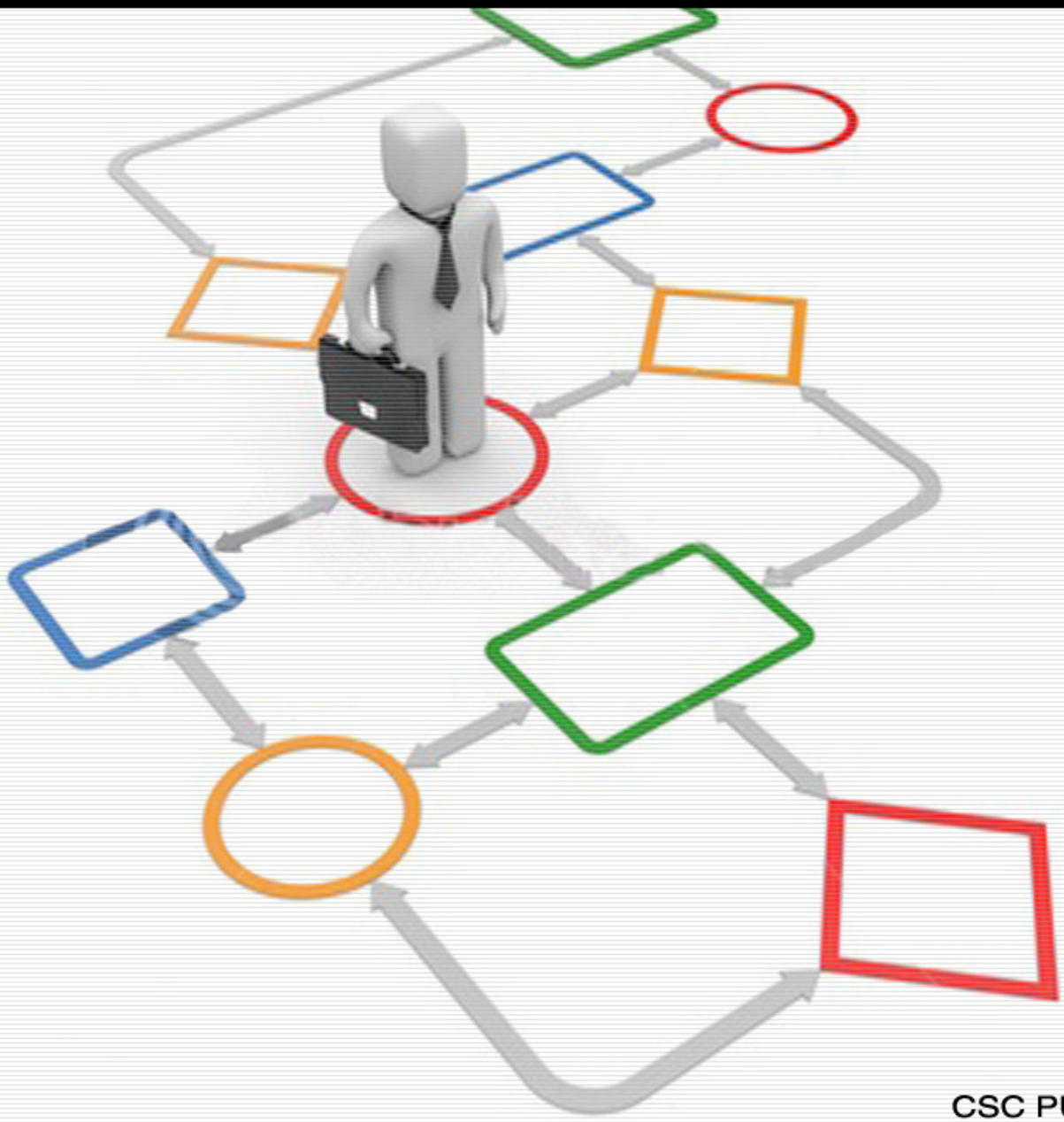
Volume 2 ▪ Issue 4 ▪ October 2011

INTERNATIONAL JOURNAL OF

SOFTWARE ENGINEERING (IJSE)

ISSN : 2180-1320

Publication Frequency: 6 Issues / Year



CSC PUBLISHERS
http://www.cscjournals.org

INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE)

VOLUME 2, ISSUE 4, 2011

EDITED BY
DR. NABEEL TAHIR

ISSN (Online): 2180-1320

I International Journal of Software Engineering (IJSE) is published both in traditional paper form and in Internet. This journal is published at the website http://www.cscjournals.org, maintained by Computer Science Journals (CSC Journals), Malaysia.

IJSE Journal is a part of CSC Publishers
Computer Science Journals
http://www.cscjournals.org

INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING (IJSE)

Book: Volume 2, Issue 4, October 2011
Publishing Date: 05 – 10 - 2011
ISSN (Online): 2180-1320

IJSE Journal is a part of CSC Publishers
http://www.cscjournals.org

Published in Malaysia

Typesetting: Camera-ready by author, data conversation by CSC Publishing Services – CSC Journals, Malaysia

CSC Publishers, 2011

EDITORIAL PREFACE

The International Journal of Software Engineering (IJSE) provides a forum for software engineering research that publishes empirical results relevant to both researchers and practitioners. It is the fourth issue of First volume of IJSE and it is published bi-monthly, with papers being peer reviewed to high international standards.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Starting with volume 2, 2011, IJSE appears in more focused issues. Besides normal publications, IJSE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

IJSE encourage researchers, practitioners, and developers to submit research papers reporting original research results, technology trend surveys reviewing an area of research in software engineering, software science, theoretical software engineering, computational intelligence, and knowledge engineering, survey articles surveying a broad area in software engineering and knowledge engineering, tool reviews and book reviews. Some important topics covered by IJSE usually involve the study on collection and analysis of data and experience that can be used to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies. IJSE is a refereed journal that promotes the publication of industry-relevant research, to address the significant gap between research and practice.

IJSE give the opportunity to researchers and practitioners for presenting their research, technological advances, practical problems and concerns to the software engineering.. IJSE is not limited to a specific aspect of software engineering it cover all Software engineering topics. In order to position IJSE amongst the most high quality journal on computer engineering sciences, a group of highly professional scholars are serving on the editorial board. IJSE include empirical studies, requirement engineering, software architecture, software testing, formal methods, and verification.

International Editorial Board ensures that significant developments in software engineering from around the world are reflected in IJSE. The submission and publication process of manuscript done by efficient way. Readers of the IJSE will benefit from the papers presented in this issue in order to aware the recent advances in the Software engineering. International Electronic editorial and reviewer system allows for the fast publication of accepted manuscripts into issue publication of IJSE. Because we know how important it is for authors to have their work published with a minimum delay after submission of their manuscript. For that reason we continue to strive for fast decision times and minimum delays in the publication processes. Papers are indexed & abstracted with International indexers & abstractors.

EDITORIAL BOARD

TABLE OF CONTENTS

Volume 2, Issue 4, October 2011

Pages

Different Software Testing Levels for Detecting Errors

Mohd. Ehmer Khan *ehmerkhan@gmail.com*
Lecturer
Department of Information Technology
Al Musanna College of Technology
P.O. Box-191, PC-314, Sultanate of Oman

Abstract

Software testing is the process to uncover requirement, design and coding errors in the program. But software testing is not a “miracle” that can guarantee the production of high quality software system, so to enhance the quality of a software and to do testing in a more unified way, the testing process could be abstracted to different levels and each level of testing aims to test different aspects of the system. In my paper, I have described different level of testing and these different levels attempt to detect different types of defects. The goal here is to test the system against requirement, and to test requirement themselves.

Keywords: Acceptance Testing, Integration Testing, Regression Testing, System Testing, Unit Testing

1. INTRODUCTION

Software testing is the process of accessing the functionality and correctness of a software through analysis. It also identifies most important defects, flaws, or errors in the application code that must be fixed. The system must be tested in steps with the planned build and release strategies. The key to successful testing strategies is selecting the right level of test at each stage in a project.

The level of testing have a hierarchical structure which build up from the bottom-up where higher level assume successful and satisfactory completion of lower level test. Each level of test is characterized by an environment i.e. type of people, hardware, data etc. and these environmental variables vary from project to project. [1] Each completed level represent a milestone on the project plan and each stage represents a known level of physical integration and quality. These integrated stages are known as level of testing.

The various levels of testing are:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing
5. Regression Testing

