Software Failures and Chaos Theory

Dinesh Kumar Saini and Moinuddin Ahmad *Member, IAENG*

***Abstract*— Software itself is very complex and studying its failure is much more complex. Complexity is a relative concept and defining complexity in software failure is chaotic in nature. Software failure of complex systems and understanding the complex system needs some sort of theoretical support for analyzing the nonlinearity in the system. Complex systems have attributes like subsystems, composed of nonlinear dynamic elements and feedback loops. Software Failure is highly disorganized and unmanageable so the chaos theory is used to study the behavior of software.**

***Index Terms*—Chaos Theory, Software systems, Software Failures, on Linear Behavior, Complexity**

I. Introduction

Lorenz gave the classic paradigm of chaos theory and the butterfly effect [1]. Earlier chaos was not much relevant to software industry but with the passage of time, the complexity increased in the systems and development, chaos theory has become popular. Chaos theory helps in understanding the situations which lead to disorganized and unmanageable systems, whilst complexity theory helps to deal with systems that have a large number of subsystems or elements and although hard to predict, these systems have structure and permit improvement [2, 3]. Order and chaos is emerging science in the software systems, which are very complex and have nonlinear properties.

Software systems are in the propensity of a system to be sensitive to initial conditions so that the system becomes unpredictable over time. In fig.1 the different phases of Software Development Life Cycle (SDLC) are shown. If we allow requirements to be changed in later phases other than Software Requirement Specification (SRS) that means the needs of the users of a software system may change over time, invalidating the requirements laid down in an earlier phase. In object-oriented software design the emphasis is on easy maintenance and reuse of the components [4, 10]. Software quality attributes like correctness, robustness, extensibility, and compatibility must also be addressed during design [5, 6, and 7].

One of the major questions organizations is "how secure are my systems from failures?" Answering such a question is often difficult. The root of most security problems is software that fails in unexpected ways when under attack [8]. Despite extensive research in security engineering, measuring security is still a difficult problem [9]. While we do not have security measurements with absolute certainty, we often rely on measurement of risk in assessing security [11, 12, and 13].

Using risk of violations to evaluate security decisions is a common practice. It provides a systematic mechanism for optimizing cost and resources [16, 17, and 18]. The difficult part lies in providing accurate information on failures and their likelihood. Since systems are typically exposed to constant changes, associated risks are often affected by such changes. However, risk assessments are not typically repeated as often as changes are introduced into systems. Over time, initial risk estimates become outdated possibly leading to less secure systems [13, 14].

