is developed by making use of Testing time and testing effort.
M94[135]	2016	1) New S shaped and concave model extended earlier dependent parameter model. 2) Model considers dependency of fault detection, imperfect fault removal and a parameter called maximum amount of faults in software.
M95[136]	2016	Incorporates uncertainty of FDR per unit of time in an operating environment.
M96[137]	2016	1) A novel NHPP model is proposed to enumerate the uncertainties related with perfect or imperfect debugging process is developed. 2) It represents the environment based uncertainties as a noise of arbitrary correlation.
M97[138]	2016	Assumes a Dynamic weighted behaviour for the model development and combines multiple models.
M98[139]	2017	1) Assumes that software debugging can be affected by many factors, such as subjective and objective influences, the difficulty and complexity of fault removal, the dependent relationships among faults, the changes in different phases of software testing and the test schedules. 2) The rate of fault introduction is not assumed as a constant, but is an irregularly fluctuating variable in software debugging.
M99[140]	2017	1) Model is based on environmental uncertainties and dynamics. These uncertainties are like non-predictable changes in requirements and number of team members. 2) It predicts several development situations that involve random factors like team skills and development environment.
M100[141]	2017	This model is based on the multiple objectives and considers time, cost and reliability of subsystems in each stage of software development.
M101[142]	2018	1) It incorporates effect of up-gradations on successive releases of software. 2) It also assumes that faults in software are of different type like soft and hard faults on the basis of effort and time consumed in removal of faults.
M102[143]	2018	Assumed as a special case of famous inflection S-shaped model and generalized GO model. A special attention is given to non-existence issues of MLE.
M103[144]	2018	A variable η is considered as a random variable to represent uncertainty of FDR in the operating environment.
M104[145]	2019	Fault Removal process for multi-release OSS systems is assumed by considering the concept of change point.
M105[146]	2018	Assumes that all of issues that are fixed in a current release of the software are deciding the next release of software, uncertainty issues are quantified by the entropy based measures.
M106[147]	2019	Complexity issues like knowledge of debugging process, coverage factor and delay time function in distributed computing environment is concerned.

2.4 Model Evolution

On the basis of literature survey given in previous section, it is analysed by authors that there is a relationship between various groups of models. Keeping in view, the notion of how one model is

evolved from other group of models, authors make a tabular description of assumptions used by various models for both group of failure rate models (Table 2.1) and NHPP based models (

Table 2.2). These are the majorly used categories of software reliability models that are extensively used in industries and academia. On the basis of specific assumptions, models are examined to see what are the specific attributes on which models are proposed and in which category of the attributes they belongs.

2.4.1 Attributes used to Categorize Software Reliability Growth Models

All models that have been surveyed by the authors are classified in terms of fifteen specific criteria. All the developed models in some or other ways belongs to these major attribute based criteria. These criteria are used by the authors to distinguish models in different categories. The key attribute based criteria are specified as follows:

- Failure rate distribution of model (Exponential and s-shaped model)
- Parametric modification based model
- Debugging behaviour of a model
- Involvement of Modular structure in model development
- Incorporation of Testing Effort in modeling
- Incorporation of Testing Coverage in model development
- Involvement of uncertain software development environment
- Bayesian Process based model
- Involvement of multi release software development environment
- Generalized model formulation
- Model evolved on the basis of types of errors
- Consideration of error complexity in modeling
- Involvement of environment factors in model development
- Incorporation of some new technology in model development
- Models evolved by combination of some models.

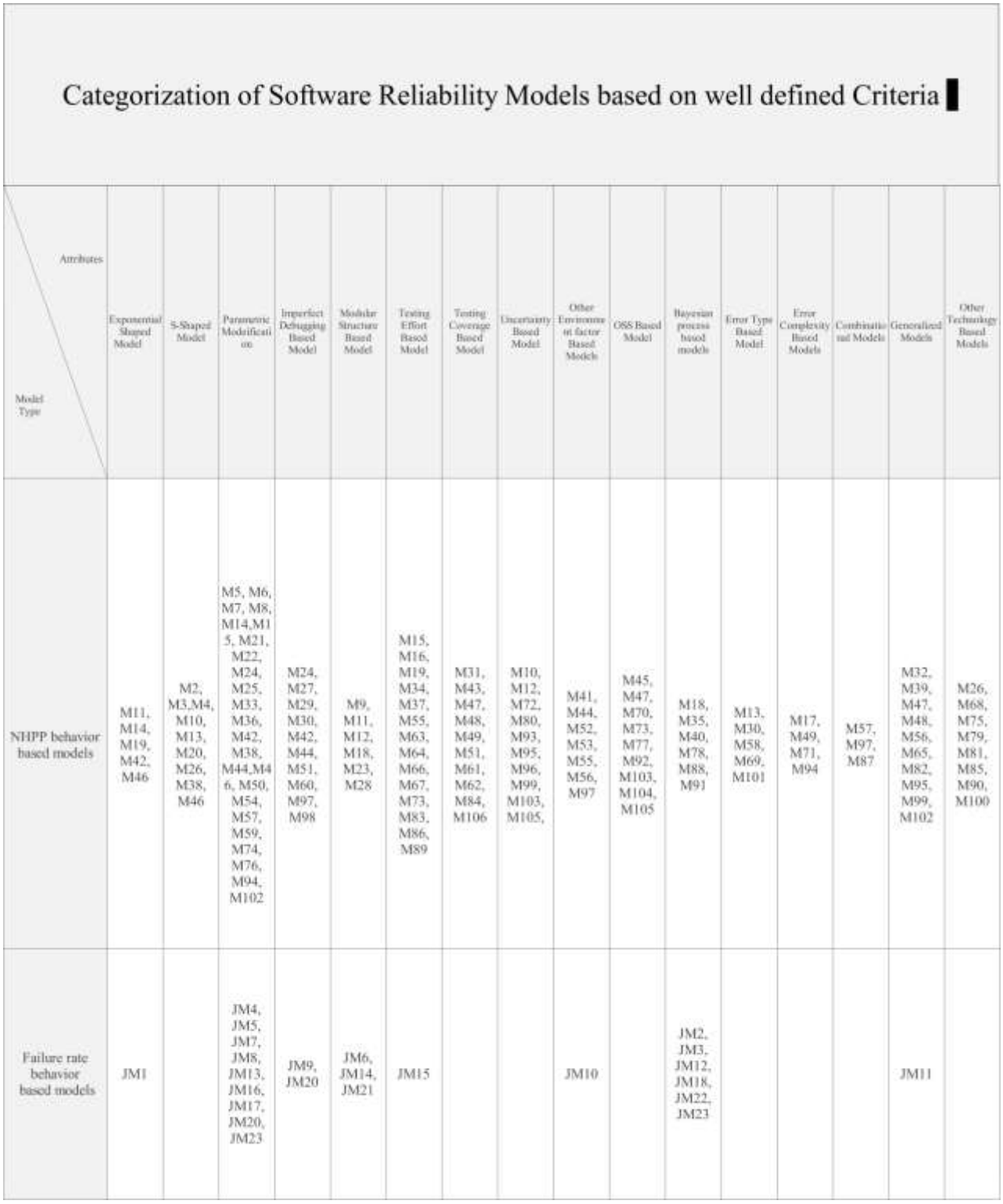

Fig. 2.3 Categorization of SRGMS based on Well-Defined Criteria

On the basis of survey done in the Table 2.1 and

Table 2.2 for SRGMs, a new classification is defined by the authors in Fig. 2.3. This classification on the basis of attribute based criteria involves either a new approach for model development or

models are enhanced using new parameters which have not been included in the development of earlier models. Only on the basis of these criteria, researchers have developed models and further these are providing a new way to the researchers to enhance field of modeling software reliability. Further, this classification illustrates how one model is evolved from other model based on their specific assumptions. Using in-depth systematic survey, an evolution diagram for the existing models in history is made by the authors in this thesis work and is shown in Fig. 2.4. Each of the models is evolved on the basis of their specific assumptions that underlies in a specific class of attribute criterion.

From the survey of software reliability models done in this chapter it is found in practice that SRGM encounters major challenges. First, a software developer rarely uses operational profile for testing of the software, so what has been observed may not be applicable for operational usage. Second, failure data published by the software companies to be used in reliability modeling is very much limited; in such a case it is hard to make statistical meaning to software reliability prediction. Third, all assumptions made in SRGM development are not realistic. Nevertheless, these difficulties can be overcome with suitable measures by making more realistic assumptions in specific software development environment.

In spite of all the difficulties in model development, there are numerous software reliability models that performs significantly well in industries and academia. There is a need to uncover all the major challenges in methods of software reliability model development so that accurate estimation of software's reliability could be done in order to avoid any kind of harmful losses in social or business environment.

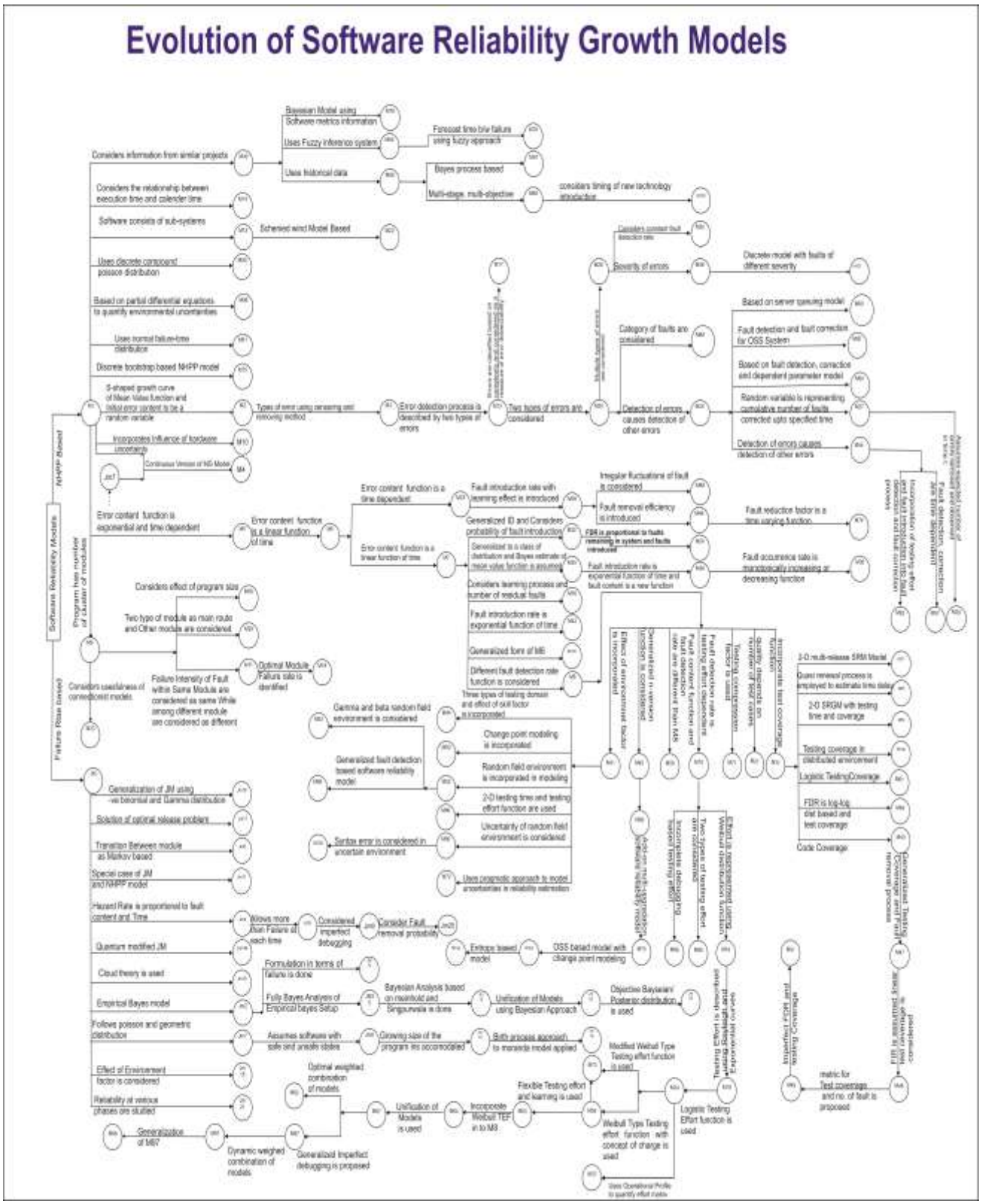

Fig. 2.4 Evolution of Software Reliability Growth Models

2.5 Parameter Estimation Technique

Parameter estimation problems for a non-linear system are represented with a function optimization problem. There is a requirement to find value of parameters which can be fitted to a

function that is to be optimized. In software reliability, MLE and LSE techniques [51] are used heavily as general technique of parameter estimation. MLE states that anticipated probability distribution is one that will make observed data "most likely" means that there is a need to have set of parameter values that will maximize likelihood function value. LSE technique requires finding parameter values that minimizes sum of squared errors. In both the techniques, it is impossible to find analytic form solution, particularly when model involves several parameters and its probability density function is in highly non-linear form. Both MLE and LSE methods require use of nonlinear algorithms of optimization. An elementary idea for non-linear optimization is to rapidly find optimal parameters that maximizes likelihood function or minimizes the sum of squared errors.

Traditional techniques have been found in use for estimation of parameters of software reliability models [4]. Fatefully, all model parameters are normally having non-linear relationships and because of this, traditional techniques for optimization of parameters suffer various problems in finding optimum value of models to better predict software reliability.

2.5.1 Meta-heuristic Algorithms in Various Application Domains

In recent years meta-heuristic algorithms have become very much popular due to their simplicity, flexibility, their derivative free nature and capability of avoiding the local optima problem [7]. These algorithms explore the feasible solution space using some specified rules. Number of nature inspired algorithms has been developed in the literature which can be applied in various domains for the solution of numerical optimization based problems [10], [148]–[151]. All nature inspired algorithms are assembled into four groups and comprises the algorithms as below:

1. Natural Evolutionary Principal Based Algorithm
2. Swarm Intelligence Behaviour Based Algorithms
3. Physics Phenomenon Based Algorithms
4. Human Intelligence Behaviour Based Algorithm

1). Evolutionary principal based algorithms are inspired from the nature's evolution process. All these are based on the principal of survival of fittest in order to form the next generation individual. Algorithms in this group are like:

- Genetic Algorithm [149]
- Genetic Programming [152]
- Differential Evolution [10]
- Biogeography Based Optimization [153] etc.

2). Physics phenomenon based algorithms are grounded on the rule of physics. There are so-many well-known optimizations algorithms like:

- Simulated Annealing [154]
- Galaxy Based Search Optimizations [155]
- Gravitational Search Algorithm [156]
- Black Hole [157]
- Charged System Search [158] etc.

3) Swarm Intelligence behaviour based algorithms follows the social activities of the flocks of animals, birds and amphibians etc. This is the widest group of nature inspired algorithms and includes algorithms as:

- Particle Swarm Optimization [148]
- Bat Algorithm [159]–[161]
- Artificial Bee Colony [162]
- Ant Colony Optimization [163]
- Dolphin Echolocation [164]
- Honey Bee Marriage [151], [165]
- Artificial Fish Swarm Optimization [166]
- Termite Algorithm [167]
- Wasp Swarm Algorithm [168]
- Monkey Search Algorithm [169]
- Wolf Pack Search Algorithm [165]
- Bee Collecting Pollen Based Algorithm [170]
- Cuckoo Search Algorithm [171]
- Dolphin Optimization [172]
- Firefly Algorithm [173]
- Fruit Fly Optimization [174]
- Whale Optimization Algorithm [175]
- Grey Wolf Optimization Algorithm [176], [177]

4) Human inspired meta-heuristic algorithms are stirred from the intelligent behaviour of human beings and include algorithms like:

- Teaching Learning Algorithm For Optimization [178]
- Tabu Search Algorithm For Problem Optimizations [179] etc.

Numbers of algorithms among these groups of algorithms are applicable in various domains and these have been found significant in their specific domains. For example, Dandy et al. [180] in 2015 proposed an improved GA for the optimization of water distribution problem. Ismail et al. [181] in 2014 applied GA for the design of renewable energy system. Kumar [182] in 2014 applied GA and MAUT in dynamic faults and release optimization based problems. Hsu et al. [183] in 2010 discussed the software reliability models parameter estimation with the applicability of modified GA. As compared to GA where chances of selection to become a parent is dependent only on fitness values of the solution, in Differential Evolution algorithm each solution is having an equivalent opportunity of being nominated as parent and has been applied in areas like design of filters with requirement of magnitude and group delays, in image segmentation etc. An improved version of DE algorithm generating random number by disordered change has been applied for the safety problem of power system described in Li and Chen [184]. DE algorithm after a few runs may generate overlapping individuals and makes it an inefficient algorithm for global search.

Jin and Jin [185]in 2016 applied improved swarm intelligence approach for parameter optimization. Mahdavi et al. [186] proposed a novel discrete PSO algorithm in 2010 and Astuty and Haryono [187] in 2016 proposed binary PSO algorithm in power system transmission in an optimized way. Malhotra and Negi [188] in 2013 applied PSO for reliability modeling. Sheta [189] in 2007 applied PSO for parameter estimation of software reliability models. Algorithms like ABC by Yang [6], [7], [190] have been well applied in various field. Basturk and karaboga [191] published a research on ABC algorithm's performance in 2008. Ozturk and Karaboga [162] proposed a new clustering approach in 2010 using ABC algorithm. Zhu [192] proposed a new globally best directed ABC algorithm in 2010. Karaboga [193] proposed ABC algorithm to solve problems based on constrained optimization in 2007. ABC algorithm and its variants for constrained optimization have also been discussed by Akay and Karaboga [190] in 2017. The food information in ABC is dispersed among the whole bees, this algorithm achieves well in global search optimization i.e. the exploration is upright in finding the better solution. Dynamic frequency based parallel K-bat algorithm [160] is developed to estimate its efficiency in big data environment.

Hybrid algorithm have also been applied for the solution of optimization problem like Mirjalili [194] developed a hybrid PSOGSA algorithm by hybridization of PSO and PSOGSA for mathematical function optimization in 2010. Mirjalili [195] proposed a binary optimization algorithm using hybrid PSOGSA algorithms in 2014. Liuand Zhoua [196] applied improved QPSO (Quantum Particle Swarm Optimization) algorithm in high dimensional complex problems in 2014. Abraham [197] proposed a hybrid differential and ABC algorithm in 2012. This algorithm applied DE after normal process of ABC for better selection of candidate solution. Li et al. [198] proposed

ABC assisted DE for ORPF (Optimal Reactive Power Flow) in 2013. They have used ABC in the DE algorithm in-order to recover DE shortcoming of large population requirement for avoiding premature convergence. Tiwari et al. [199] proposed hybrid ABC algorithm with DE in 2017 and applied it for optimization of welded beam design problems. This algorithm has provided modification to position update equation in employee bee phase and they applied DE for onlooker bee phase position update.

2.5.2 Meta-heuristic Algorithms in Ground of Software Reliability

In modeling reliability of software, parameter estimation process is having an important role. In the last decade parameter optimization by meta-heuristic algorithm is a major attention in the field of software reliability. Xiao-Li Meng in 1993 proposed MLE via ECM algorithm and also provide a general framework for this [200].

Takashi Minohra, Yoshihiro Tohma [201] applied genetic algorithm for parameter estimation of hyper geometric distribution based SRGMs and they discussed that genetic algorithm is more significant in parameter estimation and it also removes constraints from software reliability model parameters.

Hiroyuki Okamura [202] discussed an expectation and maximization principal (EM) based iterative method for maximum likelihood estimation of software reliability model parameters and then compare EM based method with traditional Newton and Fisher's Scoring methods. In Jae Myung [203] proposed MLE for statistical estimation of parameters and discussed that the LSE is mainly a descriptive tool, MLE is a favoured method of parameter estimation and is an crucial tool for various statistical modeling techniques, Specifically in non-linear modeling having non normal data.

Ohishi, Okamura, Dohi [204] proposed Gompertz model for software reliability estimation and estimate model parameters using convergence property based EM algorithm and discussed effectiveness of EM algorithm by comparing in terms of accuracy and security with Newton method. M. Casertaa, A. Márquez Uribeb [205] applied Tabu-Search-based algorithm for parameter estimation of software reliability models. It emphasized on the memory based mechanism to balance intensification and diversification with the help of short and long-term memory. Sultan and Mohammed E. El-Telbany [206] provide a method for software reliability estimation using multi-objective genetic algorithm. Evidential reasoning algorithm is used by Hu, Si and Yang [207] for software reliability prediction based model that is grounded on nonlinear optimization and provide the effectiveness of it by comparing it with several existing methods in

terms of prediction accuracy or speed. Hsu et al. discussed the software reliability models parameter estimation with the applicability of modified GA [208].

A parameter estimation method grounded on the ACO Algorithm is proposed by Zhenga, Liua, Huanga and Yaoa [209] and they also discussed that in comparison to traditional methods and PSO, ACO is about ten times more accurate.

Wason [210] proposed new method for parameter estimation using finite automata to model software reliability that is indirectly significant over the traditional models by many factors, most prominently due to a reason that a software system during execution is a finite state machine. Latha Shanmugam and Dr. Lilly Florence [211] proposed an estimation method created on ACO algorithm and compared the results with PSO algorithm with higher accuracy.

Dr. Najla Akram, AL-Saati and Marwa Abd-AlKareem [212] applied swarm intelligence based cuckoo-search algorithm for parameter estimation of SRGMs and proved its efficiency and effectiveness by comparing it with the PSO and ACO algorithms.

K.Mallikharjuna and Kodali [213] proposed a method for parameter estimation of SRGMs using ABC algorithm. TaehyounKima, KwangkyuLeeb, JongmoonBaik [214] proposed an effective approach for parameter estimation of software reliability using real valued genetic algorithm and then compared the results of proposed approach with the existing GA and other traditional methods. Wei Zhaoa, Tao Taoa, Enrico Zio [215] discussed prediction of reliability with support vector machine using combination of diagnostic selection and GA algorithm. In combination these methods allows utilization of prior knowledge for guiding GA process Inorder to avoid divergence, local optima, and to accelerate convergence. Pratik Roy, Mahapatra and Dey [216] proposed a Neuro-Genetic method and then applied it on logistic phenomenon based software reliability prediction and provide clues for the effectiveness of the proposed approach than other artificial neural network based methods.

Efficient method based on gravitational search algorithm is applied by Ankur Choudhary, Anurag Singh Baghel, Om Prakash Sangwan [217] for parameter estimation of SRGM. They also provide the efficacy of the method by comparing it with previous numerical estimation techniques, genetic algorithm and cuckoo search methods.

Harmony search is used by Ankur Choudhary, Anurag Singh Baghel and Om Prakash Sangwan [218] for parameter estimation of SRGMs. Authors tested their proposed approach against Cuckoo search and traditional numerical methods considering MSE and TS as a measure of quality. They also applied firefly optimization algorithm for parameter estimation of SRGMs [219].

2.6 Performance Measurement Methods

Quantitative techniques are required to access how accurate are the SRGMs for measuring predictions about reliability of a software. Ability of a model can be judged by the way how it replicates the perceived behavior of software and how it make predictions about the future behavior of the software by using observed failure data. In modeling software reliability, main concerned is to predict future behavior of the software. To explore the efficiency of SRGMs, here few of accuracy estimation criteria are discussed that may be used to compare model's accuracy quantitatively. There are different criteria that are utilized by various researchers for software reliability measurement and to check for accuracy estimation or comparison of software reliability models. For example Teng et al. [220] proposed Mean Squared Error, Short Term Relative Error and Mean Square Prediction Error measure etc. to represent a deviation between the predicted values and actual values. These methods provide a good measure of difference between the actual and estimated values. Li and Malaiya [221] proposed Mean Relative Error in order to access accuracy of models. These measures show the quantitative comparisons for short term predictions. Further various researchers employed some other methods of accuracy estimation as given in Table 2.3.

Table 2.3 Performance Measurement Metrics[13]

Accuracy Estimation Method	Definition	Measurement
"Mean Square Error(MSE)"	"Measures the deviation between the predicted values with the actual observations"	$MSE = \frac{\sum_{i=1}^{k} (m_i - \hat{m}(t_i))^2}{k - p}$
"Mean Absolute Error(MAE)"	"It is similar to MSE but the way of measurement is using the absolute values"	$MAE = \frac{\sum_{i=1}^{k} \lvert m_i - \hat{m}(t_i) \rvert}{k - p}$
"Sum of Squared Error(SSE)"	"It is the sum of the squared distances between the observed value and the actual values"	$SSE = \sum_{i=1}^{k} \lvert m_i - \hat{m}(t_i) \rvert^2$
"Bias"	"It is the sum of the difference between the estimated curve and the actual data"	$\text{Bias} = \frac{\sum_{i=1}^{k} (m_i - \hat{m}(t_i))}{k}$

"Mean error of Prediction(MEOP)"	"It sums the absolute value of the deviation between the actual data and estimated curve"	$MEOP = \frac{\sum_{i=1}^{k} \lvert m_i - \hat{m}(t_i) \rvert}{k - p + 1}$
"Accuracy of Estimation(AE)"	"It reflects the difference between the estimated number of all errors with the actual number of all detected errors"	$\text{AE} = \lvert \frac{m_a - a}{m_a} \rvert$ M_a and a is actual and estimated number of detected errors respectively
"Predictive Ratio Risk(PRR)"	"It observes the distance of model estimates from the actual data against the model estimate"	$\text{PRR} = \sum_{i=1}^{k} \frac{(\hat{m}(t_i) - m_i)}{(\hat{m}(t_i)}$
"Root Mean Square prediction error (RMSPE)"	"It is the measure of closeness with which the model predicts the observation"	$\text{RMSPE} = \sqrt{\frac{1}{k-1} \sum_{i=1}^{k} (m_i - \hat{m}(t_i) - Bias)^2 + Bias^2}$
"Theil Statistics (TS)"	"It is the average deviation percentage over all periods with regards to the actual values. The closer it is to zero better is the prediction capability of a model"	$\text{TS} = \sqrt{\frac{\sum_{i=1}^{k} (\hat{m}(t_i) - m_i)^2}{\sum_{1}^{k} (m_i)^2}} * 100\%$

2.7 Inference on the basis of Literature Survey

1. More than 300 models have been developed till today, but all these models cannot be recommended to potential users, because of their sheer complexity in implementing them in latest software development environment to gain benefit of improving software reliability.
2. Failure rate behaviour based models are the earliest group of models used to analyze failure rate per fault of the program. These models provides a way to examine how the failure rate changes at the time of failure in an interval of time and found to be utilized in academia and industry efficiently.
3. NHPP based models are among the most promising group of models for assessing the reliability of software. These models can observe the reliability growth of the software at the time of testing in software development process and can observe software failure phenomenon analytically during the testing phase.
4. Available research in software reliability model development is made for reliability estimation of software's developed under traditional software development lifecycle process.

5. Accurate parameter values of software reliability models are difficult to derive, because most of the software reliability models lack experimental datasets. Research publications are screening no more than 31% experimental researches and among these only 13% are purely experimental. This low number is due to reasons because public experimental data sets in software reliability estimation are very narrow and producing reliability data through experimentation usually require long time cluster. Lack of the experimental datasets has been considered as a major block in the success of software reliability model development.
6. The estimation of parameter values in model development is the primary difficulty in software reliability model development. Future prediction of the software reliability depends on the estimated parameter values. Optimum parameter valves for accurate reliability predictions involve usage of traditional methods of parameter estimation like least square estimation and maximum likelihood estimation methods.
7. Recommendations on how to use LSE and MLE to obtain accurate model parameter is largely missing in the literature. This involves not only what algorithm should be used to find accurate parameter values but also includes what parameter values should be used to start the search process.
8. To estimate the unknown parameter values of specific software reliability models, there is a need to optimize a function that may be for maximum likelihood function or for minimization of least square estimation function. There is no difference in maximization or minimization problems because values of parameter x which maximizes the function f(x) and minimizes the function f(x).
9. Existing models are grounded on the traditional software development environments. Software reliability model development must have concern about the software development methodologies used. Development environmental of the software plays a major role in reliability estimation of the software in field usage of the software.
10. In spite of the software development methodologies, there is a need to consider the effect of early stages in the software development. These may include the requirement specification for the software to be reliable or the planning of the resources that are required to fulfill the software reliability requirements.
11. Testing resource allocation in every phase of software development in a scheduled way is required to have accurate software reliability estimation. There is a need to properly plan and schedule the testing resources so that they can more accurately estimate the system quality in terms of the reliability. If the resources are having the proper allocation to the software modules only then they can perform the task allocated to them properly and within the specified time and conditions.

12. More effective factors are required to be considered for the estimation of reliability in open source software development.
13. In the last decades it is found that more than hundreds of meta-heuristic algorithms have been proposed across various domains, however there does not exists any meta-heuristic algorithm that can handle all type of projects failure data for software reliability estimation.
14. These algorithms show promising results only on specific problem set and may not show good performance on other sets of problems. No free lunch theorem is saying that there is not any meta-heuristic algorithm that is best suited for the solution of all optimization problems.
15. No free lunch theorem really has made this field a continuously growing field from the last decade and has motivated authors to propose a new optimizer for parameter optimization of software reliability models.
16. Literature perceived that hybrid algorithms are outperforming their parent algorithms. Unfortunately, no hybrid algorithm of the meta-heuristic approach has been studied for parameter estimation of software reliability models.
17. Finding optimum parameters is essentially a heuristic process in which the optimization algorithm tries to improve. Depending upon the choice of the initial parameter values, the algorithm could prematurely stop and return a sub-optimal set of parameter values. Unfortunately there exists no general solution to the local maximum problem. Instead, a variety of techniques have been developed in an attempt to avoid the problem of local maxima, though there is no guarantee of their effectiveness.

Chapter 3 ITERATIVE SOFTWARE FAILURE RATE MODEL

Failure rate-based models are among the earliest software reliability estimation models used in industries and academia. These models are grounded on Jelinski-Moranda software reliability model [1]. They need further enhancement, so that more realistic assumptions like imperfect debugging and factors for exact reliability growth estimation can be incorporated into the model. Software reliability is primarily dependent on the process of software development. Existing failure rate models cannot be applied to the current software development methodology. The software developed using latest SDLC approach can capture and implement all the user requirements within time and budget. Keeping new SDLC processes and technologies, a new failure rate model centered on iterative SDLC process has been proposed. The proposed model has been derived from the general class of failure rate behavior-based models and exploits iterative behaviour of software development process. All latest iterative SDLC processes can be used to predict the reliability by applying the proposed failure rate model. The proposed model takes care of complexity and paradigm shift of iterative based software development process by introducing modulation factor.

3.1 Introduction

With the growing advances in digital world, software development demand from industries is growing at an exponential. Due to enormous demand and lack of time and budget, software companies are not able to develop fault free software. Latest tools and techniques has been applied for development of defect free software but still, it is not possible for software developers to develop defect free software practically [222]. Software must go through exhaustive testing and debugging, which requires time and money to enhance the reliability. Occurrence of fault is inevitable in the current demand of software. There should have some means to avoid software failures so that harmful losses whether related to life or any other field could be evaded.

If we could measure the reliability of software under development, better we can predict whether the software would be operational in future or not. Early estimation of reliability provides information to the managers like what should be the release time of the software and amount of man-hour consumption etc. while developing any software [223]. Software reliability models are one of the ways to simulate software reliability estimation curve to predict reliability of the system under study. Numerous reliability estimation models for software have been developed and all are working on specific applications, specific environments, datasets and assumptions made by them. But still, there is a need to develop new software reliability models.

Need lies in the fact that, among the available research in software reliability model development, all developed models are basically made for reliability estimation of software's

developed under traditional waterfall SDLC process [51]. Further research publications are screening no more than 31% experimental researches and among them only 13% are purely experimental [17]. This low number is due to reason that public experimental data sets in software reliability estimation are very narrow and producing reliability data through experimentation usually require long time cluster. Most generally developed models are only NHPP based and very few are belonging to failure rate based models [20], [21], [109], [122], [144], [146], [224], [225].

Failure rate based models are among the earliest software reliability estimation models. These models are grounded on Jelinski-Moranda software reliability model [44] and needs further enhancement in order to incorporate more realistic assumptions like imperfect debugging and factors for exact reliability growth estimation. Further, all software reliability models are developed under waterfall SDLC process. Waterfall SDLC process for development of software assumes that requirements from the end users are stable and it delivers whole software at one shot in the end [226], [227]. This may generate risks for the users as they do not have information till end what they will get. These limitations construct a need for the use of another methodology while modeling software reliability. Yet now, no research in software reliability assessment has been found which are based on latest SDLC process.

Software reliability is primarily dependent on the process of software development. The software developed using latest SDLC approach can capture and implement all the user requirements with in time and budget. Earlier waterfall SDLC process based reliability estimation models are very much stable and found to be unsuitable for the software industries because there is a high risk and uncertainty of change in requirements. An iterative SDLC process is latest methodology of software development and is a practical method of step-wise top-down refinement approach to the software development that replaces waterfall SDLC process [33], [228]. It gathers user requirements in each of iteration of software development and implements software with less uncertainty in risk and user satisfaction. Nowadays latest software development processes are based on iterative software development process and these are like Rational Unified Process (RUP) [229], Adaptive method [230], [231] [232] [231], XP [233], Spiral model [234] and AZ [235] development processes. These SDLC processes are considered as an aid of producing reliable software based on iterative model in development. Moreover, these are found to be used practically in industries and academia for software development and promote iterative development process.