2. LEVELS OF TESTING

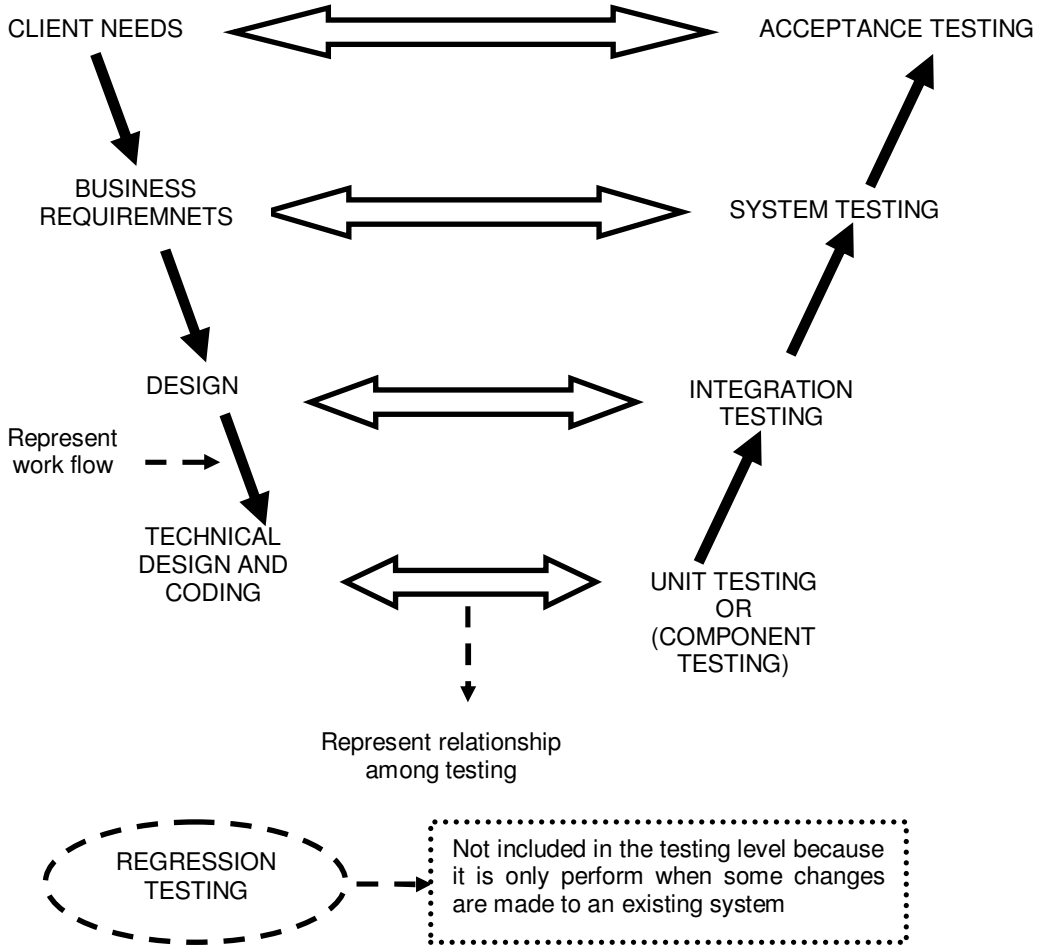


FIGURE 1: Represent various levels of testing

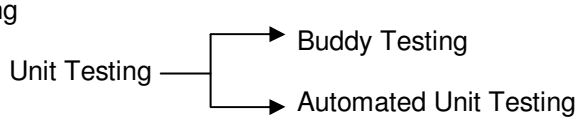
2.1 Unit Testing

Unit testing is also known as component testing, is the first and the lowest level of testing. In this level, individual units/components of software are tested and it is typically done by a programmer of the unit or module (Unit is the smallest piece of software that can be tested). Unit testing help to expose bugs that might appear to be hidden. Unit testing focuses on implementation and also require thorough understanding of the systems functional specification. [1]

Approximate level of documentation is needed for unit testing and there are certain minimum requirements for that documentation. They are as follows:

1. It must be reviewable
2. All the record must be archivable
3. Test can be repeatable

Two Types of unit testing



2.1.1 Buddy Testing

A better approach that has been encouraged is the use of a process called "Buddy Testing". It takes two-person team approach for code implementation and unit testing. Developer A writes the test cases for developer B, while B performs the unit test and vice versa. There are certain advantages of this team approach (buddy testing) – [1]

1. Models of the program specification requirements are served properly as the test cases are created prior to coding
2. Buddy testing provide cross-training on application.
3. The testing process is more objective and productive.

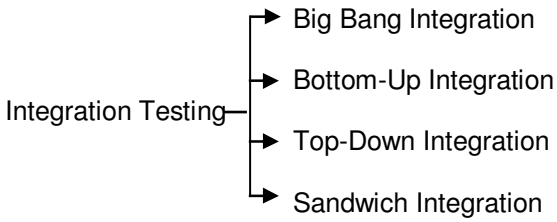
The one disadvantage with buddy testing is the extra time required to write the test cases up front and for the developer to familiarize themselves with each other specification and code.

2.1.2 Automating Unit Testing

Unit testing is usually automated and it is performed within the IDE of programmers. JUNIT (for JAVA) is an example of automated unit testing. RUTE-J a randomized unit testing example of JAVA is an effective and efficient method of testing.

2.2 Integration Testing

The level after unit testing is integration testing either the developer or an independent tester performs integration testing. It involves combining and testing different units of the program. The purpose of integration testing is to verify functional, performance and reliability requirements placed on major design items. [2] Approximately 40% of software errors are revealed during integration testing, so the need of integration testing must not be overlooked. The key purpose of integration testing is to leverage the overall integration structure to allow rigorous testing at each phase while minimizing duplication of efforts. Some different types of integration testing are



2.2.1 Big Bang Integration Testing

Big bang is an approach to integration testing where almost all the units are combined together and tested at one go. It is very effective for saving time in the integration testing process. Usage model testing is a type of big-bang testing and can be used in both software and hardware integration testing. [2]

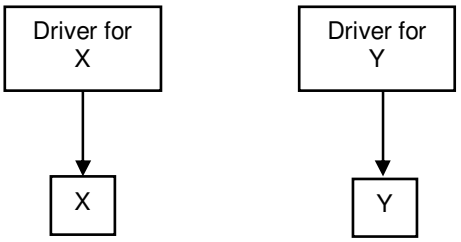
There are certain disadvantages with big bang [3]

1. High cost of repair
2. Minimum observability, diagnosabilty, efficacy and feedback
3. Defects are identified at a very late stage.
4. High probability of missing critical defects.
5. Difficult to isolate the defect found.

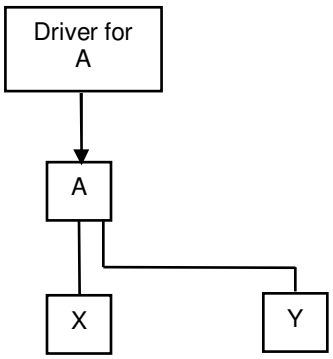
2.2.2 Bottom-Up Integration Testing

In this approach, testing starts at the bottom of the tree. Bottom-Up integration uses test drivers to drive and pass appropriate data to the lower level module. At each stage of bottom-up integration, the units at the higher levels are replaced by drivers (drivers are throw away pieces of code that are used to simulate procedure calls to child). [1]

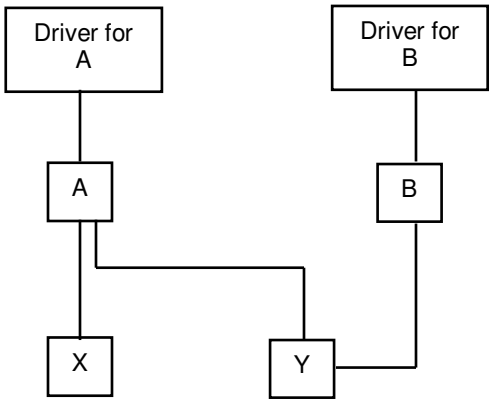
In this approach behaviour of interaction point are crystal clear. On the other hand, writing and maintaining test drivers is more difficult than entering stub.

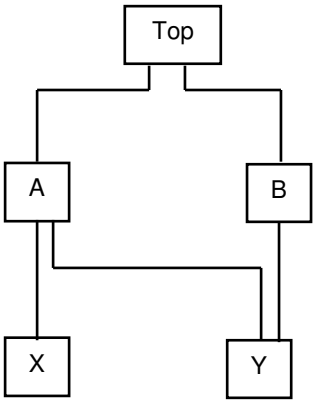


Separate drivers for each module, No. of tests are V(X) + V(Y)



Test a basis set of paths A in integration units X and Y using a driver, No. of tests V(A)



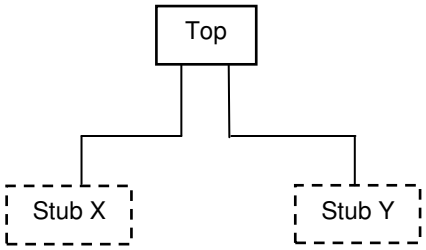


Integrated into single subsystem (Complete Bottom UP)

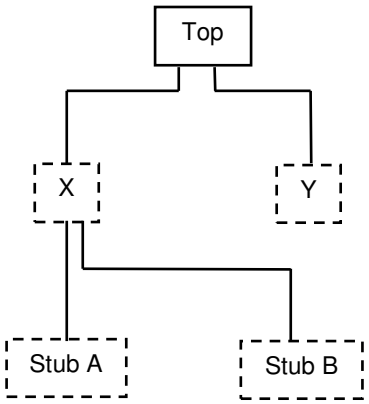
FIGURE 2: Represent complete bottom-up integration testing

2.2.3 Top-Down Integration Testing

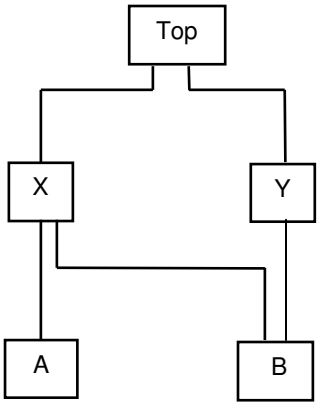
Top-down integration testing starts from the top and then proceeds to its child units. Any other lower level nodes that may be connected should be create as a stub. [4] As we add lower level code, we will replace stubs with actual components. This testing can be performed either breadth first or depth first. It is up to the tester to decide how many stubs should be replaced before the next test is performed. As the system prototype can be developed early on in the project process, this will make work easier and design defect can be found and corrected early. But one disadvantage with top-down approach is that extra work is needed to be done to produce large number of stubs.