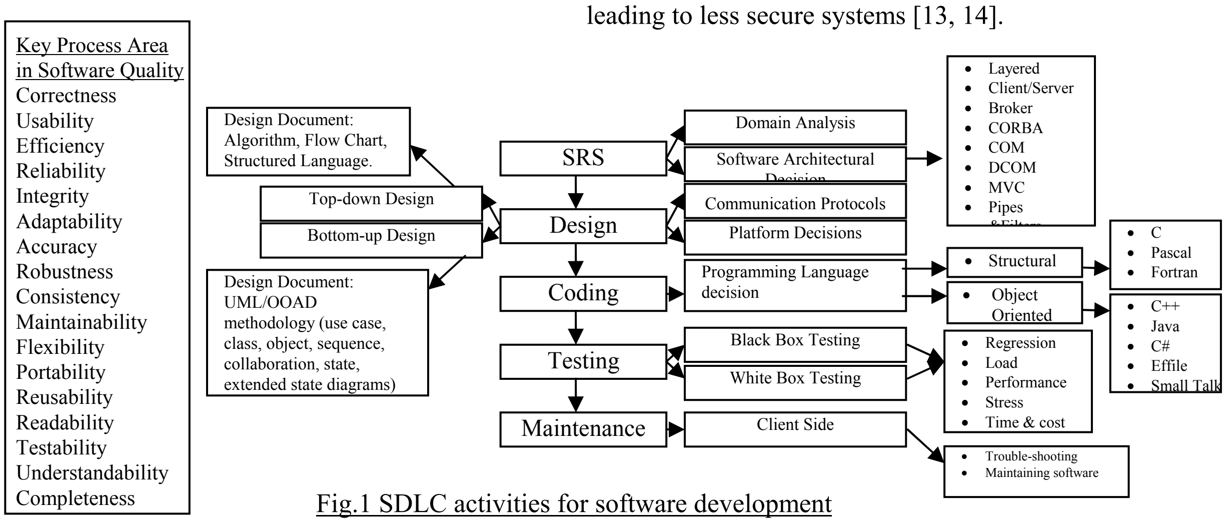


Fig.1 SDLC activities for software development

Dinesh Kumar Saini is with the Faculty of Computing, Sohar University, Oman.
He is Sr. Research Fellow, Faculty of Engineering and IT, University of Queensland, Brisbane, Australia. (Corresponding author phone:+968-95784762;
E-mail:dsaini@soharuni.edu.om, d.saini@uq.edu.au).
Moinuddin Ahmad is with the Faculty of Business, Sohar University, Oman (Phone: +968-95784715 E-mail: mahmad@soharuni.edu.om)

II. Software Failure

Software development industry is a profession from last sixty years. During this period there have been numerous success stories as well as many broadly publicized failures. These publicized failures are either associated with safety problems or substantial cost overruns and schedule delays in most of the software development project [20]. The oldest case of big failure reported is from 16th century old from the Vasa Tragedy; because of design failure the ship sank at the time of launch itself.

There are many similar examples like the Therac-25 computer-controlled radiation therapy machine is often cited as an example of a software safety problem. Because of a software bug, the Therac-25 massively overdosed six people. The other similar project failure which is reported is the automated baggage management system at the new Denver Airport is often cited as an example of a cost and schedule delay. At one point, this delay was costing the city of Denver over $1 million a day in interest and operating costs [15]. There are many more examples of these projects in the literature. There are two categories of the projects one is called projects as runaway projects and the second one is called death-march projects [19].

One recent study reported that 31.1% of all corporate software development projects are canceled before they get completed and 52.7% are costing 189% of their original estimates (Standish Group International, report). The reasons for the project failure are analyzed and finding conveys that failure happened because of either a project that has been canceled or a project that does not meet its budget, delivery, and business objectives [21, 22].

Project success is measured in terms of completing the project in budget, delivery, and business objectives. The average software project success rate in the Standish study is an abysmal 16.2%. Researchers have determined that software developers have much higher achievement needs than the general population. How do software developers reconcile their high achievement needs in a profession with an advertised failure rate of 84%? This question is not easy to answer and efforts are made in the paper to analyze the failure and its nature which is quite complicated for the project failure through the perspective of the software developers that worked on a software development project failure.

Software failure is exploited to develop a model of dynamic program complexity for the identification of failure prone software. Based on this new look at software failures the areas of interest in software reliability is examined. Software complexity is the main factor that leads to chaotic behavior in the case of failure. The use of dynamic complexity and failure models is incorporated into a software reliability model. Finally, the characteristics of software faults and failures are modeled; two degrees of freedom in the software design process are identified representing an initial step in the mathematical specification of software design objectives.

“Chaos” is a term that describes pseudorandom behavior generated by a system that is both deterministic, and nonlinear. Nonlinear dynamic methods, including Poincare map, fractal dimension, correlation dimension, Kolmogorov, entropy, and Lyapunov exponents, can analyze irregular or chaotic activities. Nonlinear dynamic analysis methods need not replace existing methods, but they could improve the array of fault analysis tools available to the tester. The combination of traditional and nonlinear dynamic analyses could potentially improve our ability to analyze test cases from the software with faults.