Proposed failure rate model in this chapter incorporates iterative SDLC process by replacing earlier waterfall development software development life cycle process. It assumes imperfect debugging during each of the iteration. There is always a possibility of fault introduction with feature addition in each of the iteration of software development. All latest iterative SDLC

processes can be used to predict the reliability by applying proposed failure rate model. Existing failure rate model cannot be applied on the latest software development methodologies. The proposed model take cares of complexity and paradigm shift of iterative based software development process by introducing modulation factor.

3.2 Iterative Software Development Life Cycle Process

In iterative process of software development, entire software is built and delivered to the end user in iterations. The process starts with simple implementation of key samples of a problem and iteratively enhances existing software releases until full system is implemented [228]. At each of iteration release feedback from the iteration is available for next iteration [236]–[238]. Feedback is mainly about the functionality and user aspect of the software. At each released stages of software, not only extensions but design modifications may be made. Each of the iteration makes step wise refinements in an effective way to converge to the full implementation of a problem. Main focus of feedback analysis is to find the amount of defects in an iteration that gets injected in the upcoming iteration [239]. Using feedback in each of iteration, analysis of changing needs in an up-coming iteration can be made. These changing needs in each iteration are essential to know because these tells information about how much more defects may occur and how much more effort and functionality is needed in making extensions and design modifications to the released iteration.

3.2.1 Reliability in Iterative Software Development Environment

It is well known that development methodology is one of the major factors that affect not only the software reliability but also other software quality attributes [227]. It is impossible to perfectly catch up all the functional requirements before development. Iterative method of software development and its enhancements are becoming major development process in software industries and these methods allow a better deal in making parallel software development and testing. This is the only reason for the failure of waterfall development process in perfectly identifying quality of complex software systems.

There is a need to be flexible in handling user's changing requirements in order to deliver reliable software in the current digital market. This will provide a remarkable improvement in handling reliability growth requirement at each of the iteration of software development process.

There is a need to introduce a factor that will engulf all the changing needs depending on the defect analysis in each of iteration. The varying needs in each of iteration of software development include bug reports, additional functionalities, and testing effort required to find the amount of defects that gets injected / removed in each of the upcoming iteration. These changing

needs are incorporated in proposed failure rate model so that precise reliability growth of the system could be estimated.

3.3 Proposed Model

3.3.1 Analytical Software Failure Rate Model for SDLC

Keeping new SDLC processes and technologies, a new failure rate model centered on iterative SDLC process has been proposed. The proposed model has been derived from the general class of failure rate behavior-based models and exploits iterative behaviour of software development process.

In iterative process of software development, entire software is built and delivered to the end user in iterations [240]. The process starts with simple implementation of key samples of a problem and iteratively enhances existing software releases until full system is implemented. At each of iteration release feedback from the iteration is available for next iteration. Feedback is mainly about the functionality and user aspect of the software. At each released stages of software, not only extensions but design modifications may be made. Each of the iteration makes step wise refinements in an effective way to converge to the full implementation of a problem. Main focus of feedback analysis is to find the amount of defects in an iteration that gets injected in the upcoming iteration. Using feedback in each iteration, analysis of changing needs in an up-coming iteration can be made. These changing needs in each iteration are essential to know because these tells information about how much more defects may occur and how much more effort and functionality is needed in making extensions and design modifications to the released iteration.

In modeling software reliability, there is a need to introduce a factor that will engulf all the changing needs depending on the defect analysis in each of iteration. The varying needs in each of iteration of software development include bug reports, additional functionalities, and testing effort required to find the amount of defects that gets injected / removed in each of upcoming iteration. These changing needs are incorporated in proposed failure rate model using modulation factor γ given in (1), so that precise reliability growth of the system could be estimated. The proposed model assumes that fault removal process is imperfect. Due to imperfect debugging regenerated faults are induced in successive iterations.

3.3.2 Proposed Model Assumptions

Proposed model is developed by extending assumptions of failure rate based models. Assumptions on which proposed model has been grounded are given below-

i. Initial software fault is unknown and constant in iteration.

ii. Each fault in iteration is independent and it may be equally likely to cause a failure while testing.

iii. The interval of time between fault occurrences in each of the iteration is independent and follows an exponential distribution.

iv. Software failure rate remains constant over the intervals between fault occurrences.

v. The software failure rate is proportional to number of faults that remains in a software and modulation factor γ.

vi. In each of the iteration the detected fault is removed with a probability p, not removed perfectly with a probability q and new fault may be introduced with a probability r. Here $p+q+r=1$ and the probability $p > r$.

Faults are injected from previous iterations along with newly introduced faults in the current iteration. Newly induced faults are caused by added and modified functionalities in a respective iteration of software development. Depending on the number of initial iterative faults, there is a need to modify the amount of resources allocated for debugging in each of iteration. The modulation factor γ reflects the modified needs that integrate iterative development processes in software failure rate models. This factor changes its value according to the Modulation parameter μ as shown in Eq. (1). Moreover, changing needs in each of the iteration is different and vary according to Eq. (1).

$$\gamma = \mu + (1-\mu)/\mu, \quad (0 < \mu \leq 1.0) \qquad (1)$$

Here, μ is the modulation parameter that represents newly added functionality and user acceptance in the current iteration. Its value is almost 0 at the beginning and becomes 1.0 in the final iteration of iterative software development process. Modulation parameter μ takes its value from 0 to 1.0 by assuming that with growing number of iterations, level of user acceptance increases from lower to higher. It shows abrupt changes in its value in initial iterations due to preliminary design changes, testing effort, and user acceptance. When it's values reaches near to one then software under development is assumed to be reliable enough and has achieved all the required functionalities to fulfill the end user needs.

3.3.3 Model Formulation

Failure rate function ($\lambda(t_i)$) with imperfect debugging is modeled in Eq. (2).

$$\lambda(t_i) = \phi[\mathrm{N} - \frac{(n_{i-1})(\gamma)}{(i-1)}(p - r)], \quad i\text{-}1, 2....N \qquad (2)$$

Where,

γ Modulation factor for representing changing needs in each of the iteration of software development

n_{i-1} Cumulative number of failures at $(i-1)^{th}$ failure interval

N Initial number of faults in software

ϕ Proportionality constant

Cumulative Density Function $F(t_i)$ and Reliability Function $R(t_i)$ is calculated in Eq. (3) and Eq. (4)

$$F(t_i) = 1 - e^{[-\phi[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)]t_i]} \quad (3)$$

$$R(t_i) = e^{[-\phi[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)]t_i]} \quad (4)$$

When $p = 1, r = 0$ and γ varies as in Eq. (5), proposed model behaves as JM model[44].

$$1, \frac{2i}{i+1}, \frac{3i}{i+4}, \frac{4i}{i+8}, \frac{5i}{i+13}.... \quad (5)$$

Following a variation of γ in Eq. (5) and considering p being the probability of fault removal and r as the fault introduction probability then model behaves as the GS Mahapatra et al. model[47] .

3.3.4 Parameter Estimation

In the proposed model, there are three unknown parameters N, n , and ϕ these parameters are estimated at different values of the γ . MLE has been used to estimate the values of the parameters. Parameter estimation by MLE method requires solutions of complex equations by maximizing the likelihood of model parameters. Probability density function $f(t_i)$ for the proposed model is given in Eq. (6).

$$f(t_i) = \phi[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)] \cdot e^{[-\phi[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)]t_i]} \quad (6)$$

The likelihood function $L(N)$ is calculated in Eq. (7) using Eq. (6).

$$L(N) = \prod f(t_i)$$

$$= \prod_{i=1}^{n} [\phi[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)] e^{[-\phi \sum_{i=1}^{n} [N - \frac{(n_{i-1})\gamma}{i-1}(p-r)]t_i]} \quad (7)$$

Taking log of $L(N)$, LLF is calculated in Eq. (8).

$$LLF = \ln L(N) = n \ln \phi + \sum_{i=1}^{n} \ln[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)]$$
$$- \sum_{i=1}^{n} \phi[N - \frac{(n_{i-1})\gamma}{i-1}(p-r)]\, t_i \qquad (8)$$

Solution using log likelihood function for parameter estimation involves calculation of its partial derivatives with respect to N, n and ϕ respectively and then equating them to value zero. MLE of parameters are calculated from Eq. (9), (10) and (11).

$$\frac{n}{\phi} = \sum_{i=1}^{n} [N - \frac{(n_{i-1})\gamma}{i-1}(p-r)]\, t_i \qquad (9)$$

$$n = \phi \sum_{i=1}^{n} [N - \frac{(n_{i-1})\gamma}{i-1}(p-r)]\, t_i \qquad (10)$$

$$\frac{1}{\sum_{i=1}^{n} [N - \frac{(n_{i-1})\gamma}{i-1}(p-r)]} = \phi \sum_{i=1}^{n} t_i \qquad (11)$$

3.4 Application Datasets used for Experimentation

Suitability of the proposed model has been tested using Tera Promise repository bug report files of Eclipse (DS1) and JDT (DS2) open source software at https://zenodo.org/record/268486#.W-QsPpMzY2w. These data sets have been given by An Ngoc Lam. The bug reports contains data like: bug_id, summary, description, report time, report time status, commit, and commit time files. Datasets have been extracted from these bug reports and reformatted in time domain format. Data given in DS1 includes eight minor releases and four major releases starting from the year 2001 to the year 2013. DS2 includes 3 major releases and 6 minor releases starting from the year 2002 to the year 2014.

3.5 Experimental setup

Model is evaluated in light of analyzing its significance as compared to other failure rate models. Model parameters include N, n, ϕ, γ, p and r. Fault removal probability p and fault introduction probability r cannot be estimated directly from DS1 and DS2 and depends on the project type and skill set of persons involved in development and testing. Based on these two factors fault removed during testing is assumed 95 % and fault introduced is assumed 3%.

The goodness of fit for the proposed model is measured using SSE and MSE for each application datasets DS1 and DS2. These statistics are used for comparison of the proposed model with existing failure rate models given in Table 3.1. These software reliability models are used for comparison of the proposed model and to find model parameters for each of iteration of DS1 and DS2.

To calculate the parameters of the models (N, n, ϕ and γ) hybrid PSO-GSA algorithm is employed along with MLE technique to maximize the log-likelihood function value as given in Eq. (9), (10) and (11) for 21 iterations datasets. Maximum likelihood estimation technique is used to get the likelihood function value of failure rate models. Log of the likelihood function is the objective function and is provided as the input to the PSOGSA algorithm. The optimum value of the objective function is used to find the values of the parameters.

Initially system is assumed to be having minimum features. Later on with time, requirements of the end users need to be fulfilled; more and more features in the system are added. These additional changes in upcoming iterations may incorporate some new errors and may reduce few of the errors from the earlier iterations. Depending on these new conditions, there is a need to make changes in the iterative system development. Accordingly in the proposed work γ parameter changes its shape. In the proposed model, estimated values of γ are representing altogether entire feature changes incorporated in the current iteration with a value differing from the previous iteration. For initial iteration releases of Eclipse and JDT open source software systems, γ value should be estimated in such a way that key sample of requirements are implemented with least features in the system. As time passes and more and more iterations are added with changing functionality in each of iteration, μ value fluctuates from lower to higher with the user acceptance and increase in functionality of the system with time. Depending on the μ values γ is estimated for each iteration release and model is implemented.

The values of μ with respect to iteration are different for different software projects and depend on the type of project and user acceptance levels. Fig. 3.1 depicts variation in μ values with respect to number of iterations for DS1. The value of μ decreases in successive iterations 1.0 to 2.0, 2.1 to 3.0 and 3.3 to 3.4 by difference of 0.064, 0.081 and 0.045 respectively. These little deeps in μ values represent variation in compliance of functionality requirements and lower values of user acceptance levels for eclipse software dataset. Overall values of μ shows increasing trends with successive iterations and finally it reaches near to 1.0 in final iteration release.

Table 3.1 Summary of Failure Rate Based Software Reliability Models for Comparison

Sr. No.	Model Name	Failure Intensity
1	Jelinski-Moranda Model (JM)[44]	$\lambda(t_i) = \phi[\mathrm{N} - (i-1)]$
2	Schick &Wolverton Model (SW)[54]	$\lambda(t_i) = \phi[\mathrm{N} - (i-1)]\, t_i$
3	Goel Okumotto imperfect debugging Model[51]	$\lambda(t_i) = \phi[\mathrm{N} - p(i-1)]$
4	G.S.Mahapatra *et al.* Model [47]	$\lambda(t_i) = \phi[N - p(i-1) + r(i-1)]$
5	Modified S-W Model(MSW)[51]	$\lambda(t_i) = \phi[\mathrm{N} - (n_{i-1})]t_i$
6	Proposed Model	$\lambda(t_i) = \phi[\mathrm{N} - \frac{(n_{i-1})(\gamma)}{(i-1)}(p - r)]$

Fig. 3.2 is revealing the change in functionalities and its corresponding increase in user acceptance for DS2 of JDT software product. At each of the iteration release there is increase in user acceptance level and it is illustrated with the values of μ in all successive iterations. For major iteration releases the value of μ are 0.0869, 0.1108 and 0.2144, for minor iteration these values are 0.1381, 0.2684, 0.3677, 0.6787, 0.7241 and 0.9539. The user acceptance increases in all iterations but a large variation of 0.31102 in user acceptance level is found in 3.4 to 3.5 iteration. It represents enhanced functionality and accomplishment of all requirements at end-users. Overall values of μ in successive iterations are reflecting variation in functionalities and user acceptance level that increases by small amount in each iteration and finally reaches near to 1.0 in last iteration release.

The implementation of the proposed algorithm is done on Intel(R) Core (TM) i5 (5^{th} gen)-62000 CPU 2.40 GHz with 4 GB RAM and 64-bit windows architecture, x64 based-processor. Mat-lab (R2015a) has been used to model proposed software reliability model and to implement PSOGSA algorithm.

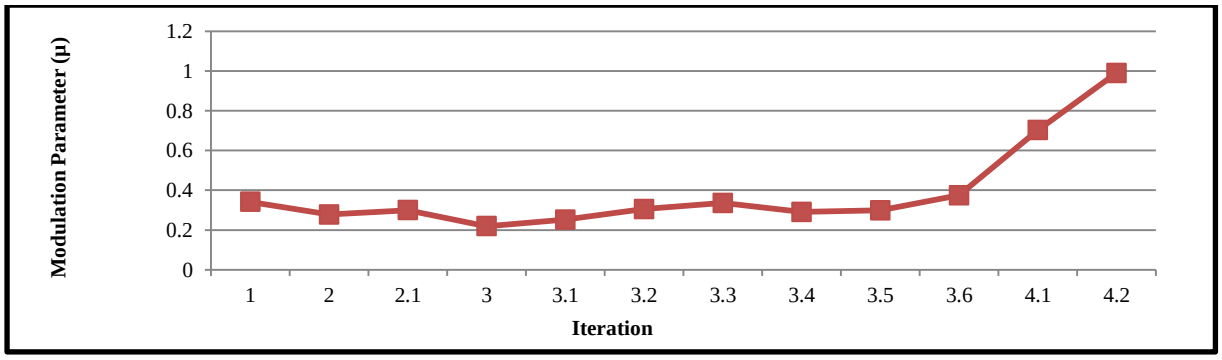

Fig. 3.1 Plot of Modulation Parameter versus Iteration for DS1

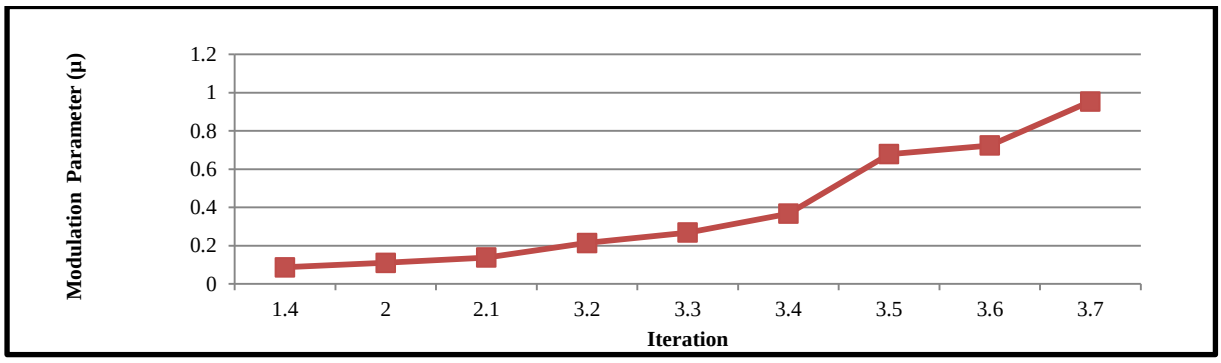

Fig. 3.2 Plot of Modulation Parameter versus Iterations for DS2

3.6 Result Analysis

3.6.1 CASE 1: Eclipse Software Dataset

In this section, the goodness-of-fit of the proposed model is calculated and compared with existing models given in Table 3.1for DS1. The value of γ for major iterative release 1.0, 2.0, 3.0 and 4.1 are 2.2666, 2.8763, 3.7891 and 1.1258 respectively. In minor iteration release, the value of γ for iterative release 2.1, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6 and 4.2 are 2.6343, 3.2217, 2.5788, 2.3121, 2.7295, 2.6434, 2.0435 and 1.00009 respectively.

The value of γ shows large changes in successive major iterations as compared to successive minor iterations due to varying needs in major and minor iterations. From the analysis of results, it is found that proposed model fits well in term of SSE in all iterations of DS1 except at iteration number 3.1 where GOI model, GS Mahapatra model, and SW model outperforms proposed model.

The proposed model has clear-cut outperform all models under comparison in 11 iterations for DS1. It shows the lowest values of SSE in 91.6 % iterations. In term of MSE proposed model is winner in nine iterations. In iteration 2.1 MSW model is performing well than the proposed model. In iteration 3.3, GOI model, SW model, and GS Mahapatra model are performing well than the proposed model. There is a tie among proposed model and GS Mahapatra model in iteration 2.0, where both of them outperform all other models. The proposed model has clear-cut outperform other models in 75 % iterations by achieving lowest value of MSE. Result in Fig. 3.3 and Fig. 3.4 shows the reliability and failure rate estimated using the proposed model respectively. All the results from the proposed model has given a significantly better fit to iterative data by adapting according to varying needs of different iterations.

The evaluation and comparison of goodness-of-fit of all models in **Table 3.2** in terms of SSE and MSE for all iterations shows that the proposed model has good technical merit in a sense that it

provides development terms with both iterative SDLC requirements and traditional reliability measures.

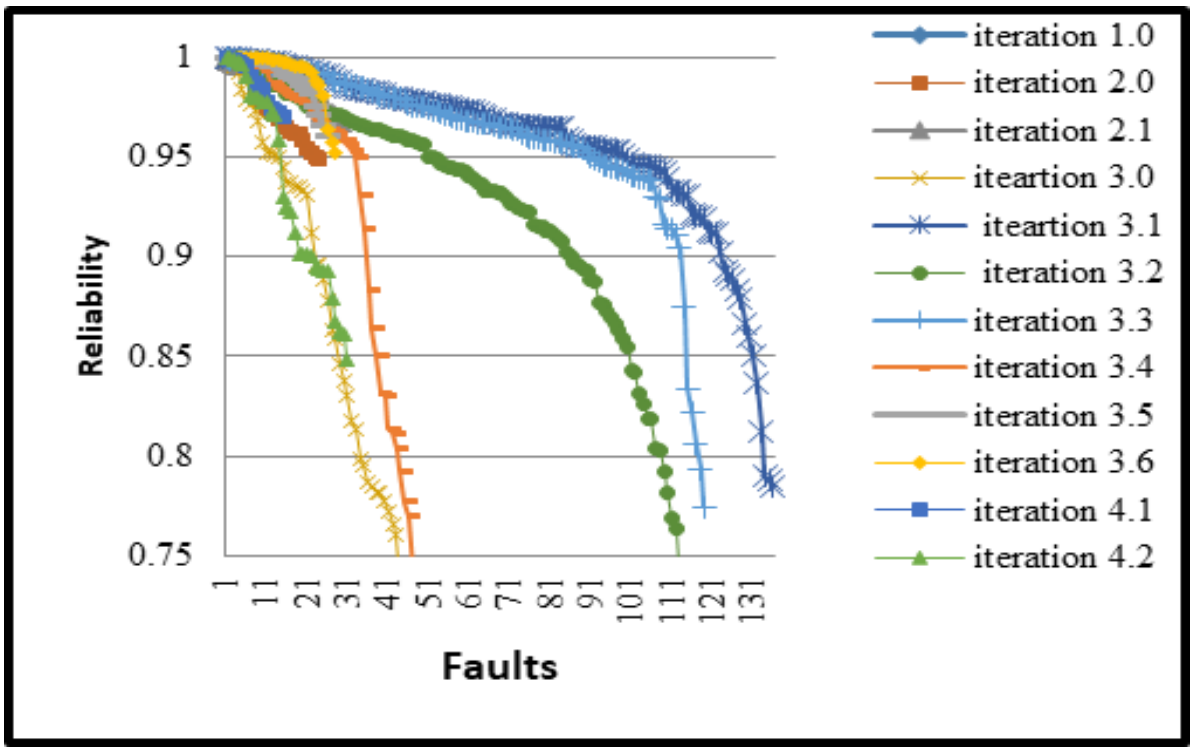

Fig. 3.3 Plot of Reliability versus Number of Faults

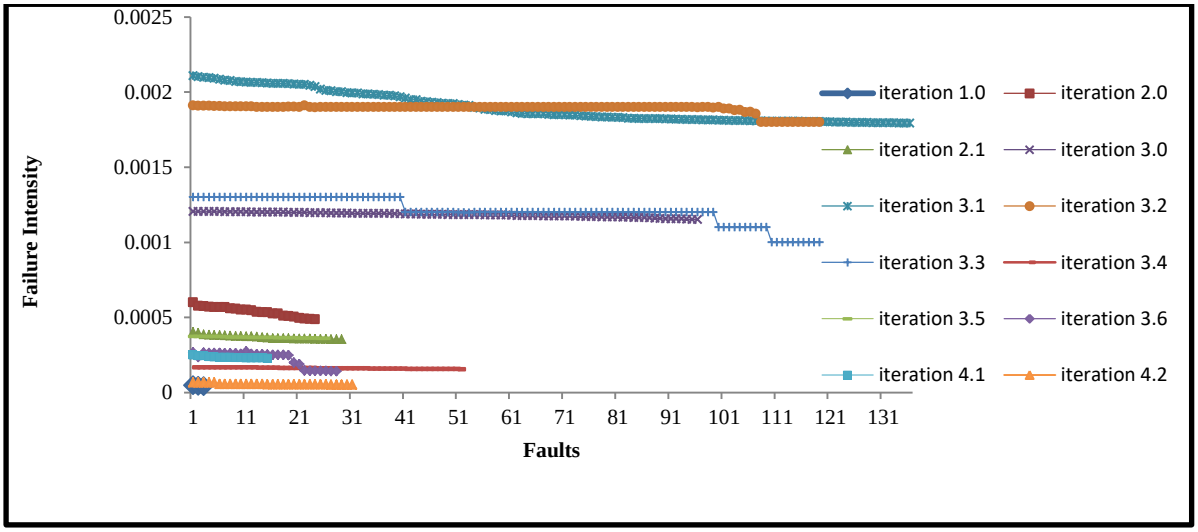

Fig. 3.4 Plot of Fault Intensity in Various Iterations

3.6.2 CASE 2: JDT Dataset

To test the applicability of the proposed model in this section dataset DS2 is used. **Table 3.3** is showing the estimated values of model parameters and goodness of fit criteria. The value of γ for major iterative release 1.4, 2.0 and 3.2 are 10.5954, 8.1358 and 3.8778 respectively. In minor iteration releases the value of γ for iterative release 2.1, 3.3, 3.4, 3.5, 3.6 and 3.7 are 6.3815, 2.9946, 2.0872, 1.1521, 1.1052 and 1.0022, respectively. These values of γ are showing large deviations in successive major releases as compared to successive minor releases due to the varying needs in each of the major and minor iterations. **Table 3.3** is showing the goodness-of-fit measures of all models depicted in Table 3.1for DS2. Out of total nine iterations in DS2, proposed model outperform other models under comparison in eight iterations, in terms of SSE values. JM model performs well than proposed model in iteration 3.4. Proposed model attains lower-most value of SSE in 88.8 % of iterations. In terms of MSE values for nine iterations proposed model out-performs other models in six iterations. In iteration 2.0, GS Mahapatra model and GOI model are performing better than the proposed model. In iteration 3.7, proposed model outperforms JM, GOI, SW and MSW models except GS Mahapatra model. The proposed model outperforms GOI , SW, GS Mahapatra and MSW model except JM model in iteration number 3.6. The proposed model has performed better than all other models under comparison in 66.6% iterations by attaining the minimum value of MSE. Results in terms of estimated reliability and failure rate estimation by the proposed model are shown using Fig. 3.5 and Fig. 3.6 respectively.

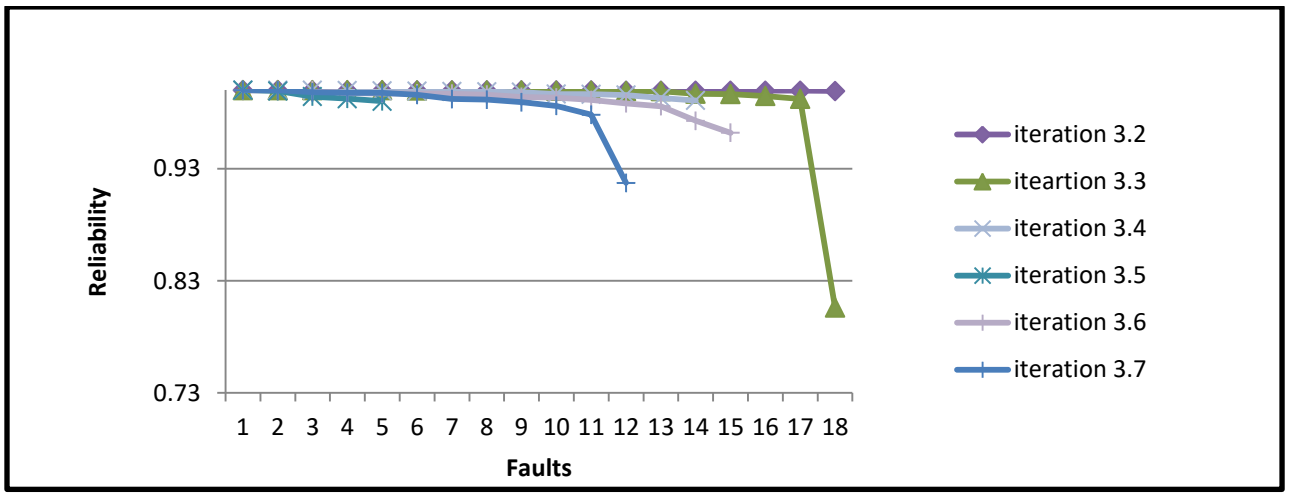

Fig. 3.5 Plot of Reliability versus Iterations

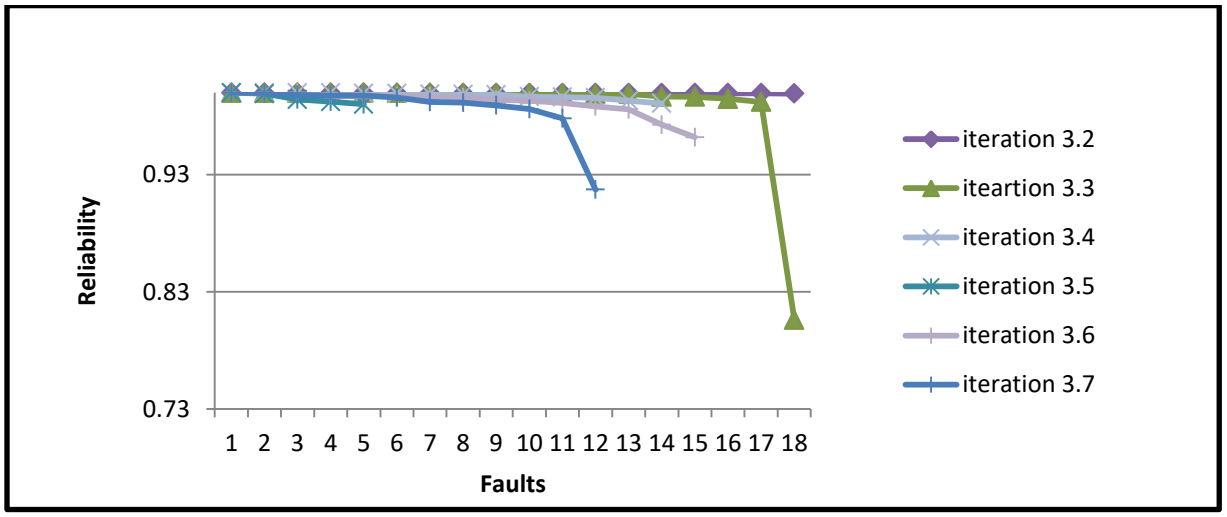

Fig. 3.6 Plot of Failure Intensity versus Iterations

Result analysis in **Table 3.3** shows that the proposed model has significantly better fit to iterative data by fine-tuning to varying needs of different iterations.