Working from top level towards bottom



Write stubs of used module at each step in construction

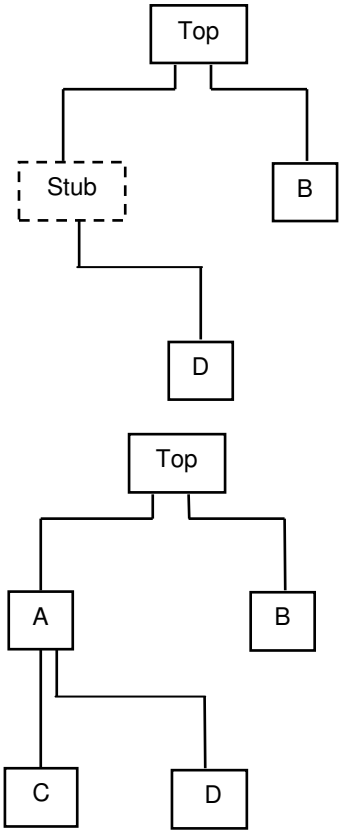


Top down is complete, and all functionality can be tested

FIGURE 3: Represent complete top-down integration testing

2.2.4 Sandwich Integration Testing

This approach combines the functionality of both bottom-up and top-down approach. The lower section unit are tested using bottom-up integration and higher section unit are tested by using top-up integration.[1] Less throw away codes are used by sandwich testing as compare to top down approach. [5]



Working from extremes toward center

It is flexible but complex to plan

FIGURE 4: Represent complete sandwich integration testing

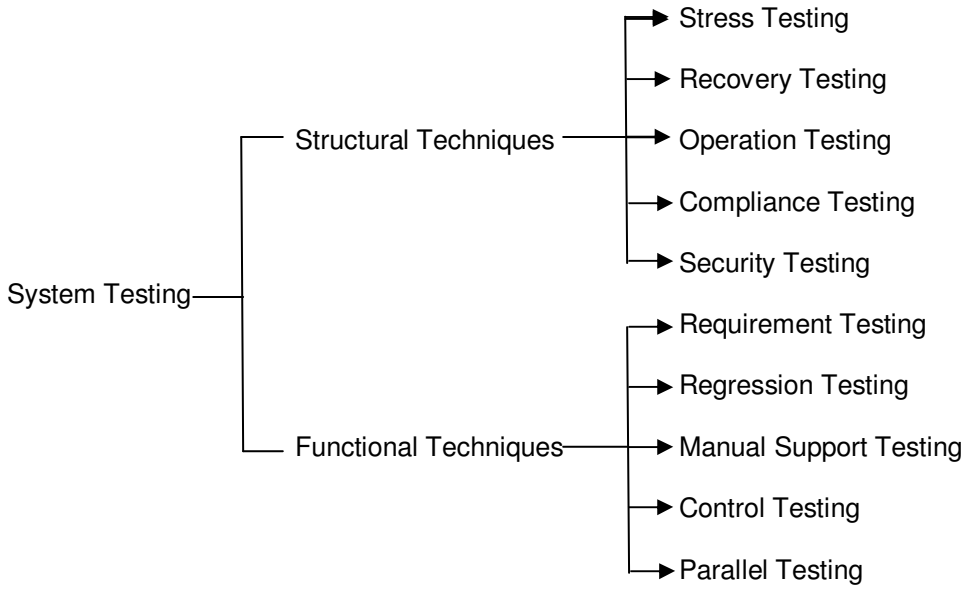
2.3 System Testing

Major level of testing or we can say the core of testing comes in this level, it is called system testing. This phase demands additional skills from a tester because various structural and functional techniques are carried out in this phase. System testing occurs when the system has been deployed onto a standard environment and all necessary components has been released internally.

Besides functional testing, system testing may include testing configuration, security, optimal utilization of resources and performance of the system. System testing is needed as:

1. It reduces cost
2. It increase productivity
3. It reduces commercial risk

The main goal of system testing is to evaluate the system as a whole and not its part. Various forms of testing under system testing are [6]



2.3.1 Structural Techniques

2.3.1.1 Stress Testing

It puts the program under heavy load or stress or we can also define stress testing as type of performance testing conducted to evaluate a system or components at all beyond limits of its specified work load. Stress testing may have a more specified meaning in certain industries, such as fatigue testing for material. [7]

2.3.1.2 Recovery Testing

It is a process of testing to determine the recoverability of the software. Recovery testing is executed to show that whether the recovery function of a system work in a correct manner or not. It also handles how the system recovers from the failure and it handles corrupted data such as data in DBMS and operating system.

2.3.1.3 Operation Testing

Testing conducted to evaluate a component or system in its operational environment. [8] Operation testing also test how the system is fits in with existing operations and procedure in the user organization.

2.3.1.4 Compliance Testing

It is usually done to determine the compliance of a system. It tests adherence to standards.

2.3.1.5 Security Testing

It helps to protect data and maintains the functionality of the system. The main concepts covered by security testing are [7]

1. Confidentiality
2. Integrity
3. Authentication
4. Authorization
5. Availability
6. Non Duplication

Internet based application are good candidate for security testing due to the continuous growth in the number of e-commerce applications.

2.3.2 Functional Techniques

2.3.2.1 Requirement Testing

It is the most fundamental form of testing and it checks and make sure that the system does what it is required to do.

2.3.2.2 Regression Testing

It is performed to uncover new errors, in existing functionality after changes have been made to the software. It assures that a change, such as a bugfix, did not introduce new bugs. [7] Regression testing ensures that the unchanged functionality remains unchanged.

2.3.2.3 Manual Support Testing

It includes user documentation and tests whether the system can be used properly or not. [6]

2.3.2.4 Control Testing

It is the process of testing various required control mechanism for system.

2.3.2.5 Parallel Testing

In parallel testing the same input is feed into two different versions of the system to make sure that both the versions of the system produces the same result. [6]

2.4 Acceptance Testing

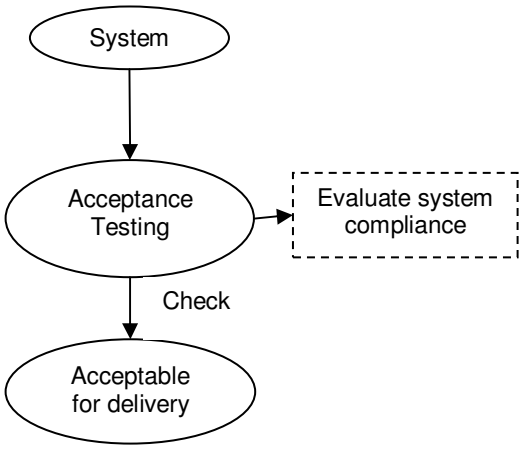
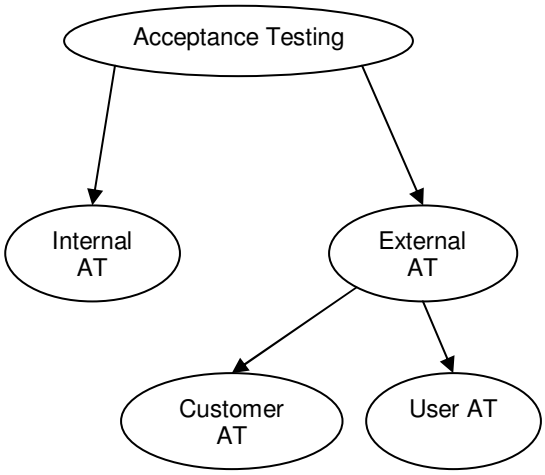


FIGURE 5: Represent acceptance testing

In software engineering acceptance testing is a level of software testing where the system is tested for user acceptability. Acceptance testing checks the system against requirement.

Acceptance testing is performed after system testing and before making the system available for actual use. [9] Sometimes acceptance testing also involves compatibility testing, it happens when a new system is developed to replace the old one.

Types of system testing



2.4.1 Internal Acceptance Testing
It is also known as alpha testing, which is simulated or actual operational testing and it is carried out by the test team who are not directly involved in the project.

2.4.2 External Acceptance Testing
It is performed by the external (not employed in an organization that developed the system)

2.4.2.1 Customer Acceptance Testing
Testing is performed by the customers who asked the organization to develop the software for them (software not being owned by the organization that developed it) [9]

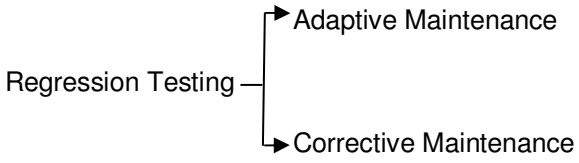
2.4.2.2 User Acceptance Testing
It is also known as beta testing which is operational testing by potential and existing customer at an external site not otherwise involved with the developer, to determine whether or not system satisfies customer needs.

Hence the goal of acceptance testing should verify the overall quality, correct operation, scalability, completeness, usability, portability and robustness of the functional component which is supplied by software system [6]

2.5 Regression Testing
Regression testing is performed when the software or its environment is changed. It is testing of a previously tested program following modification to ensure that defects have not been introduce or uncovered in unchanged areas of software, as a result of the changes made. [8]

Another important reason for regression testing is that it is often extremely difficult for a programmer to find out how the changes in one part of the software effects the other part. [10] Regression testing is a very important aspect of the system maintenance.

Two strategies of regression testing are [1]



2.5.1 Adaptive Maintenance
In adaptive maintenance the system specification are modified. Adaptive regression requires the generation of new test cases to suit the new specification.

2.5.2 Corrective Maintenance
In corrective maintenance the system specification are not modified and its support the reuses of test cases.

3. COMPARING DIFFERENT SOFTWARE TESTING LEVELS

Comparing different levels of testing that are done throughout the software development process are outlined in the table below

Level	Description	Importance
Unit Testing	Verify the functionality of a specific section of code at functional level.	White Box
Integration Testing	Tests the interfaces between component against a software design.	White Box / Black Box
System testing	Test a completely integrated software system and verifies that it satisfies the requirement.	Black Box
Acceptance Testing	It is performed as a part of hand-off process between any two phases of software development process.	Black Box
Regression Testing	Tests the defects that are occurred after a major code change i.e. tests new functionality in a program.	White Box

TABLE 1: Comparison between different software testing levels.

4. CONCLUSION

Software testing has the potential to save time and money by identifying errors early, and to improve customer satisfaction by delivering a more error free product. Software testing normally involves different levels of test case specification, test case generation, test execution, test evaluation and regression testing. Each of these levels plays an important role in the production of the program and meets their desired specification.