In today’s world of competition when companies are competing very tightly with each other in the software development environment, development cost must be optimized, time to market must be reduced and decisions needed for performing the same must be supported with the testing data and failure rate. Software development is done both in-house and outsources and field failure rate helps the development environment. Software reliability growth models help to cheek the robustness of the software developed but still it doesn’t give guaranties for the failures.

Fault removal is non instantaneous and most of the software development environments involve third parties, off the self and semi custom hardware and software. Most of the software development has supplier focus on development of high value applications and system integration. When software failure occurs it create an uncertainty in the environment and removing this fault require lot of efforts and longer time [23].

Parameter estimation and variation in the parameter estimation help in predicting the software fault and confidence interval for the same [24, 25].

Sensitivity analyses are conducted to estimate the uncertainties in the failure rate prediction. Process control and on line transaction processing are some of the application that require high availability and failures cannot be accepted for such applications. For some critical applications like server applications, downtime leads to lost productivity and lost business.

All most all the companies and organizations of any nature highly depend on the software and its failure leads to loss of business and time. With the tremendous growth of e-commerce, almost every kind of organization is becoming dependent upon highly available systems. System
availability is highly affected by the software failure. Unfortunately, software failures severely reduce system availability. Arecent Gartner Dataquest surveyed around three hundred companies of all sizes and across diverse industry segments showed that software defects account for 27% of system failure. Concurrency and memory-related bugs are common software defects, causing more than 60% of system vulnerabilities [CERT/CC] and accounting for 33–43% of the reported bugs in mature database management systems and operating systems. For this reason, software companies invest enormous effort and resources on software testing and bug detection prior to releasing software [27, 28].

However, software failures still occur during production runs since some bugs inevitably slip through even the strictest testing. Therefore, to achieve higher system availability, mechanisms must be devised to allow systems to survive the effects of un-eliminated software bugs to the largest extent possible.

Earlier work on surviving software failures can be classified into four categories.

The first category encompasses those techniques which were designed to handle either transient hardware failures or nondeterministic software bugs, which could not deal with deterministic software bugs, a major cause of software failures [29 and 30].
Another major limitation of these methods is service unavailability
However, it requires legacy software to be reconstructed in a loosely coupled fashion. The second category includes general check pointing and recovery. The most straight forward method in this category is to checkpoint, rollback upon failures, and then re executes either on the same machine. Similar to the whole program restart approach, these techniques were also proposed to deal with hardware failures, and therefore suffer from the same limitations in addressing software failures.
In particular, they also cannot deal with failures caused by deterministic bugs. Progressive retry and environment diversity are interesting improvement over these approaches. They increase the degree of program execution non determinism by either reordering messages or creating diverse OS environment. While proposing a promising direction, they limit the technique to message reordering or diverse OS environment creation, such as process migration. As a result, they cannot handle bugs unrelated to message order or OS environment. For example, if a server receives a malicious request that exploits a buffer overflow bug. Their approaches will not solve the problem. The most aggressive approaches in the check pointing/recovery category include recovery both of which rely on different implementation versions upon failures.

III. Fault and Bug Analysis and Software Failure

Software bug is a very complicated issue and when it affects the software, failure is certain, now how to analyze the bug and its impact is a tough task for the testers and developers. In this papers we are trying to find out the reasons of the failures and failure effects in the development environment. The software bug can be of deterministic or non-deterministic nature, now the question arises is how to find out which kind of the bug appeared in the software and how it will affects the development. These approaches may be able to survive deterministic bugs under the assumption that different versions of the software fail independently. This is not guarantees because programmers tend to make similar mistakes [31, and 32].

There are other issues and to approach them is very expensive and if adopted by software companies, the software development costs get doubled. An alternative to N-version programming is data diversity that tries to execute multiple copies of the same program, each with a different form of the input [33].