Table 3.2GOODNESS-OF-FIT ES/TIMATED USING DS1 (ECLIPSE SOFTWARE FAILURE DATASET)

		Iteration1.0			Iteration2.0		
Sr. No.	Model	Estimated Parameter values	SSE	MSE	Estimated Parameter values	SSE	MSE
1	JM Model	Φ=5.74E-06, N=4	5.23	5	Φ=2.68E-05, N=8	375.51	17.05
2	GOI Model	Φ=1.64E-05, N=3	1.51	1	Φ=2.89E-06, N=28	92.01	4.18
3	SW Model	Φ=2.124E-06, N=3	17.31	17.00	Φ=4.70E-06, N=28	218.81	9.91
4	GS Mahapatra Model	Φ=7.77E-07, N=3	1.26	1	Φ=2.53 E-05, N=23	55.46	2.5
5	MSW Model	Φ= 2.23E-05, N=2, n=58	5.0	4.9	Φ=1.52E-06, N=9, n=209	528.7	15.89
6	Proposed Model	Φ= 2.86E-05, N=5, n=11, ϒ=2.2666,	1.24	0.66	Φ=4.70E-06, N=27, n=362, ϒ=2.8763	45.12	2.5

		Iteration2.1			Iteration 3.0		
Sr. No.	Model	Estimated Parameter values	SSE	MSE	Estimated Parameter values	SSE	MSE
1	JM Model	Φ=5.039E-06, N=27	452.2	16.74	Φ=2.40E-06, N=124	56948	605.82
2	GOI Model	Φ=1.95E-06, N=26	133.34	4.93	Φ=1.37 E-06, N=100	45510.1	559.22
3	SW Model	Φ=3.83E-06, N=29	126.6	4.66	Φ=2.92 E-05, N=121	39981	494.69
4	GS Mahapatra Model	Φ=3.20E-06, N=28	131.31	4.85	Φ=1.25 E-06, N=125	55122.2	586.40
5	MSW Model	Φ=1.36E-05, N=18, n=316	118.67	3.89	Φ=1.29E-06, N=123, n=298	66890.4	679.32
6	Proposed Model	Φ=1.67E-06, N=30,n=415, ϒ=2.6343,	101.02	4.39	Φ=2.98E-05, N=102,n=5690,	39705.2	484.14

		Iteration3.1			Iteration3.2		
Sr. No.	Model	Estimated Parameter values	SSE	MSE	Estimated Parameter values	SSE	MSE
1	JM Model	Φ=2.99 E-05, N=100	6091.1	45.45	Φ=2.21E-06, N=122	13940.73	119.14
2	GOI Model	Φ=3.07E-07, N=126	1018.8	7.59	Φ=8.61E-06, N=106	7246.05	61.93
3	SW Model	Φ=2.99E-05, N=125	1123	8.38	Φ=5.40E-06, N=90	12585	107.56
4	GS Mahapatra Model	Φ=6.62E-06, N=130	1015	7.57	Φ=1.25E-06, N=115	7794.28	68.97
5	MSW Model	Φ=1.36E-05, N=135, n=4979	2000.7	15.9	Φ=2.08E-06, N=132, n=4108	9685.12	89.67
6	Proposed Model	Φ=2.96E-05, N=136,n=6132,ϒ=3.2217	1589.82	12.22	Φ=2.98 E-05 , N=122,n=6910,ϒ=2.5788	952.2	8.14

Table 3.2 **continued…**

		Iteration3.3			Iteration 3.4		
Sr. No.	Model	Estimated Parameter values	SSE	MSE	Estimated Parameter values	SSE	MSE
1	JM Model	Φ=2.95E-05, N=117	6810.14	58.24	Φ=8.28E-06, N=53	1153.32	23.53
2	GOI Model	Φ=1.25E-06, N=115	9371.1	80.09	Φ=3.06E-06, N=50	1545.41	1320.9
3	SW Model	Φ=2.95 E-05, N=123	7318.81	62.55	Φ=2.93 E-05, N=54	1153.73	23.53
4	GS Mahapatra Model	Φ=3.06E-06, N=114	2217.4	18.99	Φ=1.63E-05, N=55	905.4	18.46
5	MSW Model	Φ=5.31E-06, N=121,n=3129	2903.9	27.87	Φ=1.72E-05, N=52, n=2094	3589.96	29.79
6	Proposed Model	Φ=2.96E-05, N=123,n=7879, ϒ=2.3121	2203.05	19.49	Φ=2.92 E-05, N=53, n=1411, ϒ=2.7295	520.23	11.55
		Iteration3.5			Iteration3.6		
Sr. No.	Model	Parameter estimated values	SSE	MSE	Parameter estimated values	SSE	MSE
1	JM Model	Φ=1.63E-06, N=30	373.3	15.54	Φ=6.24E-07, N=28	199.91	7.65
2	GOI Model	Φ=3.81E-06, N=29	137.74	5.71	Φ=3.208E-06, N=28	160.01	6.15
3	SW Model	Φ=1.70E-07, N=29	551.1	22.96	Φ=3.74E-06, N=27	154.54	5.92
4	GS Mahapatra Model	Φ=3.81E-06, N=29	137.8	5.71	Φ=6.65E-07, N=28	152.23	5.86
5	MSW Model	Φ=1.26E-05, N=17, n=2663	489.74	18.24	Φ=2.97E-05, N=17, n=2624	315.00	14.92
6	Proposed Model	Φ=2.82 E-05, N=25, n=416, ϒ=2.6434	86.62	4.3	Φ=2.94E-05, N=30, n=401, ϒ=2.0435	91.15	4.14
		Iteration4.1			Iteration4.2		
Sr. No.	Model	Parameter estimated values	SSE	MSE	Parameter estimated values	SSE	MSE
1	JM Model	Φ= 2.20 E-05, N=19	286.65	22	Φ=1.92E-05, N=31	441.12	15.21
2	GOI Model	Φ=1.58 E-05, N=17	111.21	8.54	Φ=3.20E-06, N=30	121.21	4.17
3	SW Model	Φ=1.93E-05, N=19	284.86	22.1	Φ=6.50E-06, N=31	193.34	6.66
4	GS Mahapatra Model	Φ=5.53E-06, N=17	129	9.92	Φ=7.29E-07, N=31	166.65	5.72
5	MSW Model	Φ=2.68E-05, N=8, n=1529	178.22	7.34	Φ=2.97E-05, N=30, n=3729	704.86	26.87
6	Proposed Model	Φ=2.87E-05, N=16,n=136, ϒ=1.1258	49.95	5.44	Φ=2.82E-05, N=32, n=513, ϒ=1.00009	72.24	2.88

Table 3.3GOODNESS-OF-FIT ESTIMATED USING DS2 (JDT SOFTWARE FAILURE DATASET)

		Iteration 1.4			Iteration 2.0		
Sr. No.	Model	Estimated Parameter values	SSE	MSE	Estimated Parameter values	SSE	MSE
1	JM Model	Φ= 1.46E-05, N=22	1510	62.91	Φ= 5.63E-08, N=13	1792.56	66.37
2	GOI Model	Φ= 4.11E-06, N=15	1517	75.85	Φ= 4.80E-08, N=18	1486.81	55.04
3	SW Model	Φ= 8.18E-07 N=9	2165	90.21	Φ= 6.99E-06, N=17	2427.45	187.9
4	GS Mahapatra Model	Φ= 9.00E-06 N=12	1871	77.96	Φ= 1.08E-06, N=16	1434.79	53.11
5	MSW Model	Φ=7.43E-06 N=19,n=242.98	2909.88	121.09	Φ=1.62E-05, N=15,n=282.93	1870.67	69.26
6	Proposed Model	Φ=2.99E-05, N=13, n=160, ϒ=10.5954	1379	57.54	Φ=2.97E-05, N=17, n=199, ϒ=8.1358	1292	56.34
		Iteration 2.1			Iteration 3.2		
Sr. No.	Model	Estimated Parameter values	SSE	MSE	Estimated Parameter values	SSE	MSE
1	JM Model	Φ= 2.81E-05, N=37	2638.12	203.02	Φ= 2.35E-06, N=50	179.79	11.19
2	GOI Model	Φ= 5.53E-06, N=35	1650.04	127.90	Φ= 8.16E-07, N=56	317.67	5.56
3	SW Model	Φ= 5.36E-06, N=38	3278.45	252.87	Φ= 2.17E-06, N=58	1056	15.68
4	GS Mahapatra Model	Φ= 1.21E-06, N=57	3808.85	293.48	Φ= 1.21E-06, N=57	56500.09	991.23
5	MSW Model	Φ=8.73E-06, N=29,n=2569.9	1741.90	164.48	Φ=2.91E-05, N=52,n=1723	2989.06	29.69
6	Proposed Model	Φ=2.82E-05, N=37, n=234, ϒ=6.3815	683.70	75.88	Φ=2.91E-05, N=60, n=1788, ϒ=3.8778	171.89	3.226
		Iteration 3.3			Iteration 3.4		
Sr. No.	Model	Estimated Parameter values	SSE	MSE	Estimated Parameter values	SSE	MSE
1	JM Model	Φ= 2.78E-05, N=9	404	33.666	Φ=8.86E-06, N=18	37	12.33
2	GOI Model	Φ=1.48E-05, N=16	989.9	61.825	Φ=2.35E-06, N=18	206.56	17.16
3	SW Model	Φ=4.70E-06, N=8	589.1	40.78	Φ=3.83E-06, N=16	750.9	68.00
4	GS Mahapatra Model	Φ=8.92E-06, N=26	604.01	37.75	Φ=1.03E-05, N=28	1241.01	103.41
5	MSW Model	Φ=2.99E-05, N=9, n=1028	486.32	33.960	Φ=2.91E-05, N=13, n=1124	1839.56	146.92
6	Proposed Model	Φ=2.97E-05, N= 19,n=179, ϒ=2.9946	31	2.583	Φ=2.92E-05, N=18, n=119, ϒ= 2.0871	91	11.37

Table 3.3 Continued…

		Iteration 3.5			Iteration 3.6		
Sr. No.	Model	Estimated Parameter values	SSE	MSE	Estimated Parameter values	SSE	MSE
1	JM Model	Φ=2.49E-05, N=8	57	4.384	Φ=2.93 E-05, N=4	76	7.6
2	GOI Model	Φ=1.06E-06, N=13	76.53	25.33	Φ=2.01E-05, N=11	46721	359.39
3	SW Model	Φ=2.92 E-05, N=10	3019.9	119.05	Φ=2.99E-05, N=29	12089	367.83
4	GS Mahapatra Model	Φ=4.78 E-07, N=5	38225	318.54	Φ=1.26E-05, N=85	11534	384.48
5	MSW Model	Φ=2.31E-05, N=3, n=137	20687	156.94	Φ=1.34E-05, N=15, n=134	34834.09	329.21
6	Proposed Model	Φ=2.96E-05, N=7, n=25, ϒ=1.1521	18	-24.002	Φ=2.67E-05,N=18, =171,ϒ=1.1051	192	21.33

Iteration 3.7

Sr. No.	Model	Estimated Parameter values	SSE	MSE
1	JM Model	Φ=1.50E-06, N=16	280	45.02
2	GOI Model	Φ=1.58E-06, N=13	90.011	9.01
3	SW Model	Φ=5.40E-06, N=26	200.9	11.08
4	GS Mahapatra Model	Φ=5.30E-06, N=12	952	8.14
5	MSW Model	Φ=2.69E-05, N=14, n=1599	1109.23	57.34
6	Proposed Model	Φ=2.74E-05,N=15, n=78,ϒ=1.0022	54	9.00

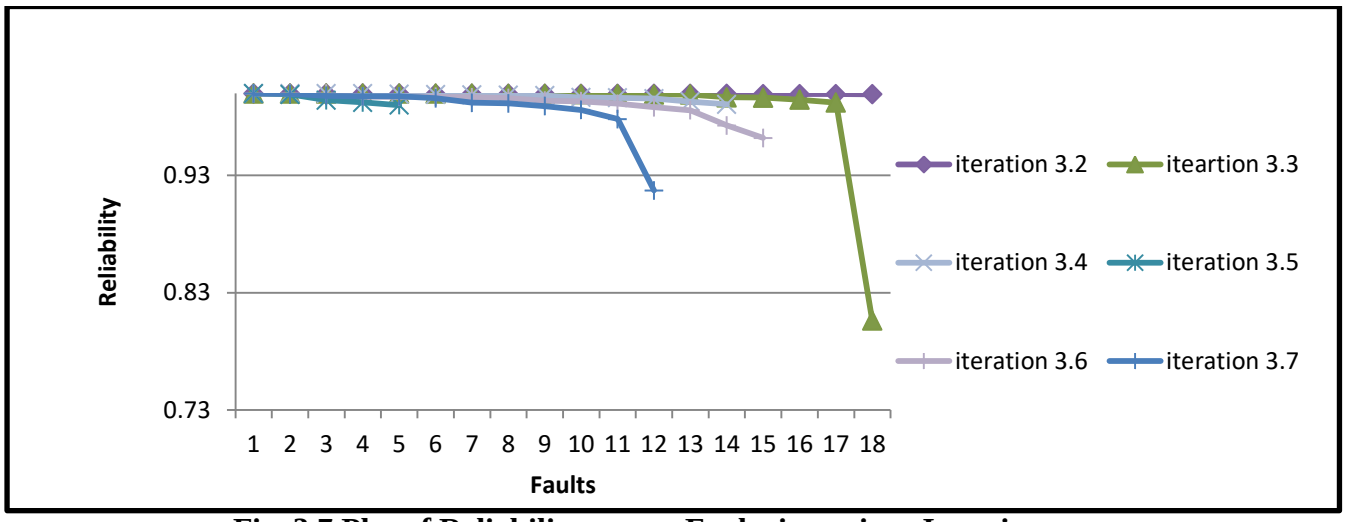

Fig. 3.7 Plot of Reliability versus Faults in various Iterations

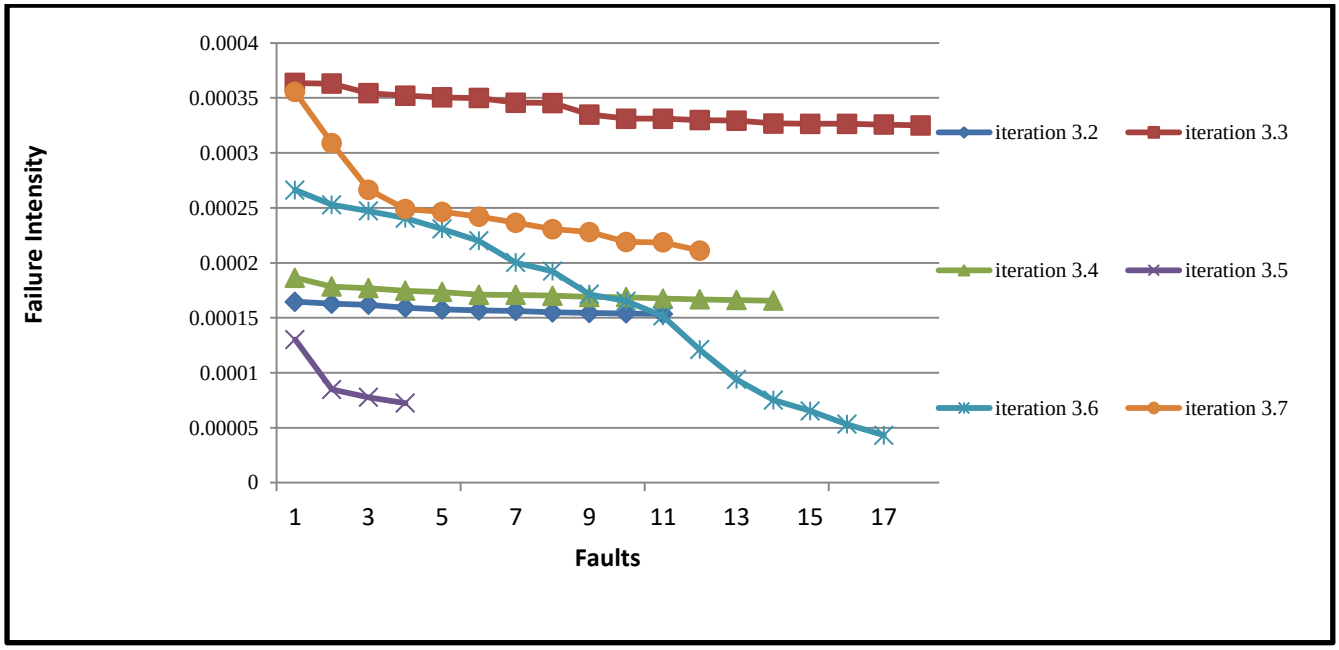

Fig. 3.8 Plot of Failure Intensity in several Iterations

From the analysis it is found that the proposed model is suitable on time domain data sets. Proposed model is more adaptive to observed time domain failure data sets than other failure rate models. Adaptation has been made possible because of the modulation parameter μ used in the model. As the software development moves towards completion, this parameter changes its values according to added functionalities and user acceptance level in each of successive iteration of software development. This parameter assumes that in each iterative development cycle there is added functionality and user's acceptance. According to the functionality addition in each of the iteration there is change in requirements of iterative software development. All the varying needs from the users are reflected with modulation factor γ in the proposed model. Proposed model provides good fit of the observed failure data. Values estimated in **Table 3.2** and **Table 3.3** shows

acceptable parameter values at all values of γ. With each iteration, functionality values and user acceptance increases (shown by value of modulation parameter that increases from lower to higher) and there is corresponding change in needs for iteration development (shown by modulation factor). Values of modulation factor and modulation parameter change suitably with each of upcoming iteration in both the failure datasets, showing well the iterative software development behaviour. Depending on the estimated parameters values, number of remaining faults in terms of expectation is calculated. The prediction deviates by small amount in some cases due to introduction and removal of mutually dependent and independent faults in each of iteration and this verifies imperfect debugging phenomenon associated with each iterative software development. Goodness of fit for models is also shown in **Table 3.2** and **Table 3.3** in terms of SSE, MSE, AE and TS values. Proposed model fits well in both the failure datasets used.

In data analysis of both the failure datasets, proposed model is having outstanding performance than JM, GOI, SW, Mahapatra et al. and SWM models in terms of goodness values. In all iterations of datasets, proposed model reliability increases as number of iterations proceeds. Last iteration is assumed to be the reliable iteration. This change in reliability is showing value-added software development process. All models have been implemented using hybrid PSO-GSA algorithm for parameter estimation. Proposed model shows considerable performance with hybrid algorithm even with more number of parameters then other models.

3.7 Conclusion

Failure rate models available in literature are centered on most traditional waterfall SDLC process. However, new software development processes have been developed and found to be more beneficial than waterfall SDLC process, like iterative life cycle processes. Keeping in view, new software development environments and technologies, a new failure rate model by exploiting iterative behaviour of software development process is proposed. The changing needs in each of the iteration are reflected in the proposed model using a modulation factor. Calculated values of γ parameter are significantly reflecting all changing requirements for each upcoming iterations numerically. These values are meaningfully representing how much impact is of adding and removing new features with the level of user acceptance in each of upcoming iteration. In order to compare the performance of the proposed model, five well-known software failure rate models JM, GOI, SW, GS Mahapatra and MSW have been applied for dataset DS1 and DS2. The proposed model is a clear-cut winner in 11 iterations in SSE and 9 iterations in MSE out of 12 iterations and 8 iteration in SSE and 6 iterations in MSE out of 9 iterations for DS1 and DS2 respectively. Overall, in 83.33 % of iterations for DS5 and 77.77 % of iterations for DS2, the proposed model has shown better results in terms of goodness of fit by successfully incorporating varying needs in

each of iteration. The data collected from real applications and comparison of goodness of fit shows that the proposed model successfully incorporated varying needs in each of iteration and performed better than JM, GOI, SW, GS Mahapatra and MSW models.

Chapter 4 MODELING AND ANALYSIS OF OPEN SOURCE SOFTWARE SYSTEM RELIABILITY

Growing software demand in the present virtual world introduces new competitive dynamics for software developers. Recently, Open Source Software systems are providing a faster way of software production. To survive in the competitive market, developed OSS system needs enhancement in previous versions. Each enhanced versions are found to be more liable to risks of failures. In recent process of software development, primary concern of researchers is always to find new ways for assessing the reliability of developed OSS versions. To incorporate modern software development environments and technologies, new model for reliability estimation of multiple versions of OSS systems has been developed in this chapter. Proposed model incorporates a new testing effort factor for integrating varying needs in each release of software development. It comprises phenomenon of imperfect debugging with a possibility of fault introduction. Proposed model is validated on various releases of Firefox and Genome project failure data set. Parameter estimation for the proposed model has been done using proposed algorithm. Experimental results have shown the enhanced capability of the proposed model in comparison to Goel-Okumotto model, Inflection S-shaped model and PTZ model in simulating real OSS development environment.

4.1 Introduction

In today's growing cyber world, due to a revolution in the digital world, software is in significant use of commercial to ordinary people lifestyles. Everything is in the fingerprints of smartphone user's, whether it is related to search for any information, communication, government documents, workplace, news articles, media, transportation and so on. The software has been strongly implanted into every facet of human lives. There is a tremendous demand for developing

new applications from the software industries. Since 1990's there is an increasing interest of software companies and academicians in developing OSS systems. A major success of OSS systems has led them towards development of OSS, leaving closed source software development behind [241]. These are providing a new way of developing software systems worldwide. Development of OSS systems is much faster than traditional software projects as there are no firm plans, schedules, and system design processes. Before time software's are released, to fulfill huge software demand in the market. OSS systems are more prone to failures as these are not undergoing every phase of software development and may pose a critical danger to life and properties.

There are numbers of OSS systems like Eclipse, Gnome, Linux, Mozilla, Apache, and Android found to be in wide-spread use. Due to increasing demand, lack of time and budget in developing OSS systems, OSS developers are not able to create fault-free software. There is always a significant probability of fault occurrence in the newly developed software releases. To meet requirements and to remove faults from previous versions, new features are added in upcoming software releases. The existence of a fault in forthcoming versions is inevitable.

There should have some means to avoid software failures so that bad losses could be evaded. There should be a level of quality assurance from the software developers before any software release [242], for this there is need to have an idea of the probability of failure in the field environment. There is an immense need for estimation of reliability of hastily spread out OSS systems. In engineering software reliability, software reliability growth models are providing a way to depict software reliability based on fault content. Researchers have developed numerous software reliability estimation models, but new software development environments, latest technologies and different development modes for software development have led to create a need for the development of new model for estimation of software reliability.

OSS development provides a potential for flexible and quicker innovative technology. In each version release of OSS systems, new features are added and software moves towards refinements in each developed versions. Reliability also varies in each future releases because new functionality is added and new bugs are identified and fixed by the co-developers. Precise version reliability estimation should be done via software reliability estimation models. Models developed for OSS system reliability estimation are mainly based on NHPP. No model has yet been developed for OSS systems that incorporate testing effort based fault content function; these models need further modification for adaption to the realistic development environment of OSS systems.

Identifying factors that will engulf varying needs in each version release of software development is very much essential and need to be incorporated into OSS system reliability estimation models. For example, in each version development process of OSS systems, there is a

need to know all crucial changes that should be incorporated into current version development apart from the previous version releases. In new version development, there is need of adding new features, removing bugs from the previous version or it may involve alterations in testing efforts depending on the end user requirements. All the testing effort changes in each version should be revealed with a new factor that could be incorporated in the OSS system reliability assessment model development. There are no studies available in the literature that concentrates on incorporating such factors in the ensemble for reliability estimation via software reliability models for OSS systems. Keeping multi-release open source software policies in mind, work discussed in this chapter proposed a new model that introduces a new testing effort factor. This factor is showing the change in fault content function with the amount of testing effort in each version of OSS system development. Altogether it is reflecting complete testing effort functionality added or upgraded in each version of the software. Effort factor changes its value according to the effort coefficient which takes its value from 0 to 1 by assuming that complexity value added in each version increases from lower to higher. Effort factor has a change in its value, depending on whether it is a minor or a major release.

For precise estimation of OSS system reliability, there is a need to have a parameter estimation method that could provide optimum parameter values of models. The techniques for optimizing the parameters of software reliability models are available through various classical methods of parameter estimation [51][203]. However, these methods are based on a number of constraints. An alternative to these classical mathematical optimization methods is nature-inspired optimization algorithms for the solution of non-differential, non-linear and multi-modal problems [6], [173], [243]. Proposed model parameters are estimated using proposed ABCDE algorithm that incorporates simple and efficient features of ABC and DE algorithms and then results have been compared with other models for validation. The goodness-of-fit measure of proposed model is calculated using eight criterion and results are compared with other NHPP based models. Estimated numbers of faults are calculated in each release using the estimated values of parameters.

OSS development provides a potential for flexible and quicker innovative technology. In each version release of OSS systems, new features are added and software moves towards refinements in each developed versions. Reliability also varies in each future releases because new functionality is added and new bugs are identified and fixed by the co-developers. Precise version reliability is estimated using software reliability estimation models. Models developed for OSS system reliability estimation are mainly based on Non-homogeneous Poison Process. Having multi-release OSS policies in mind, a new NHPP model is proposed that introduces a new testing effort factor. This factor is showing the change in fault content function with the amount of testing effort

in each version of OSS system development. Altogether it is reflecting complete testing effort functionality added or upgraded in each version of the software. Effort factor has a change in its value, depending on whether it is a minor or a major release. There are no studies available in the literature that concentrates on incorporating such factors in the ensemble for reliability estimation via software reliability models for OSS systems.

4.2 Modeling Procedure for OSS Systems

In this section step by step method to fit the proposed model to version based failure data set has been described. The flowchart in Fig. 4.1 is illustrating how the proposed model could be fitted to OSS system data sets. The proposed model is tested on different versions of OSS system failure data. Performance measures of the model are calculated in terms of the estimated number of errors in a system using multi-version failure data. The objective of the proposed model is to help in decision making like what is the right time to release software is more testing required or is there any need of developing a new version. Fitting of the proposed model on a multi-version data-set with decision-making process based on performance measures is described below [12]:

Step1. Study of Open Source Failure Data

The first step in model development includes a selection of data-set for which model is being proposed. There is a need to carefully study particular data-set so that accurate insight could be made about the nature of the process that is being modeled. In this chapter version based data-set has been used. Keeping in mind multi-version based reliability modeling; interval domain data has been collected and normalized.

Step2. Choose a Model for OSS System Reliability

Next step is to choose a particular model from the assumptions and nature of data-sets. In this chapter version based model has been proposed. Care has been taken to include all the necessary variables required to reflect the OSS system development environment. A new testing effort factor has been introduced in the proposed model that reflects altogether different effort needs in terms of manpower for system development in each successive version of OSS development life cycle.

Step3. Obtain Proposed Model Parameter Estimates

This step includes the parameter estimation process used for exact estimates of parameters of the model. Maximum likelihood estimation process has been used in this chapter for parameter estimation. For optimization of the model parameters, hybrid ABCDE algorithm has been applied.

Step4. Find Fitted Model

Using estimated parameters, fitted model estimates are acquired. At this time we could get fitted model value based on data-sets and proposed model.

Step5. Perform test for Goodness-of-Fit of model

Goodness-of-Fit of the model is calculated using various criteria.

Step6. Obtain Performance Measures of Proposed Model

The performance measures of fitted model are calculated using estimated number of errors in the system. Other performance measure includes failure rate behaviour estimation and reliability estimation.

Step7. Make Decision

The ultimate objective is to make decisions about the time of the release of versions of OSS system. These decisions help in deciding whether it is to release a version or not depending on the decisions made in the earlier steps.

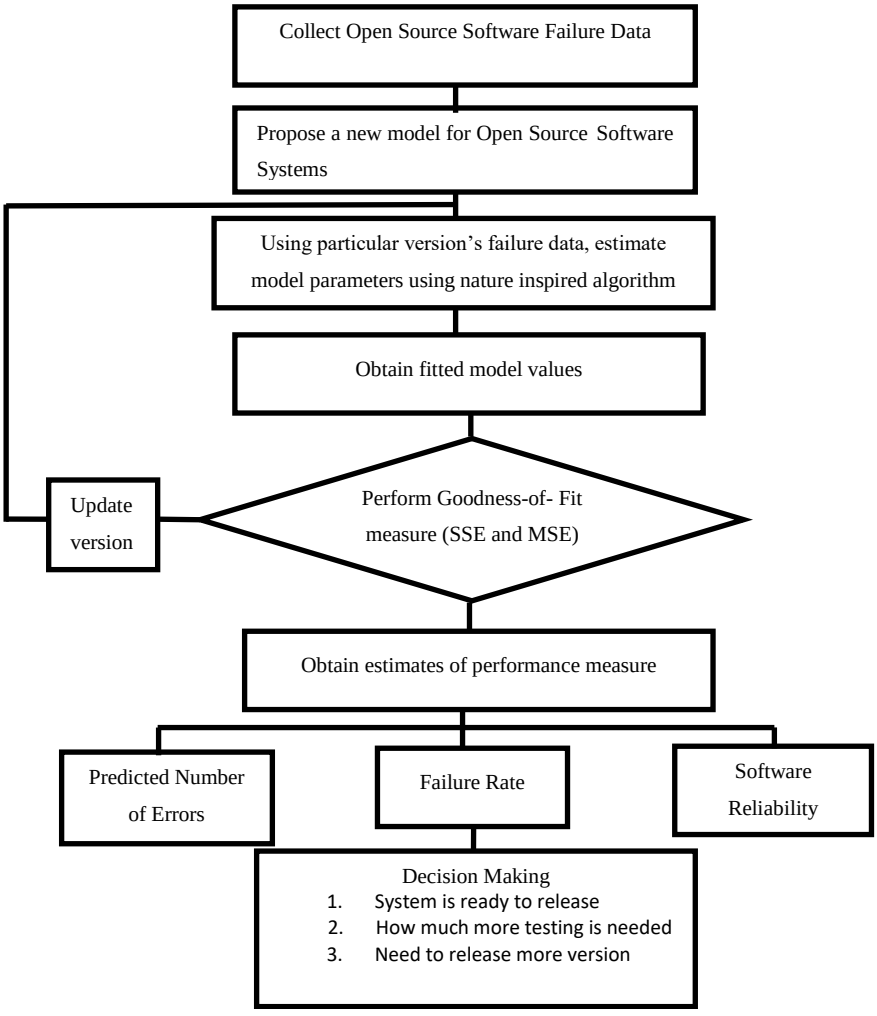

Fig. 4.1 Software Reliability Modeling process for OSS systems

4.3 Proposed NHPP Model

4.3.1 Non Homogenous Poisson Process based model for OSS System Development

Maximum of existing reliability models emphasis only on single version process of software development. In OSS development, co-developers develop software, test and releases several versions of the software. There is a need to focus on changes that are incorporated into multiple version releases of software. In OSS system development, there is need of adding new features, removing bugs from the previous version or it may involve alterations in testing efforts depending on the end user requirements. Change in the amount of effort in each OSS system development is revealed with a new factor in proposed reliability assessment model.

4.3.2 Proposed Model Incorporating Effort Based Fault Content Function

The proposed model incorporates testing effort as the main factor in total fault content of the system at a particular time and ensuring it, assumes fault content function as an exponential function of time. The rate of fault detection is assumed to be constant. Further, at the beginning phase of testing, testers are not having the knowledge of the testing environment and by the time only testers gain knowledge about the software environment. Testing effort is having an effective impact on increasing or decreasing the failure rate of the system. The number of errors may be introduced while testing progresses. The fault introduction rate at the beginning phase is assumed to be higher than later. Effect of testing effort on total fault content has been considered that affects the whole system reliability.

Assuming realistic development environment, the model proposed is based on following assumptions:

a. Fault detection and fault removal process are modeled as NHPP.
b. At the time of testing the rate of detection of the fault is a constant value.
c. The rate of failure for software is assumed to be proportional to remaining number-of faults in software.
d. There is always a chance to introduce fault with some probability when detected faults are removed.
e. Fault content is assumed to be an exponentially increasing function with an effective impact of the testing effort.

With the increase in time, the amount of total fault content changes. Relation of testing effort consumption and change in fault content is incorporated using factor ψ in each release of OSS system development. Testing effort is assumed to be at a maximum when the system is

initially implemented and it varies till at the end. It becomes stable when most of the errors are removed and the system gains full functionality and maximum reliability. Testing effort is the most valuable resource in enhancing system reliability. Testing effort factor ψ reflects the fundamental changes of volunteer testing effort by some quantified amount that changes in each release of open source software failure rate models. Testing effort factor changes its value according to the effort coefficient as shown in Equation (6). Moreover, testing effort needs in OSS development in each release is different and varies according to Equation (6). 'r' is the effort coefficient that represents newly added functionality and system complexity in each release due to mutually dependent and independent faults in the system at a particular time. Its value is almost 0 at the beginning and becomes 1.0 as OSS development proceeds. Testing effort coefficient changes its value from 0 to 1 by assuming that with upcoming releases system complexity changes depending on a number of faults present in the system and its functionality. It shows abrupt changes in its value initially due to preliminary changes in system functionality. When its values reach near to one then software under development is assumed to be reliable enough and now effort consumption is assumed to be lowest because system has achieved all the required functionality to fulfill the end user needs.

4.3.3 Model Formulation

The general class of NHPP based software reliability models assumes that a rate of failure in software is proportional to total fault content that is remaining in the software. Equation (1), Equation (2) and Equation (3) are used for obtaining the general class of SRGMs. The intensity of failure λ(t) in software is given using Equation (1)

$$\lambda(t)= \frac{dm(t)}{dt} \qquad (1)$$

$$\frac{dm(t)}{dt} = b(t)[a(t)-m(t)] \qquad (2)$$

Where a(t) and b(t) are the initial content of fault in software and rate of fault detection respectively. The generalized solution for Equation (2) in terms of mean value function m(t) is given in Equation (3) and Equation (4).

$$m(t)=e^{-B(t)}[m_0+\int_{t_0}^{t} a(t)b(t)e^{B(t)}dt] \qquad (3)$$

$$\text{where } B(t)= \int_{t_0}^{t} b(t)dt \; m(t_0)=m_0 \qquad (4)$$

The total content of fault a(t) in software is modeled using a new parameter ψ_r in Equation (5).

$$a_r(t_i)=a\psi_r+a(1-e^{-\alpha_r t})=a(\psi_r+(1-e^{-\alpha t})) \text{ , } i=1,2...n \qquad (5)$$

Where,

$a>0$, $0 \leq \alpha$ is fault introduction probability.

ψ_r is the effort factor that changes in each release (r =1,2…k) according to effort coefficient given in Equation (6).

$$\psi_r = \chi + ((1-\chi)/\chi) \quad 0 < \chi \leq 1 \qquad (6)$$

Number of faults when testing starts in each release is given as

$a(0) = a\psi = a$ (as ψ is assumed to be 1)

$a(\infty) = a\psi + a = a(\psi + 1)$

Total number of faults generated because of imperfect debugging while testing progresses is given in Equation (7)

$$a(\infty) - a(0) = a \qquad (7)$$

This content of faults specifies the total number of faults that gets generated or introduced throughout a testing phase. Thus total number of faults that may be introduced after a time t in each release is given by Equation (8)

$$a(\infty) - a(t) = a(\psi + 1) - (a\psi + a(1 - e^{-\alpha t})) = ae^{-\alpha t} \qquad (8)$$

The fault detection rate is assumed as a constant value as given in Equation (9).

$$b(t) = b \qquad (9)$$

The expected number of faults i.e. mean value function in Equation (11) and failure intensity function in Equation (12) can be obtained by putting the value of a(t) and b(t) in Equation (3).

$$m(t) = be^{-bt} \int_0^t (a\psi + a(1 - e^{-\alpha t}))e^{bx} dx$$

$$= be^{-bt} \left(\int_0^t a\psi e^{bx} dx + \int_0^t ae^{bx} dx - \int_0^t ae^{(b-\alpha)x} dx \right.$$

$$= abe^{-bt} [\psi [\frac{e^{bx}}{b}]_0^t + [\frac{e^{bx}}{b}]_0^t - [\frac{e^{(b-\alpha)x}}{b-\alpha}]_0^t$$

$$= abe^{-bt} [\frac{(e^{bt} - 1)(\psi + 1)}{b} - \frac{e^{(b-\alpha)t} - 1}{b - \alpha}] \qquad (10)$$

$$m(t) = a(\psi + 1)(1 - e^{-bt}) - \frac{ab(e^{-\alpha t} - e^{-bt})}{b - \alpha} \qquad (11)$$

$$\lambda(t) = \frac{dm(t)}{dt} = \frac{d(a(\psi + 1)(1 - e^{-bt}) - \frac{ab(e^{-\alpha t} - e^{-bt})}{b - \alpha})}{dt}$$

$$= ab(\psi + 1)(e^{-bt}) - \frac{ab(be^{-bt} - \alpha e^{-\alpha t})}{b - \alpha} \qquad (12)$$

For the multi-release OSS system, the mean value function and failure intensity at each release are calculated using Equation (13) to Equation (18).