We have seen different level of testing so far. Starting from unit testing which is at the lowest level and ensures that the implementation fits the functional specification. Integration testing is next to unit testing and it tests the communication between different components of the system. After integration testing, system testing comes which tests the functionality of software as a complete system. The last level is acceptance testing and it verifies whether the end user is satisfy with the

system or not. Lastly, the regression testing which ensures that the modification applied to a system has not adversely change system behavior.

Irrespective of different levels of testing the testing should encompass the following

1. Cost of failure
2. Identify defect before customer finds them
3. Reduce the risk of releasing
4. Evaluation of product with an independent perspective

5. REFRENCES

[1] Levels of Testing written by Patrick Oladimeji supervised by Prof. Dr. Holger Schlingloff & Dr. Markus Roggenbach published on 1/12/2007 available at http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/OladimejiP.pdf

[2] Integration testing available at http://en.wikipedia.org/wiki/Integration_testing

[3] Big bang integration available at http://www.testinggeek.com/big-bang-integration-testing

[4] Integration testing by Thomas Bradley (368100) published on February 8, 2008 available at http://sucs.org/~tobeon/testing.pdf

[5] Integration testing & Component based software testing chapter # 21 by Mauro & Michal Young, 2007 (c) available at http://ix.cs.uoregon.edu/~michal/book/slides/pdf/PezzeYoung-Ch21-integration.pdf

[6] Cognizant Technology Solution available at pp 44, 50, 52 & 61 available at http://www.scribd.com/doc/6749799/Software-Testing-COGNIZANT-Notes

[7] System testing available at http://en.wikipedia.org/wiki/System_testing

[8] Standard glossary of terms used in Software Testing (ISTQB) version 2.1 (dd. April 1st, 2010) by Erik van Veenendaal (The Netherlands) available at http://istqb.org/download/attachments/2326555/ISTQB+Glossary+of+Testing+Terms+2+1.pdf

[9] Levels of software testing available at http://softwaretestingfundamentals.com/software-testing-levels/

[10] Regression testing available at http://en.wikipedia.org/wiki/Regression_testing

Pareto Type II Based Software Reliability Growth Model

Dr.R.Satya Prasad *profrsp@gmail.com*
Associate Professor, Dept.of Computer Science & Engg.
Acharya Nagarjuna University,
Nagarjuna Nagar- 520510.
INDIA.

N.Geetha Rani *geetha.neppala@gmail.com*
Associate Professor, Department of Computer Science,
Abhinav Institute of Management & Tech.
Singaryakonda – INDIA

Prof.R.R.L.Kantam *kantam_rrl@rediffmail.com*
Professor, Department of Statistics
Acharya Nagarjuna University,
Nagarjuna Nagar- 520510.
INDIA.

Abstract

The past 4 decades have seen the formulation of several software reliability growth models to predict the reliability and error content of software systems. This paper presents Pareto type II model as a software reliability growth model, together with expressions for various reliability performance measures. Theory of probability, distribution function, probability distributions plays major role in software reliability model building. This paper presents estimation procedures to access reliability of a software system using Pareto distribution, which is based on Non Homogenous Poisson Process (NHPP).

Keywords: Software Reliability, NHPP, Pareto Type II Distribution, Parameter Estimation.

1. INTRODUCTION

Software reliability is the probability of failure free operation of software in a specified environment during specified duration [Musa 1998]. Several models have been proposed during the past 4 decades for accessing reliability of a software system for example Crow and Basu(1988), Goel and Okumoto (1979,1984), Musa(1980), Pham(2005), Ramamurthy and Bastani(1982), Zhang,Teng and Pham(2003), Malaiya, Karunanithi and Verma(1992) and Wood(1996). The objective of such models is to improve software performance. These models are concerned with forecasting future system operability from the failure data collected during the testing phase of a software product. Most of the models assume that the time between failure follows an exponential distribution with parameter that varies with the number of errors remaining in the software system. A software system is a product of human work and is very likely to contain faults. The accuracy of software reliability growth models when validated using the very few available data sets varies significantly and thus despite the existence of numerous models, none of them can be recommended unreservedly to potential users.

This paper presents a Pareto type II model to analyze the reliability of a software system. Our objective is to develop a parsimonious model whose parameters have a physical interpretation and which can yield quantitative measure for software performance assessment. The layout of the paper is as follows: Section 2 describes the development and interpretation of the mean value function for the underlying NHPP. Section 3 discusses parameter estimation of Pareto type II model based on time between failure data. Section 4 describes the techniques used for software failure data analysis for a live data and Section 5 contains conclusions.

2. PARETO MODEL DEVELOPMENT

Software reliability models can be classified according to probabilistic assumptions. When a Markov process represents the failure process, the resultant model is called Markovian Model. Second one is fault counting model which describes the failure phenomenon by stochastic process like Homogeneous Poisson Process (HPP), Non Homogeneous Poisson Process (NHPP) and Compound Poisson Process etc. A majority of failure count models are based upon NHPP described in the following lines.

A software system is subject to failures at random times caused by errors present in the system. Let {N(t), t >0} be a counting process representing the cumulative number of failures by time t. Since there are no failures at t=0 we have

$$N(0) = 0$$

It is to assume that the number of software failures during non overlapping time intervals do not affect each other. In other words, for any finite collection of times t1<t2<….<tn the 'n' random variables N(t1), {N(t2)-N(t1)}, ….. {N(tn) - N(tn-1)} are independent. This implies that the counting process {N(t), t>0} has independent increments.

Let m(t) represent the expected number of software failures by time 't'. Since the expected number of errors remaining in the system at any time is finite, m(t) is bounded, non decreasing function of 't' with the following boundary conditions.

$$m(t) = 0, \quad t = 0$$
$$= a, \quad t \to \infty$$

where a is the expected number of software errors to be eventually detected.

Suppose N(t) is known to have a Poisson probability mass function with parameters m(t) i.e.

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

then N(t) is called an NHPP. Thus the stochastic behavior of software failure phenomena can be described through the N(t) process. Various time domain models have appeared in the literature (Kantam and Subbarao, 2009) which describe the stochastic failure process by an NHPP which differ in the mean value functions m(t).

In this paper we consider m(t) as given by

$$m(t) = \alpha\left[1 - \frac{c^b}{(t+c)^b}\right] \qquad (2.1)$$

where [m(t)/a] is the cumulative distribution function of Pareto type II distribution (Johnson et al, 2004) for the present choice.

$$P\{N(t) = n\} = \frac{[m(t)]^n . e^{-m(t)}}{n!}$$

$$\lim_{n\to\infty} P\{N(t) = n\} = \frac{e^{-a} . a^n}{n!}$$

which is also a Poisson model with mean 'a'.

Let N(t) be the number of errors remaining in the system at time 't'

$$
\begin{aligned}
N(t) &= N(\infty) - N(t) \\
E[N(t)] &= E[N(\infty)] - E[N(t)] \\
&= a - m(t) \\
&= a - a\left[1 - \frac{c^b}{(t+c)^b}\right] \\
&= \frac{ac^b}{(t+c)^b}
\end{aligned}
$$

Let be the time between (k-1)th and kth failure of the software product. Let be the time up to the kth failure. Let us find out the probability that time between (k-1)th and kth failures, i.e. exceeds a real number 's' given that the total time up to the (k-1)th failure is equal to x, i.e. P[> s / = x]

$$R_{S_k/X_{k-1}}(s / x) = e^{-[m(x+s)-m(s)]} \qquad (2.2)$$

This Expression is called Software Reliability.

3. PARAMETER ESTIMATION OF PARETO TYPE II MODEL

In this section we develop expressions to estimate the parameters of the Pareto type II model based on time between failure data. Expressions are now derived for estimating 'a', 'b' and 'c' for the model.

Let S_1, S_2 …. be a sequence of times between successive software failures associated with an NHPP N(t). Let X_k be equal to

$\sum_{i=1}^{k} S_i$, k = 1, 2, 3 ….

which represents the time to failure k. Suppose we are given 'n' software failure times say $x_1, x_2, \ldots x_n$, there are 'n' time instants at which the first, second, third … nth failures of a software are observed. This is a special case of a life testing experiment in which only one product is put to test and its successive failures are recorded alternatively separated by error detections and debugging.

The mean value function of Pareto type II model is given by

$$m(t) = a\left[1 - \frac{c^b}{(t+c)^b}\right], \quad t \geq 0 \qquad (3.1)$$

The constants 'a' , 'b' and 'c' which appear in the mean value function and various other expressions are called parameters of the model. In order to have an assessment of the software reliability a, b and c are to be known or they are to be estimated from software failure data. Expressions are now derived for estimating 'a', 'b' and 'c' for the model.

The required likelihood function is given by

$$L = e^{-m(x_n)} \cdot \prod_{i=1}^{n} m'(x_i) \qquad (3.2)$$

values of a, b and c that would maximize L are called maximum likelihood estimators (MLEs) and the method is called maximum likelihood (ML) method of estimation.

$$L = e^{-a\left[1 - \frac{c^b}{(x_n+c)^b}\right]} \cdot \prod_{i=1}^{n} \frac{ab c^b}{(x_i+c)^{b+1}} \qquad (3.3)$$

Then the log likelihood equation to estimate the unknown parameters a, b and c are given by

$$\text{LogL} = -a\left[1-\frac{c^b}{(x_n+c)^b}\right]+\sum_{i=1}^{n}\left[\log a+\log b+b\log c-(b+1)\log(x_i+c)\right] \qquad (3.4)$$

Accordingly parameters 'a', 'b' and 'c' would be solutions of the equations

$$\frac{\partial LogL}{\partial a}=0, \quad \frac{\partial LogL}{\partial b}=0, \quad \frac{\partial^2 LogL}{\partial b^2}=0,$$

$$\frac{\partial LogL}{\partial c}=0, \quad \frac{\partial^2 LogL}{\partial c^2}=0$$

Substituting the expressions for m(t) (3.1) in the above equations, taking logarithms, differentiating with respect to 'a', 'b', 'c' and equating to zero, after some joint simplications we get

$$a = \frac{n\,(x_n+c)^b}{(x_n+c)^b - c^b} \qquad (3.5)$$

$$g(b)= \frac{n\log\frac{c}{x_n+c}}{(x_n+1)^b-1} + \frac{n}{b} - \sum_{i=1}^{n}\log(x_i+1) \qquad (3.6)$$

Second order partial derivative of L with respect to the parameter 'b'

$$g'(b) = -n\log\frac{1}{x_n+1}\left[\frac{(x_n+1)^b\log(x_n+1)}{[(x_n+1)^b-1]^2}\right] - \frac{n}{b^2} \qquad (3.7)$$

$$g(c) = \frac{n}{x_n+c} + \frac{n}{c} - \sum_{i=1}^{n}\frac{2}{x_i+c} \qquad (3.8)$$

Second order partial derivative of L with respect to the parameter 'c'

$$g'(c) = -\frac{n}{(x_n+c)^2} - \frac{n}{c^2} + \sum_{i=1}^{n}\frac{2}{(x_i+c)^2} \qquad (3.9)$$

The values of 'b' and 'c' in the above equations can be obtained using Newton Raphson Method. Solving the above equations simultaneously, yields the point estimates of the parameters a, b and c. These equations are to be solved iteratively and their solutions in turn when substituted in the log likelihood equation of 'a' would give analytical solution for the MLE of 'a'. However when 'b' is assumed to be known only one equation that of 'c' has to be solved by numerical methods to proceed for further evaluation of reliability measures.