A theoretical framework can be proposed but still the validation is required from the practical tests which are not easy to work out in non-monitored software development environment. In particular, it does not answer how to apply the idea transparently without modifying the application and without causing major performance degradation during normal execution. Some applications comprise application-specific recovery mechanisms, such as the multi process model (MPM), exception handling, etc. Some multi processed applications, such as the old version of the Apache HTTP Server and the CVS server, spawn a new process for each client connection and therefore can simply kill a failed process and start a new one to handle the unanswered request. While simple and capable of surviving certain software failures, this technique has several limitations.

There are different approaches for the bug analysis and these includes First, if the bug is deterministic, the new process will most likely fail again at the same place given the same request (e.g., a malicious request). Second, if a shared data structure is corrupted, simply killing the failed process and restarting a new one will not restore the shared data to a consistent state, therefore potentially causing subsequent failures in other processes. Other application-specific recovery mechanisms require software to be failure-aware, which adversely affects programmability and code readability. The fourth category includes several recent nonconventional proposals such as failure-oblivious computing [34] and the reactive immune system [34]. Failure-oblivious computing proposes to deal with buffer overflows by providing artificial values for out-of-bound reads, while the reactive immune system returns a speculative error code for functions that suffer software failures (e.g., crashes). While these approaches are fascinating and may work for certain types of applications or certain types of bugs, they are unsafe to use for correctness-critical applications (e.g., on-line banking systems) because they “speculate” on programmers’ intentions, which can lead to program misbehavior. Even worse, the problem becomes much harder to detect if the speculative “fix” introduces a silent error that does not manifest itself immediately. In addition, such problems, if they occur, are very hard for programmers to diagnose since the application’s execution has been forcefully perturbed by those speculative “fixes.” Besides the above individual limitations, existing work provides insufficient feedback to developers for debugging. For example, the information provided to developers may include only a core dump, several recent checkpoints, and
an event log for deterministic replay of a few seconds of recent execution. To save programmers’ debugging effort, it is desirable if the runtime system can provide information regarding the bug type, under what conditions the bug is triggered, and how it can be avoided. Such diagnostic information can guide programmers in the debugging process and thereby improve their efficiency.

IV. Chaos Theory

There is several prediction methods described in chaotic series. Some of the common methods are the first-order local-region method, the whole domain method, the local-region method, the weighted first-order local-region method, the Lyapunov exponent method and neural networks. The Lyapunov exponent is one of the main methods which characteristics' indices of chaos and can be used to assess chaotic characteristics as well as chaotic prediction with great accuracy. Considering the prediction accuracy and the characteristics of software failure prediction, Lyapunov exponent method is selected.

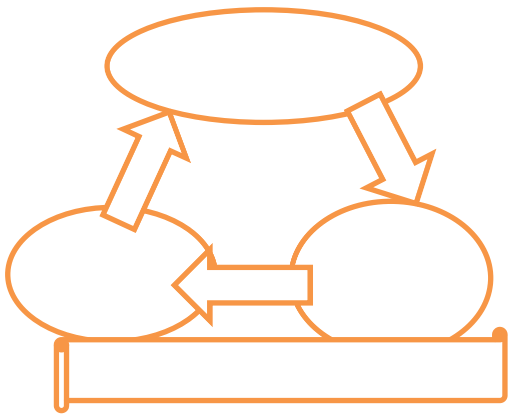


Figure 2 Chaos Theory Triangle

Chaos identification is the process of identifying the behavior of the software which is highly un- deterministic and that leads for the behavior of software failures to be chaotic. Correlation dimension is one of characteristic parameters of chaos. Based on chaos theory, the correlation dimension D increases with the embedding dimension m [11]. If D, converges at a stable value, the time
Series is chaotic, else it is stochastic.