Release 1

$$m_1(t_i) = a_1(\psi_1 + 1)(1 - e^{-b_1 t_i}) - \frac{a_1 b_1 (e^{-\alpha_1 t_i} - e^{-b_1 t_i})}{b_1 - \alpha_1} \quad (13)$$

$$\lambda_1(t_i) = a_1 b_1 (\psi_1 + 1)(e^{-b_1 t_i}) - \frac{a_1 b_1 (b_1 e^{-b_1 t_i} - \alpha_1 e^{-\alpha_1 t_i})}{b_1 - \alpha_1} \quad (14)$$

Release 2

$$m_2(t_i) = a_2(\psi_2 + 1)(1 - e^{-b_2 t_i}) - \frac{a_2 b_2 (e^{-\alpha_2 t_i} - e^{-b_2 t_i})}{b_2 - \alpha_2} \quad (15)$$

$$\lambda_2(t_i) = a_2 b_2 (\psi_2 + 1)(e^{-b_2 t_i}) - \frac{a_2 b_2 (b_2 e^{-b_2 t_i} - \alpha_2 e^{-\alpha_2 t_i})}{b_2 - \alpha_2} \quad (16)$$

Release k

$$m_k(t_i) = a_k(\psi_k + 1)(1 - e^{-b_k t_i}) - \frac{a_k b_k (e^{-\alpha_k t_i} - e^{-b_k t_i})}{b_k - \alpha_k} \quad (17)$$

$$\lambda_k(t_i) = a_k b_k (\psi_k + 1)(e^{-b_k t_i}) - \frac{a_k b_k (b_k e^{-b_k t_i} - \alpha_k e^{-\alpha_k t_i})}{b_k - \alpha_k} \quad (18)$$

4.4 Results and Discussion

In this section, the objective of result analysis is to test the validity of proposed model on two well-known data sets DS3 and DS4. Two example sections are comparing the results in terms of estimated number of parameter values and goodness of fit measures analyzed in terms of SSE, MSE, MAE, MEOP, AE, AIC, TS and PRR [13] with other three most famous NHPP models named GO model [244] , Inflection s-shaped model [18] and PTZ model [19] .

4.4.1 Example 1: Firefox Dataset Analysis

This section analyzes the proposed model on three releases of Firefox open-source software failure data-sets. In each release, the amount of testing effort is varying with the change in the amount of fault content and functional complexity of the released version.

Estimated parameter values and goodness-of-fit measurements of the proposed model are shown in Table 4.1, Table 4.2 and Table 4.3**.** The goodness of fit of the proposed model is calculated in terms of eight criteria SSE, MSE, MAE, MEOP, AE, AIC, TS and PRR. In Firefox 3.0 release, the proposed model is performing very well in seven criteria except in terms of AE where Inflection S-shaped model and PTZ model behaves well. Among eight measures of fitness, the proposed model has clear- cut outperforms by 87.25% than other used models. In Firefox 3.5

release, the proposed model is having lesser values of SSE, MSE, MAE, AIC, TS, and PRR. It has major beating values in terms of goodness of fit measures except for values of MEOP where Inflection s-shaped model performed well and AE values where inflection s-shaped model and PTZ model performs well. Overall, the proposed model is having 75% major beating values. In Firefox 3.6, the proposed model has major beating criterion except in term of AIC where Inflection s-shaped model performed better than the proposed model. Fig. 4.2, Fig. 4.3 and Fig. 4.4 shows the estimated number of faults using proposed model. Overall, the proposed model is performing 87.5% superior to other used models.

Table 4.1 Result Analysis using Firefox 3.0

Model Name	**Estimated parameter values**	**SSE**	**MSE**	**MAE**	**MEOP**	**AE**	**AIC**	**TS**	**PRR**
GO model	110.067, 0.991, 0	335052.21	14567.487	3.156	3.025	0.979	5.481	1838.632	0.985
Inflection s-shaped model	103.466, 0.992, 0.291	366508.08	15935.133	3.218	3.084	0.95	5.472	2162.14	0.9944
PTZ model	131.454, 0.987, 0.241, 0.09664, 0.000490	265036.58	11523.329	2.819	2.702	0.956	5.417	2067.275	0.9857
Proposed model	118.711, 0.936, 0.264, 3.786	63468.217	2759.487	1.458	1.397	0.977	5.459	899.77	0.9737

Table 4.2 Result Analysis using Firefox 3.5

Model Name	**Estimated parameter values**	**SSE**	**MSE**	**MAE**	**MEOP**	**AE**	**AIC**	**TS**	**PRR**
GO model	51.186, 0.997	1.65E+04	3.44E+02	0.74	0.725	0.993	5.446	242.413	9.947
Inflection s-shaped model	43.803, 0.983, 0.295	6.85E+04	1.43E+03	34.009	0.199	0.974	5.415	493.99	0.953
PTZ model	21.938, 0.993, 0.277, 0.000389, 0.000548	1.83E+04	3.81E+02	0.647	0.633	0.983	5.408	255.23	0.957
Proposed model	69.345,0.973, 0.292, 2.894	1.25E+04	2.61E+02	0.529	0.519	0.986	5.404	211.167	0.939

Table 4.3 Result Analysis using Firefox 3.6

Model Name	**Estimated parameter values**	**SSE**	**MSE**	**MAE**	**MEOP**	**AE**	**AIC**	**TS**	**PRR**

GO model	119.863, 0.999	4.53E+05	1.01E+04	1.995	2.392	0.982	5.452	1346.2	0.985
Inflection S-shaped model	124.494, 0.966, 0.292	2.44E+05	5.42E+03	1.882	1.841	0.986	5.416	987.36	0.981
PTZ model	135.836 0.997, 1.414755,0.00058	3.17E+05	7.06E+03	2.098	2.052	0.984	5.451	1126.9	0.539
Proposed model	155.922, 0.997, 0.274, 1.746	2.31E+05	5.14E+03	1.84	1.8	0.986	5.419	962.24	0.981

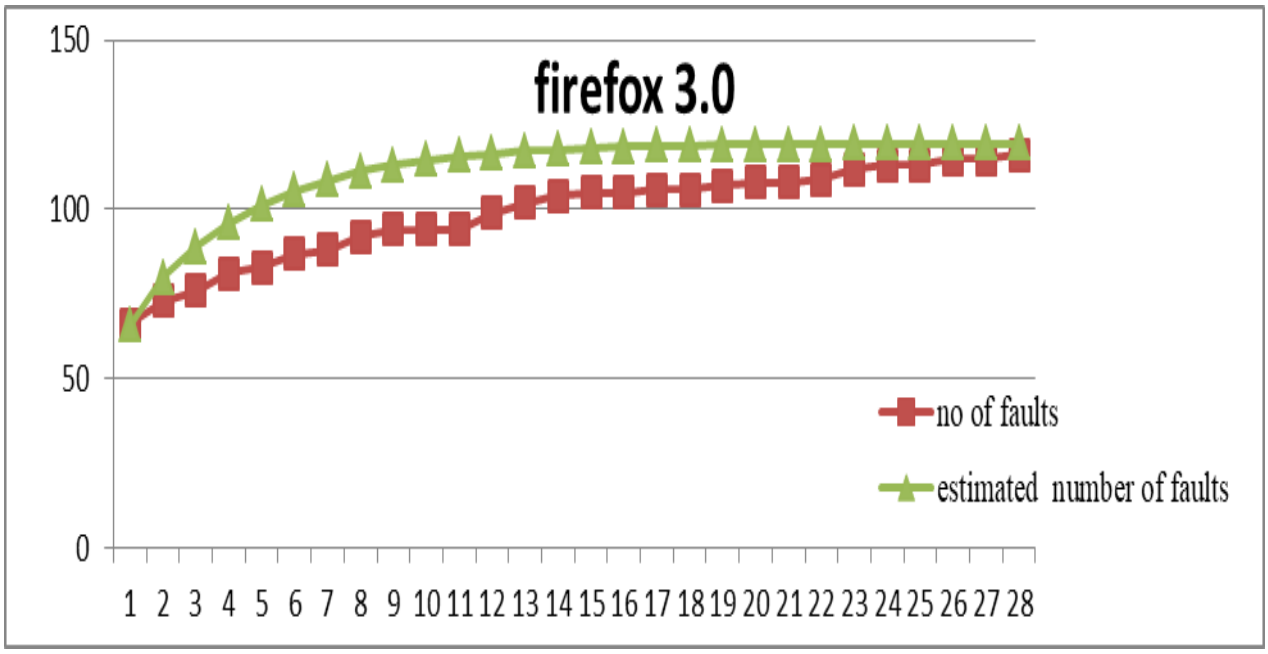

Fig. 4.2 Estimated Number of Faults using Firefox 3.0

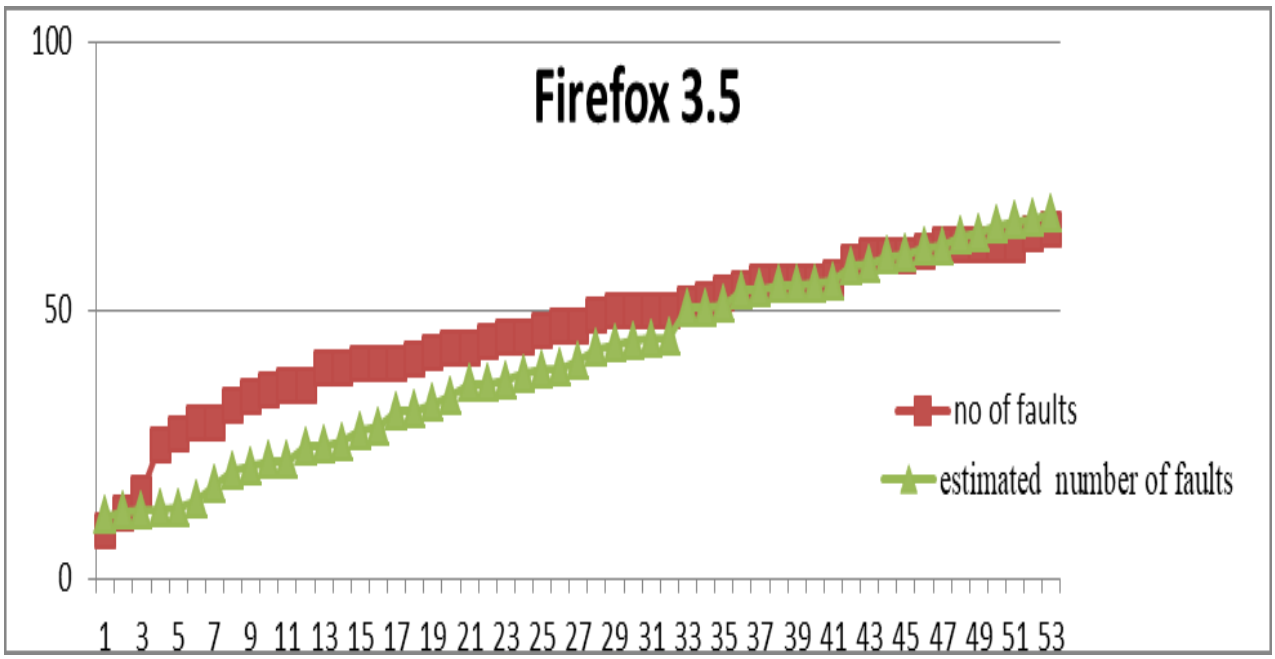

Fig. 4.3 Estimated Number of Faults using Firefox 3.5

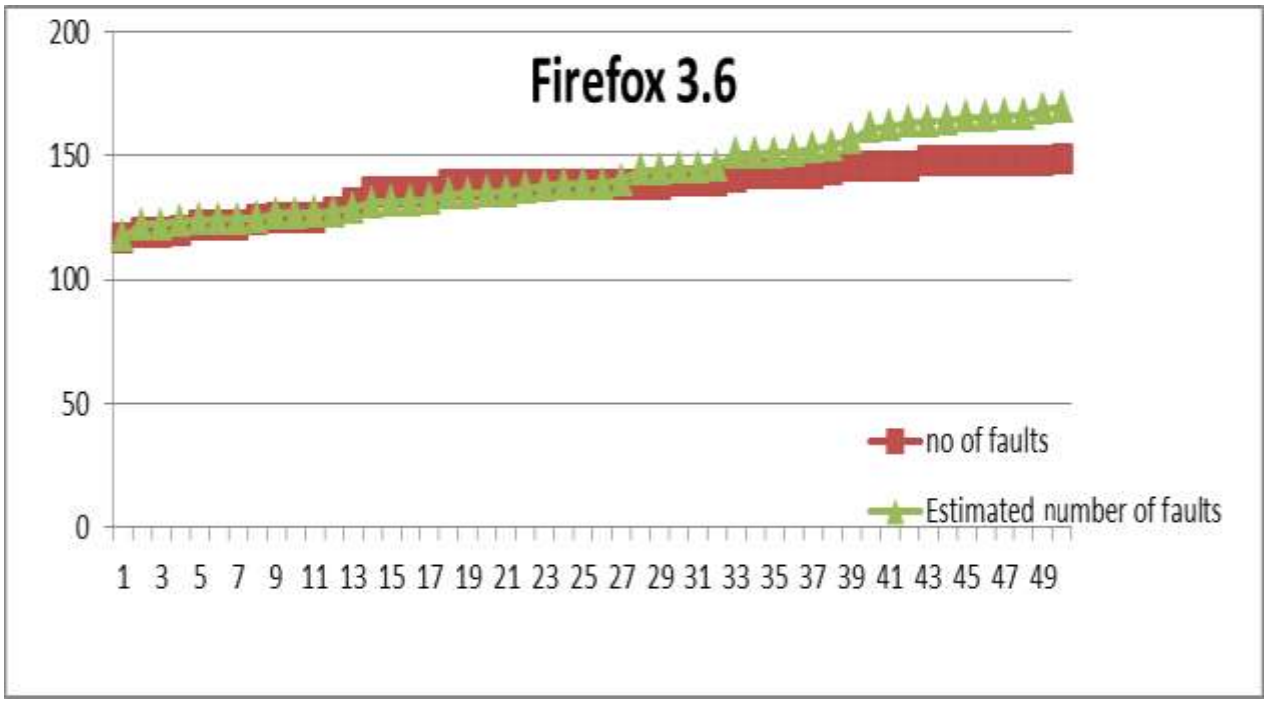

Fig. 4.4 Estimated Number of Faults using Firefox 3.6

4.4.2 Example 2: Result analysis for Genome failure data-sets

In this example failure data of the Genome project is used to check the validity of the proposed model. Three releases of Genome OSS system are used to test the proposed model behaviour. Using Genome 2.0 release failure data-set, the proposed model is showing major beating fitness values shown in Table 4.4 and found to be about 62.5% better than all other models

used. Proposed model behaves well in terms of SSE, MSE, MAE, MEOP and TS values except in terms of AE, AIC and PRR values where other models suits well to the Genome 2.0 data-set. Using Genome 2.2 data-set, the proposed model is behaving very well as shown in Table 4.5 in terms of all comparison criterions except AE values where all other models are performing better. Overall, proposed model is about 87.50 % more suitable on Genome 2.2 data-set. In Table 4.6, the proposed model is analyzed using Genome 2.3 data-set. Results are proving that the proposed model fits very reasonably on Genome 2.3 data-set except in terms of SSE and MLE where Inflection s-shaped model and GO model fits well. Estimated errors using the proposed model using three releases of Genome data-set is shown in Fig. 4.5, Fig. 4.6 and Fig. 4.7. In Genome data-set estimated number of faults are found to be much closer to the actual number of faults in a given failure data-sets and proves suitability of the proposed model.

Table 4.4 Result Analysis using Genome 2.0

Model Name	**Estimated parameter values**	**SSE**	**MSE**	**MAE**	**MEOP**	**AE**	**AIC**	**TS**	**PRR**
GO model	58.034, 0.947	2.31E+04	1.28E+03	2.229	2.117	0.479	5.996	660.86	0.974
Inflection s-shaped model	90.739, 0.934, 0.0014	1.67E+04	9.28E+02	1.874	1.775	0.513	5.985	561.79	0.968
PTZ model	71.965, 0.993,0.0001, 0.0076, 0.000348	2.28E+04	1.27E+03	2.227	2.11	0.484	5.997	656.79	0.475
Proposed model	88.366, 0.958, 0.002, 2.746	6.37E+03	3.54E+02	1.399	1.326	0.676	5.999	347.009	0.959

Table 4.5 Result Analysis using Genome 2.2

Model Name	**Estimated parameter values**	**SSE**	**MSE**	**MAE**	**MEOP**	**AE**	**AIC**	**TS**	**PRR**
GO model	19.938, 0.995	9.20E+03	7.08E+02	2.194	2.037	0.334	5.983	399.722	0.9648
Inflection s-shaped model	63.969, 0.935, 0.005	1.35E+04	1.04E+03	2.613	2.426	0.264	5.99	483.38	0.9702
PTZ model	55.048, 0.951, 0.0016,0.029,0.007576	8.81E+03	6.77E+02	2.26	2.099	0.395	5.996	391.782	0.9665
Proposed model	54.298,0.928, 0.0082, 2.549,	7.32E+02	56.27538	0.79	0.734	0.671	5.943	12.699	0.9077

Table 4.6 Result Analysis using Genome 2.3

Model Name	Estimated parameter values	SSE	MSE	MAE	MEOP	AE	AIC	TS	PRR
GO model	44.895, 0.983	5.37E+03	2.98E+02	1.287	1.219	0.681	5.993	318.19	0.957
Inflection s-shaped model	49.615, 0.916, 0.0002	3.63E+33	2.05E+02	1.123	1.084	0.731	5.991	264.43	0.953
PTZ model	66.211 , 0.946, 0.005, 0.00008,0.00035	9.50E+03	5.28E+02	1.62	1.54	0.631	6.000	374.24	0.966
Proposed model	54.265, 0.928, 0.003, 0.416,	3.70E+03	2.01E+02	1.149	1.069	0.713	5.99	261.82	0.952

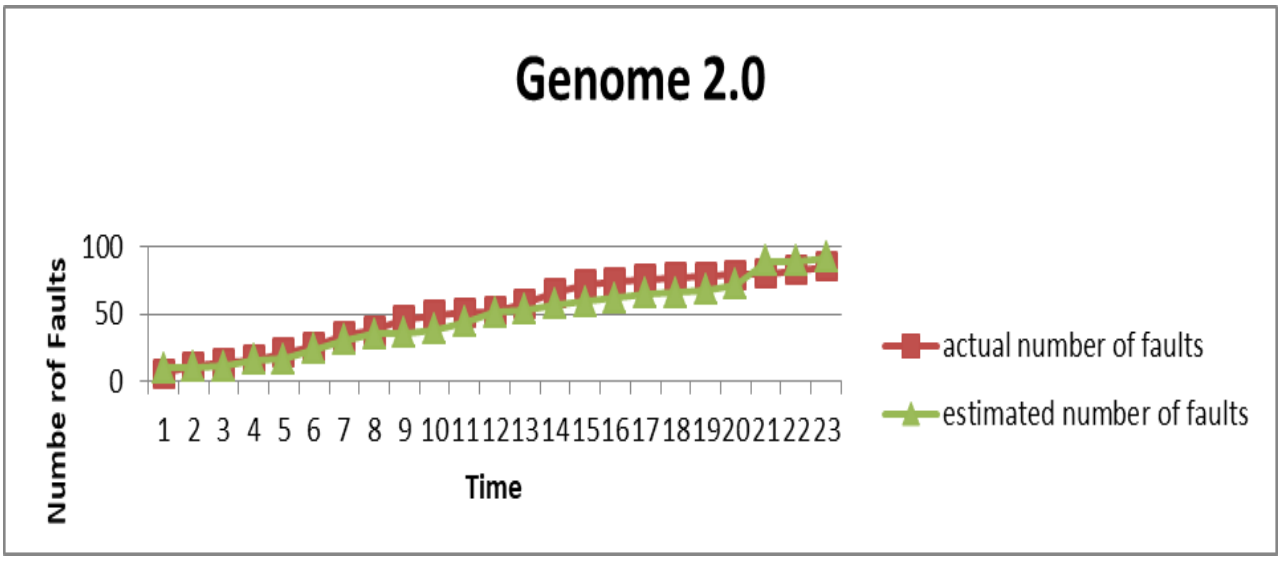

Fig. 4.5Estimated Number of Faults using Genome 2.0

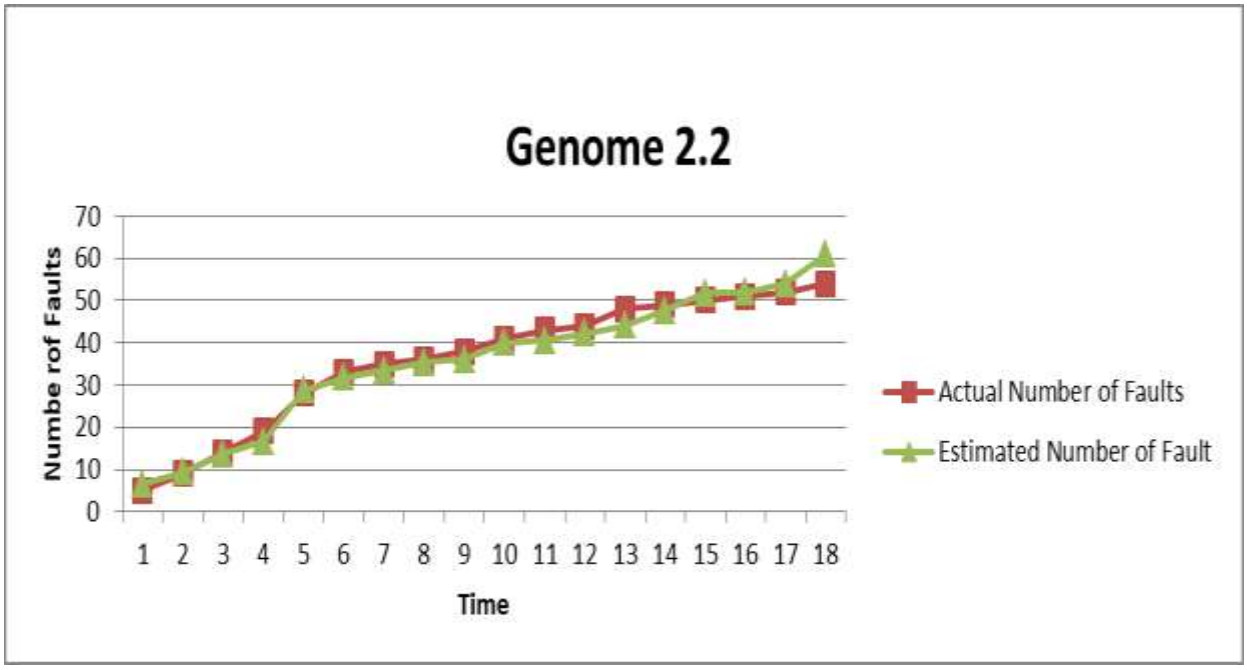

Fig. 4.6Estimated Number of Faults using Genome 2.2

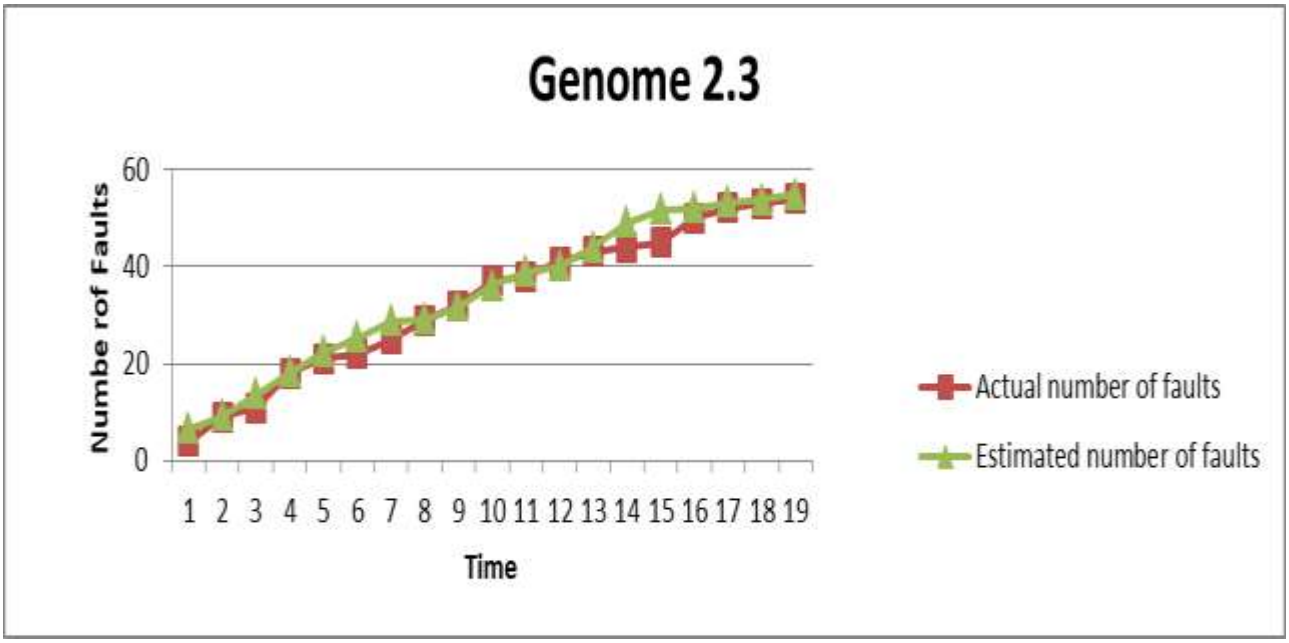

Fig. 4.7 Estimated Number of Faults using Genome 2.3

4.5 Conclusion

The authors proposed a new NHPP software reliability model for open source software based on new testing effort behaviour based fault content function. The fault content has been modified and assumed to be an exponential function with the impact of fault introduction probability and testing effort coefficient. Incorporated testing effort coefficient is depicting that in each new released version due to added functionality and number of fault content, there is a need of change in amount of incorporated testing effort, more is the effort devoted in software testing and debugging, more reliable software will be released in future. Impact of testing effort has been reasonably incorporated to fit well in the latest software development technologies and environments. Results are showing the suitability of the proposed model on two well-known Firefox and Genome open source software failure datasets. The proposed model is found to be beating other models in most of the goodness of fit criterion and fits very well for software reliability estimation.

Chapter 5 PARAMETER ESTIMATION ALGORITHM

The software reliability prediction by mathematical models is entirely centered on the parameter values. From the survey in the field of parameter estimation it is found that meta-heuristic algorithms are performing better than other traditional methods for the solution of optimization problems. Authors in this chapter proposed a new algorithm based on ecological space, Differential Evolution and Artificial Bee Colony for optimization of the parameter values. The exploration capability in Artificial Bee Colony algorithm has been improved by introducing the concept of ecological space. Onlooker bee ecological space is one of the important factors for evolution and reflects the expansion of individual bee in search space. Differential Evolution technique provides the diversity of bee's population and faster convergence. Proposed algorithm has been tested with four standard failure datasets. Proficiency of proposed algorithm is also compared with other meta-heuristic algorithms namely Artificial Bee Colony, Genetic Algorithm and Particle Swarm Optimization. Further validation of proposed algorithm is done through comparing its efficiency with hybrid PSO and Gravitational Search Algorithm. Simulation results verify that proposed hybrid algorithm is very much effective in field of software reliability estimation and would be a competitive one among meta-heuristic optimization algorithms.

5.1 Introduction

Reliability is the major quality aspect of the software. Software reliability estimation process must be precise in order to provide the information to the software developers like release time of the software, extent of man-hour consumption etc. Accurate software reliability estimation is mainly dependent on the selection of optimum parameter values of the models.

The techniques for optimizing parameter values of the software reliability models are available through various classical methods of parameter estimation. These methods are based on the number of constraints, may fall in local maxima and do not converge to global maxima in the multimodal cases. Alternative to these classical mathematical optimization methods, are the nature inspired optimization algorithms [6], [173], [245] to solve non-linear, non-differential and multimodal problems.

Nature inspired meta-heuristic algorithms are assembled into four main groups that includes algorithms based on evolutionary principals, Swarm Intelligence behaviour, physics

phenomenon's and Human Intelligence behaviour. For example, Holland [246], Price and Storn[10] proposed algorithms related with the natural evolutionary principal. Khelif [247]et al, Basturk and karaboga[191], Fahad and Mohamed[248], Ozturk and Karaboga[162], Kennedy and Eberhart [249] proposed few Swarm Intelligence behaviour based algorithms. Gelatt et al. [154] proposed a physics phenomenon based algorithm. Lim and Isa[178] proposed an algorithm evolved from the human Intelligence behaviour. Among these groups most promising optimization capability are from Swarm Intelligence and Evolutionary principal based algorithms. Darwin's principal based evolutionary algorithms are like GA, GP and DE etc. Swarm Intelligence behaviour based algorithms imitates the social activities of the creatures like flocks of animals, birds and amphibians etc. and utilizes the social ability of learning and adaptation. These algorithms are the most propitious area of research for numerical optimization and can be successfully applied for parameter optimization of software reliability models. There are numerous such algorithms available, how researchers should use them? To answer this, crucial is to compare these algorithms.

For designing a new algorithm the foremost goal is to know how behaviour is pursuing the evolutionary and swarm intelligence capabilities. To obtain optimum solution, there should be equilibrium between exploration and exploitation geographies of an algorithm. The necessary condition for swarm intelligence is self-organization (comprises feedback, variations and multiple collaborations) and division of labour. For self-organization, fluctuations are vital to provide a level of randomness and for finding the new better solutions effectively. It is important to get rid of stagnation for enhancing the exploitation. There is a need to find the factors in the algorithm which are playing a key role in calculating the fitness probabilities of the candidate solutions. These factors need to be related to the ecological space fitness as the survival of fittest is not only the driving factor of evolution as discussed in [8].

A new ecological space based hybrid Swarm Evolutionary algorithm is proposed in this chapter. Proposed algorithm is centered on the social behaviour of artificial bee colonies given in Yang [6] and evolutionary behaviour of DE algorithm given by Price and Storn [10]. The swarm intelligence of employee bee is enhanced for providing exploitation to provide better local search of the neighbour-hood positions using the evolutionary principle based DE algorithm. Onlooker bee phase has been improved by incorporating a new factor, showing the fitness probability of

the ecological space. The implementation results on the reliability models are showing the validity of the proposed algorithm for reliability estimation.

The chapter is organized into various sections: Section 1 describes introduction part. Section 2 discusses meta-heuristic algorithms in different domains. Section 3 describes a new proposed hybrid algorithm. Section 4 describes experimental setup. Section 5 discusses results and comparison with other algorithms. Last section provides conclusion of the work done.

5.2 General Study of Meta-heuristic algorithms

Although, there are number of meta-heuristic algorithms available in the literature but in the field of software reliability assessment, there are only few of the algorithms that has been implemented for software reliability model parameter estimation. Authors analysed capability of few of the well-known algorithms that has been successfully used for solution of optimization problem in various domains for parameter estimation. This section discusses these meta-heuristic algorithms.

5.2.1 Artificial Bee Colony

ABC algorithm was proposed by Karaboga in 2005 based on the foraging behaviour of honeybee for numerical optimization problems [191]. The algorithm welfares includes its Robustness, flexibility and simplicity, easy implementation with fewer number of parameters required.

The ABC artificial agents are categorized into three parts. Each part is performing its task for finding the nectar as the food source. These ABC agents include the employee bees, Onlooker Bees and the Scout Bees. The process of finding the food source is as follows-

1. Exploitation process (Evaluating the nectar quality)-The exploitation is performed by the employee bees and the onlooker bees. The process involves first sending the employee bee to the initial location of food source and then sharing the information regarding the food source as the probability proportion of the profitability of the source of food with the bees in the hive. Employee bees then choose the neighbour-hood positions of the memory for another food source. Onlooker bees after getting the information of the food source in the hive employ itself at the most profitable source and calculates the nectar quality. There is the positive feedback and negative feedback of the food sources

depending on the amount of nectar at the food source, if it increases then more number of onlooker bees will move at that food source and if it decreases then exploitation of the food source will be stopped by the bees.

2. Exploration process (Discovering new food source) - A random search process is carried out by the scouts for searching the new food source.

Three control parameters i.e. the size of the swarm, limit, and the maximum number of iterations are used in this work for the implementation of ABC algorithm. Fig. 5.1 illustrates the flowchart of Artificial Bee Colony algorithm.