4. NTDS SOFTWARE FAILURE DATA ANALYSIS

In this Section, we present the analysis of NTDS software failure data, taken from Jelinski and Mornda(1972). The data are originally from the U.S. Navy Fleet Computer Programming Centre, and consists of the errors in the development of software for the real time, multi computer complex which forms the core of the Naval Tactical Data Systems (NTDS). The NTDS software consisted of some 38 different modules. Each module was supposed to follow three stages; the production (development) phase, the test phase and the user phase. The data are based on the trouble reports or 'software anomaly reports' for one of the larger modules denoted as A-module. The times (days) between software failures and additional information for this module are summarized in the below table.

Error Number n	Time between Errors Sk days	Cumulative Time xn = $\sum S_k$ days
Production (Checkout) Phase		
1	9	9
2	12	21
3	11	32
4	4	36
5	7	43
6	2	45
7	5	50
8	8	58
9	5	63
10	7	70
11	1	71
12	6	77
13	1	78
14	9	87
15	4	91
16	1	92
17	3	95
18	3	98
19	6	104
20	1	105
21	11	116
22	33	149
23	7	156
24	91	247
25	2	249
26	1	250
Test Phase		
27	87	337
28	47	384
29	12	396
30	9	405
31	135	540
User Phase		
32	258	798
Test Phase		
33	16	814
34	35	849

TABLE 4.1 NTDS Data

The data set consists of 26 failures in 250 days. 26 software errors were found during production phase and five additional errors during test phase. One error was observed during the user phase and two more errors are noticed in a subsequent test phase indicating that a network of the module had taken place after the user error was found.

Solving equations in section 3 by Newton Raphson Method (N-R) method for the NTDS software failure data, the iterative solutions for MLEs of a, b and c are

a^ = 55.018710
b^ = 0.998899
c^ = 278.610091

Hence, we may accept these three values as MLEs of a, b, c. The estimator of the reliability function from the equation (2.2) at any time x beyond 250 days is given by

$$R_{S_k/X_{k-1}}(s/x) = e^{-[m(x+s)-m(s)]}$$

$$R_{S_{27}/X_{26}}(250/50) = e^{-[m(50+250)-m(250)]}$$

$$= 0.081677$$

5. CONCLUSION

In this paper we have presented Pareto software reliability growth model with a mean value function. It provides a plausible description of the software failure phenomenon. This is called Pareto Type II Model. This is a simple method for model validation and is very convenient for practitioners of software reliability.

6. REFERENCES

[1] CROW, .H, and BASU, A.P. (1988). "Reliability growth estimation with missing data-II", Proceeding annual Reliability and Maintainability Symposium, 26-28.

[2] Goel, A.L., Okumoto, K., 1979. Time- dependent error-detection rate model for software reliability and other performance measures. IEEE Trans. Reliab. R-28, 206-211.

[3] Jelinski, Z and Moranda, P.B (1972) "Software reliability research", In:W.Freiberger,(Ed) Statistical Computer Performance Evaluation, New York:Academic Press 465-497.

[4] Musa J.D, Software Reliability Engineering MCGraw-Hill, 1998.

[5] Musa,J.D. (1980) "The Measurement and Management of Software Reliability", Proceeding of the IEEE vol.68, No.9, 1131-1142.

[6] Pham. H (2005) "A Generalized Logistic Software Reliability Growth Model", Opsearch, Vol.42, No.4, 332-331.

[7] Ramamurthy, C.V., and Bastani, F.B.(1982). "Software Reliability Status and Perspectives", IEEE Transactions on Software Engineering, Vol.SE-8, 359-371.

[8] R.R.L.Kantam and R.Subbarao, 2009. "Pareto Distribution: A Software Reliability Growth Model". International Journal of Performability Engineering, Volume 5, Number 3, April 2009, Paper 9, PP: 275- 281.

[9] J.D.Musa and K.Okumoto,"A Logorithmic Poisson Execution time model for software reliability measure-ment", proceeding seventh international conference on software engineering, orlando, pp.230-238,1984.

[10] ZHANG,X., TENG,X. and PHAM,H. CONSIDERING FAULT REMOVAL EFFICIENCY IN SOFTWARE RELIABILITY ASSESSMENT, IEEE Transactions on Systems, Man and Cybernetics-part A, Vol.33, No.1, 2003; 114-120.

[11] MALAIYA, Y.K., KARUNANITHI, N., and VERMA, P. PREDICTABILITY OF SOFTWARE RELIABILITY MODELS, IEEE Transactions on Reliability, Vol, No.4. 1992; 539-546.

[12] WOOD, A. predicting software Reliability, IEEE Computer, 1996; 2253-2264.

Software Effort Estimation Using Particle Swarm Optimization With Inertia Weight

Prasad Reddy.P.V.G.D *prasadreddy.vizag@gmail.com*
Department of Computer Science & Systems Engineering
Andhra University
Visakhapatnam, 530003, India

CH.V.M.K.Hari *kurmahari@gmail.com*
Department of IT
GITAM University
Visakhapatnam, 530045, India

Abstract

Software is the most expensive element of virtually all computer based systems. For complex custom systems, a large effort estimation error can make the difference between profit and loss. Cost (Effort) Overruns can be disastrous for the developer. The basic input for the effort estimation is size of project. A number of models have been proposed to construct a relation between software size and Effort; however we still have problems for effort estimation because of uncertainty existing in the input information. Accurate software effort estimation is a challenge in Industry. In this paper we are proposing three software effort estimation models by using soft computing techniques: Particle Swarm Optimization with inertia weight for tuning effort parameters. The performance of the developed models was tested by NASA software project dataset. The developed models were able to provide good estimation capabilities.

Keywords--PM- Person Months, KDLOC-Thousands of Delivered Lines of Code, PSO - Particle Swarm Optimization, Software Cost Estimation.

1. INTRODUCTION

The modern day software industry is all about efficiency. With the increase in the expanse and impact of modern day software projects, the need for accurate requirement analysis early in the software development phase has become pivotal. The provident allocation of the available resources and the judicious estimation of the essentials form the basis of any planning and scheduling activity. For a given set of requirements, it is desirable to cognize the amount of time and money required to deliver the project prolifically. The chief aim of software cost estimation is to enable the client and the developer to perform a cost – benefit analysis. The software, the hardware and the human resources involved add up to the cost of a project. The cost / effort estimates are determined in terms of person-months (pm) which can be easily interchanged to actual currency cost.

The basic input parameters for software cost estimation is size, measured in KDLOC (Kilo Delivered Lines Of Code). A number of models have been evolved to establish the relation between Size and Effort [13]. The parameters of the algorithms are tuned using Genetic Algorithms [5] ,Fuzzy models[6][14], Soft-Computing Techniques[7][9][10][15], Computational Intelligence Techniques[8],Heuristic Algorithms, Neural Networks, Radial Basis and Regression [11][12] .

1.1 Basic Effort Model

A common approach to the estimation of the software effort is by expressing it as a single variable function - project size. The equation of effort in terms of size is considered as follows:

$$\text{Effort} = a * (\text{Size})^{b} \quad (1)$$

Where a, b are constants. The constants are usually determined by regression analysis applied to historical data.

1.2 Standard PSO with Inertia Weights

In order to meet the needs of modern day problems, several optimization techniques have been introduced. When the search space is too large to search exhaustively, population based searches may be a good alternative, however, population based search techniques cannot guarantee you the optimal (best) solution. We will discuss a population based search technique, Particle Swarm Optimization (PSO) with Inertia Weights [Shi and Eberhart 1998]. Particle Swarm has two primary operators: Velocity update and Position update. During each generation each particle is accelerated toward the particles previous best position and the global best position. At each iteration a new velocity value for each particle is calculated based on its current velocity, the distance from its previous best position, and the distance from the global best position. The new velocity value is then used to calculate the next position of the particle in the search space. The inertia weight is multiplied by the previous velocity in the standard velocity equation and is linearly decreased throughout the run. This process is then iterated a set number of times or until a minimum error is achieved.

The basic concept of PSO lies in accelerating each particle towards its Pbest and Gbest locations with regard to a random weighted acceleration at each time. The modifications of the particle's positions can be mathematically modeled by making use of the following equations:

$$V_i^{k+1} = w * V_i^k + c_1 * rand()_1 * (Pbest - S_i^k) + c_2 * rand()_2 * (Gbest - S_i^k) \quad (2)$$

$$S_i^{k+1} = S_i^k + V_i^{k+1} \quad (3)$$

Where,

S_i^k is current search point,
S_i^{k+1} is modified search point,
V_j^k is the current velocity,
V^{k+1} is the modified velocity,
V_{pbest} is the velocity based on Pbest ,
V_{gbest} = velocity based on Gbest,
w is the weighting function,
c_j is the weighting factors,
Rand() are uniformly distributed random numbers between 0 and 1.