A. Theory Formulation

1. Instability of behavior and system.
2. Non – linearity.
3. Emergent order

$$\frac{dx(t)}{dt} = f(x(t)) \ldots\ldots\ldots\ldots\ldots 1$$

$f(x)$ …… Is a nonlinear vector field

$$x(t) = f(x(0)t) \ldots\ldots\ldots\ldots\ldots\ldots\ldots 2$$

F: m $\longrightarrow$ m represents the flow that determines the evaluation of (x) t.

x (t) for initial condition x(0).

B. CHAOS

Initial condition sensitivity is measured by determine number of exponent Lyapunov (λ)

Mathematical representation of the formal equation of exponent Lyapunov (λ)

For given i, [P_i (t)] dimension is λ=
$$\lim_{t\to\infty}\left(\frac{1}{t}\right)^2 \log\left[\frac{Pi(t)}{Pi(0)}\right]\ldots\ldots\ldots\ldots 3$$
Logistic function calculation will be
$$\lambda = \frac{1}{N}\sum_{n=1}^{N}\log_2(r - 2rXn)\ldots\ldots\ldots\ldots\ldots\ldots 4$$
λ = exponent Lyapunov
t = t – period
P_i (t) = data I for t - period
P_i (0) = data I for initial period
X_n = data – n
N = number of data
r = input parameters

Chaos
$\lambda < 0$ state of the system stable
$\lambda = 0$ system in steady state
$\lambda > 0$ condition of system is chaotic
Fractal dimension
Accommodating objects in its space.
Arranging variable in dynamic system
Fractal dimension

$$N * d^D = 1 \ldots\ldots\ldots\ldots\ldots\ldots 5$$

Where N = number of circles
d = diameter of the circle
D = Fractal dimension

$$D = \frac{\log N}{\log_{d}-1}\ldots\ldots\ldots\ldots\ldots\ldots\ldots 6$$

Correlation dimension can be used to measure the fractal dimension with correlation integral $C_m(R)$.

Correlation integral is probability in attractor which have a distance R among the other point.

$$C_m(R) = \left(\frac{1}{N^2}\right) * \sum_{\substack{i,j=1 \\ i\neq j}}^{N} z(R - |x_i - x_j|)\ldots\ldots\ldots 7$$

Where $z(x) = 1, if\ (R - |x_i - x_j|) > 0$
N = Number of observation
R= distance
C_m= correlation integral of m-dimension

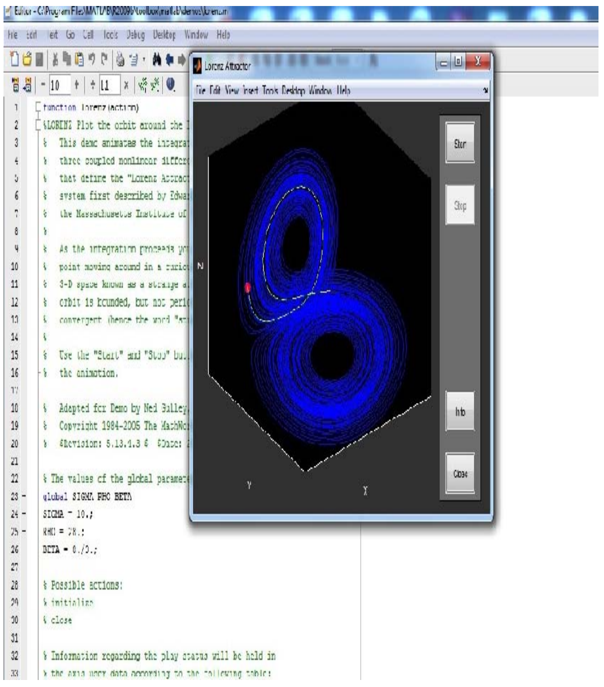
$$C_m = R^D\ldots\ldots\ldots\ldots\ldots 8$$

$$\log(C_m) = D * \log(R) + constant\ldots\ldots\ldots\ldots 9$$

Chaos is the phenomena which seems random in nature and irregular which emerge from deterministic systems [5]. It can be describe chaos as the outcome of tangible systems with nonlinear interdependent variables with sensitive dependence on initial conditions. Such systems are determined by precise laws, but due to the sensitive dependence on initial conditions, their behavior possesses an amount of randomness.