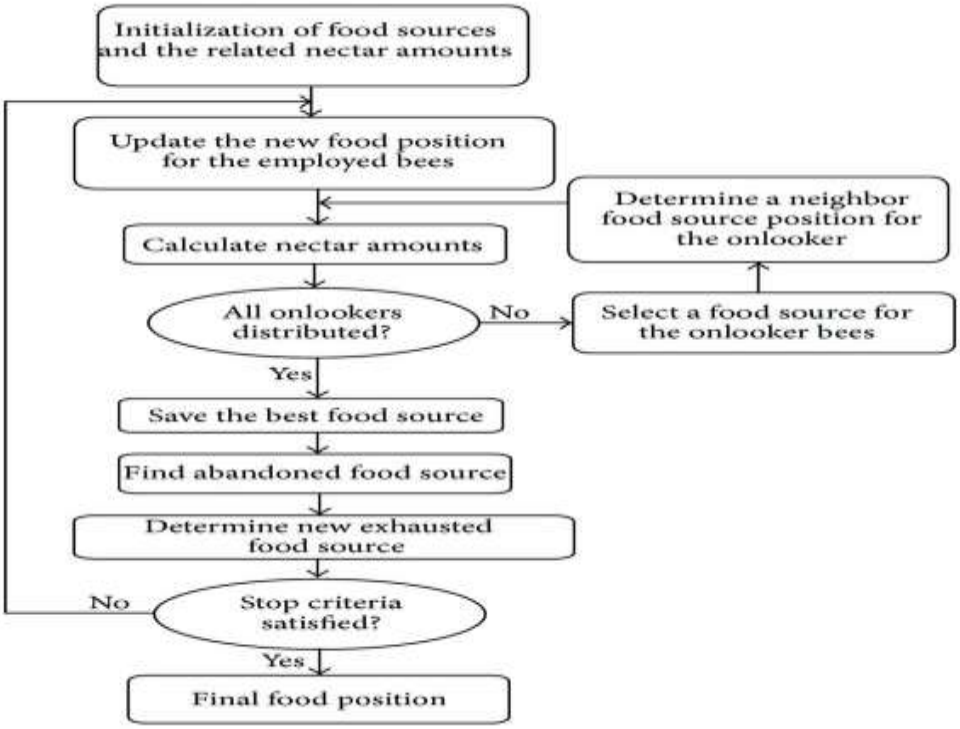

Fig. 5.1 Artificial Bee Colony Algorithm

5.2.2 Particle Swarm Optimization

Particle swarm optimization is a meta-heuristic method to optimize the problem solution iteratively [249]. Only few of the assumptions are to be made for the solution that is to be optimized. The problem is solved by having a candidate solution in terms of particles and then it moves the particles in the search space by using mathematical formulas for the position and

velocity updates. Each movement of the particles is influenced by the particles local best position and also moves toward the global best positions which are considered to be the better positions found by other particles in swarm. By this process whole swarm is expected to be moved towards the best solution. Fig. 5.2 illustrates the general process of PSO algorithm.

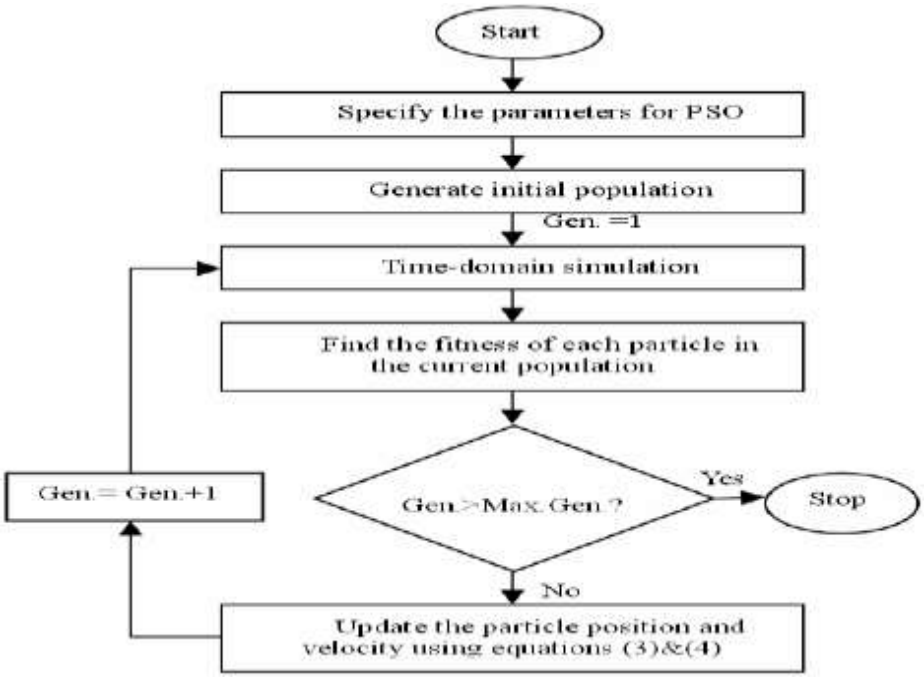

Fig. 5.2 Particle Swarm Optimization Algorithm

5.2.3 Differential Evolution Algorithm

Differential Evolution is developed by Ken Price and Storn in 1997 for global optimization of the problems [10]. DE is very much similar to Genetic algorithm as it is also using three operators mutation, cross over and selection, but with a difference that genetic algorithm trusts on cross over operation while differential evolution relies on mutation operation. Fig. 5.3 describes the general process of Differential algorithm.

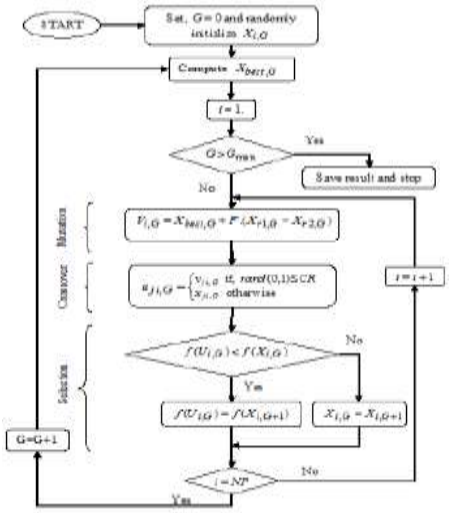

Fig. 5.3 Differential Evolution Algorithm

5.2.4 Hybrid Particle Swarm Optimization and Gravitational Search Algorithm

Using gravitational affects, hybrid particle swarm optimization and gravitational search algorithm helps in finding the best solution for guiding the heavy masses towards the global optimum positions. Fig. 5.4 illustrates the process of PSOGSA algorithm. This process also increases the speed and overall movement of particles and masses as well and will enhance the exploitation capability of PSOGSA algorithm. Especially, some works [194], [195] applied the idea of PSO on GSA (which is memory less, originally), and modified the GSA velocity term by combining it with the PSO velocity term (which is memory based).

Fig. 5.4 Hybrid PSOGSA Algorithm

After studying literature of the evolutionary algorithm including the ABC, DE, Particle Swarm optimization, Hybrid particle swarm optimization and gravitational search algorithms, authors found number of advantages of artificial Bee colony optimization and decided to find out the application of ABC for parameter optimization of software reliability growth models as the reliability of the software is a key concern while developing any software product. It is found that ABC algorithm being a simple algorithm has a premature convergence with unbalanced exploration and exploitation process. There is an advantage of using DE in providing the diversity of the population and in providing the improved local search capability to the ABC algorithm. DE is also having faster convergence capability then ABC algorithm. The proposed work in this section hybridizes Artificial be colony algorithm with the best features of DE algorithm.

5.3 Proposed Algorithm

A new ecological space based hybrid Swarm-Evolutionary algorithm has been proposed for software reliability models parameter estimation. ABC algorithm is simple and flexible swarm intelligence based algorithm having fewer numbers of parameters, but needs further modification to improve its efficiency in terms of exploitation and exploration. ABC algorithm may be modified by enhancing the exploitation and exploration capability of employee bees and the onlooker bees. Food source information shared by the employee bees must be accurate. Employee bees must calculate and share the candidate ecological living space fitness information. There is also a need to incorporate a candidate's ecological living space fitness factor for fitness probability calculations by the onlooker bees so that fitness probabilities could be calculated in precise. This ecological space factor will enhance the exploration capability in ABC.

Proposed hybrid algorithm combines DE algorithm capabilities with ABC algorithm. DE algorithm has the proficiency in improving the diversity of the population by providing an equal probability of being selected as a parent solution from the candidate solutions in space and could better improve local search capability of the employee bee phase in ABC algorithm. DE algorithm also has a faster convergence capability then other evolutionary algorithms and could better fit in ABC algorithm for enhancing its exploitation capability.

5.3.1 Mathematical Formulation for Artificial Bee Colony Algorithm

1. Initialization phase- This phase involves the initialization of the swarm and setting of the appropriate values for the control parameters in the algorithm. Initialization is done using equation (1).

$$x_I(i)=lb_i+rand(0,1)\times(ub_i-lb_i) \quad (1)$$

2. Employee Bee Phase: Each employee bee searches for the new food source with the large nectar amount. When the new food source is found its fitness value is calculated and new food source is defined by the equation (2).

$$y(i) = x_l(i) + \phi_i(x_b(i) - x_c(i)) \quad (2)$$

where x_c is the value of the food source selected randomly and
ϕ_i is a random number and b $\neq c$ and c=(1...n(number of employee bee))

Greedy selection is done after obtaining the new value of the food source, if the value of the difference between (x(i)-y(i)) is higher, then exploration will happen and if this difference is small then exploitation process will happen. The fitness value f (xi) is given by equation (3)

maximum fitness value is calculated as
$f(y_i) \geq f(x_i)$ then $x_i = y_i$ and

$$f(x_i) = f(y_i) \quad (3)$$

3. Onlooker Bee phase: The fitness value probabilities are calculated and these calculated probabilities and information that has been shared by the employee bees are used by the Onlooker bees for selecting their food sources .The food source whose fitness valve is highest is selected by the onlooker bees.

4. Scout Bee phase: The scout bees are unemployed bees and replaces the abandoned food sources that is the food source that has not been improved from certain number of cycles.

5. Food source memorization phase: In this phase best fitness value and the positions related with that value are memorized.

6. Termination criteria phase: Termination will happen if the termination condition is met and otherwise repeat from employee bee phase till the termination condition happens.

The implementation of ABC requires the balance between the exploitation and exploration process. The exploitation process done by the employee bees and onlooker bees during their execution requires the enhancement of the local search capability i.e. the exploration. The fitness probability calculation by the onlooker bees requires additional attention so that exact fitness of the food source can be calculated.

5.3.2 Mathematical Differential Evolution Algorithm for Global Solution of a Problem

Mutation phase-The new solution from the original solution for expanding the search space is created using equation (4)

$$V_i = v_{i1}, v_{i2}.....v_{iD}$$
$$V_i = X_{r1} + C(X_{r2} - X_{r3}) \quad (4)$$

Where r1,r2,r3 are the random integer numbers different from the i value

C is a real valued number ranging from {0...2}.and used in scaling the value of $(X_{r2} - X_{r3})$.

Cross over phase-This phase is used for enhancing the diversity of the population. Here the target vector is diversified with the mutant vector and a different trial vector is generated having more diversity then the mutant vector. The trial vector is given as in equation (5).

$$O_i = o_{i1}, o_{i2}.....o_{iD}$$
$$o_{ij} = \begin{cases} v_{ij} & \text{if } r_{ij} \le C_r \text{ or } j=j_{random} \\ x_{ij} & \text{otherwise} \end{cases} \quad (5)$$

where r_{ij} is random number between 0..1, C_r is the probability of the crossover and

j_{random} ensures that the randomly chosen number in X_i is atleast from V_i

Selection phase - All the solutions in the solution space are having an equal probability of being selected as the parent without considering their fitness value. So selection phase is to find which one among v_{ij} or x_{ij} should be the member of the next generation using the equation (6).

$$X_i' = \begin{cases} V_i & \text{if } f(V_i) \ge f(X_i) \\ X_i & \text{otherwise} \end{cases} \quad (6)$$

where f() is the fitness function and X_i' is the new individual of the population

5.3.3 Hybridization of Proposed Swarm-Evolutionary Algorithm

5.3.3.1 Employee Bee phase

A level of randomness is essential for maintaining diversity throughout the search space. The quality of the updated solution is influenced greatly by the selection of random solutions. In basic ABC algorithm, solution is updated only using random solution of the current search space. Rather than depending only on this random selection, to maintain population diversity employee bees find updated positions of food source from the current positions using DE method, in this each particle breeds locally new and enhanced information from the swarm for finding the neighbour-hood positions. The position update in Employee bees is done as below:

First employee bees search for the updated positions is done using Eq. (7).

$$Y_1^{t+1}(i) = P_l(i) + \phi_i(P_l(i) - P_m(i)) \quad (7)$$

And then employee bees again searches for the neighbour-hood solution of the current position for each employee bee using DE. New trial vector is generated first using mutation and then cross over is done for new offspring generation. At last greedy selection is done from the current and offspring solution for finding better candidate selection. Using Eq. (8), Eq. (9) and Eq. (10) neighbour-hood solutions are selected.

Generation of m neighbors

For k_0 =1-m(neighborhood size) do

Mutation

$$V^t_{i,k_0} = Y_1^t \mu 1 + C(Y_1^t \mu 2 - Y_1^t \mu 3) \quad (8)$$

For (d'=1-D)do

Cross over

$$\text{if random}_{ij} \le C_\mu \text{ or d'=j}_{random} \text{Then } O^t_{k_0,\,d'} = V^t_{k_0,\,d'} \text{ else } O^t_{k_0,\,d'} = Y_1^{t+1}{}_{k_0,d'} \quad (9)$$

End for

Selection

$$\text{If fitness } (Y_1^{t+1}{}_{k_0}) > \text{fitness}\left(O^t_{k_0}\right) \text{Then } O^t_{k_0} = Y_1^{t+1}{}_{k_0} \text{ else } O^t_{k_0} \quad (10)$$

End for

5.3.3.2 Onlooker Bee phase

In the original ABC algorithm [250], Onlooker bee selects the food sources depending on the information about nectar amount shared by employee bee in the hive. But this is not only the factor for calculating the fitness probabilities of the food source, ecological space fitness information is also one of the driving factor showing the nectar quality and is being shared by employee bees in hive in the proposed algorithm.

As per with the state of art algorithms and theories in the literature, ecological living space is a factor that is necessary for the organism (food source) to flourish in the environment and it will enhance the diversity by matching the available living space as described in Michael et al.[8]. The candidate solution and ecological diversity are very much closely related. More favourable is the ecological conditions more quality food will be available to the bees. The diversity will appear to increase depending on the favourable ecological space. The diversity is assessed from ecological, morphological or genetic perspective. Understanding correlation among them can define better the history of life of candidate solutions on earth. This is the first application exploring the link between ecological diversity and candidate solution. In the proposed concept candidate food source fitness probability is calculated by onlooker bees using Eq. (11).

$$\left(0.5 * \text{fitness}\left(i\right) / \max \text{fitness}\right)+\eta_i \qquad (11)$$

Further onlooker bees make position update using ecological space factor as given in Eq. (12).

$$Y_1^{t+1}(i)=P_l(i)+\phi_j(P_l(i)-P_m(i)) + \eta_i(P_g(i)-P_l(i)) \qquad (12)$$

Where a factor η is the candidate's available living ecological space fitness probability

Higher fitness probability of the ecological living space will increase the chance of food source as being selected by the onlooker bees.

5.3.3.3 Scout Bee phase

Scout bee on random regenerate the exhausted food source solutions in the space using ecological space fitness information and other information from the employee bee in the solution space.

Fig. 5.5 is a two dimensional space created using the cellular grids. Each position in the cellular grid is a possible candidate solution. Current positions of the food source $Y_{1(i)}$ and updated positions of food source $Y_{1(i+1)}$ are represented by circle and the circle with plus indicate the possible number of neighbours found using the cross over process. The $Y_{1\mu1}, Y_{1\mu2}$ and $Y_{1\mu3}$ cells in the grid are selected as the possible random candidate solutions and are selected for mutation to generate the V_i vector. This whole process will terminate after fixed number of iterations and are equal to the swarm size specified.

5.3.4 Framework of the proposed algorithm

Proposed hybrid algorithm as shown in Fig. 5.6 and Fig. 5.7 starts with the initialization of the control parameters, swarm size(S), number of available food sources, limit, maximum number of iterations, neighbour-hood size (m). Other parameters used in the algorithm are ϕ, C and D. After initialization phase each employee bee searches for the food source using Eq. (7) and then using DE (Eq. (8), Eq. (9) and Eq. (10)) neighbour-hood is defined for each employee bee with three random states in the grid using mutation vector, cross over vector and selection vector. Then onlooker bee phase takes place that calculates the fitness probability of each state using Eq. (11) and then selects the food source with highest fitness probability. If fitness probability of food source still is not satisfying the termination criterion then onlooker bee updates positions using Eq. (12). At last scout bee regenerates the exhausted food source in the solution space and finally outputs the best generated solution.

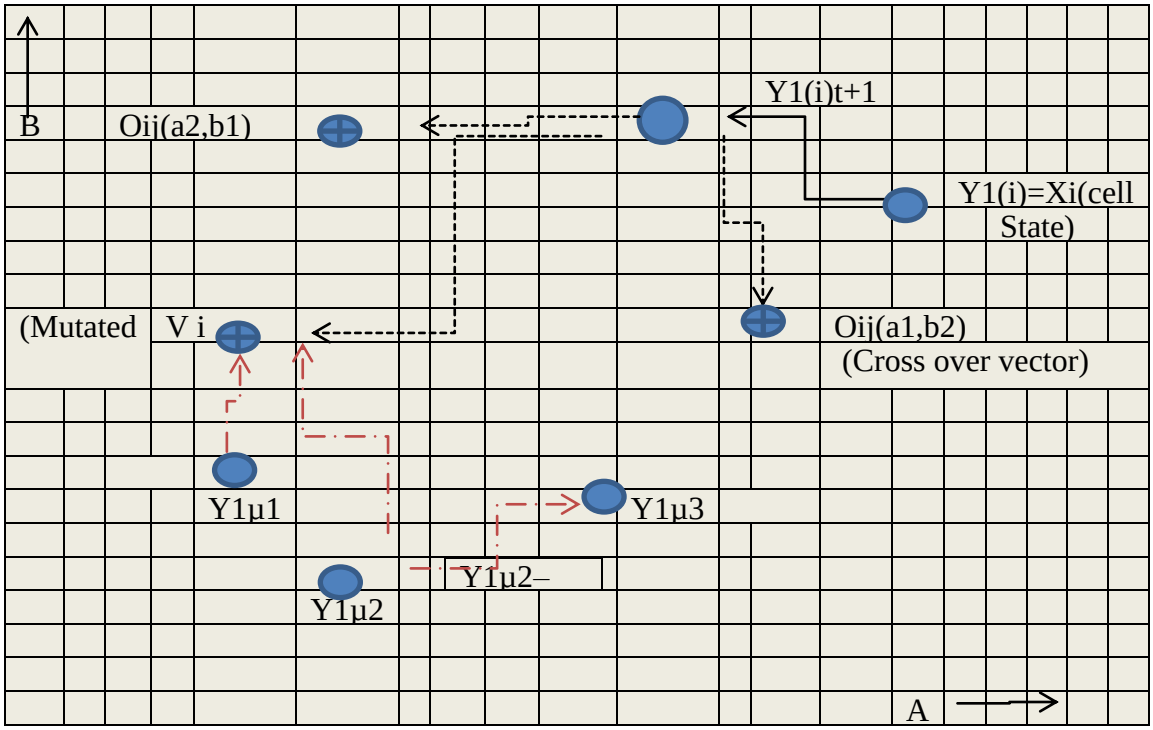

Fig. 5.5 2D View of Hybrid DE assisted ABC Algorithm

Initialize the swarm and Set the value of the control parameters
While (termination condition) do
//Employee bee phase :
for (each particle i= $(1\ldots..S\ (\text{swarm size}))$)
Searches the new food source solution for each employee bee in the swarm using Eq. (7)
Go for the neighbor-hood solution of the current position for each employee bee
for (k_0=1-m(neighbour-hood size)
do apply mutation to generate trial vector V^t_{i,k_0} as solution using Eq. (8)
Apply crossover to generate offspring $O^t_{k_0,d'}$ using Eq. (9)
do greedy selection for finding the better solution from the current and generated solutions using Eq. (10)
end for
end for
//Onlooker bee phase:
for (each of onlooker bee's solution)
Based on fitness value probability calculated using Eq. (11),
Select the food source having highest fitness probability
Then generate neighbor-hood solution based on the probability calculation and position update using Eq. (12)
end for
//Scout bee phase:
Scout bee on random regenerate the exhausted food source solutions in the space using ecological space fitness information and other information from the employee bee in the solution space.
end while loop
output the best solution

Fig. 5.6 Ecological space based Hybrid Swarm-Evolutionary (DE assisted ABC) Algorithm

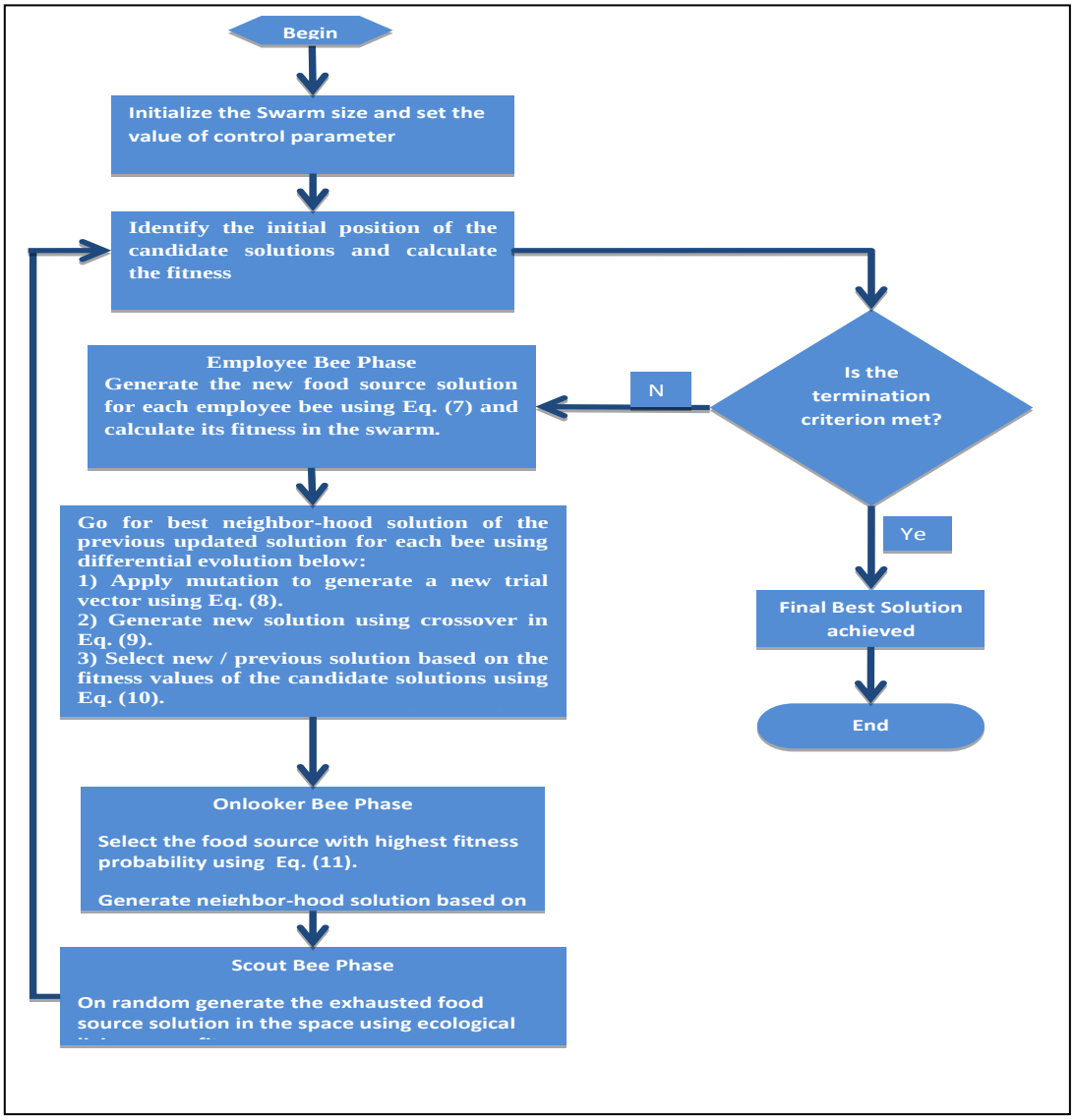

Fig. 5.7 DE Assisted ABC Algorithm

5.4 Experimental Setup

The implementation of the proposed algorithm is done on Intel(R) Core (TM) i5(5th gen)-62000 CPU 2.40 GHz with 4 GB RAM and 64 bit windows architecture, x64 based-processor. Proposed algorithm has been implemented on most generally used perfect debugging

and imperfect debugging behaviour based models. The models used for experimentation work are:

1. Perfect debugging behaviour based two parameter Goel and Okumotto model [244]

2. Imperfect debugging behaviour based three parameter Inflection s shaped model [18]

3. Imperfect debugging with fault introduction behaviour based five parameter PTZ model [19].

The performance of the proposed algorithm has been compared with other meta-heuristic algorithms like PSO, DE, ABC and hybrid PSOGSA. Four bench-mark datasets [51] have been used for experimental analysis.

5.5 Results and Discussion

The statistical results for estimated model parameter values, Sum of squared errors, Mean square error and elapsed time taken by algorithms in seconds are shown using different cases. The best results are shown in bold for each of the tables. Implementation of the algorithms has been done with more than 1000 iterations.

5.5.1 Result analysis

5.5.1.1 Case1. Analysis of GO Model Using Real Time Command and Control System

This analyzes various meta-heuristic algorithms using GO model and DS5. Fig. 5.8 is showing the results of estimated number of errors i.e. the mean value function, calculated in the software at time t and proves that the results estimated by the hybrid algorithms (PSOGSA and ABCDE) are much better and close to the actual number of detected errors at time t in the software than the other used algorithms. The reliability estimation at each detected faults ranges from 78-90%.

5.5.1.2 Case2. Analysis of Inflection-S Shaped Model using Real Time Command and Control System

The behaviour of all the algorithms for mean value function analysis using inflection s-shaped model and DS5 is shown in Fig. 5.9 and the best results are found using the PSO, Hybrid PSO-GSA and Hybrid ABC-DE (HABCDE) algorithms. The reliability behaviour of the software after estimating the errors in the system are found to be increasing from 79 to 93% in proposed HABCDE algorithm.

The implementation results for parameter estimation using evolutionary approaches for the GO and inflection s-shaped model are given in bold in Table 5.1 and Table 5.5 respectively and are found to be very much satisfactory. The MSE and the Sum of squared errors calculated are shown in Table 5.2, Table 5.3 and in Table 5.6 and Table 5.7 respectively for both the models and are showing that the hybrid algorithms are having less error as compared to the other approaches used. Hybrid ABCDE is having better MSE values in average case and satisfactory in other cases, SSE values in the worst case are found to be good using the hybrid algorithms. For inflection s-shaped model best values for MSE and SSE are estimated using the Hybrid ABCDE algorithm. Elapsed time showing the convergence behaviour of the algorithm is displayed in the Table 5.16 and

Table 5.20. The results are showing that the hybrid ABC-DE algorithm better converges and do not trap in to a local optimum condition, the elapsed time is better than DE and little more than others due to the large number of steps taken by the hybrid approach.

5.5.1.3 Case3. Analysis of PTZ Model using US Navel Tactical Data Systems

In this case analysis of the meta-heuristic nature inspired algorithms has been done on a more complex model i.e. PTZ model having five parameters and DS6. Estimation of expected number of errors is shown in Fig. 5.10; HABCDE results are very much close to actual number of errors in the system than other used approaches. Reliability estimated is found to be less at the starting phase and gradually increases when the resources are enough and learning proceeds.

Table 5.9 specifies the results estimated using five parameter PTZ model and these are very much satisfactory. The MSE and SSE values calculated are showing the best behaviour of HABCDE algorithm in all the best, worst and average cases as shown in Table 5.10 and Table 5.11. Even number of parameters to be estimated by the algorithms is large, convergence of HABCDE is satisfactorily good and do not trap in to a local minima or maxima as shown in Table 5.12.

5.5.1.4 Case4. Analysis of GO Model using Tandem Computers Software Projects

This case analyzes algorithm's behaviour using GO model on time domain dataset DS6. Fig. 5.11 is showing the number of errors estimated at time t in the software for the GO model and satisfactory results are obtained by all the approaches including HABCDE algorithm. The highest reliability is analysed with HABCDE algorithm as compared to the other algorithms using DS7 and GO model.

5.5.1.5 Case5. Analysis of Inflection S-shaped Model using Tandem Computers Software Projects

This case uses three parameter inflection s shaped model and DS7 for analysis. Results analyzed for parameter estimation of GO model and Inflection s-shaped model using hybrid ABCDE algorithm are best among all the other algorithms and are close to the actual number of errors detected given in DS7 as shown in

Table 5.13 and

Table 5.17.The MSE and SSE results in

Table 5.14, Table 5.15 and Table 5.18, Table 5.19 respectively for both the models are showing the satisfactory results of all the approaches used but among them for the best and average case behaviour of MSE and SSE values Hybrid ABCDE algorithm outperforms. The convergence properties are shown in the Table 5.16 and

Table 5.20 respectively for both the models.

5.5.1.6 Case6. Analysis of PTZ Model using Real Time Control System

This case analyzes expected number of errors and reliability using PTZ model and DS8 using hybrid algorithms and results are shown in Fig. 5.12. From the statistical results in

Table 5.21 calculated for the PTZ model it is found that the hybrid algorithms are performing better. The MSE and SSE and convergence rate calculated in

Table 5.22 , Table 5.23 and Table 5.24 are showing the better performance using DE and hybrid ABCDE algorithms than the other used approaches.