2. THE STANDARD PSO WITH INERTIA WEIGHT FOR SOFTWARE EFFORT ESTIMATION

The software effort is expressed as a function of a single variable as shown in equation-1. In this parameters a, b are measured by using regression analysis applied to historical data. Now in order to tune these parameters we use the standard PSO with inertia weights. A nonzero inertia weight introduces a preference for the particle to continue moving in the same direction it was going on the previous iteration. Decreasing the inertia over time introduces a shift from the exploratory (global search) to the exploitative (local search) mode. The updating of weighting function is done with the following formula.

$$W_{new} = [(T_{mi} - T_{ci}) * (W_{iv} - W_{fv})] / T_{mi} + W_{fv} \quad (4)$$

Where

W_{new} is new weight factor,
T_{mi} is the maxium numer of iteration specified,
T_{ci} is the current iteration number,
W_{iv} is the initial value of the weight,
W_{fv} is the final value of the weight.

Empirical experiments have been performed with an inertia weight set to decrease linearly from 0.9 to 0.4 during the course of simulation. In the first experiment we keep the parameters c1 and c2 (weighting factors) fixed, while for the following experiment we change c1 and c2 (weighting factors) during subsequent iterations by employing the following equations [Rotnaweera, A. Halgamog S.K. and Watson H.C, 2004].

$$C_1(t) = 2.5 - 2 * (t / max\_iter), \text{ which is the cognitive learning factor.} \quad (5)$$

C_2 (t) = 0.5 + 2* (t / max_iter), which is the social coefficient. (6)

The particles are initialized with random position and velocity vectors the fitness function is evaluated and the Pbest and Gbest of all particles is found out. The particles adjust their velocity according to their Pbest and Gbest values. This process is repeated until the particles exhaust or some specified number of iterations takes place. The Gbest particle parameters at the end of the process are the resultant parameters.

3. MODEL DESCRIPTION

In this model we have considered "The standard PSO with inertia weights" with /without changing the weighting factors (c1, c2). PSO is a robust stochastic optimization technique based on the movement of swarms. This swarm behavior is used for tuning the parameters of the Cost/Effort estimation. As the PSO is a random weighted probabilistic model the previous benchmark data is required to tune the parameters, based on that data, swarms develop their intelligence and empower themselves to move towards the solution. The following is the methodology employed to tune the parameters in each proposed models following it.

3.1 METHODOLOGY (ALOGORITHM)

Input: Size of Software Projects, Measured Efforts, Methodology (Effort Adjustment factor-EAF).
Output: Optimized Parameters for Estimating Effort.
The following is the methodology used to tune the parameters in the proposed models for Software Effort Estimation.
Step 1: Initialize "n" particles with random positions P_i and velocity vectors V_i of tuning parameters .We also need the range of velocity between [- V_{max},V_{max}]. The Initial positions of each particle are Personally Best for each Particle.
Step 2: Initialize the weight function value w with 0.5 and weightening parameters cognitive learning factor c1, social coefficient c2 with 2.0.
Step 3: Repeat the following steps 4 to 9 until number of iterations specified by the user or Particles Exhaust.
Step 4: for i = 1,2, ……, n do // For all the Particles
For each particle position with values of tuning parameters, evaluate the fitness function. The fitness function here is Mean Absolute Relative Error (MARE). The objective in this method is to minimize the MARE by selecting appropriate values from the ranges specified in step 1.
Step 5: Here the Pbest is determined for each particle by evaluating and comparing measured effort and estimated effort values of the current and previous parameters values.
If fitness (p) better than fitness (Pbest) then: Pbest = p.
Step 6: Set the best of 'Pbests' as global best – Gbest. The particle value for which the variation between the estimated and measured effort is the least is chosen as the Gbest particle.
Step 7: Update the weightening function is done by the following formula

$$W_{new} = [(T_{mi} - T_{ci}) * (W_{iv} - W_{fv})] / T_{mi} + W_{fv} \quad (7)$$

Step 8: Update the weightening factors is done with the following equations for faster convergence.

$$C_1(t) = 2.5 - 2 * (T_{ci} / T_{mi}) \quad (8)$$

$$C_2 (t) = 0.5 + 2* (T_{ci} / T_{mi}), \quad (9)$$

Step 9: Update the velocity and positions of the tuning parameters with the following equations
for j = 1, 2, ……m do // For number of Parameters, our case m is 2or 3 or 4
begin

$$V_{ji}^{k+1} = w * V_{ji}^{k} + c_1 * rand()_1 * (Pbest - S_{ji}^{k}) + c_2 * rand()_2 * (Gbest - S_{ji}^{k}) \quad (10)$$

$$S_{ji}^{k+1} = S_{ji}^{k} + V_{ji}^{k+1} \quad (11)$$

end;
Step 10: Give the Gbest values as the optimal solution.
Step 11: Stop

3.2 PROPOSED MODELS

3.2.1 MODEL 1:

A prefatory approach to estimating effort is to make it a function of a single variable , often this variable is project size measure in KDLOC (kilo delivered lines of code) and the equation is given as ,

$$Effort = a\ (size)^b$$

Now in our model the parameters are tuned using above PSO methodology.

The Update of velocity and positions of Parameter "a" is

$$V_{ai}^{k+1} = w * V_{ai}^{k} + c_1 * rand()_1 * (Pbest - S_{ai}^{k}) + c_2 * rand()_2 * (Gbest - S_{ai}^{k}) \quad (12)$$

$$S_{ai}^{k+1} = S_{ai}^{k} + V_{ai}^{k+1}$$

The Update of velocity and positions of Parameter "b" is

$$V_{bi}^{k+1} = w * V_{bi}^{k} + c_1 * rand()_1 * (Pbest - S_{bi}^{k}) + c_2 * rand()_2 * (Gbest - S_{bi}^{k})$$

$$S_{bi}^{k+1} = S_{bi}^{k} + V_{bi}^{k+1}$$

TABLE 1: Effort Multipliers

COST FACTORS	DESCRIPTION	RATING				
		VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
	Product					
RELY	Required software reliability	0.75	0.88	1	1.15	1.4
DATA	Database size	-	0.94	1	1.08	1.16
CPLX	Product complexity	0.7	0.85	1	1.15	1.3
	Computer					
TIME	Execution time constraint	-	-	1	1.11	1.3
STOR	Main storage constraint	-	-	1	1.06	1.21
VIRT	Virtual machine volatility	-	0.87	1	1.15	1.3
TURN	Computer turnaround time	-	0.87	1	1.07	1.15
	Personnel					
ACAP	Analyst capability	1.46	1.19	1	0.86	0.71
AEXP	Application experience	1.29	1.13	1	0.91	0.82
PCAP	Programmer capability	1.42	1.17	1	0.86	0.7
VEXP	Virtual machine volatility	1.21	1.1	1	0.9	-
LEXP	Language experience	1.14	1.07	1	0.95	-
	Project					
MODP	Modern programming practice	1.24	1.1	1	0.91	0.82
TOOL	Software tools	1.24	1.1	1	0.91	0.83
SCED	Development schedule	1.23	1.08	1	1.04	1.1

3.2.2 MODEL 2

Instead of having resources estimates as a function of one variable, resources estimates can depend on many different factors, giving rise to multivariable models. Such models are useful as they take into account the subtle aspects of each project such as their complexity or other such factors which usually create a non linearity. The cost factors considered are shown below. The product of all the above cost factors is the Effort Adjustment Factor (EAF).A model of this category starts with an initial estimate determined by using the strategic single variable model equations and adjusting the estimates based on other variable which is methodology.

The equation is,

$$Effort = a * (size)^b + c * (ME).$$

Where ME is the methodology used in the project.

The parameters a, b, c are tuned by using above PSO methodology.

The Update of velocity and positions of Parameter "a", "b" are shown in Model 1 and Parameter "c" is

$V_{ci}^{k+1} = w * V_{ci}^{k} + c_1 * rand()_1 * (Pbest - S_{ci}^{k}) + c_2 * rand()_2 * (Gbest - S_{ci}^{k})$

$S_{ci}^{k+1} = S_{ci}^{k} + V_{ci}^{k+1}$

3.2.3 MODEL 3

There are a lot of factors causing uncertainty and non linearity in the input parameters. In some projects the size is low while the methodology is high and the complexity is high, for other projects size is huge but the complexity is low. As per the above two models size and effort are directly proportional. But such a condition is not always satisfied giving rise to eccentric inputs. This can be accounted for by introducing a biasing factor (d). So the effort estimation equation is:

$$Effort = a *(size)^{b} + c* (ME).+ d$$

a,b,c,d parameters are tuned by using above PSO methodology.

The Update of velocity and positions of Parameter "a", "b", "c" are shown in Model 1,2 and Parameter "d" is

$V_{di}^{k+1} = w * V_{di}^{k} + c_1 * rand()_1 * (Pbest - S_{di}^{k}) + c_2 * rand()_2 * (Gbest - S_{di}^{k})$

$S_{di}^{k+1} = S_{di}^{k} + V_{di}^{k+1}$

4. MODEL ANALYSIS

4.1 Implementation

We have implemented the above methodology for tuning parameters a,b,c and d in "C" language. For the parameter' a 'the velocities and positions of the particles are updated by applying the following equations:

$V_{ai}^{k+1} = w * V_{ai}^{k} + c_1 * rand_1 * (Pbesta - S_{ai}^{k}) + c_2 * rand_2 * (Gbest - S_{ai}^{k})$

$S_{ai}^{k+1} = S_{ai}^{k} + V_{ai}^{k+1}$, w=0.5 , c1=c2=2.0.

and similarly for the parameters b,c and d the values are obtained for the first experiment and weight factor w changed during the iteration and C1 and C2 are constant. For the second experiment we changed the C1, C2 weighting factors by using equations 4 and 5.