Chaos is a new kind of state of nonlinear systems. Sensitive dependence on initial conditions is the key feature of chaos. For example, there are deterministic systems in which an initial difference of one unit between two states will eventually increase to a hundred units, or even a million units. However small the initial difference is, it will evolve bigger and bigger with time. It is this that renders long-term predictability impossible [6] the behavior of the chaos is plotted which comes like a butterfly and it is called butterfly effect.

V. BEHAVIOR MODELING OF CHAOS THEORY


Editor - C:\Program Files\MATLAB\R2009b\toolbox\matlab\demos\lorenz.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Lorenz Attractor
File Edit View Insert Tools Desktop Window Help
Start
Stop
Info
Close
Z
Y
X

```
function lorenz(action)
%LORENZ Plot the orbit around the L
%   This demo animates the integra
%   three coupled nonlinear differ
%   that define the "Lorenz Attract
%   system first described by Edwa
%   the Massachusetts Institute of
%
%   As the integration proceeds yo
%   point moving around in a curio
%   3-D space known as a strange a
%   orbit is bounded, but not peri
%   convergent (hence the word "st
%
%   Use the "Start" and "Stop" but
%   the animation.

%   Adapted for Demo by Ned Gulley
%   Copyright 1984-2005 The MathWo
%   $Revision: 5.13.4.3 $  $Date: 

% The values of the global paramet
global SIGMA RHO BETA
SIGMA = 10.;
RHO = 28.;
BETA = 8./3.;

% Possible actions:
% initialize
% close

% Information regarding the play status will be held in
% the axis user data according to the following table:
```

Figure 3 Chaos Theory Behavior Plot

In the paper chaos theory is to be applied in the software systems, one the best way to do is to apply chaos theory on software reliability assessment which can be quantified and can be observed. Conventional SRGM specify the form of a random process that describes the behavior of software failures which is deterministic in some way. It possesses chaos as well as randomness.

Because of resource and time constrained we did not tested our chaotic nature but other parallel studies support the argument that as shown in the results, it fits well for the actual data sets which are chaotic. The goodness-fit of the proposed method based on chaos theory with the conventional stochastic SRGM JM model and NHPP model can be tested on the real data sets. Comparison can be done for finding how the proposed method fits better than the conventional stochastic SRGM JM model and NHPP mode for the actual data sets which are chaotic.

VI. CONCLUSION

Software development and software projects are very complex systems and success or failure of the software systems have great impact on the business. In the paper we started with the concept of SDLC, its various activities, software bug analysis and software failures. All the possible causes of failure and its effects on the systems and its natures is studied in the paper. Formal theory of Chaos is formulated for the software systems and software development process.
Software success or failure concept needs a new theory or paradigm which can suggest why and how failure happened. An extensive framework must be formulated for quantified success and failure for the software development and projects.
The current definition of software project success may be too narrowly defined and may create negative perceptions about software developers. There also may be instances when these failure statistics are used as fear-based advertisements for consultant services.

VII. LIMITATIONS

Limited by test conditions and data sources, only a preliminary research on chaotic prediction of the spatial series of our software failure desorption index of software is carried out. The universality of our conclusions still needs further testing, analyses and arguments.

ACKNOWLEDGEMENT

This paper is presented from the conference support fund from the Sohar University. Authors are thankful for the Sohar University research office and faculty colleagues for the support. The authors are thankful for the conference editor and reviewer for the valuable review and suggestions.