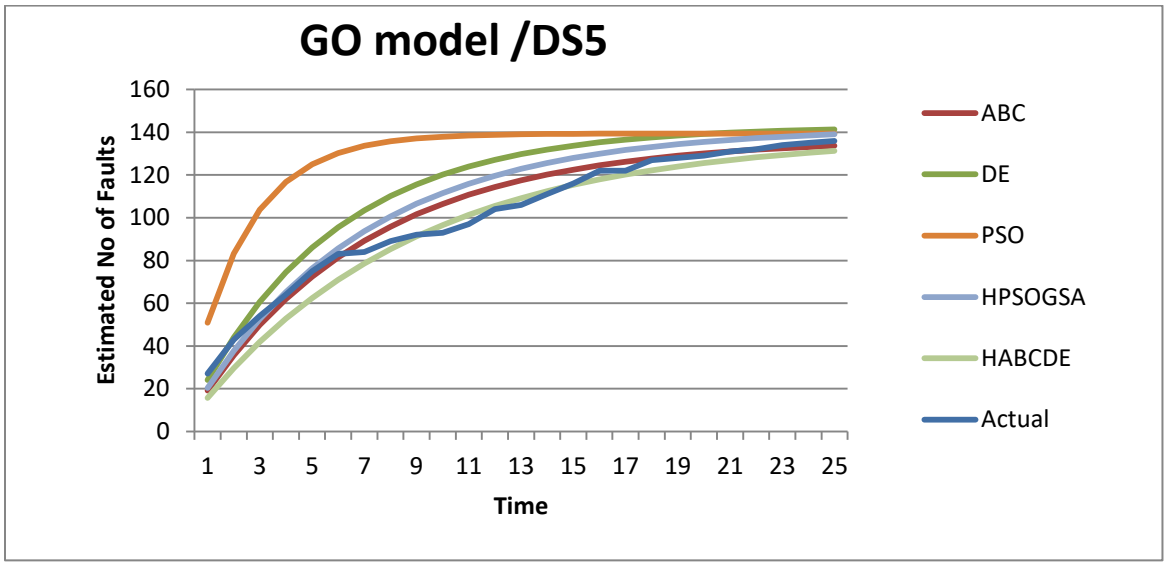

Fig. 5.8 Estimated Number of Errors at Time t Using GO Model and DS5

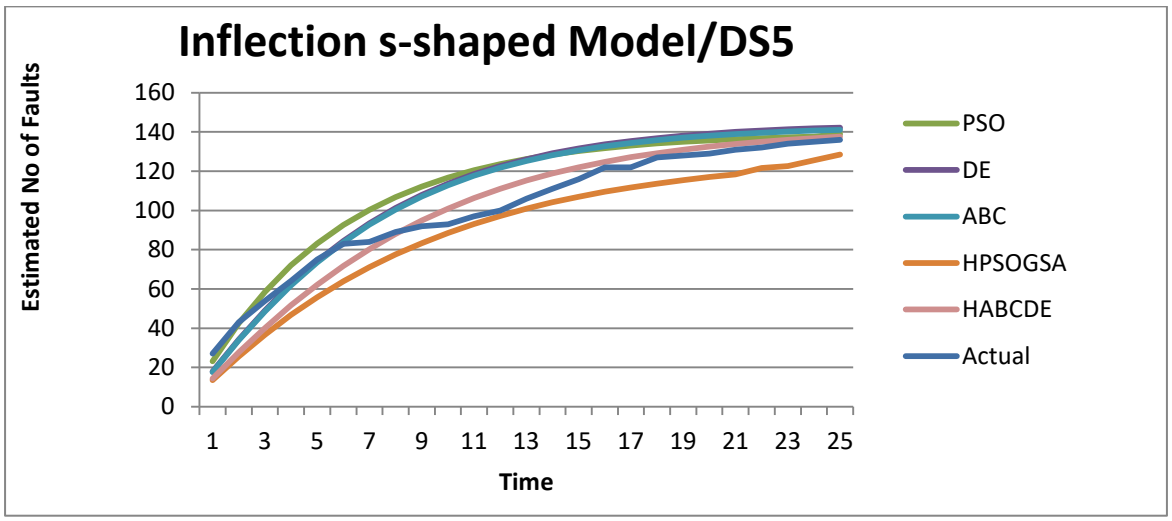

Fig. 5.9 Estimated Number of Errors at Time t using Inflection S Shaped Model and DS5

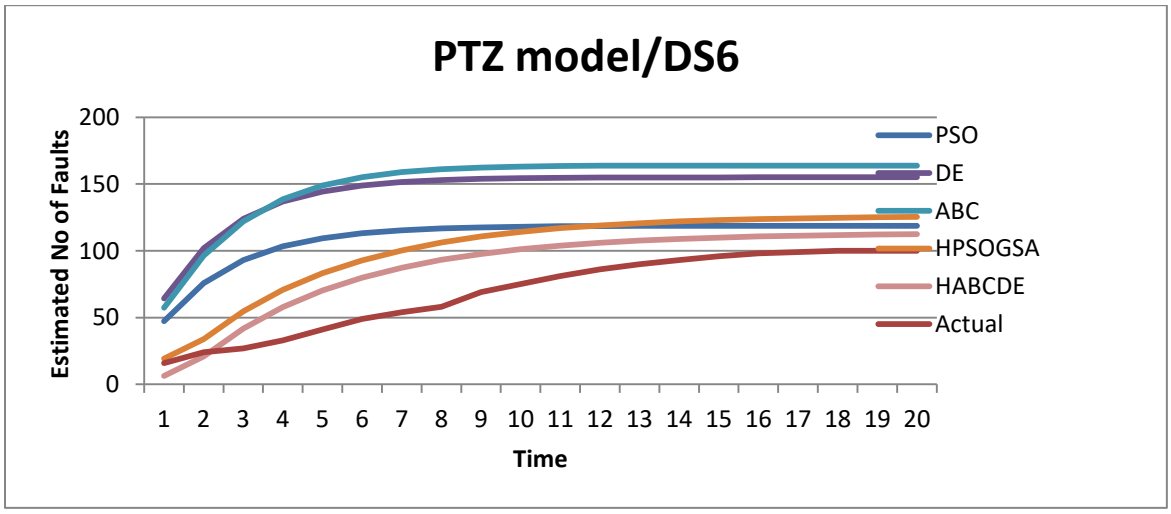

Fig. 5.10 Estimated Number of Errors at Time t using PTZ Model and DS6

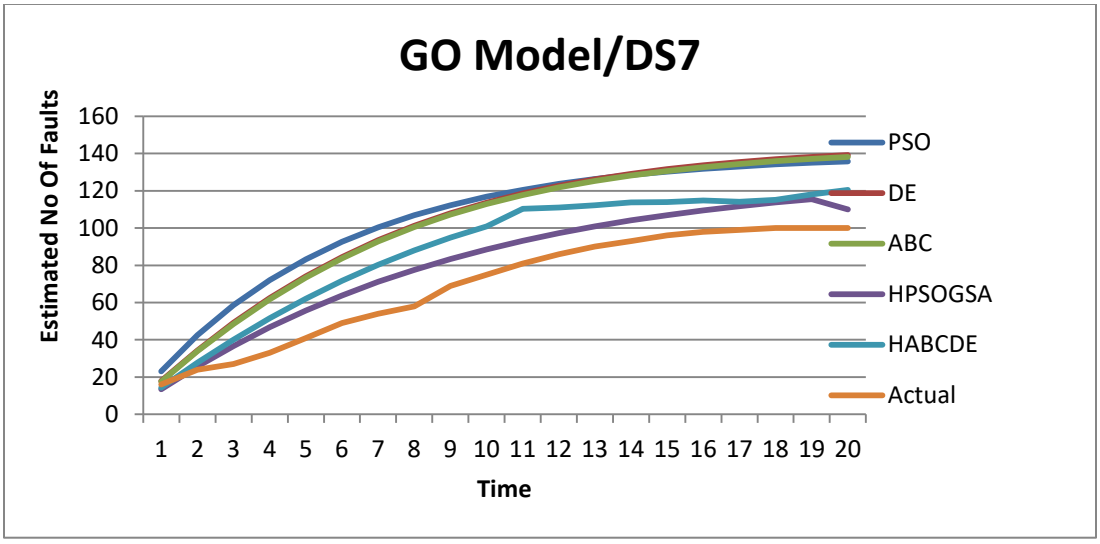

Fig. 5.11 Estimated Number of Faults using GO Model and DS7

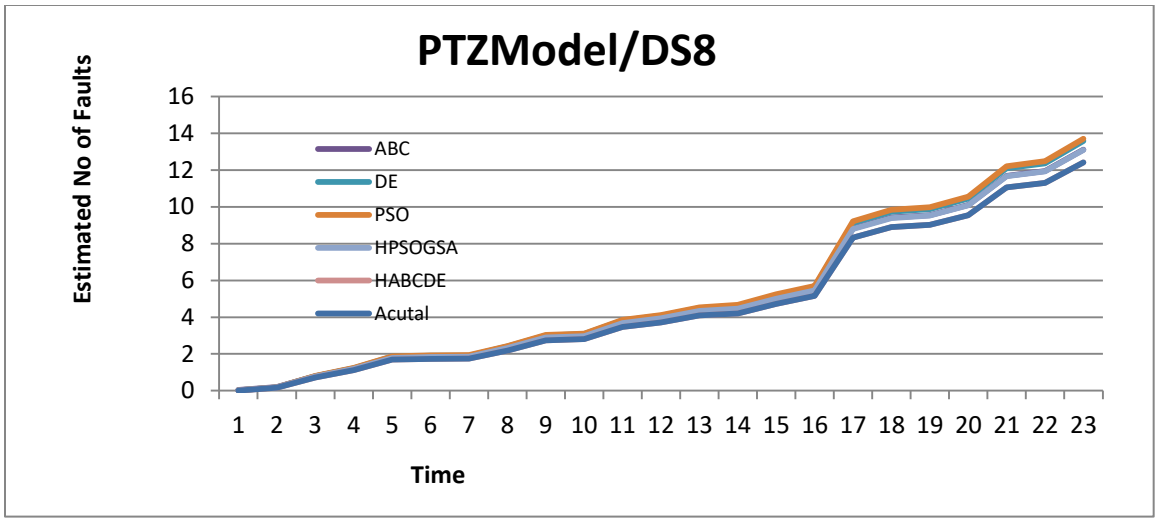

Fig. 5.12 Estimated Number of Errors using PTZ Model and DS8

Table 5.1 Statistical Results of Parameter Estimation for GO Model

Sr. no	Algorithm	a	b
1	ABC	139.4072	0.18173
2	DE	138.9734	0.16789
3	PSO	139.4080	0.18173
4	HPSOGSA	138.0168	0.19733
5	HABCDE	139.4070	0.18526

Table 5.2 Statistical Results of Mean Squared Errors for GO model

MSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	7.16E+02	8.10E+03	1.27E+04	1.03E+02	183.5E+00
Worst	8.62E+03	1.12E+04	1.53E+04	8.37E+02	8.05E+02
Average	2.34E+03	9.99E+03	1.44E+04	4.236E+03	4.24E+02

Table 5.3 Statistical Results of Sum of Squared Errors for GO model

SSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	1790.469	2.03E+05	3.17E+05	2.58E+02	4.59E+02
Worst	215576.4	2.80E+05	3.82E+05	2.09E+05	2.05E+05
Average	58458.92	2.50E+05	3.59E+05	6.41E+03	1.26E+03

Table 5.4 Statistical Results of Elapsed Time in seconds for GO Model

Elapsed time	ABC	PSO	DE	HPSOGSA	HABCDE
Best	27.430	4.25219	28.563	14.28191	25.45811
Worst	28.00795	5.503333	29.782	14.64772	30.18599
Average	27.60951	4.433742	25.890	14.48766	28.25369

Table 5.5 Statistical Result for Parameter Estimation using Inflection S Shaped Model

Sr.	Parameter Estimation	A	b	beta
1	ABC	143.5475	0.199477	0.494249
2	DE	139.4072	0.1817297	0.4938127
3	PSO	135.2333	0.197125	0.478044
4	HPSOGSA	134.0954	0.148224	0.370924
5	HABCDE	138.1621	0.159543	0.367240

Table 5.6 Statistical Result for Mean Squared Errors using Inflection S Shaped Model

MSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	8.57E+01	5.17E+03	1.00E+04	4.62E+03	9.73E+00
Worst	8.57E+03	1.06E+04	1.06E+04	8.25E+03	4.56E+03
Average	2.443E+03	8.941E+03	1.025E+04	6.35E+03	8.18E+02

Table 5.7 Statistical Result for SSE using Inflection S Shaped Model

SSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	2.14E+03	1.291E+05	2.450E+05	1.16E+05	2.43E+02
Worst	2.14E+05	2.65E+05	2.66E+05	2.06E+05	1.14E+05
Average	6.10E+04	2.241E+05	2.56E+05	1.591E+05	2.04E+04

Table 5.8 Statistical Result for Elapsed Time in seconds using Inflection S-Shaped Model

Elapsed time	ABC	DE	PSO	HPSOGSA	HABCDE
Best	28.13206	75.68854	4.012182	14.89633	25.25601
Worst	29.00085	78.809	4.290052	15.29576	28.70718
Average	28.55347	77.07568	4.164199	15.01432	26.74783

Table 5.9 Statistical Result for Parameter Estimation using PTZ Model

Sr.	Parameter Estimation	a	b	c	alpha	beta
1	ABC	108.789	0.06790	0.09664	1.414755	0.000490
2	DE	104.7803	0.09026	1.485479	0.000361	0.000573
3	PSO	109.9867	0.09852	1.692284	0.004389	0.000548
4	HPSOGSA	104.4378	0.069748	0.52414	0.000386	0.000581
5	HABCDE	107.7895	0.067909	0.095645	0.0000141	0.000491

Table 5.10 Statistical Result for MSE using PTZ Model

MSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	5.11E+03	1.97E+04	1.78E+04	2.241E+03	1.59E+03
Worst	1.84E+04	2.44E+04	2.151E+04	1.142E+04	1.67E+04
Average	9.36E+03	2.287E+04	1.95E+04	6.212E+03	6.17E+03

Table 5.11 Statistical Result for SSE using PTZ Model

SSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	1.02E+03	3.931E+05	3.57E+05	4.48E+04	3.178E+04
Worst	3.67E+05	4.872E+05	4.29E+05	2.29E+05	3.33E+05
Average	1.87E+05	4.55E+05	3.90E+05	1.291E+05	1.330E+05

Table 5.12 Statistical Result for Elapsed Time in seconds using PTZ Model

Elapsed time	ABC	DE	PSO	HPSOGSA	HABCDE
Best	25.48469	23.29652	2.534641	14.79884	24.00024
Worst	29.67125	26.44875	2.909702	14.8681	26.55558
Average	26.8195	24.37326	2.650514	14.85586	26.2141

Table 5.13 Statistical Result for Parameter Estimation using GO Model

Sr. no	Optimization method used	a	b
1	ABC	31.7865	0.0039
2	DE	34.3050	0.0068
3	PSO	35.9554	0.0068
4	Hybrid PSOGSA	30.0675	0.0051
5	HABCDE	27.09773	0.0066

Table 5.14 Statistical Results for MSE using GO Model

MSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	13.70259	4.582402	4.945849	4.571759	4.64046
Worst	234.853	19.5016	18.22528	10.16137	37.9492
Average	132.7339	13.02272	11.99917	5.725057	21.8947

Table 5.15 Statistical Result for SSE using GO Model

SSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	3.556E+02	**1.19E+02**	1.291E+02	1.193E+02	1.21E+02
Worst	6.11E+03	**5.07E+02**	4.74E+02	2.64E+02	6.19E+02
Average	3.45E+03	3.349E+02	3.12E+02	1.49E+02	**2.91E+02**

Table 5.16 Statistical Result for Elapsed Time in seconds by Various Algorithms using GO Model

Elapsed time	ABC	DE	PSO	HPSOGSA	HABCDE
Best	25.15042	50.8989	1.548572	16.96231	26.72567
Worst	38.96231	63.50719	18.13354	21.03597	27.69542
Average	25.82883	53.37236	2.073709	18.77478	26.9957

Table 5.17 Statistical Result for Parameter Estimation using Inflection S Shaped Model

Sr.no.	Optimization method	a	b	beta
1	ABC	31.77672	7.24E-05	8.80E-05
2	DE	35.77057	0.006895	0.000333
3	PSO	31.08892	0.006895	0.000333
4	HPSOGSA	31.95481	9.54E-05	0.000106
5	HABCDE	27.28924	0.002422	7.81E-05

Table 5.18 Statistical Result for MSE using Inflection S Shaped Model

MSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	1.42E+02	8.25691	4.750727	2.25E+02	15.71997
Worst	2.31E+02	19.46594	15.11254	2.33E+02	229.2785
Average	2.08E+02	14.30301	9.784142	2.30E+02	83.36798

Table 5.19 Statistical Result for SSE using Inflection S Shaped Model

SSE	ABC	DE	PSO	HPSOGSA	HABCDE
Best	3.678E+03	2.15E+02	1.24E+02	5.85E+03	4.09E+02
Worst	6.00E+03	5.056E+02	3.93E+02	6.056E+03	5.96E+03
Average	5.41E+03	3.72E+02	2.54E+02	5.99E+03	2.15E+02

Table 5.20 Statistical Result For Elapsed Time in seconds using Inflection S Shaped Model

Elapsed time	ABC	DE	PSO	HPSOGSA	HABCDE
Best	27.58383	79.07232	1.279148	14.52722	27.54145
Worst	40.85063	87.04153	1.502138	15.24965	30.54829
Average	29.92456	81.44662	1.401541	14.72697	28.13046

Table 5.21 Statistical Result for Parameter Estimation using PTZ Model

Sr.	Approach used	a	b	c	alpha	beta
1	ABC	139.9816	0.000602	3.99E-05	8.05E-05	9.69E-05
2	DE	138.3598	0.000988	3.98E-05	0.009964	0.000348
3	PSO	139.7796	0.000998	3.99E-05	0.009884	0.007576
4	HPSOGSA	136.9941	0.000933	3.30E-05	0.008679	9.53E-05
5	HABCDE	138.4123	0.000106	3.80E-05	0.006291	0.000352

Table 5.22 Statistical Result for MSE using PTZ Model

MSE	**ABC**	**DE**	**PSO**	**HPSOGSA**	**HABCDE**
Best	64.29368	**49.98509**	57.25215	58.49756	56.32027
Worst	1148.084	**53.31243**	58.89849	94.35258	1558.339
Average	339.4261	**51.4039**	58.11268	77.54565	382.322

Table 5.23 Statistical Result for SSE using PTZ Model

SSE	**ABC**	**DE**	**PSO**	**HPSOGSA**	**HABCDE**
Best	8.74E+03	6.80E+03	7.79E+03	7.96E+03	**6.66E+03**
Worst	1.56E+05	7.25E+03	8.01E+03	1.28E+04	**2.12E+03**
Average	4.62E+04	6.99E+03	7.90E+03	1.05E+04	**5.20E+03**

Table 5.24 Statistical Result of Elapsed Time using various Algorithms for PTZ Model

Elapsed time	ABC	DE	PSO	HPSOGSA	HABCDE
Best	45.51209	156.644	51.426	31.22995	50.9900
Worst	46.58381	161.9628	53.83177	31.78577	62.18178
Average	45.91114	159.0327	52.61924	31.472	56.80561

5.6 Conclusions

A new algorithm based on swarm and evolutionary behaviour along with the impact of ecological space factor is proposed in this chapter. The hybridization has been done in different phases of ABC algorithm. In employee bee phase for finding the better neighbour-hood positions, DE has been used, as DE algorithm outperforms in local search capability. This hybridization will enhance the exploitation capability of original ABC algorithm. In onlooker bee phase for calculating the fitness probabilities of the candidate solutions a new ecological space factor has been used showing the fitness of the available living space of the candidate solution. This factor is having its importance as survival of fittest is not only the factor for evolution; there is the need of having better living ecological space conditions. This factor has been used further in position update calculation of onlooker bees. Using ecological space factor exploration capability has been improved as this factor enhances the diversity of the candidate solution. In scout bee phase exhausted food source solutions has been regenerated on random using ecological space fitness information along with other information shared by employee bee in the solution space.

Further comparative analysis of the proposed work with various other nature inspired algorithms has been done in this chapter. The implementation results are showing the enhanced capability of the proposed algorithm over the ABC, GA, PSO and HPSOGSA in various means. The proposed algorithm may have higher complexity in some cases, yet outperforms in other cases. The algorithm shows its convergence in less number of iterations as compared to the other used approaches. For the future work, proposed algorithm may be used as a generalized algorithm for parameter estimation of other SRGMs. Further proposed algorithm may be applied in other domains for solution of the optimization problems.

Chapter 6 FAILURE DATASETS

6.1 Introduction

All the developed models are based on two types of software failure datasets [51]. One type of the data deals with the time of software failure occurrence and the second one are about the time between the failure occurrences in the software. These two groups are equivalent and are considered as the basis of software reliability model development. Finding suitable data for model verification and improvement are difficult to find. Software companies are not providing their projects failure datasets in fear of their competitors in growing digital world. This difficulty in finding latest real software failure datasets also makes it difficult to develop a reliable software model for estimation of software failure behaviour. New models are developed, validated and verified only using already existing failure datasets available in the literature or using the datasets which are published somewhere. Early works on software reliability model development have used calendar time software failure data [251]. But Musa affirms that the time of execution provides a better measure of the software behaviour as it can vary according to CPU load, man hours etc. as compared to available calendar time data [252], [253]. Despite various difficulties, datasets was collected from the published literature and from the available dataset repositories. The ongoing section lists various datasets to be used in this work along with available auxiliary information about them in the literature.

6.2 Datasets

Variants of software failure datasets are available in the literature but all of them cannot be used in their actual form for software reliability model development. Some of the available data are used in their actual form while others are utilized after extracting useful information from the available failure datasets. The available datasets in the literature is both from the failure history of closed source software and open source software. The work utilizes both kinds of failure datasets to illustrate the accuracy estimation of the proposed research work. Table 6.1 illustrates DS1, DS2, DS3, DS4, DS5, DS6, DS7 and DS8 datasets that are from the Eclipse project failure datasets, JDT project failure datasets, Firefox and Genome project failure datasets and datasets from closed software [51] . Table 6.2, Table 6.3, Table 6.4 and Table 6.5 describes in detail failure datasets from various versions of Eclipse projects, JDT project, Firefox and Genome projects. Table 6.6,

Table 6.8, Table 6.7 and Table 6.9 represents the failure datasets of Real time Command and Control System, US Naval Tactical Data system software failure data, Tandem Computer Software Project Datasets and failure data of Real Time Control System respectively.

Table 6.1 Failure Datasets used for Implementation

DS1-Eclipse Project Failure Datasets (In different Iterations)				
1	Iteration1.0	3	65 days	Time domain data
2	Iteration 2.0	24	272 days	Time domain data
3	Iteration 2.1	29	331 days	Time domain data
4	Iteration 3.0	96	569 days	Time domain data
5	Iteration 3.1	136	718 days	Time domain data

6	Iteration 3.2	119	605 days	Time domain data
7	Iteration 3.3	119	588 days	Time domain data
8	Iteration 3.4	51	469 days	Time domain data
9	Iteration 3.5	26	283 days	Time domain data
10	Iteration 3.6	28	283 days	Time domain data
11	Iteration 4.1	15	407 days	Time domain data
12	Iteration 4.2	31	407 days	Time domain data
DS2-JDT Project Failure Dataset				
1	Version 1.4	9	291 days	Time Domain Dataset
2	Version 2.0	15	351 days	Time Domain Dataset
3	Version 2.1	35	503 days	Time Domain Dataset
4	Version 3.2	59	707 days	Time Domain Dataset
5	Version 3.3	18	1285 days	Time Domain Dataset
6	Version 3.4	14	568 days	Time Domain Dataset
7	Version 3.5	5	311 days	Time Domain Dataset
8	Version 3.6	15	641 days	Time Domain Dataset
9	Version 3.7	12	944 days	Time Domain Dataset

DS3-Firefox Failure Dataset				
1	Version 3.0	2435	53 days	Interval Domain Dataset
2	Version 3.5	2771	28 days	Interval Domain Dataset
3	Version 3.6	50	6840 days	Interval Domain Dataset
DS4-Genome Failure Datasets				
1	Version 2.0	85	24 days	Interval Domain Dataset
2	Version 2.3	54	46 days	Interval Domain Dataset
3	Version 2.4	54	24 days	Interval Domain Dataset
Dataset	Number of faults	Time(sec/hours/days)	Type of application	Type of the data
DS5	136	25 hours	Real time command and control system	Interval domain data
DS6	34	849 days	US Navel Tactical data systems	Time domain data
DS7	100	10000 hours	Tandem computers software projects.	Interval domain data
DS8	136	88682 sec	Real time control system	Time domain data

Table 6.2 Eclipse dataset (DS1)

Version 1			Version 2.1			Version 3.0		
Fault	TBF	Cum. TBF	Fault	TBF	Cum. TBF	Fault	TBF	Cum. TBF
1	10	10	1	20	20	1	14	14
2	1	11	2	9	29	2	60	74
3	54	65	3	50	79	3	5	79
Version 2.0			4	27	106	4	18	97
Fault	TBF	Cum. TBF	5	27	133	5	20	117
1	9	9	6	37	170	6	7	124
2	14	23	7	1	171	7	1	125
3	21	44	8	19	190	8	34	159
4	8	52	9	1	191	9	44	203
5	45	97	10	9	200	10	6	209
6	29	126	11	13	213	11	1	210
7	15	141	12	14	227	12	7	217
8	1	142	13	6	233	13	8	225
9	1	143	14	1	234	14	5	230

10	4	147	15	6	240	15	6	236
11	2	149	16	8	248	16	14	250
12	7	156	17	5	253	17	6	256
13	19	175	18	1	254	18	12	268
14	8	183	19	1	255	19	5	273
15	1	184	20	1	256	20	14	287
16	5	189	21	21	277	21	1	288
17	9	198	22	14	291	22	5	293
18	3	201	23	1	292	23	1	294
19	2	203	24	21	313	24	1	295
20	1	204	25	1	314	25	21	316
21	21	225	26	1	315	26	1	317
22	18	243	27	3	318	27	2	319
23	28	271	28	6	324	28	6	325
24	1	272	29	7	331	**29**	6	331
30	1	332	66	7	444	4	2	53
31	1	333	67	2	446	5	9	62

32	1	334	68	6	452	6	29	91
33	18	352	69	5	457	7	3	94
34	6	358	70	1	458	8	6	100
35	1	359	71	6	464	9	7	107
36	3	362	72	4	468	10	4	111
37	10	372	73	4	472	11	13	124
38	1	373	74	8	480	12	1	125
39	17	390	75	6	486	13	6	131
40	3	393	76	7	493	14	1	132
41	2	395	77	1	494	15	4	136
42	4	399	78	1	495	16	1	137
43	1	400	79	6	501	17	1	138
44	1	401	80	8	509	18	1	139
45	1	402	81	1	510	19	5	144
46	2	404	82	3	513	20	1	145
47	3	407	83	1	514	21	2	147
48	2	409	84	3	517	22	14	161

49	3	412	85	5	522	23	5	166
50	2	414	86	1	523	24	5	171
51	1	415	87	3	526	25	2	173
52	1	416	88	1	527	26	1	174
53	1	417	89	1	528	27	14	188
54	1	418	90	22	550	28	2	190
55	1	419	91	4	554	29	1	191
56	1	420	92	2	556	30	8	199
57	1	421	93	1	557	31	4	203
58	4	425	94	4	561	32	1	204
59	1	426	95	3	564	33	6	210
60	1	427	96	5	569	34	3	213
61	1	428	Version 3.1			35	1	214
62	4	432	Fault	TBF	Cum. TBF	36	1	215
63	3	435	1	25	25	37	4	219
64	1	436	2	2	27	38	2	221
65	1	437	3	24	51	39	1	222

40	1	223	71	2	304	102	8	408
41	5	228	72	1	305	103	13	421
42	1	229	73	1	306	104	5	426
43	1	230	74	4	310	105	3	429
44	3	233	75	4	314	106	19	448
45	4	237	76	1	315	107	6	454
46	5	242	77	1	316	108	1	455
47	5	247	78	1	317	109	1	456
48	2	249	79	1	318	110	17	473
49	5	254	80	1	319	111	13	486
50	2	256	81	1	320	112	6	492
51	1	257	82	1	321	113	26	518
52	6	263	83	1	322	114	1	519
53	5	268	84	1	323	115	1	520
54	2	270	85	1	324	116	21	541
55	4	274	86	1	325	117	9	550
56	1	275	87	3	328	118	9	559

57	1	276	88	1	329	119	4	563
58	1	277	89	4	333	120	5	568
59	1	278	90	1	334	121	1	569
60	1	279	91	2	336	122	21	590
61	1	280	92	5	341	123	12	602
62	1	281	93	1	342	124	14	616
63	2	283	94	1	343	125	3	619
64	8	291	95	9	352	126	7	626
65	1	292	96	11	363	127	11	637
66	1	293	97	11	374	128	1	638
67	1	294	98	12	386	129	9	647
68	6	300	99	5	391	130	7	654
69	1	301	100	1	392	131	7	661
70	1	302	101	8	400	132	1	662
133	4	666	31	3	248	67	1	341
134	1	667	32	1	249	68	2	343
135	1	668	33	1	250	69	7	350

136	50	718	34	1	251	70	1	351
Version 3.2			35	2	253	71	1	352
Fault	TBF	Cum. TBF	36	3	256	72	3	355
1	10	10	37	3	259	73	1	356
2	23	33	38	6	265	74	1	357
3	28	61	39	2	267	75	2	359
4	33	94	40	2	269	76	3	362
5	7	101	41	5	274	77	4	366
6	16	117	42	3	277	78	1	367
7	15	132	43	1	278	79	1	368
8	13	145	44	3	281	80	3	371
9	1	146	45	4	285	81	1	372
10	4	150	46	4	289	82	1	373
11	28	178	47	3	292	83	6	379
		178	48	1	293	84	3	382
13	2	180	49	1	294	85	7	389
14	2	182	50	2	296	86	1	390

15	2	184	51	1	297	87	3	393
16	5	189	52	5	302	88	2	395
17	1	190	53	1	303	89	5	400
18	4	194	54	1	304	90	3	403
19	7	201	55	1	305	91	7	410
20	6	207	56	6	311	92	12	422
21	8	215	57	1	312	93	6	428
22	4	219	58	1	313	94	2	430
23	1	220	59	1	314	95	8	438
24	6	226	60	18	332	96	6	444
25	1	227	61	3	335	97	14	458
26	1	228	62	2	337	98	1	459
27	13	241	63	1	338	99	15	474
28	2	243	64	1	339	100	3	477
29	1	244	65	1	340	101	1	478
30	1	245	66	0	340	102	2	480
103	7	487	13	13	175	44	1	278

104	6	493	14	2	177	45	1	279
105	22	515	15	4	181	46	1	280
106	4	519	16	2	183	47	11	291
107	1	520	17	12	195	48	4	295
108	23	543	18	1	196	49	1	296
109	2	545	19	2	198	50	5	301
110	3	548	20	6	204	51	1	302
111	15	563	21	1	205	52	1	303
112	6	569	22	1	206	53	5	308
113	1	570	23	5	211	54	1	309
114	1	571	24	4	215	55	1	310
115	5	576	25	12	227	56	1	311
116	3	579	26	1	228	57	1	312
117	21	600	27	1	229	58	4	316
118	3	603	28	3	232	59	2	318
119	2	605	29	7	239	60	1	319
Version 3.3			30	5	244	61	1	320

Fault	TBF	Cum. TBF	31	5	249	62	1	321
1	20	20	32	4	253	63	1	322
2	8	28	33	1	254	64	1	323
3	44	72	34	1	255	65	1	324
4	3	75	35	2	257	66	1	325
5	48	123	36	5	262	67	6	331
6	1	124	37	3	265	68	1	332
7	1	125	38	3	268	69	1	333
8	2	127	39	3	271	70	2	335
9	19	146	40	2	273	71	1	336
10	1	147	41	2	275	72	5	341
11	9	156	42	1	276	73	3	344
12	6	162	43	1	277	74	2	346
75	9	355	111	4	515	26	2	276
76	2	357	112	1	516	27	2	278
77	1	358	113	7	523	28	4	282
78	3	361	114	6	529	29	1	283

79	2	363	115	14	543	30	4	287
80	2	365	116	4	547	31	4	291
81	2	367	117	11	558	32	3	294
82	5	372	118	16	574	33	14	308
83	1	373	119	14	588	34	21	329
84	1	374	Version 3.4			35	17	346
85	1	375	Fault	TBF	Cum. TBF	36	7	353
86	1	376	1	28	28	37	6	359
87	4	380	2	36	64	38	0	359
88	1	381	3	48	112	39	1	360
89	1	382	4	10	122	40	12	372
90	1	383	5	12	134	41	3	375
91	1	384	6	1	135	42	5	380
92	1	385	7	3	138	43	42	422
93	4	389	8	1	139	44	5	427
94	1	390	9	1	140	45	1	428
95	1	391	10	23	163	46	14	442

96	1	392	11	22	185	47	4	446
97	1	393	12	7	192	48	2	448
98	5	398	13	5	197	49	8	456
99	3	401	14	15	212	50	8	464
100	6	407	15	7	219	51	5	469
101	1	408	16	5	224	Version 3.5		
102	5	413	17	15	239	Fault	TBF	Cum. TBF
103	1	414	18	4	243	1	16	16
104	4	418	19	1	244	2	1	17
105	37	455	20	1	245	3	15	32
106	16	471	21	4	249	4	1	33
107	7	478	22	5	254	5	15	48
108	11	489	23	2	256	6	1	49
109	4	493	24	7	263	7	18	67
110	18	511	25	11	274	**8**	2	69
9	1	70	12	8	103	11	12	138
10	4	74	13	7	110	12	6	144

11	2	76	14	10	120	13	5	149
12	25	101	15	5	125	14	22	171
13	9	110	16	7	132	15	2	173
14	20	130	17	5	137	Version 4.2		
15	15	145	18	8	145	Fault	TBF	Cum. TBF
16	44	189	19	2	147	1	2	2
17	4	193	20	28	175	2	15	17
18	22	215	21	1	176	3	30	47
19	14	229	22	20	196	4	18	65
20	2	231	23	7	203	5	1	66
21	11	242	24	6	209	6	28	94
22	7	249	25	54	263	7	34	128
23	19	268	26	7	270	8	1	129
24	8	276	27	2	272	9	1	130
25	2	278	28	11	283	10	2	132
26	5	283	Version 4.1			11	14	146
Version 3.6			Fault	TBF	Cum. TBF	12	38	184

Fault	TBF	Cum. TBF	1	4	4	13	3	187
1	5	5	2	16	20	14	47	234
2	1	6	3	7	27	15	11	245
3	7	13	4	18	45	16	2	247
4	15	28	5	5	50	17	6	253
5	7	35	6	11	61	18	24	277
6	4	39	7	13	74	19	5	282
7	2	41	6	11	85	20	2	284
8	12	53	7	13	98	21	4	288
9	8	61	8	6	104	22	2	290
10	17	78	9	6	110	23	1	291
11	17	95	10	16	126	24	8	299
25	24	323						
26	14	337						
27	16	353						
28	5	358						
29	1	359						

30	14	373
31	34	407

Table 6.3 JDT Dataset (DS2)

Version 1.4			12	6	90	16	6	54	57	119	615	Version 3.5		
Fault	TBF	Cum. TBF	13	3	93	17	6	60	58	1	616	Fault	TBF	Cum. TBF
1	1	1	14	1	94	18	1	61	59	91	707	1	1	1
2	82	83	15	3	97	19	1	62	Version 3.3			2	138	139
3	1	84	16	5	102	20	1	63	Fault	TBF	Cum. TBF	3	9	148
4	46	130	17	11	113	21	1	64	1	1	1	4	81	229
5	1	131	18	1	114	22	11	75	2	221	222	5	82	311
6	6	137	19	19	133	23	1	76	3	12	234	Version 3.6		
7	15	152	20	15	148	24	1	77	4	23	257	Fault	TBF	Cum. TBF
8	77	229	21	3	151	25	1	78	5	149	406	1	1	1
9	62	291	22	3	154	26	2	80	6	7	413	2	49	50
Version 2.0			23	8	162	27	4	84	7	24	437	3	41	91
Fault	TBF	Cum. TBF	24	3	165	28	13	97	8	23	460	4	18	109

1	1	1	25	15	180	29	2	99	9	4	464	5	42	151
2	1	2	26	8	188	30	7	106	10	16	480	6	1	152
3	29	31	27	4	192	31	2	108	11	24	504	7	1	153
4	86	117	28	0	192	32	0	108	12	3	507	8	16	169
5	43	160	29	99	291	33	6	114	13	6	513	9	13	182
6	33	193	30	6	297	34	9	123	14	8	521	10	34	216
7	1	194	31	34	331	35	12	135	15	22	543	11	160	376
8	37	231	32	1	332	36	9	144	16	1	544	12	129	505
9	25	256	33	5	337	37	16	160	17	17	561	13	55	560
10	6	262	34	77	414	38	13	173	18	724	1285	14	24	584
11	14	276	35	89	503	39	2	175	Version 3.4			15	57	641
12	6	282	Version 3.2			40	6	181	Fault	TBF	Cum. TBF	Version 3.7		
13	3	285	Fault	TBF	Cum. TBF	41	3	184	1	1	1	Fault	TBF	Cum. TBF
14	24	309	1	0	0	42	6	190	2	30	31	1	1	1
15	42	351	2	16	16	43	34	224	3	94	125	2	257	258
Version 2.1			3	1	17	44	24	248	4	103	228	3	22	280