4.2 Performance Measures

We consider three performance criterions:

1) Variance accounted – For(VAF)

$$\%VAF = \left[1 - \frac{var(ME-EE)}{var(ME)}\right] * 100$$

2) Mean Absolute Relative Error

$$\%MARE = mean\left[\frac{abs(ME-EE)}{(ME)}\right] * 100$$

3) Variance Absolute Relative Error (VARE)

$$\%VARE = var\left[\frac{abs(ME-EE)}{(ME)}\right] * 100$$

Where ME represents Measured Effort, EE represents Estimated Effort.

5. MODEL EXPERIMENTATION

EXPERIMENT – 1

For the study of these models we have taken data of 10 NASA [13]

TABLE 2: NASA software projects data

Project No	Size In KDLOC	Methodology (ME)	Measured Effort
13	2.1	28	5
10	3.1	26	7
11	4.2	19	9
17	12.5	27	23.9
3	46.5	19	79
4	54.5	20	90.8
6	67.5	29	98.4
15	78.6	35	98.7
1	90.2	30	115.8
18	100.8	34	138.3

By running the "C" implementation of the above methodology we obtain the following parameters for the proposed models.

Model 1 : a=2.646251 and b=0.857612 .
The range of a is [1, 10] and b is [-5,5] .

Model 2: a=2.771722, b=0.847952 and c= -0.007171.
The range of a is [1, 10], b is [-5,5] and c is [-1,1].

Model 3: a =3.131606 , b=0.820175 , c=0.045208 and d= -2.020790.
The ranges are a[1,10],b[-5,5], c[-1,1] and d[1,20].

EXPERIMENT -2:

The following are the results obtained by running the above PSO algorithm implemented in "C" with changing weighting factors on each iteration.

Model 1: a=2.646251 and b=0.857612.
The range of a is [1,10] and b is[-5,5]

Model 2: a=1.982430, b=0.917533 and c= 0.056668.
The range of a , b, c is [1,10] , [-5,5] and [-1,1] respectively.

Model 3: a= 2.529550 , b= h0.867292 , c= -0.020757 and d=0.767248.
The ranges of a,b,c,d is [1,10] , [-5,5] , [-1,1] and [0,20] respectively.

6. RESULTS AND DISCUSSIONS

One of the objectives of the present work is to employ Particle Swarm Optimization for tuning the effort parameters and test its suitability for software effort estimation. This methodology is then tested using NASA dataset and COCOMO data set provided by Boehm. The results are then compared with the models in the literature such as Baily-Basili, Alaa F. Sheta, TMF, Gbell and Harish models. The Particle Swarm Optimization to tune parameters in Software Effort Estimation has an advantage over the other models as the PSO process determines effective parameter values which reduces the Mean Absolute Relative Error, which may easily be analyzed and the implementation is also relatively easy. The following table shows estimated effort of our proposed model:

EXPERIMENT -1:

TABLE 3: Estimated Efforts of Proposed Models

SL. NO	SIZE	MEASURED EFFORT	METHODOLOGY	ESTIMATED EFFORTOF OUR MODELS C1,C2 ARE CONSTANT DURING THE ITERATION (CASE-I)			ESTIMATED EFFORTOF OUR MODELS C1,C2 ARE CHANGED DURING THE ITERATION(CASE-II)		
				MODEL-I	MODEL-II	MODEL-III	MODEL-I	MODEL-II	MODEL-III
1	2.1	5	28	5.000002	4.998887	5.000007	5.000002	5.502722	5.000001
2	3.1	7	26	6.982786	7.047925	7.07543	6.982786	7.071439	6.975912
3	4.2	9	19	9.060186	9.222874	8.999259	9.060186	8.47359	9.154642
4	12.5	23.9	27	23.08629	23.40447	24.05549	23.08629	21.65101	22.82118
5	46.5	79	19	71.2293	71.75396	71.84614	71.2293	68.24138	71.03909
6	54.5	90.8	20	81.61792	82.10557	82.04368	81.61792	78.82941	81.44935
7	67.5	98.4	29	98.05368	98.39988	98.39998	98.05368	96.18965	97.79541
8	78.6	98.7	35	111.7296	111.9449	111.8526	111.7296	110.7037	111.4518
9	90.2	115.8	30	125.7302	125.8721	125.048	125.7302	125.0572	125.6834
10	100.8	138.3	34	138.3002	138.3003	137.2231	138.3002	138.523	138.2999

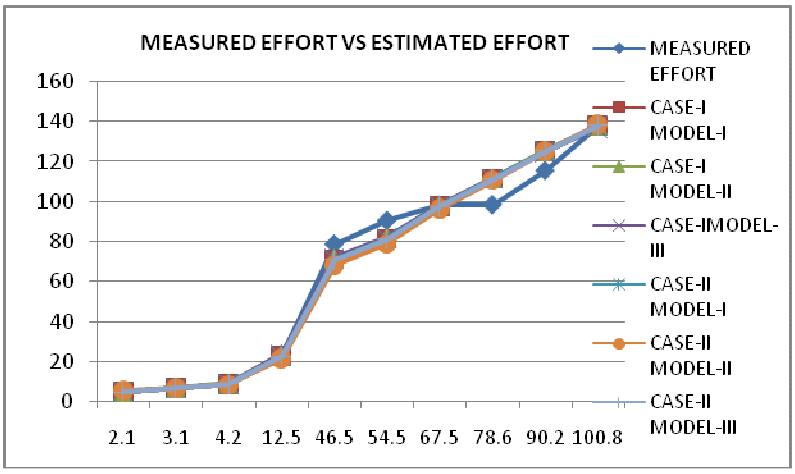


FIGURE 1: Measured Effort Vs Estimated Efforts of Proposed Models

COMPARISON WITH OTHER MODELS

TABLE 4: Measured Efforts of Various Models

Measured effort	Bailey – Basili Estimate	Alaa F. ShetaG. E.Model Estimate	Alaa F. Sheta Model 2 Estimate	Harish model1	Harish model2	CASE-I MODEL-I	CASE-I MODEL-II	CASE-I MODEL-III	CASE-II MODEL-I	CASE-II MODEL-II	CASE-II MODEL-III
5	7.226	8.44	11.271	6.357	4.257	5.000002	4.998887	5.000007	5.000002	5.502722	5.000001
7	8.212	11.22	14.457	8.664	7.664	6.982786	7.047925	7.07543	6.982786	7.071439	6.975912
9	9.357	14.01	19.976	11.03	13.88	9.060186	9.222874	8.999259	9.060186	8.47359	9.154642
23.9	19.16	31.098	31.686	26.252	24.702	23.08629	23.40447	24.05549	23.08629	21.65101	22.82118
79	68.243	81.257	85.007	74.602	77.452	71.2293	71.75396	71.84614	71.2293	68.24138	71.03909
90.8	80.929	91.257	94.977	84.638	86.938	81.61792	82.10557	82.04368	81.61792	78.82941	81.44935
98.4	102.175	106.707	107.254	100.329	97.679	98.05368	98.39988	98.39998	98.05368	96.18965	97.79541
98.7	120.848	119.27	118.03	113.237	107.288	111.7296	111.9449	111.8526	111.7296	110.7037	111.4518

115.8	140.82	131.898	134.011	126.334	123.134	125.7302	125.8721	125.048	125.7302	125.0572	125.6834
138.3	159.434	143.0604	144.448	138.001	132.601	138.3002	138.3003	137.2231	138.3002	138.523	138.2999

7. PERFORMANCE ANALYSIS

Model	VAF (%)	Mean Absolute Relative Error (%)	Variance Absolute Relative Error (%)
Bailey –Basili Estimate	93.147	17.325	1.21
Alaa F. Sheta G.E.model I Estimate	98.41	26.488	6.079
Alaa F. Sheta Model II Estimate	98.929	44.745	23.804
Harish model1	98.5	12.17	80.859
Harish model2	99.15	10.803	2.25
CASE-I MODEL -I	98.92	4.6397	0.271
CASE-I MODEL-II	98.92	4.6122	0.255
CASE-I MODEL-III	98.9	4.4373	0.282
CASE-II MODEL -I	98.92	4.6397	0.271
CASE-II MODEL-II	98.89	7.5	0.253
CASE-II MODEL-III	98.95	4.9	0.257