REFERENCES

[1] E. Lorenz, "*Deterministic no periodic flow*", Journal of Atmospheric Science 20 (1963) 130–141
[2] A. Zahra, C. Ryan, "*From chaos to cohesion—complexity in tourism structures: an analysis of New Zealand's regional tourism organizations*", Tourism Management 28 (3) (2007) 854–862.
[3] R. Axelrod, M. Cohen, "*Harnessing Complexity: Organizational Implications of a Scientific Frontier*", the Free Press, New York, 1999.B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.
[4] Dinesh Kumar Saini and Nirmal Gupta "*Fault Detection Effectiveness in GUI Components of Java Environment through Smoke Test*", Journal of Information Technology, ISSN 0973-2896 Vol.3, issue3, 7-17 September 2007.
[5] Gu, w., kalbarczyk, z., iyer, R. K., and yang, z.-y. "*Characterization of Linux kernel behavior under errors*". In Proceedings of the International Conference on Dependable Systems and Networks. 2003.
[6]]Dinesh Kumar Saini and Nirmal Gupta "*Class Level Test Case Generation in Object Oriented Software Testing*", International Journal of Information Technology and Web Engineering, (IJITWE) Vol. 3, Issue 2, pp. 19-26 pages, march 2008.

[7] Prvulovic, M. And Torrellas, J, "Reenact*: using thread-level speculation mechanisms to debug data races in multithreaded codes"* In proceedings of the 30th international symposium on computer architecture. 2003

[8] Dinesh Kumar Saini and Hemraj Saini *"VAIN: A Stochastic Model for Dynamics of Malicious Objects"*, the ICFAI Journal of Systems Management, Vol.6, No1, pp. 14- 28, February 2008.

[9] Hemraj Saini and Dinesh Kumar Saini *"Malicious Object dynamics in the presence of Anti Malicious Software"* European Journal of Scientific Research ISSN 1450-216X Vol.18 No.3 (2007), pp.491-499 © Euro Journals Publishing, Inc. 2007 http://www.eurojournals.com/ejsr.htm

[10] Hallem, S., Chelf, B., Xie, Y., and Eengler, D. "*A system and language for building system specific, static analyses".* In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. 69–82., 2002.

[11] Dinesh Kumar Saini, Jabar H. Yousif, and Wail M. Omar *"Enhanced Inquiry Method for Malicious Object Identification"* ACM SIGSOFT Volume 34 Number 3 May 2009, ISSN: 0163-5948, USA.

[12] Dinesh Kumar Saini "Sense the Future" Campus Volume 1-Issue 11, Page No14-17, February 2011.

[13] Dinesh Kumar Saini and Moinuddin Ahmad *"Modeling of Object Oriented Software Testing Cost"* The 2011 International Conference on Software Engineering Research and Practice (SERP'11), World Congress in computer Science and Engineering, July 18-21, 2011, Las Vegas, USA. Pp. 333-339.

[14] Dinesh Kumar Saini and Moinuddin Ahmad *"Enhanced Software Quality Economics for Defect Detection Techniques Using Failure Prediction"* The 2011 International Conference on Software Engineering Research and Practice (SERP'11) World Congress in computer Science and Engineering July 18-21, 2011, Las Vegas, USA, PP. 346-351.

[15] Rinard, M., Cadar, C., Dumitran, D., Roy, D. M., Leu, T., and Beebee, jr., W. S, "*Enhancing server availability and security through failure-oblivious computing",* In proceedings of the 6th symposium on operating system design and implementation. . 2004

[16] Dinesh Kumar Saini, Sanad Al Maskari and Hemraj Saini, *"Malicious Object Trafficking in the Network"* IEEE IDCTA-2011, Korea, August 13-16, 201

[17]] Dinesh Kumar Saini, Sanad Al Maskari and Lingaraj Hadimani "*Mathematical Modeling of Software Reusability"* 3rd IEEE International Conference on Machine Learning and Computing (ICMLC, 2011) Singapore, February 26-28, 2011, IEEEXplore, 978-1-4244-9253-4/11

[18] Dinesh Kumar Saini and Hemraj Saini *"Achieving Quality Through Testing Polymorphism in Object Oriented Systems",*3rd International Conference on Quality, Reliability and INFOCOM Technology (Trends and Future Directions), 2-4 December, 2006, Indian National Sciences and Academics, New Delhi (India). Conference proceeding.