Fault	TBF	Cum. TBF	4	1	18	45	33	281	5	18	246	4	16	296
1	1	1	5	9	27	46	14	295	6	10	256	5	131	427
2	4	5	6	3	30	47	1	296	7	56	312	6	16	443
3	19	24	7	1	31	48	1	297	8	2	314	7	23	466
4	9	33	8	1	32	49	7	304	9	12	326	8	1	467
5	8	41	9	1	33	50	5	309	10	19	345	9	48	515
6	3	44	10	1	34	51	42	351	11	107	452	10	55	570
7	16	60	11	5	39	52	20	371	12	18	470	11	34	604
8	19	79	12	5	44	53	35	406	13	1	471	12	340	944
9	2	81	13	2	46	54	31	437	14	97	568			
10	1	82	14	1	47	55	1	438						
11	2	84	15	1	48	56	58	496						

Table 6.4 Firefox Failure Dataset (DS3)

Firefox 3.0					
Time	No. 0f faults	Cumm Fault	Time	No. 0f faults	Cumm Fault
1	9	9	28	49	1010

2	12	21	29	50	1060
3	16	37	30	50	1110
4	25	62	31	50	1160
5	27	89	32	50	1210
6	29	118	33	51	1261
7	29	147	34	52	1313
8	32	179	35	53	1366
9	34	213	36	54	1420
10	35	248	37	55	1475
11	36	284	38	55	1530
12	36	320	39	55	1585
13	39	359	40	55	1640
14	39	398	41	56	1696
15	40	438	42	59	1755
16	40	478	43	60	1815
17	40	518	44	60	1875
18	41	559	45	60	1935

19	42	601	46	61	1996
20	43	644	47	62	2058
21	43	687	48	62	2120
22	44	731	49	62	2182
23	45	776	50	62	2244
24	45	821	51	62	2306
25	46	867	52	64	2370
26	47	914	53	65	2435
27	47	961			

Firefox 3.5

Time	No. 0f faults	Cumm Fault	Time	No. 0f faults	Cumm Fault
1	66	66	15	105	1338
2	73	139	16	105	1443
3	76	215	17	106	1549
4	81	296	18	106	1655
5	83	379	19	107	1762
6	87	466	20	108	1870

7	88	554	21	108	1978
8	92	646	22	109	2087
9	94	740	23	112	2199
10	94	834	24	113	2312
11	94	928	25	113	2425
12	99	1027	26	115	2540
13	102	1129	27	115	2655
14	104	1233	28	116	2771

Firefox 3.6

Time	No of faults	Cumm Faults	Time	No of faults	Cumm Faults
1	117	117	15	135	1868
2	119	236	16	135	2003
3	119	355	17	135	2138
4	120	475	18	138	2276
5	122	597	19	138	2414
6	122	719	20	138	2552
7	122	841	21	138	2690

8	124	965	22	138	2828
9	125	1090	23	138	2966
10	125	1215	24	138	3104
11	125	1340	25	138	3242
12	127	1467	26	138	3380
13	131	1598	27	138	3518
14	135	1733	28	138	3656
Time	No of faults	Cumm Faults	Time	No of faults	Cumm Faults
29	138	3794	43	148	5803
30	140	3934	44	148	5951
31	140	4074	45	148	6099
32	140	4214	46	148	6247
33	141	4355	47	148	6395
34	143	4498	48	148	6543
35	143	4641	49	148	6691
36	143	4784	50	149	6840
37	143	4927			

38	144	5071			
39	146	5217			
40	146	5363			
41	146	5509			
42	146	5655			

Table 6.5 Genome Failure Dataset(DS4)

Genome 2.0					
Time	No of faults	Cumm Faults	Time	No of faults	Cumm Faults
1	6	6	13	6	58
2	5	11	14	8	66
3	3	14	15	6	72
4	2	16	16	2	74
5	5	21	17	2	76
6	5	26	18	1	77
7	8	34	19	1	78

8	4	38	20	1	79
9	8	46	21	1	80
10	3	49	22	2	82
11	2	51	24	3	85
12	1	52			

Genome 2.2

Time	No of faults	Cumm Faults	Time	No of faults	Cumm Faults
1	5	5	10	3	41
2	4	9	11	2	43
3	5	14	13	1	44
4	5	19	15	4	48
5	9	28	16	1	49
6	5	33	17	1	50
7	2	35	18	1	51
8	1	36	22	1	52
9	2	38	24	2	54

Genome 2.3

Time	No of faults	Cumm Faults	Time	No of faults	Cumm Faults
1	4	4	11	1	38
2	5	9	12	3	41
3	2	11	15	2	43
4	7	18	18	1	44
5	3	21	19	1	45
6	1	22	20	5	50
7	3	25	21	2	52
8	4	29	23	1	53
9	3	32	46	1	54
10	5	37			

Table 6.6 Failure Dataset(DS5)

Real Time Command and Control Data					
Time (in Hrs)	No. of faults	Cumm. Faults	Time(in Hrs)	No. of faults	Cumm. Faults
1	27	27	14	5	111
2	16	43	15	5	116

3	11	54	16	6	122
4	10	64	17	0	122
5	11	75	18	5	127
6	7	83	19	1	128
7	2	84	20	1	129
8	5	89	21	2	131
9	3	92	22	1	132
10	1	93	23	2	134
11	4	97	24	1	135
12	7	104	25	1	136
13	2	106			

Table 6.7 Failure Dataset (DS6)

US Naval Tactical Data System Software Failure Data (NTDS)					
Fault	**Time Between**	**Cumulative**	**Fault**	**Time Between**	**Cumulative**

number	faults	Time	number	faults	Time
1	9	9	18	3	98
2	12	21	19	6	104
3	11	32	20	1	105
4	4	36	21	11	116
5	7	43	22	33	149
6	2	45	23	7	156
7	5	50	24	91	247
8	8	58	25	2	249
9	5	63	26	1	250
10	7	70	27	87	337
11	1	71	28	47	384
12	6	77	29	12	396
13	1	78	30	9	405
14	9	87	31	135	540
15	4	91	32	258	798
16	1	92	33	16	814

17	3	95	34	35	849

Table 6.8 Failure Dataset (DS7)

Tandem Computer Software Project Data							
Time(in Week)	**CPU hours**	**Number of Faults**	**Cum. Faults**	**Time(in Week)**	**CPU hours**	**Number of Faults**	**Cum.Faults**
1	519	16	16	11	6539	81	527
2	968	24	40	12	7083	86	613
3	1430	27	67	13	7487	90	703
4	1893	33	100	14	7846	93	796
5	2490	41	141	15	8205	96	892
6	3058	49	190	16	8564	98	990
7	3625	54	244	17	8923	99	1089
8	4422	58	302	18	9282	100	1189
9	5218	69	371	19	9641	100	1289
10	5823	75	446	20	10000	100	1389

Table 6.9 Failure Dataset (DS8)

Real Time Control System Data					
Fault No.	**Time Between Faults**	**Cum. Time**	**Fault No.**	**Time Between Faults**	**Cum. Time**
1	3	3	41	97	6477
2	30	33	42	263	6740
3	113	146	43	452	7192
4	81	227	44	255	7447
5	115	342	45	197	7644
6	9	351	46	193	7837
7	2	353	47	6	7843
8	91	444	48	79	7922
9	112	556	49	816	8738
10	15	571	50	1351	10089
11	138	709	51	148	10237
12	50	759	52	21	10258
13	77	836	53	233	10491

14	24	860	54	134	10625
15	108	968	55	357	10982
16	88	1056	56	193	11175
17	670	1726	57	236	11411
18	120	1846	58	31	11442
19	26	1872	59	369	11811
20	114	1986	60	748	12559
21	325	2311	61	0	12559
22	55	2366	62	232	12791
23	242	2608	63	330	13121
24	68	2676	64	365	13486
25	422	3098	65	1222	14708
26	180	3278	66	543	15251
27	10	3288	67	10	15261
28	1146	4434	68	16	15277
29	600	5034	69	529	15806
30	15	5049	70	379	16185

31	36	5085	71	44	16229
32	4	5089	72	129	16358
33	0	5089	73	810	17168
34	8	5097	74	290	17458
35	227	5324	75	300	17758
36	65	5389	76	529	18287
37	176	5565	77	281	18568
38	58	5623	78	160	18728
39	457	6080	79	828	19556
40	300	6380	80	1011	20567
81	445	21012	109	875	49171
82	296	21308	110	245	49416
83	1755	23063	111	729	50145
84	1064	24127	112	1897	52042
85	1783	25910	113	447	52489
86	860	26770	114	386	52875
87	983	27753	115	446	53321

88	707	28460	116	122	53443
89	33	28493	117	990	54433
90	868	29361	118	948	55381
91	724	30085	119	1082	56463
92	2323	32408	120	22	56485
93	2930	35338	121	75	56560
94	1461	36799	122	482	57042
95	843	37642	123	5509	62551
96	12	37654	124	100	62651
97	261	37915	125	10	62661
98	1800	39715	126	1071	63732
99	865	40580	127	371	64103
100	1435	42015	128	790	64893
101	30	42045	129	6150	71043
102	143	42188	130	3321	74364
103	108	42296	131	1045	75409
104	0	42296	132	648	76057

105	3110	45406	133	5485	81542
106	1247	46653	134	1160	82702
107	943	47596	135	1864	84566
108	700	48296	136	4116	88682

Chapter 7 RESULTS AND SCOPE FOR FUTURE RESEARCH

In this chapter major conclusion of the research work done by the authors in this thesis work are discussed. It also explores the possibilities of future scope of research in the field of software reliability assessment and parameter estimation algorithms.

7.1 Introduction

The major consideration in software development process is to develop reliable software at the very end of testing phase. Based on the comprehensive survey in software reliability model development, authors carefully analyzed how various models are evolved from existing models using assumptions made by the models. Each model has used specific attributes for model development and has specific assumptions. Using these assumptions models are categorized in twelve classes depending on various attributes. These classes are further analysed to see how various models are belonging to these categories and how they have been evolved. The major issues in software reliability model development are considered in depth by analysing previous work done by the researchers. After finding major issues in software reliability model development, two models are proposed that are based on conduct of failure rate models and NHPP models. Proposed models are validated by using various performance measurement methods and by their comparisons with well-established models in the field of software reliability estimation. In literature software reliability model parameter estimation methods are also discussed. Evolutionary algorithm based methods are found to be better than traditional methods of parameter estimation. However in the field of software reliability estimation these algorithms are not found to be exploited in number of publications. Only few of the publications are there where these methods are employed. A new hybrid algorithm has been proposed to adopt evolutionary algorithms in the field of software reliability model parameter estimation.

7.2 Major findings

1. Fault forecasting is the major area for software reliability estimation. Forecasting methods estimates software reliability through the use of statistical models development. These models can estimate future reliability of the software significantly by making use of available software failure data and helps in decision making for the software

developers about the number of resources required to enhance future reliability of the software.

2. Existing models are explored on the basis of their assumptions and behaviour. Analysis of twenty three failure rate models and one hundred and six NHPP behaviour based models is done in chapter1. Detailed analysis examined how one model is extending the features of other existing model. To represent evolution of models from other existing models an evolution diagram is made that shows how a new model development is enhancing existing model features to make better reliability estimation.
3. Fifteen attributes are identified and defined; these attributes are making primary help in model classification.
4. Among the large group of software reliability models, a group of failure rate behaviour based models are found to be the earliest software reliability estimation models. These models can predict program failure rate depending on the amount of faults present in software at a particular interval of time and assumes that with a change in amount of remaining faults, program failure rate changes accordingly. These models are only applicable in traditional software development environments. Further improvements in these models are made so that they can be adopted in the latest software development environment. A new failure rate model is proposed in chapter 3, which will work in iterative software development environment to estimate reliability of software developed under iterative SDLC process.
5. A new modulation factor is used to reflect all the changing requirement of the software during each phase of iterative software development environment. Modulation factor values are calculated using a modulation parameter. This modulation parameter is specifying the level of acceptance of the end users and helps in making decision to identify the requirements that need to be changed in the upcoming iteration. Modulation factor is well defining the varying needs of the software development during each of the iteration.
6. An existing classification scheme is protracted to make it applicable in latest software development environment. Depending on the iterative software development

environment, software reliability estimation models are categorized according to various stages of software development. Proposed classification is easy and can be used in industries and academia.

7. There is a limited availability of the software failure datasets as the software companies are not interested in revealing their software's faults. Keeping this factor in mind, in this thesis work new data set is collected and reformatted to validate proposed software reliability model in real world environment of software development. Collected raw dataset is of Eclipse and JDT projects failure dataset over a wide range from 2003-2013. Twelve versions of Eclipse project failure datasets and six versions of JDT project datasets are used to validate an applicability of proposed iterative software reliability model.

8. NHPP group of models are among most popular group of models and these are also found to be significantly good in software industries for accurate software reliability estimation. There are hundreds of models published in this NHPP group of models. But no model is well applicable in heterogeneous environment and their failure datasets. Keeping in mind that every specific system needs enhancements, a new NHPP model is developed that can significantly estimate reliability of open source software systems.

9. NHPP based software reliability model for OSS systems has been proposed in this work. This model incorporates a new testing effort behaviour based fault content function. Incorporation of testing effort coefficient is depicting that in each new released version due to added functionality and number of fault content there is need of change in testing effort, more is the effort incorporated in the software testing and debugging, more reliable software will be released in future. Impact of testing effort has been reasonably incorporated to fit well in the latest software development technologies and environments. Model has been validated using three versions of Firefox project failure data and three versions of Genome project failure data.

10. Nature inspired algorithms are found to be extensively used for solution of non-linear optimization problems in various application domains. However these algorithms have not been adopted well for reliability analysis of software developed under latest development environment. Chapter 5 has proposed a new hybrid algorithm for parameter

estimation of proposed models which will work efficiently in software reliability analysis.

11. Five well-known algorithms are analysed in this work. Famous artificial bee colony algorithm has been selected for enhancement. ABC algorithm is having number of advantages and found to have good performance for reliability estimation on the existing software reliability but somewhere there are limitations in which it traps to a local optima. Proposed algorithm has overcome the limitations of existing artificial bee colony algorithm by making a new hybrid Swarm Evolutionary algorithm.
12. Differential Evolution algorithm has been utilized to overcome the above limitation of ABC algorithm. ABC algorithm is having premature convergence with unbalanced exploration and exploitation process. The advantage of using DE is mainly in providing diversity of population that helps in providing the improved local search to ABC algorithm. DE is also having faster convergence capability then ABC algorithm.
13. ABC and DE algorithms are hybridized by making a use of ecological living space factor. Ecological living space is a factor that is necessary for the organism to flourish in the environment and it will enhance the diversity by matching the available living space. The candidate solution and ecological diversity are very much closely related. More favourable is the ecological conditions more quality food will be available to the bees.
14. Proposed algorithm is tested on existing software reliability models and proposed failure rate behaviour based software reliability model. Proposed model is outperforming than other failure rate based models on all the used datasets. However for few datasets HPSOGSA is also performing well. From the results it is concluded that hybrid algorithms will adapt finely in parameter estimation of proposed software reliability model.
15. In this work MLE technique is used for parameter estimation. The accuracy of proposed failure rate model has been compared with other five well-known JM, SW, SWM, GOI and Mahapatra software reliability models. The proposed model has clear-cut outperform all models under comparison in 11 iterations for DS1. It shows the lowest values of SSE in 91.6 % iterations. In term of MSE proposed model is winner in nine iterations. The

proposed model has clear-cut outperform other models in 75 % iterations by achieving lowest value of MSE. Result shows that proposed model has given a significantly better fit to iterative data by adapting according to varying needs of different iterations.

16. To check the strength of the proposed NHPP model it has been compared with three well known GO, Inflection s-shaped and PTZ NHPP models. The goodness-of-fit of proposed model is calculated in terms of eight criteria SSE, MSE, MAE, MEOP, AE, AIC, TS and PRR. In Firefox 3.0 release, among eight measures of fitness, the proposed model has clear- cut outperforms by 87.25% than other used models. In Firefox 3.5 release, overall proposed model is having 75% major beating values. In Firefox 3.6, the proposed model has major beating criterion except in term of AIC where Inflection s-shaped model performed better than the proposed model. Overall, the proposed model is performing 87.5% better than other used models.

Thus, the goal of software reliability analysis using software reliability model development and parameter estimation using nature inspired meta-heuristic algorithm has been successfully attempted in this thesis.

7.3 Future Scope

The development of Iterative SDLC process based software reliability growth models and hybrid Swarm-Evolutionary algorithm with their application for Open source software system's reliability estimation are viewed as the initial point in this area. Thus, there is a widespread scope of research in this area. Following are some of the major area for future extension of the proposed work:

1. Proposed model can be further extended and validated using more techniques of software development. More real world software failure datasets can be used to validate the efficiency of proposed software reliability models.
2. Proposed hybrid nature inspired algorithm can be applied on other group of software reliability model parameter estimation in order to make better estimation of software reliability.

References

[1] IEEE, “IEEE standard Glossary of Software Engineering Terminology, IEEE std. 729-19833, IEEE CS order no. 729.” 1983.

[2] H. Pham, “Software Reliability Modeling,” in *Springer Series in Reliability Engineering*, Springer London, 2006, pp. 153–177.

[3] M. Xie, *Software reliability modelling*. World Scientific Pub Co Pte Lt, 1991.

[4] K. G. Santosh, *Numerical Methods for Engineer*, 3rd ed. Age New International, 2015.

[5] X.-S. Yang, *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[6] X.-S. Yang, “Nature-inspired mateheuristic algorithms: Success and new challenges,” *J Comput. Eng. Inf. Technol*, vol. 1, no. 1, pp. 1–3, 2012.

[7] X.-S. Yang, “Analysis of Algorithms,” in *Nature-Inspired Optimization Algorithms*, Elsevier, 2014, pp. 23–44.

[8] S. Sahney, M. J. Benton, and P. A. Ferry, “Links between global taxonomic diversity, ecological diversity and the expansion of vertebrates on land,” *Biol. Lett.*, vol. 6, no. 4, pp. 544–547, 2010.

[9] D. Karabo\uga and B. Ba\cstürk, “Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems,” *LNCS Adv. Soft Comput. Found. Fuzzy Log. Soft Comput.*, vol. 4529, pp. 789–798, 2007.

[10] R. Storn and K. Price, “Differential evolution--a simple and efficient heuristic for global optimization over continuous spaces,” *J. Glob. Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[11] H. Pham, *System software reliability*. Springer Science & Business Media, 2007.

[12] A. L. Goel, "Software reliability models: Assumptions, limitations, and applicability," *IEEE Trans. Softw. Eng.*, vol. 12, pp. 1411–1423, 1985.

[13] K. Sharma, R. Garg, C. K. Nagpal, and R. K. Garg, "Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach," *{IEEE} Trans. Reliab.*, vol. 59, no. 2, pp. 266–276, Jun. 2010.

[14] A. L. Goel and K. Okumoto, "A Markovian model for reliability and other performance measures of software systems," in *National Computer Conference*, 1979, pp. 769–774.

[15] K. Sahu and R. K. Srivastava, "Revisiting Software Reliability," in *Data Management, Analytics and Innovation*, Springer Singapore, 2018, pp. 221–235.

[16] M. Xie and M. Zhao, "The Schneidewind software reliability model revisited," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 184–192, 1992.

[17] J. Xavier, A. Macêdo, R. Matias, and L. Borges, "A survey on research in software reliability engineering in the last decade," in *Proceedings of the 29th Annual {ACM} Symposium on Applied Computing - {SAC} {\textquotesingle}14*, 2014.

[18] M. Ohba, "Inflection S-shaped software reliability growth model," in *Stochastic models in reliability theory*, Springer, 1984, pp. 144–162.

[19] X. Zhang, X. Teng, H. Pham, and S. Member, "Considering Fault Removal Efficiency in Software Reliability Assessment," vol. 33, no. 1, pp. 114–120, 2003.

[20] X. Li, Y. F. Li, M. Xie, and S. H. Ng, "Reliability analysis and optimal version-updating for open source software," *Inf. Softw. Technol.*, vol. 53, no. 9, pp. 929–936, Sep. 2011.

[21] M. Xie, G. Y. Hong, and C. Wohlin, “Software reliability prediction incorporating information from a similar project,” *J. Syst. Softw.*, vol. 49, no. 1, pp. 43–48, 1999.

[22] H. Pham, L. Nordmann, and Z. Zhang, “A general imperfect-software-debugging model with S-shaped fault-detection rate,” *IEEE Trans. Reliab.*, vol. 48, no. 2, pp. 169–175, 1999.

[23] C. Huang, M. R. Lyu, S. Member, and S. Kuo, “A Unified Scheme of Some Nonhomogenous Poisson Process Models for Software Reliability Estimation,” vol. 29, no. 3, pp. 261–269, 2003.

[24] C. Hsu and C. Huang, “Optimal Weighted Combinational Models for Software Reliability Estimation and Analysis,” vol. 63, no. 3, pp. 731–749.

[25] T. Li and K. Wu, “A NHPP software reliability growth model considering learning process and number of residual faults,” *J. Converg. Inf. Technol.*, vol. 7, no. 13, pp. 127–134, 2012.

[26] S. Yamada, “Software Reliability Growth Modeling:,” no. 12, pp. 1431–1437, 1985.

[27] S. Yamada, “S-Shaped Reliability Growth Modeling for Software Error Detection V) 3-Fte,” no. 5, pp. 475–479, 1983.

[28] S. Inoue and S. Yamada, “Two-Dimensional Software Reliability Measurement Technologies,” pp. 223–227, 2009.

[29] S. Inoue, S. Yamada, and A. T. Model, “A Bootstrap Method for Software Reliability Assessment Based on a Discretized NHPP Model,” no. 2, pp. 23–27.

[30] S. Yamada, M. Ohba, and S. Osaki, “S-shaped reliability growth modeling for software

error detection," *IEEE Trans. Reliab.*, vol. 32, no. 5, pp. 475–484, 1983.

[31] S. Yamada, H. Ohtera, and M. Ohba, "Testing-domain dependent software reliability models," *Comput. Math. with Appl.*, vol. 24, no. 1–2, pp. 79–86, 1992.

[32] S. Yamada, K. Tokuno, and S. Osaki, "Imperfect debugging models with fault introduction rate for software reliability assessment," *Int. J. Syst. Sci.*, vol. 23, no. 12, pp. 2241–2252, 1992.

[33] V. R. Basil and A. J. Turner, "Iterative enhancement: A practical technique for software development," *{IEEE} Trans. Softw. Eng.*, vol. {SE}-1, no. 4, pp. 390–396, Dec. 1975.

[34] RADC-TR-87-171, "Rome Laboratory, Methodology for software reliability prediction and assessment Technical Report," 1987.

[35] C. Smidts, M. Stutzke, and R. W. Stoddard, "Software reliability modeling: an approach to early reliability prediction," *IEEE Trans. Reliab.*, vol. 47, no. 3, pp. 268–278, 1998.

[36] K. Goševa-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Perform. Eval.*, vol. 45, no. 2–3, pp. 179–204, Jul. 2001.

[37] S. S. Gokhale, W. E. Wong, K. S. Trivedi, and J. R. Horgan, "An analytical approach to architecture-based software reliability prediction," in *Proceedings. {IEEE} International Computer Performance and Dependability Symposium. {IPDS}{\textquotesingle}98 (Cat. No.98TB100248)*.

[38] S. S. Gokhale, M. R. Lyu, and K. S. Trivedi, "Analysis of Software Fault Removal Policies Using a Non-Homogeneous Continuous Time Markov Chain," *Softw. Qual. J.*, vol. 12, no. 3, pp. 211–230, Sep. 2004.

[39] W.-L. Wang, Y. Wu, and M.-H. Chen, “An architecture-based software reliability model,” in *Proceedings 1999 Pacific Rim International Symposium on Dependable Computing*.

[40] SS Gokhale and K. Trivedi, “A Time Structure Based Model of Software Reliability,” *Ann. Softw. Eng.*, vol. 8, pp. 85–121, 1999.

[41] K. S. M. Swapna S Gokhale Peter N and Trivedi, “Important milestones in software reliability modeling,” in *Software Engineering and Knowledge Engineering (SEKE’96), Lake Tahoe, NV*, 1996, pp. 345–352.

[42] S. N. Weiss and E. J. Weyuker, “An extended domain-based model of software reliability,” *{IEEE} Trans. Softw. Eng.*, vol. 14, no. 10, pp. 1512–1524, Oct. 1988.

[43] B. Littlewood, “Software reliability model for modular program structure,” *IEEE Trans. Reliab.*, vol. 28, no. 3, pp. 241–246, 1979.

[44] Z. Jelinski and P. Moranda, *Software Reliability Research*. Academic Press New York and London, 1972.

[45] P. B. Moranda, “An error detection model for application during software development,” *IEEE Trans. Reliab.*, vol. 30, no. 4, pp. 309–312, 1981.

[46] A. N. Sukert, “An investigation of software reliability models,” in *Annual Reliability and Maintainability Symposium, Philadelphia, Pa*, 1977, pp. 478–484.

[47] G. S. Mahapatra, P. Roy, G. S.Mahapatra, and P. Roy, “Modified Jelinski-Moranda Software Reliability Model with Imperfect Debugging Phenomenon,” *Int. J. Comput. Appl.*, vol. 48, no. 18, pp. 38–46, Jun. 2012.

[48] J. D. Musa and K. Okumoto, “A logarithmic Poisson execution time model for software reliability measurement,” in *Proceedings of the 7th international conference on Software engineering*, 1984, pp. 230–238.

[49] L. I. Al Turk and E. G. Alsolami, “Jelinski-Moranda Software Reliablity Growth Model : A Brief Literature and Modification,” *Int. J. Softw. Eng. Appl.*, vol. 7, no. 2, pp. 33–44, Mar. 2016.

[50] Y.-C. Chang and C.-T. Liu, “A generalized {JM} model with applications to imperfect debugging in software reliability,” *Appl. Math. Model.*, vol. 33, no. 9, pp. 3578–3588, Sep. 2009.

[51] H. Pham, *Springer handbook of engineering statistics*. Springer Science & Business Media, 2006.

[52] B. Littlewood and J. L. Verrall, “A Bayesian reliability growth model for computer software,” *J. R. Stat. Soc. Ser. C (Applied Stat.*, vol. 22, no. 3, pp. 332–346, 1973.

[53] T. A. Mazzuchi and R. Soyer, “A Bayes empirical-Bayes model for software reliability,” *IEEE Trans. Reliab.*, vol. 37, no. 2, pp. 248–254, 1988.

[54] G. J. Schick and R. W. Wolverton, “An analysis of competing software reliability models,” *IEEE Trans. Softw. Eng.*, vol. SE-4, no. 2, pp. 104–120, 1978.

[55] S. Yamada, K. Tokuno, and Y. Kasano, “Quantitative assessment models for software safety/reliability,” *Electron. Commun. Japan, Part II Electron. (English Transl. Denshi Tsushin Gakkai Ronbunshi)*, vol. 81, no. 5, pp. 33–43, 1998.

[56] J. G. Shanthikumar, “A general software reliability model for performance prediction,”

Microelectron. Reliab., vol. 21, no. 5, pp. 671–682, 1981.

[57] W. S. Jewell, "Bayesian extensions to a basic model of software reliability," *IEEE Trans. Softw. Eng.*, no. 12, pp. 1465–1471, 1985.

[58] H. Joe and N. Reid, "On the Software Reliability Models of Jelinski-Moranda and Littlewood," no. 3, pp. 216–218, 1985.

[59] T. F. Ho, W. C. Chan, and C. G. Chung, "A quantum modification to the Jelinski-Moranda software reliability model," *Midwest Symp. Circuits Syst.*, vol. 1, pp. 339–342, 1991.

[60] P. J. Boland and H. Singh, "A birth-process approach to Moranda's geometric software-reliability model," *IEEE Trans. Reliab.*, vol. 52, no. 2, pp. 168–174, 2003.

[61] K. Rinsaka and T. Dohi, "Who solved the optimal software release problems based on Markovian software reliability model?," in *The 2004 47th Midwest Symposium on Circuits and Systems, 2004. MWSCAS'04.*, 2004, vol. 3, pp. iii--475.

[62] A. Washburn, "A sequential Bayesian generalization of the Jelinski--Moranda software reliability model," *Nav. Res. Logist.*, vol. 53, no. 4, pp. 354–362, 2006.

[63] Z. Luo, P. Cao, G. Tang, and L. Wu, "A modification to the Jelinski-Moranda software reliability growth model based on cloud model theory," in *2011 Seventh International Conference on Computational Intelligence and Security*, 2011, pp. 195–198.

[64] N. D. Singpurwalla, "A UNIFICATION OF SOME SOFTWARE RELIABILITY MODELS*," vol. 6, no. 3, pp. 781–790, 1985.

[65] Y. Lian, Y. Tang, and Y. Wang, "Objective Bayesian analysis of JM model in software

reliability," *Comput. Stat. Data Anal.*, vol. 109, pp. 199–214, 2017.

[66] A. L. Goel, M. Ieee, and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Trans. Reliab.*, vol. 28, no. 3, pp. 206–211, 1979.

[67] S. YAMADA and S. OSAKI, "S-shaped software reliability growth models with four types of software error data," *Int. J. Syst. Sci.*, vol. 14, no. 6, pp. 683–692, 1983.

[68] R. K. Iyer and P. Velardi, "Hardware-Related Software Errors: Measurement and Analysis.," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 2, pp. 223–231, 1985.

[69] S. Yamada and S. Osaki, "Software reliability growth modeling: Models and applications," *IEEE Trans. Softw. Eng.*, no. 12, pp. 1431–1437, 1985.

[70] N. D. Singpurwalla and R. Soyer, "Assessing (Software) Reliability Growth Using a Random Coefficient Autoregressive Process and Its Ramifications," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 12, pp. 1456–1464, 1985.

[71] H. Yamada, Shigeru and Osaki, Shunji and Narihisa, "A software reliability growth model with two types of errors," *RAIRO-Operations Res.*, vol. 19, no. 1, pp. 87–104, 1985.

[72] V. K. P. Kumar, S. Hariri, and C. S. Raghavendra, "Distributed Program Reliability Analysis," *IEEE Trans. Softw. Eng.*, vol. SE-12, no. 1, pp. 42–50, 1986.

[73] S. Yamada, H. Ohtera, and H. Narihisa, "Software reliability growth models with testing-effort," *IEEE Trans. Reliab.*, vol. 35, no. 1, pp. 19–23, 1986.

[74] S. Nakagawa, Yutaka and Hanata, "An error complexity model for software reliability measurement," in *11th International Conference on Software Engineering,IEEE*, 1989,

pp. 230–236.

[75] Allenwood, “Predicting software reliability,” 1996.

[76] H. Yamada, Shigeru and Ohtera, “Software reliability growth models for testing-effort control,” *Eur. J. Oper. Res.*, vol. 46, no. 3, pp. 343–349, 1990.

[77] P. Kareer, Nishi and Kapur, PK and Grover, “An S-shaped software reliability growth model with two types of errors,” *Microelectron. Reliab.*, vol. 30, no. 6, pp. 1085–1090, 1990.

[78] M. Xie and M. Zhao, “The Schneidewind software reliability model revisited,” in *[1992] Proceedings Third International Symposium on Software Reliability Engineering*, 1992, pp. 184–192.

[79] N. Karunanithi, D. Whitley, and Y. K. Malaiya, “Prediction of software reliability using connectionist models,” *IEEE Trans. Softw. Eng.*, no. 7, pp. 563–574, 1992.

[80] P. K. Kapur and R. B. Garg, “A software reliability growth model for an error-removal phenomenon,” *Softw. Eng. J.*, vol. 7, no. 4, pp. 291–294, 1992.

[81] M. Sahinoglu, “Compound-Poisson software reliability model,” *IEEE Trans. Softw. Eng.*, no. 7, pp. 624–630, 1992.

[82] S. Yamada, K. Tokuno, and S. Osaki, “Software reliability measurement in imperfect debugging environment and its application,” *Reliab. Eng. Syst. Saf.*, vol. 40, no. 2, pp. 139–147, 1993.

[83] P. K. Kapur and S. Younes, “Modelling an imperfect debugging phenomenon in software reliability,” *Microelectron. Reliab.*, vol. 36, no. 5, pp. 645–650, 1996.

[84] W. Lynch, Tom and Pham, Hoang and Kuo, "Modeling software-reliability with multiple failure-types and imperfect debugging," in *Proceedings of Annual Reliability and Maintainability Symposium (RAMS),IEEE*, 1994, pp. 235–240.

[85] S. Gokhale, T. Philip, P. N. Marinos, and K. S. Trivedi, "Unification of Finite Failure NHPP Models through Test Coverage," in *Proc. of Intl. Symposium on Software Reliability Engineering (ISSRE'96)*, pp. 289–299.

[86] H. Pham and X. Zhang, "An NHPP software reliability model and its comparison," *Int. J. Reliab. Qual. Saf. Eng.*, vol. 4, no. 03, pp. 269–282, 1997.

[87] Y. Huang, Chin-Yu and Kuo, Sy-Yen and Chen, "Analysis of a software reliability growth model with logistic testing-effort function," in *Proceedings The Eighth International Symposium on Software Reliability Engineering,IEEE*, 1997, pp. 378--388.