TABLE 5: Performance Measures

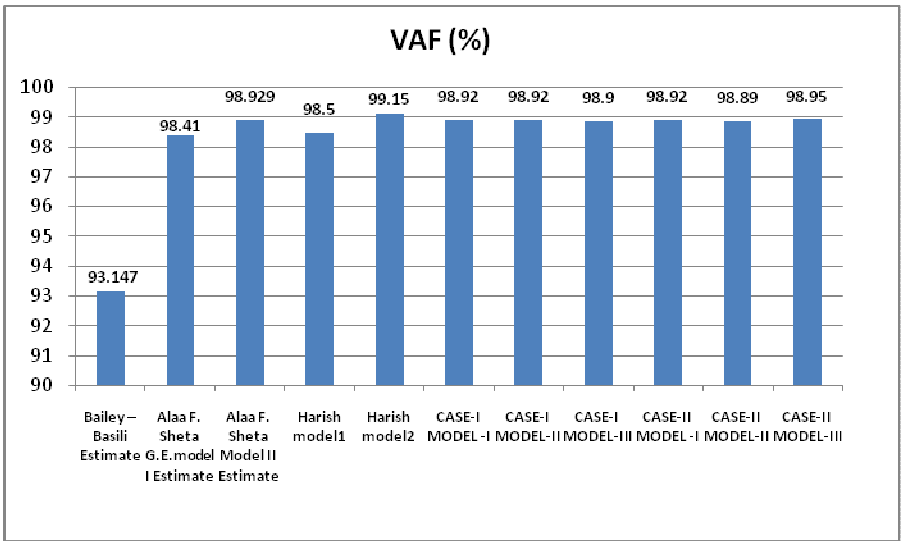


FIGURE 2: Variance Accounted For %

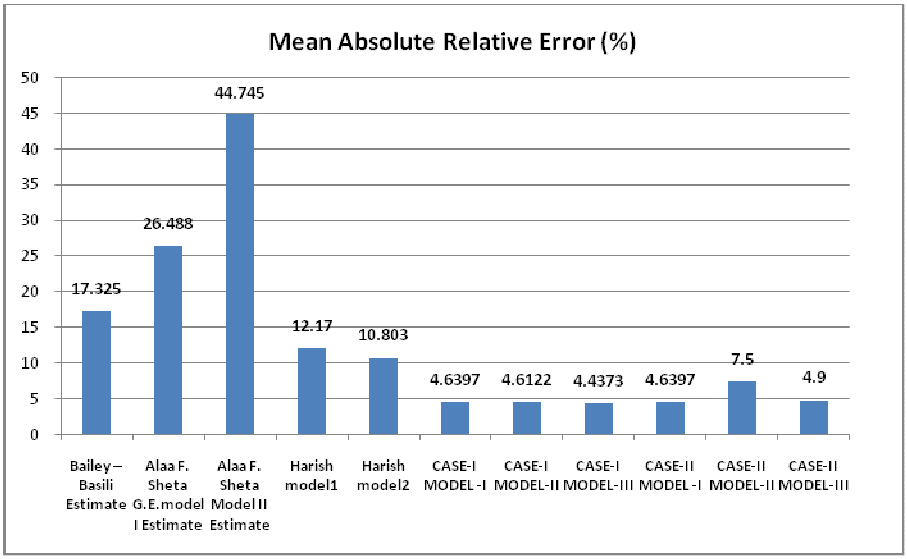


FIGURE 3: Mean Absolute Relative Error (MARE)

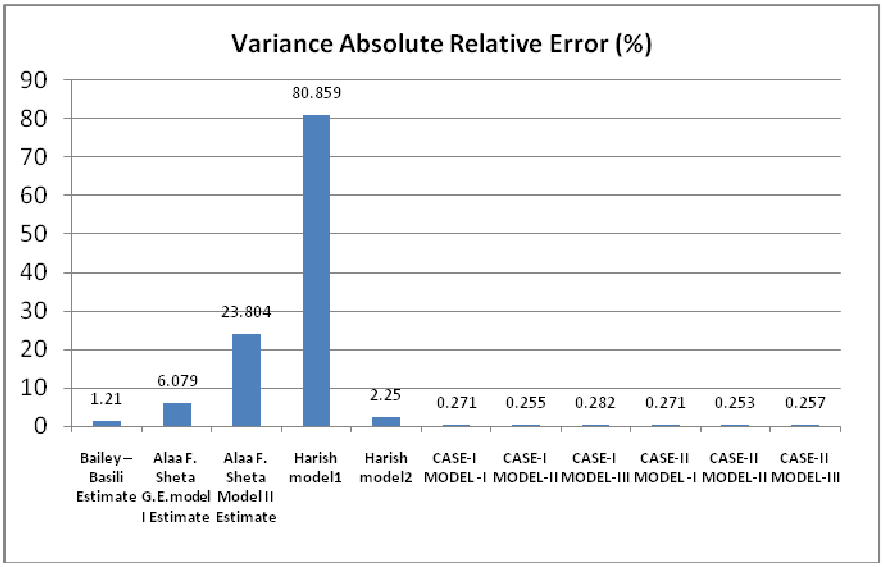


FIGURE 4: Variance Absolute Relative Error %

8 . CONCLUSION

Software cost estimation is based on a probabilistic model and hence it does not generate exact values. However if good historical data is provided and a systematic technique is employed we can generate better results. Accuracy of the model is measured in terms of its error rate and it is desirable to be as close to the actual values as possible. In this study we have proposed new models to estimate the software effort. In order to tune the parameters we use particle swarm optimization methodology algorithm. It is observed that PSO gives more accurate results when juxtaposed with its other counterparts. On testing the performance of the model in terms of the MARE, VARE and VAF the results were found to be futile. These techniques can be applied to other software effort models.

9. REFERENCES

[1] D. E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989.

[2] K. Deb. Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley and Sons, 2002.

[3] C.A. Coello Coello et al. Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer, 2002.

[4] Robert T. F. Ah King and Harry C. S. Rughooputh, "Elitist Multi evolutionary algorithm for environmental/economic dispatch", IEEE 2003.

[5] Alaa F. Sheta , "Estimation of the COCOMO Model Parameters Using Genetic Algorithms for NASA Software Projects", Journal of Computer Science 2 (2): 118-123, ISSN 1549-36362006, 2006.

[6] Alaa Sheta, David Rine and Aladdin Ayesh," Development of Software Effort and Schedule Estimation Models Using Soft Computing Techniques", 2008 IEEE Congress on Evolutionary Computation (CEC 2008), DOI: 978-1-4244-1823-7/08, 2008

[7] Tad Gonsalves, Atsushi Ito, Ryo Kawabata and Kiyoshi Itoh, (2008), Swarm Intelligence in the Optimization of Software Development Project Schedule, DOI 587 10.1109/COMPSAC.2008.179, PP: 587-592, 2008.

[8] J.S.Pahariya, V. Ravi, M. Carr (2009), Software Cost Estimation using Computational Intelligence Techniques, IEEE Transaction, 978-1-4244-5612-3/09/PP: 849-854@2009 IEEE

[9] Parvinder S. Sandhu, Porush Bassi, and Amanpreet Singh Brar (2008), Software Effort Estimation Using Soft Computing Techniques, PP: 488-491, 2008.

[10] Iman Attarzadeh and Siew Hock Ow (2010), Soft Computing Approach for Software Cost Estimation, Int.J. of Software Engineering, IJSE Vol.3 No.1, PP: 1-10, January 2010.

[11] Xishi Huang, Danny Ho, Jing Ren, Luiz F. Capretz (2005), Improving the COCOMO model using a neuro-fuzzy approach, doi:10.1016/j.asoc.2005.06.007, Applied Soft Computing 7 (2007) PP: 29–40, @2005 Elsevier.

[12] Alaa Sheta, David Rine and Aladdin Ayesh (2008), Development of Software Effort and Schedule Estimation Models Using Soft Computing Techniques, IEEE Transaction, 978-1-4244-1823-7/08/PP: 1283-1289@2008 IEEE.

[13] John w. Bailey and victor R.Basili,(1981) "A meta model for software development resource expenditures", Fifth International conference on software Engineering, CH-1627-9/81/0000/0107500.75@ 1981 IEEE, PP 107-129,1981.

[14] Anish M, Kamal P and Harish M, Software Cost Estimation using Fuzzy logic, ACM SIGSOFT Software Engineering Notes,Vol.35 No.1, ,November 2010, pp.1-7

[15] Iman A and Siew H.O, Soft Computing Approach for Software Cost Estimation, Int.J. of Software Engineering, IJSE Vol.3 No.1, January 2010, pp.1-10.

INSTRUCTIONS TO CONTRIBUTORS

The International Journal of Software Engineering (IJSE) provides a forum for software engineering research that publish empirical results relevant to both researchers and practitioners. IJSE encourage researchers, practitioners, and developers to submit research papers reporting original research results, technology trend surveys reviewing an area of research in software engineering and knowledge engineering, survey articles surveying a broad area in software engineering and knowledge engineering, tool reviews and book reviews. The general topics covered by IJSE usually involve the study on collection and analysis of data and experience that can be used to characterize, evaluate and reveal relationships between software development deliverables, practices, and technologies. IJSE is a refereed journal that promotes the publication of industry-relevant research, to address the significant gap between research and practice.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Starting with volume 2, 2011, IJSE appears in more focused issues. Besides normal publications, IJSE intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

We are open to contributions, proposals for any topic as well as for editors and reviewers. We understand that it is through the effort of volunteers that CSC Journals continues to grow and flourish.

IJSE LIST OF TOPICS

The realm of International Journal of Software Engineering (IJSE) extends, but not limited, to the following:

- Ambiguity in Software Development
- Application of Object-Oriented Technology to Engin
- Architecting an OO System for Size Clarity Reuse E
- Composition and Extension
- Computer-Based Engineering Techniques
- Data Modeling Techniques
- History of Software Engineering
- IDEF
- Impact of CASE on Software Development Life Cycle
- Intellectual Property
- Iterative Model
- Knowledge Engineering Methods and Practices
- Licensing
- Modeling Languages
- Object-Oriented Systems
- Project Management
- Quality Management
- Rational Unified Processing
- SDLC
- Software Components
- Software Deployment
- Software Design and applications in Various Domain
-
-
- Software Engineering Demographics
- Software Engineering Economics
- Software Engineering Methods and Practices
- Software Engineering Professionalism
- Software Ergonomics
- Software Maintenance and Evaluation
- Structured Analysis
- Structuring (Large) OO Systems
- Systems Engineering
- Test Driven Development
- UML
-

CALL FOR PAPERS

Volume: 3 - **Issue:** 1 - February 2012

i. Paper Submission: November 30, 2011 **ii. Author Notification:** January 01, 2012

iii. Issue Publication: January / February 2012

CONTACT INFORMATION

Computer Science Journals Sdn BhD
B-5-8 Plaza Mont Kiara, Mont Kiara
50480, Kuala Lumpur, MALAYSIA

Phone: 006 03 6207 1607
006 03 2782 6991

Fax: 006 03 6207 1697

Email: cscpress@cscjournals.org

CSC PUBLISHERS © 2011
COMPUTER SCIENCE JOURNALS SDN BHD
M-3-19, PLAZA DAMAS
SRI HARTAMAS
50480, KUALA LUMPUR
MALAYSIA

PHONE: 006 03 6207 1607
006 03 2782 6991

FAX: 006 03 6207 1697
EMAIL: cscpress@cscjournals.org