[19] Li, z., Tan, l., Wang, X., lu, s., Zhou, Y., and Zhai, C. "*Have things changed now? an empirical study of bug characteristics in modern open source software"*. In Procedings of the 1st Workshop on Architecture and System Support for Improving Software Dependability. 2006

[20] Dinesh Kumar Saini and Hemraj Saini *"Static Code Analysis",* NSCOMCS-2005 Proceeding of National Seminar on Mathematics and Computer Science sponsored by UGC.

[21] Lowell, D. E., Chandra, S., and Chen,P. M. "*Exploring failure transparency and the limits of generic recovery*". In proceedings of the 4th symposium on operating system design and implementation. 2000.

[22] Dinesh Kumar Saini and Hemraj Saini *"Identification and characterization of software testing process for object oriented systems",* National Conference on Mathematical Analysis and its Applications in Real -World Problems, Berhampur University, September 2006

[23] Plank,J. S., Li, K., and Puening, M. A. "*Diskless check pointing"* . IEEE transactions on parallel and distrib. Syst. 9, 10, 972–986. 1998.

[24] Dinesh Kumar Saini and Hemraj Saini *"Software Metrics and Mathematical Models in the Software Development Environment for Improving its Quality"*, National Conference on Mathematical Modeling, BITS Pilani, Oct.2005

[25] Qin, F., Lu, S., and Zhou, Y. *"Safemem: exploiting ecc-memory for detecting memory leaks and memory corruption during production runs",* In proceedings of the 11th international symposium on high-performance computer architecture. 2005.

[26] [18]Dinesh Kumar Saini and Hemraj Saini *"Analytical Study of Mathematical Models For Software Reusability Metrics in Software Development Environment"* National Conference on Mathematical Modeling and Analysis – October 2004.

[27] Rodrigues, R., Castro, M., and Liskov, B.. Base: "*using abstraction to improve fault tolerance",* in proceedings of the 18th symposium on operating systems principles. 2001

[28] Sidiroglou, S., Locasto,m. E., Boyd, S.W., and Keromytis, A.D*. "Building a reactive immune System for software services",* In proceedings of the usenix annual technical conference. 2005.

[29] Dinesh Kumar Saini and Hemraj Saini *"Compliance Management Framework and Methodology", 8*th International Research Conference on Quality, Innovation and Knowledge Management, Feb 11-14, 2007 at IMT New Delhi Sponsored by CII, IBEF, Monash University .

[30] Swift, M., Annamalai, M., Bershad, B. N., and Levy, H. M. "Recovering device drivers", In proceedings of the 6th symposium on operating system design and implementation. 2004.

[31] Dinesh Kumar Saini and Hemraj Saini *"Statistical Modeling of Extensibility in software"* 3rd International Conference on Quality, Reliability and INFOCOM Technology (Trends and Future Directions), 2-4 December, 2006, Indian National Sciences and Academics, New Delhi (India). ISBN 81–7446–434–4 Conference proceeding.

[32] Russinovich,m. And cogswell, b. Replay for concurrent non-deterministic shared-memory applications. In proceedings of the acmsigplan conference on programming language design and implementation. 1996.

[33] Srinivasan, s., andrews, c., kandula, s., and zhou, y. Flashback: a light-weight extension for rollback and deterministic replay for software debugging. In proceedings of the usenix annual technical conference. 2004.

[34] Staniford, s., paxson, v., and weaver, nHow to own the internet in your spare time. In proceedings of the 11th usenix security symposium. . 2002.

[35] Stoller, s. D.. Testing concurrent java programs using randomized scheduling. In proceedings of the 2nd workshop on runtime verification. 2002