[88] L. Kuo, J. C. Lee, K. Choi, and T. Y. Yang, "Bayes inference for S-shaped software-reliability growth models," *IEEE Trans. Reliab.*, vol. 46, no. 1, pp. 76–80, 1997.

[89] M. E. Helander, M. Zhao, and N. Ohlsson, "Planning models for software reliability and cost," *IEEE Trans. Softw. Eng.*, vol. 24, no. 6, pp. 420–434, 1998.

[90] S. S. Gokhale, K. S. Trivedi, and B. College, "Log-Logistic Software Reliability Growth Model University of California s ds dm t dt."

[91] M.-H. Chen, M. R. Lyu, and W. E. Wong, "Effect of code coverage on software reliability measurement," *IEEE Trans. Reliab.*, vol. 50, no. 2, pp. 165–170, 2001.

[92] S. Yamada and T. FUJIWARA, "Testing-domain dependent software reliability growth models and their comparisons of goodness-of-fit," *Int. J. Reliab. Qual. Saf. Eng.*, vol. 8,

no. 03, pp. 205–218, 2001.

[93] X. Teng and H. Pham, “A software-reliability growth model for N-version programming systems,” *IEEE Trans. Reliab.*, vol. 51, no. 3, pp. 311–321, 2002.

[94] S. INOUE and S. YAMADA, “Testing-coverage dependent software reliability growth modeling,” *Int. J. Reliab. Qual. Saf. Eng.*, vol. 11, no. 04, pp. 303–312, 2004.

[95] C.-Y. Huang, C.-T. Lin, S.-Y. Kuo, M. R. Lyu, and C.-C. Sue, “Software reliability growth models incorporating fault dependency with various debugging time lags,” in *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, 2004, pp. 186–191.

[96] H. Pham, “A generalized logistic software reliability growth model,” *Opsearch*, vol. 42, no. 4, pp. 322–331, 2005.

[97] X. Teng and H. Pham, “A new methodology for predicting software reliability in the random field environments,” *IEEE Trans. Reliab.*, vol. 55, no. 3, pp. 458–468, 2006.

[98] J. H. Lo and C. Y. Huang, “An integration of fault detection and correction processes in software reliability analysis,” *J. Syst. Softw.*, vol. 79, no. 9, pp. 1312–1323, 2006.

[99] C.-T. Lin and C.-Y. Huang, “Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models,” *J. Syst. Softw.*, vol. 81, no. 6, pp. 1025–1038, 2008.

[100] S. Inoue and S. Yamada, “Generalized discrete software reliability modeling with effect of program size,” *IEEE Trans. Syst. Man, Cybern. A Syst. Humans*, vol. 37, no. 2, pp. 170–179, 2007.

[101] Y. P. Wu, Q. P. Hu, M. Xie, and S. H. Ng, "Modeling and analysis of software fault detection and correction process by considering time dependency," *IEEE Trans. Reliab.*, vol. 56, no. 4, pp. 629–642, 2007.

[102] P. K. Kapur, A. Kumar, K. Yadav, and S. K. KHATRI, "Software reliability growth modelling for errors of different severity using change point," *Int. J. Reliab. Qual. Saf. Eng.*, vol. 14, no. 04, pp. 311–326, 2007.

[103] H. Pham, "An imperfect-debugging fault-detection dependent-parameter software," *Int. J. Autom. Comput.*, vol. 4, no. 4, pp. 325–327, 2007.

[104] C. Y. Huang and W. C. Huang, "Software reliability analysis and measurement using finite and infinite server queueing models," *IEEE Trans. Reliab.*, vol. 57, no. 1, pp. 192–203, 2008.

[105] H. Li, Q. Li, and M. Lu, "Software reliability modeling with logistic test coverage function," in *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, 2008, pp. 319–320.

[106] S. Inoue and S. Yamada, "Two-dimensional software reliability assessment with testing-coverage," *Proc. - 2nd IEEE Int. Conf. Secur. Syst. Integr. Reliab. Improv. SSIRI 2008*, pp. 150–157, 2008.

[107] P. K. Kapur, D. N. Goswami, A. Bardhan, and O. Singh, "Flexible software reliability growth model with testing effort dependent learning process," *Appl. Math. Model.*, vol. 32, no. 7, pp. 1298–1307, 2008.

[108] N. Ahmad, M. G. M. Khan, and L. S. Rafi, "A study of testing-effort dependent inflection S-shaped software reliability growth models with imperfect debugging," *Int. J. Qual.*

Reliab. Manag., vol. 27, no. 1, pp. 89–110, 2010.

[109] S. Hwang and H. Pham, "Quasi-renewal time-delay fault-removal consideration in software reliability modeling," *IEEE Trans. Syst. Man, Cybern. A Syst. Humans*, vol. 39, no. 1, pp. 200–209, 2008.

[110] S. Inoue and S. Yamada, "Two-dimensional software reliability measurement technologies," in *2009 IEEE International Conference on Industrial Engineering and Engineering Management*, 2009, pp. 223–227.

[111] P. K. Kapur, O. Shatnawi, A. G. Aggarwal, and R. Kumar, "Unified framework for developing testing effort dependent software reliability growth models," *WSEAS Trans. Syst.*, vol. 8, no. 4, pp. 521–531, 2009.

[112] M. Staron, W. Meding, and B. Söderqvist, "A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation," *Inf. Softw. Technol.*, vol. 52, no. 10, pp. 1069–1079, 2010.

[113] P. Kumar and Y. Singh, "A software reliability growth model for three-tier client server system," *Int. J. Comput. Appl.*, vol. 975, p. 8887, 2010.

[114] P. K. Kapur, A. Tandon, and G. Kaur, "Multi up-gradation software reliability model," *2010 2nd Int. Conf. Reliab. Saf. Hazard, ICRESH-2010 Risk-Based Technol. Physics-of-Failure Methods*, pp. 468–474, 2010.

[115] C.-Y. Huang and C.-T. Lin, "Analysis of software reliability modeling considering testing compression factor and failure-to-fault relationship," *IEEE Trans. Comput.*, vol. 59, no. 2, pp. 283–288, 2009.

[116] S. K. Chandran, A. Dimov, and S. Punnekkat, “Modeling uncertainties in the estimation of software reliability--a pragmatic approach,” in *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, 2010, pp. 227–236.

[117] N. Quadri, SMK and Ahmad, “Software Reliability Growth Modeling with New Modified Weibull Testing–effort and Optimal Release Policy,” *Int. J. Comput. Appl.*, vol. 6, no. 12, pp. 1–10, 2010.

[118] J.-R. Hsu, Chao-Jung and Huang, Chin-Yu and Chang, “Enhancing software reliability modeling and prediction through the introduction of time-variable fault reduction factor,” *Appl. Math. Model.*, vol. 35, no. 1, pp. 506–511, 2011.

[119] P. K. Kapur, H. Pham, A. G. Aggarwal, and G. Kaur, “Two dimensional multi-release software reliability modeling and optimal release planning,” *IEEE Trans. Reliab.*, vol. 61, no. 3, pp. 758–768, 2012.

[120] M. P. Wiper, A. P. Palacios, and J. M. Marin, “Bayesian software reliability prediction using software metrics information,” *Qual. Technol. Quant. Manag.*, vol. 9, no. 1, pp. 35–44, 2012.

[121] D. K. Yadav, S. K. Chaturvedi, and R. B. Misra, “Forecasting time-between-failures of software using fuzzy time series approach,” in *2012 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS)*, 2012, pp. 1–8.

[122] H. Pham, “A new software reliability model with Vtub-shaped fault-detection rate and the uncertainty of operating environments,” *Optimization*, vol. 63, no. 10, pp. 1481–1490, 2014.

[123] H. Okamura, T. Dohi, and S. Osaki, “Software reliability growth models with normal

failure time distributions," *Reliab. Eng. Syst. Saf.*, vol. 116, pp. 135–141, 2013.

[124] S. Zhang, Ce and Cui, Gang and Liu, Hongwei and Meng, Fanchao and Wu, "A Unified and Flexible Framework of Imperfect Debugging Dependent SRGMs with Testing-Effort," *J. Multimed.*, vol. 9, no. 2, 2014.

[125] R. Peng, Y. F. Li, W. J. Zhang, and Q. P. Hu, "Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction," *Reliab. Eng. Syst. Saf.*, vol. 126, pp. 37–43, 2014.

[126] H. Pham, "Loglog fault-detection rate and testing coverage software reliability models subject to random environments," *Vietnam J. Comput. Sci.*, vol. 1, no. 1, pp. 39–45, Nov. 2013.

[127] H. B. Yadav and D. K. Yadav, "Early software reliability analysis using reliability relevant software metrics," *Int. J. Syst. Assur. Eng. Manag.*, vol. 8, no. 4, pp. 2097–2108, 2017.

[128] C.-J. Hsu and C.-Y. Huang, "Optimal weighted combinational models for software reliability estimation and analysis," *IEEE Trans. Reliab.*, vol. 63, no. 3, pp. 731–749, 2014.

[129] R. Rana *et al.*, "Selecting software reliability growth models and improving their predictive accuracy using historical projects data," *J. Syst. Softw.*, vol. 98, pp. 59–78, 2014.

[130] Q. Li, H. Li, and M. Lu, "Incorporating S-shaped testing-effort functions into NHPP software reliability model with imperfect debugging," *J. Syst. Eng. Electron.*, vol. 26, no. 1, pp. 190–207, 2015.

[131] Z. Li, M. Mobin, and T. Keyser, “Multi-objective and multi-stage reliability growth planning in early product-development stage,” *IEEE Trans. Reliab.*, vol. 65, no. 2, pp. 769–781, 2015.

[132] M. Wayne and M. Modarres, “A Bayesian model for complex system reliability growth under arbitrary corrective actions,” *IEEE Trans. Reliab.*, vol. 64, no. 1, pp. 206–220, 2015.

[133] J. Yang, Y. Liu, M. Xie, and M. Zhao, “Modeling and analysis of reliability of multi-release open source software incorporating both fault detection and correction processes,” *J. Syst. Softw.*, vol. 115, pp. 102–110, May 2016.

[134] S. Inoue, Shinji and Ikeda, Jun and Yamada, “Bivariate change-point modeling for software reliability assessment with uncertainty of testing-environment factor,” *Ann. Oper. Res.*, vol. 244, no. 1, pp. 209–220, 2016.

[135] M. Zhu and H. Pham, “A software reliability model with time-dependent fault detection and fault removal,” *Vietnam J. Comput. Sci.*, vol. 3, no. 2, pp. 71–79, 2016.

[136] H. Pham, “A generalized fault-detection software reliability model subject to random operating environments,” *Vietnam J. Comput. Sci.*, vol. 3, no. 3, pp. 145–150, 2016.

[137] J. Xu and S. Yao, “Software Reliability Growth Model with Partial Differential Equation for Various Debugging Processes,” *Math. Probl. Eng.*, vol. 2016, 2016.

[138] S. Ramasamy and A. M. J. Muthu Kumaran, “Dynamically weighted combination of fault - Based Software Reliability Growth Models,” *Indian J. Sci. Technol.*, vol. 9, no. 22, pp. 10–13, 2016.

[139] J. Wang, “An imperfect software debugging model considering irregular fluctuation of fault introduction rate,” *Qual. Eng.*, vol. 29, no. 3, pp. 377–394, 2017.

[140] K. Honda, H. Washizaki, and Y. Fukazawa, “Generalized Software Reliability Model Considering Uncertainty and Dynamics: Model and Applications,” *Int. J. Softw. Eng. Knowl. Eng.*, vol. 27, no. 6, pp. 967–993, 2017.

[141] Z. Li, M. Mobin, and T. Keyser, “Multi-Objective and Multi-Stage Reliability Growth Planning in Early Product-Development Stage,” *IEEE Trans. Reliab.*, vol. 65, no. 2, pp. 769–781, 2016.

[142] A. G. Aggarwal, P. K. Kapur, and N. Nijhawan, “A discrete SRGM for multi-release software system with faults of different severity,” *Int. J. Oper. Res.*, vol. 32, no. 2, pp. 156–168, 2018.

[143] P. Erto, M. Giorgio, and A. Lepore, “The Generalized Inflection S-Shaped Software Reliability Growth Model,” *IEEE Trans. Reliab.*, 2018.

[144] K. Lee, Da and Chang, In and Pham, Hoang and Song, “A software reliability model considering the syntax error in uncertainty environment, optimal release time, and sensitivity analysis,” *Appl. Sci.*, vol. 8, no. 9, p. 1483, 2018.

[145] A. Aggarwal, Anu G and Dhaka, Vikas and Nijhawan, Nidhi and Tandon, *Reliability growth analysis for multi-release open source software systems with change point*. 2019.

[146] V. B. Singh, M. Sharma, and H. Pham, “Entropy Based Software Reliability Analysis of Multi-Version Open Source Software,” *{IEEE} Trans. Softw. Eng.*, vol. 44, no. 12, pp. 1207–1223, Dec. 2018.

[147] A. Gupta, Ritu and Jain, Madhu and Jain, *Software Reliability Growth Model in Distributed Environment Subject to Debugging Time Lag*. 2019.

[148] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of {ICNN}{\textquotesingle}95 - International Conference on Neural Networks*.

[149] J. H. Holland, "Genetic Algorithm," *Nat. Sci. Am.*, vol. 267, no. 1, pp. 66–72, 1992.

[150] A. Alrajeh, "Machine Translation Systems," 2011.

[151] H. A. Abbass, "{MBO}: marriage in honey bees optimization-a Haplometrosis polygynous swarming approach," in *Proceedings of the 2001 Congress on Evolutionary Computation ({IEEE} Cat. No.01TH8546)*.

[152] J. R. Koza, "Survey of genetic algorithms and genetic programming," in *Proceedings of {WESCON}{\textquotesingle}95*.

[153] D. Simon, "Biogeography-Based Optimization," *{IEEE} Trans. Evol. Comput.*, vol. 12, no. 6, pp. 702–713, Dec. 2008.

[154] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science (80-.).*, vol. 220, no. 4598, pp. 671–680, May 1983.

[155] H. Shah-Hosseini and H. S. Hosseini, "Principal components analysis by the galaxy-based search algorithm: a novel metaheuristic for continuous optimisation," *Int. J. Comput. Sci. Eng.*, vol. 6, no. 1–2, pp. 132–140, 2011.

[156] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, "{GSA}: A Gravitational Search Algorithm," *Inf. Sci. (Ny).*, vol. 179, no. 13, pp. 2232–2248, Jun. 2009.

[157] A. Hatamlou, "Black hole: A new heuristic optimization approach for data clustering," *Inf.*

Sci. (Ny)., vol. 222, pp. 175–184, Feb. 2013.

[158] A. Kaveh and S. Talatahari, “A novel heuristic optimization method: charged system search,” *Acta Mech.*, vol. 213, no. 3–4, pp. 267–289, Jan. 2010.

[159] X.-S. Yang, “A New Metaheuristic Bat-Inspired Algorithm,” in *Nature Inspired Cooperative Strategies for Optimization ({NICSO} 2010)*, Springer Berlin Heidelberg, 2010, pp. 65–74.

[160] A. K. Tripathi, K. Sharma, and M. Bala, “Dynamic frequency based parallel k-bat algorithm for massive data clustering (DFBPKBA),” *Int. J. Syst. Assur. Eng. Manag.*, vol. 9, no. 4, pp. 866–874, Sep. 2018.

[161] T. Ashish, S. Kapil, and B. Manju, “Parallel bat algorithm-based clustering using mapreduce,” in *Networking Communication and Data Knowledge Engineering*, Springer, 2018, pp. 73–82.

[162] D. Karaboga and C. Ozturk, “A novel clustering approach: Artificial Bee Colony (ABC) algorithm,” *Appl. Soft Comput.*, vol. 11, no. 1, pp. 652–657, 2011.

[163] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” *{IEEE} Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.

[164] A. Kaveh and N. Farhoudi, “A new optimization method: Dolphin echolocation,” *Adv. Eng. Softw.*, vol. 59, pp. 53–70, May 2013.

[165] C. Yang, X. Tu, and J. Chen, “Algorithm of Marriage in Honey Bees Optimization Based on the Wolf Pack Search,” in *The 2007 International Conference on Intelligent Pervasive Computing ({IPC} 2007)*, 2007.

[166] Li.X, “A new intelligent optimization-artificial fish swarm algorithm [Doctor thesis,” 2003.

[167] R. Martin and W. Stephen, “Termite: A swarm intelligent routing algorithm for mobilewireless Ad-Hoc networks,” in *Studies in Computational Intelligence*, Springer Berlin Heidelberg, 2006, pp. 155–184.

[168] P. C. Pinto, T. A. Runkler, and J. M. C. Sousa, “Wasp Swarm Algorithm for Dynamic {MAX}-{SAT} Problems,” in *Adaptive and Natural Computing Algorithms*, Springer Berlin Heidelberg, pp. 350–357.

[169] A. Mucherino, O. Seref, O. Seref, O. E. Kundakcioglu, and P. Pardalos, “Monkey search: a novel metaheuristic search for global optimization,” in *{AIP} Conference Proceedings*, 2007.

[170] X. Lu and Y. Zhou, “A Novel Global Convergence Algorithm: Bee Collecting Pollen Algorithm,” in *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, Springer Berlin Heidelberg, pp. 518–525.

[171] X.-S. Yang and S. Deb, “Cuckoo Search via L&#x00E9$\mathsemicolon$vy flights,” in *2009 World Congress on Nature & Biologically Inspired Computing ({NaBIC})*, 2009.

[172] Y. Shiqin, J. Jianjun, and Y. Guangxing, “A Dolphin Partner Optimization,” in *2009 {WRI} Global Congress on Intelligent Systems*, 2009.

[173] X. S. Yang, “Firefly algorithm, stochastic test functions and design optimisation,” *Int. J. Bio-Inspired Comput.*, vol. 2, no. 2, p. 78, 2010.

[174] W.-T. Pan, “A new Fruit Fly Optimization Algorithm: Taking the financial distress model

as an example," *Knowledge-Based Syst.*, vol. 26, pp. 69–74, Feb. 2012.

[175] S. Mirjalili and A. Lewis, "The Whale Optimization Algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016.

[176] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[177] A. K. Tripathi, K. Sharma, and M. Bala, "A novel clustering method using enhanced grey wolf optimizer and mapreduce," *Big data Res.*, vol. 14, pp. 93–100, 2018.

[178] W. H. Lim and N. A. M. Isa, "Teaching and peer-learning particle swarm optimization," *Appl. Soft Comput.*, vol. 18, pp. 39–58, May 2014.

[179] G. Barbarosoglu and D. Ozgur, "A tabu search algorithm for the vehicle routing problem," *Comput. Oper. Res.*, vol. 26, no. 3, pp. 255–270, Mar. 1999.

[180] W. Bi, G. C. Dandy, and H. R. Maier, "Improved genetic algorithm optimization of water distribution system design by incorporating domain knowledge," *Environ. Model. Softw.*, vol. 69, pp. 370–381, 2015.

[181] M. S. Ismail, M. Moghavvemi, and T. M. I. Mahlia, "Genetic algorithm based optimization on modeling and design of hybrid renewable energy systems," *Energy Convers. Manag.*, vol. 85, pp. 120–130, 2014.

[182] B. Pachauri, A. Kumar, and J. Dhar, "Software reliability growth modeling with dynamic faults and release time optimization using GA and MAUT," *Appl. Math. Comput.*, vol. 242, pp. 500–509, 2014.

[183] C.-J. Hsu and C.-Y. Huang, "A study on the applicability of modified genetic algorithms

for the parameter estimation of software reliability modeling," in *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*, 2010, pp. 531–540.

[184] L. C. and Y. P. Chen, "Application of improved differential evolution calculation method based on contious power flow to analyasis of marginal static voltage stability," *Power Eng.*, vol. 26, pp. 756–760, 2006.

[185] C. Jin and S.-W. Jin, "Parameter optimization of software reliability growth model with S-shaped testing-effort function using improved swarm intelligent optimization," *Appl. Soft Comput.*, vol. 40, pp. 283–291, 2016.

[186] H. Shayeghi, M. Mahdavi, and A. Bagheri, "Discrete PSO algorithm based optimization of transmission lines loading in TNEP problem," *Energy Convers. Manag.*, vol. 51, no. 1, pp. 112–121, 2010.

[187] T. Haryono and others, "Novel binary PSO algorithm based optimization of transmission expansion planning considering power losses," in *IOP Conference Series: Materials Science and Engineering*, 2016, vol. 128, no. 1, p. 12023.

[188] R. Malhotra and A. Negi, "Reliability modeling using particle swarm optimization," *Int. J. Syst. Assur. Eng. Manag.*, vol. 4, no. 3, pp. 275–283, 2013.

[189] A. Sheta and J. Al-Salt, "Parameter estimation of software reliability growth models by particle swarm optimization," *management*, vol. 7, p. 14, 2007.

[190] B. Akay and D. Karaboga, "Artificial bee colony algorithm variants on constrained optimization," *An Int. J. Optim. Control*, vol. 7, no. 1, p. 98, 2017.

[191] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (ABC)

algorithm," *Appl. Soft Comput.*, vol. 8, no. 1, pp. 687–697, 2008.

[192] G. Zhu and S. Kwong, "Gbest-guided artificial bee colony algorithm for numerical function optimization," *Appl. Math. Comput.*, vol. 217, no. 7, pp. 3166–3173, 2010.

[193] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony ({ABC}) algorithm," *J. Glob. Optim.*, vol. 39, no. 3, pp. 459–471, Apr. 2007.

[194] S. Mirjalili and S. Z. M. Hashim, "A new hybrid PSOGSA algorithm for function optimization," in *Computer and information application (ICCIA), 2010 international conference on*, 2010, pp. 374–377.

[195] S. Mirjalili, G.-G. Wang, and L. dos S. Coelho, "Binary optimization using hybrid particle swarm optimization and gravitational search algorithm," *Neural Comput. Appl.*, vol. 25, no. 6, pp. 1423–1435, 2014.

[196] F. Liu and Z. Zhou, "An improved QPSO algorithm and its application in the high-dimensional complex problems," *Chemom. Intell. Lab. Syst.*, vol. 132, pp. 82–90, 2014.

[197] A. Abraham, R. K. Jatoth, and A. Rajasekhar, "Hybrid differential artificial bee colony algorithm," *J. Comput. Theor. Nanosci.*, vol. 9, no. 2, pp. 249–257, 2012.

[198] Y. Li, Y. Wang, and B. Li, "A hybrid artificial bee colony assisted differential evolution algorithm for optimal reactive power flow," *Int. J. Electr. Power Energy Syst.*, vol. 52, pp. 25–33, 2013.

[199] S. S. Jadon, R. Tiwari, H. Sharma, and J. C. Bansal, "Hybrid artificial bee colony algorithm with differential evolution," *Appl. Soft Comput.*, vol. 58, pp. 11–24, 2017.

[200] D. B. R. XIAO-LI MENG, “Maximum likelihood estimation via the ECM algorithm: A general framework,” *Biometrika*, vol. 80, no. 2, pp. 267–278, 1993.

[201] Y. Minohara, Takashi and Tohma, “Parameter estimation of hyper-geometric distribution software reliability growth model by genetic algorithms,” in *Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE’95*, 1995, pp. 324–329.

[202] T. Okamura, Hiroyuki and Watanabe, Yasuhiro and Dohi, “An iterative scheme for maximum likelihood estimation in software reliability modeling,” in *14th International Symposium on Software Reliability Engineering, 2003. ISSRE*, 2003, pp. 246–256.

[203] I. J. Myung, “Tutorial on maximum likelihood estimation,” vol. 47, pp. 90–100, 2003.

[204] T. Ohishi, Koji and Okamura, Hiroyuki and Dohi, “Gompertz software reliability model: Estimation algorithm and empirical validation,” *J. Syst. Softw.*, vol. 82, no. 3, pp. 535–543, 2009.

[205] A. M. Caserta, Marco and Uribe, “Tabu search-based metaheuristic algorithm for software system reliability problems,” *Comput. \& Oper. Res.*, vol. 36, no. 3, pp. 811–822, 2009.

[206] M. E. Aljahdali, Sultan H and El-Telbany, “Software reliability prediction using multi-objective genetic algorithm,” in *009 IEEE/ACS International Conference on Computer Systems and Applications,IEEE*, 2009, pp. 293–300.

[207] J.-B. Hu, Chang-Hua and Si, Xiao-Sheng and Yang, “System reliability prediction model based on evidential reasoning algorithm with nonlinear optimization,” *Expert Syst. Appl.*, vol. 37, no. 3, pp. 2550–2562, 2010.

[208] C.-Y. Hsu, Chao-Jung and Huang, "A study on the applicability of modified genetic algorithms for the parameter estimation of software reliability modeling," in *2010 IEEE 34th Annual Computer Software and Applications Conference*, 2010, pp. 531–540.

[209] Y. Zheng, Changyou and Liu, Xiaoming and Huang, Song and Yao, "A parameter estimation method for software reliability models," *Procedia Eng.*, vol. 15, pp. 3477–3481, 2011.

[210] M. Q. Wason, Ritika and Ahmed, P and Rafiq, "New Paradigm for Software Reliability Estimation," *Int. J. Comput. Appl.*, vol. 44, no. 14, pp. 39–44, 2012.

[211] L. Shanmugam, Latha and Florence, "A comparison of parameter best estimation method for software reliability models," *Int. J. Softw. Eng. \& Appl.*, vol. 3, no. 5, p. 91, 2012.

[212] M. AL-Saati, Dr and Akram, Najla and Abd-AlKareem, "The use of cuckoo search in estimating the parameters of software reliability growth models," *arXiv Prepr. arXiv1307.6023*, 2013.

[213] A. Mallikharjuna, Rao K and Kodali, *An efficient method for parameter estimation of software reliability growth model using artificial bee colony optimization*. 2014.

[214] J. Kim, Taehyoun and Lee, Kwangkyu and Baik, "An effective approach to estimating the parameters of software reliability growth models using a real-valued genetic algorithm," *J. Syst. Softw.*, vol. 102, pp. 134–144, 2015.

[215] E. Zhao, Wei and Tao, Tao and Zio, "System reliability prediction by support vector regression with analytic selection and genetic algorithm parameters selection," *Appl. Soft Comput.*, vol. 30, pp. 792–802, 2015.

[216] K. Roy, Pratik and Mahapatra, GS and Dey, “Neuro-genetic approach on logistic model based software reliability prediction,” *Expert Syst. Appl.*, vol. 42, no. 10, pp. 4709–4718, 2015.

[217] O. P. Choudhary, Ankur and Baghel, Anurag Singh and Sangwan, “An efficient parameter estimation of software reliability growth models using gravitational search algorithm,” *Int. J. Syst. Assur. Eng. Manag.*, vol. 8, no. 1, pp. 79–88, 2017.

[218] O. P. Choudhary, Ankur and Baghel, Anurag Singh and Sangwan, “Efficient parameter estimation of software reliability growth models using harmony search,” *IET Softw.*, vol. 11, no. 6, pp. 286–291, 2017.

[219] O. P. Choudhary, Ankur and Baghel, Anurag Singh and Sangwan, “Parameter Estimation of Software Reliability Model Using Firefly Optimization,” in *Data Engineering and Intelligent Computing*, 2018, pp. 407–415.

[220] X. Teng and H. Pham, “A New Methodology for Predicting Software Reliability in the Random Field Environments,” vol. 55, no. 3, pp. 458–468, 2006.

[221] Y. K. Li, Naixin and Malaiya, “Enhancing accuracy of software reliability prediction,” in *Proceedings of 1993 IEEE International Symposium on Software Reliability Engineering*, 1993, pp. 71–79.

[222] S. Kumar and P. Ranjan, “A phase wise approach for fault identification,” *J. Inf. Optim. Sci.*, vol. 39, no. 1, pp. 223–237, Nov. 2017.

[223] R. K. R. K. Garg *et al.*, “Ranking of software engineering metrics by fuzzy-based matrix methodology,” *Softw. Testing, Verif. Reliab.*, vol. 23, no. 2, pp. 149–168, May 2011.

[224] R. Rana *et al.*, "Selecting software reliability growth models and improving their predictive accuracy using historical projects data," *J. Syst. Softw.*, vol. 98, pp. 59–78, 2014.

[225] H. Pham and X. Zhang, "NHPP software reliability and cost models with testing coverage," *Eur. J. Oper. Res.*, vol. 145, no. 2, pp. 443–454, 2003.

[226] B. W. Boehm, "Software engineering economics," *IEEE Trans. Softw. Eng.*, vol. 1, pp. 4–21, 1984.

[227] W. W. Royce, "Managing the development of large software systems: concepts and techniques," in *Proceedings of the 9th international conference on Software Engineering*, 1987, pp. 328–338.

[228] C. Larman and V. R. Basili, "Iterative and incremental developments. a brief history," *Computer (Long. Beach. Calif).*, vol. 36, no. 6, pp. 47–56, Jun. 2003.

[229] P. Kruchten, "The rational unified process: an introduction," in *Addison-Wesley Professional*, 2004.

[230] L. Tal, "Getting Started with Agile Manager," in *Agile Software Development with {HP} Agile Manager*, Apress, 2015, pp. 15–28.

[231] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes," *{IEEE} Trans. Softw. Eng.*, vol. 33, no. 6, pp. 402–419, Jun. 2007.

[232] C. Larman, *Agile and iterative development: a manager's guide,Addison-Wesley*

Professional. 2004.

[233] Abrahamsson, “Extreme programming: first results from a controlled case study,” in *Proceedings of the 20th {IEEE} Instrumentation Technology Conference (Cat No 03CH37412) {EURMIC}-03*, 2003.

[234] B. Boehm, “A spiral model of software development and enhancement,” *{ACM} {SIGSOFT} Softw. Eng. Notes*, vol. 11, no. 4, pp. 22–42, Aug. 1986.

[235] M. A. Akbar *et al.*, “Improving the Quality of Software Development Process by Introducing a New Methodology{\textendash}{AZ}-Model,” *{IEEE} Access*, vol. 6, pp. 4811–4823, 2018.

[236] P. K. KAPUR and R. B. GARG, “Optimal release policies for software systems with testing effort,” *Int. J. Syst. Sci.*, vol. 22, no. 9, pp. 1563–1571, Sep. 1991.

[237] C.-Y. Huang and M. R. Lyu, “Optimal Release Time for Software Systems Considering Cost, Testing-Effort, and Test Efficiency,” *{IEEE} Trans. Reliab.*, vol. 54, no. 4, pp. 583–591, Dec. 2005.

[238] D. Greer and G. Ruhe, “Software release planning: an evolutionary and iterative approach,” *Inf. Softw. Technol.*, vol. 46, no. 4, pp. 243–253, Mar. 2004.

[239] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, “A Survey on Software Fault Localization,” *{IEEE} Trans. Softw. Eng.*, vol. 42, no. 8, pp. 707–740, Aug. 2016.

[240] V. R. Larman, Craig and Basili, “Iterative and incremental developments. a brief history,” *Computer (Long. Beach. Calif).*, vol. 36, no. 6, pp. 47–56, 2003.

[241] C. Gacek and B. Arief, “The many meanings of open source,” *{IEEE} Softw.*, vol. 21, no.

1, pp. 34–40, Jan. 2004.

[242] S. P. Craig, Rick David and Jaskiel, “Systematic software testing,” in *Artech house*, 2002.

[243] X.-S. Yang, “Flower pollination algorithm for global optimization,” in *International conference on unconventional computing and natural computation*, 2012, pp. 240–249.

[244] A. L. Goel and K. Okumoto, “Time-dependent error-detection rate model for software reliability and other performance measures,” *IEEE Trans. Reliab.*, vol. 28, no. 3, pp. 206–211, 1979.

[245] M. N. A. Wahab, S. Nefti-Meziani, and A. Atyabi, “A Comprehensive Review of Swarm Optimization Algorithms,” *{PLOS} {ONE}*, vol. 10, no. 5, p. e0122827, May 2015.

[246] J. H. Holland, “Genetic Algorithms,” *Sci. Am.*, vol. 267, no. 1, pp. 66–72, Jul. 1992.

[247] A. Chakri, R. Khelif, M. Benouaret, and X.-S. Yang, “New directional bat algorithm for continuous optimization problems,” *Expert Syst. Appl.*, vol. 69, pp. 159–175, 2017.

[248] F. S. Abu-Mouti and M. E. El-Hawary, “Overview of Artificial Bee Colony (ABC) algorithm and its applications,” in *Systems Conference (SysCon), 2012 IEEE International*, 2012, pp. 1–6.

[249] R. C. Eberhart and J. Kennedy, “Particle swarm optimization, proceeding of IEEE International Conference on Neural Network,” *Perth, Aust.*, pp. 1942–1948, 1995.

[250] X.-S. Yang, *Nature-Inspired Metaheuristic*. Beckington, UK: Luniver press, 2008.

[251] J. D. Musa, “A theory of software reliability and its application,” *IEEE Trans. Softw. Eng.*, vol. 3, pp. 312–327, 1975.

[252] K. Musa, John D and Okumoto, “A logarithmic Poisson execution time model for

software reliability measurement," in *Proceedings of the 7th international conference on Software engineering*, 1984, pp. 230–238.

[253] K. Musa, John D and Okumoto, "Validity of execution-time theory of software reliability," *IEEE Trans. Reliab.*, vol. 28, no. 3, p. 1979, 1979.