International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 4– No.7, December 2012 – www.ijais.org

Return on Investment and Effort Expenditure in the Software Development Environment

Dinesh Kumar Saini
Faculty of Computing and IT, Sohar University, Oman
Faculty of Engineering and IT, University of Queensland, Australia

Moinuddin Ahmad
Faculty of Business, Sohar University, Oman

ABSTRACT

Software development is tedious, expensive and requires lot of resources and investment. Justification of the resources and investments are highly required in the software industry. Development of reliable and quality software has become increasingly important in today's world. A static model of the software development life cycle that treats all modules similarly is becoming inadequate to the task. In this paper efforts are made to quantify the concept of return on investment and factors responsible for improving the return on investment. In the second part of the paper, we are trying to justify the effort expenditure and how to optimize the effort expenditure.

General Terms

Business, Software Systems, Software Development Environment, Software Testing

Keywords

Software, effort, Return on Investment, Testing, Model

1. INTRODUCTION

A dynamic model of reengineering in the development processes is needed to achieve the high level of quality in the software development. This paper focuses on reengineering the business processes that produce commercial software to take advantage of software quality modeling technology. In particular, this paper presents how to analyses the costs and benefits of the accuracy of a software quality and cost of quality by measuring the efforts [1, 2]. A cost-benefit analysis gives insight into the implications of implementing the recommendations of a software quality model in the context of a dynamic development process [3]. Logistic regression in conjunction with a specialized mathematical model, which we have been proposed, predicted whether each module was *fault-prone* or not [4].

2. RETURN ON INVESTMENT

Successes and failures of large-scale software development and Information Technology projects have negative repercussions on stock prices and market capitalization of the enterprise [5]. It also impacts existing and future customers' perceptions about the enterprise. Financial metrics and ROI are becoming mandatory tools to convey effective decision making and value creation ability of a successful software development enterprise. Business value drivers are established using ROI. Progress monitoring should be done over the project duration and it helps put milestones and metrics in place. ROI is used in review and decision making in the software development environment. Mission and vision of the enterprise is established using ROI and it enables management and developers to participate in a shared common vision of business success and to identify risks. The data collected and analyzed during the ROI calculation is useful tool for future projects and new contracts[6].

ROI analyses can be extended to measure and govern other metrics such as days sales outstanding (DSO), receivables outstanding, excess inventory, inventory cycle rates, and collection efficiency.

One of the characteristics of a good ROI analysis is identifying risks, trade-offs, and challenges associated with maximizing benefits and minimizing costs. Care should be exercised that risks are properly weighted so that they are not marginalized and similarly a healthy dose of skepticism should be exercised with rewards. Once the ROI analysis is completed and areas of potential improvement identified, it becomes a perfect opportunity to step back and look at the entire scenario. The business value benefits, the new environments, the software applications, the resources, the cost outlay, and the processes will become clear. A good ROI analysis injects a healthy dose of reality. Applications providing self-service search, virtual customer representatives, and natural language interactions are more complex to test [7]. Users may interact with the applications in various ways and the interaction varies significantly between individual users

ROI = (Net Benefits/Net Costs)* 100

ROI Percentage = (Net Benefits/Net Costs) x100%

NPV = Initial Investment+ [Net Benefit for Year 1/(1 + discount rate)]+ [Net Benefit for Year 2/(1 + discount rate)]2+ … ….+ [Net Benefit for Year N/(1 + discount rate)].

Net benefits can be either direct, in terms of incremental revenue generated, productivity gained, or expense saved, or indirect, the redeployment of resources or tasks that the organization would alternatively have had to hire new and like resources to perform. Net costs include recruiting, salaries, and benefits, software licensing, and general and administrative overheads.

3. HOW ROI IMPROVES THE SOFTWARE DEVELOPMENT ENVIRONMENT

Productivity and ROI generally improve with software developers who follow the guidelines listed below.

- Software quality and software productivity are closely related. Both have to be simultaneously improved. More software development does not translate to better quality and vice versa, and a balance of both must be constantly maintained.
- Working long hours on a project does not signify success or higher productivity. When the business model and requirements are not greater, more effort is spent in writing the code.
- Automation tools are used where appropriate at each stage of the project. Manual and repetitive component building and testing leads to lowered productivity, efficiency, and developer dissatisfaction.
- Better tools in the hands of bad developers do not make better code. There is no magic the best people should be driving the most crucial aspects of software application development.
- Define and manage interfaces at the design level and not at the code level.

4. WHEN ROI IS HARD TO QUANTIFY

What we have discussed so far are quantifiable aspects of ROI analysis. Several business values of ROI are hard to quantify. These include the improvements attained in customer satisfaction, customer service, and support organizations [8]. Additional issues such as business brand name and

- Time-to-market improvements.
- Improvements in customer service and support leading to overall improvement in customer expectation management and satisfaction.
- Improved business agility.
- Reduction in uncertainty that may have existed due to lack of business process automation.
- Improvement in brand name and value of the enterprise.
- Increased business-to-business collaboration.

4.1 Software Quality, Testing and ROI

ISO/IEC 15939 standard describes the organizational elements required to support a measurement process and it organizes measurement into four key activities:

- Establish capability
- Plan measurement
- Perform the measurement process
- Evaluate measurement

5. SOFTWARE TESTING ROI

Let us begin with the costs of quality and testing and how to determine baseline ROI. We will discuss how co-sourcing and automated test tools improve ROI. We will discuss several models with examples and cases to illustrate best mechanisms of Applied ROI [9]

Validation is the process of evaluating a system or component during or at the end of the development process to determine whether or not it satisfies specified requirements. Validation activities can be divided into the following [10]

Low level testing:

- Unit testing
- Integration testing

- High level testing:
  - Functional testing
  - System testing
  - Sanity testing
  - Regression testing
  - Acceptance testing
  - Stress testing
  - Usability testing
  - Security testing

5.1 Cost of Quality and Testing

Let us discuss the cost metrics for software development and quality that apply to software testing. It will be good to establish a KPI for each of these cost metrics [11]. Some of these cost metrics are reviewed below.

5.2 Defects per 1000 Lines of Code

Most organizations have between 9 and 10 defects in every 1000 lines of code [6]. Some organizations have reported, as per a Verizon IT study listed earlier, reduction of defects by approximately 25 percent.

5.3 Fully Loaded Tester Cost

Most enterprises in the United States, the European Union, and Japan use this metric to include costs of salary, benefits, workers compensation, taxes for payroll and state disability fund, operational costs including computers, utilities, and office space. This is placed at between $10,000 Software Quality and Test ROI and $15,000 per tester. These costs are sometimes substantially lower for outsourced locations where wages are less. A co-sourcing model as we will discuss later where in-house SQA teams are supplemented by outsourced quality and testing can lead to best of both worlds[13].

5.4 Developer Time and Cost to Fix Defects

According to published studies [7, 8], on average it takes about 6.3 hours for a developer to find and fix a defect. Usually, the average time spent fixing each bug is multiplied by the average number of bugs in a project to arrive at metrics for all the department’s projects. Automated bug tracking tools, issue management processes, and good debugging tool sets reduce this time.

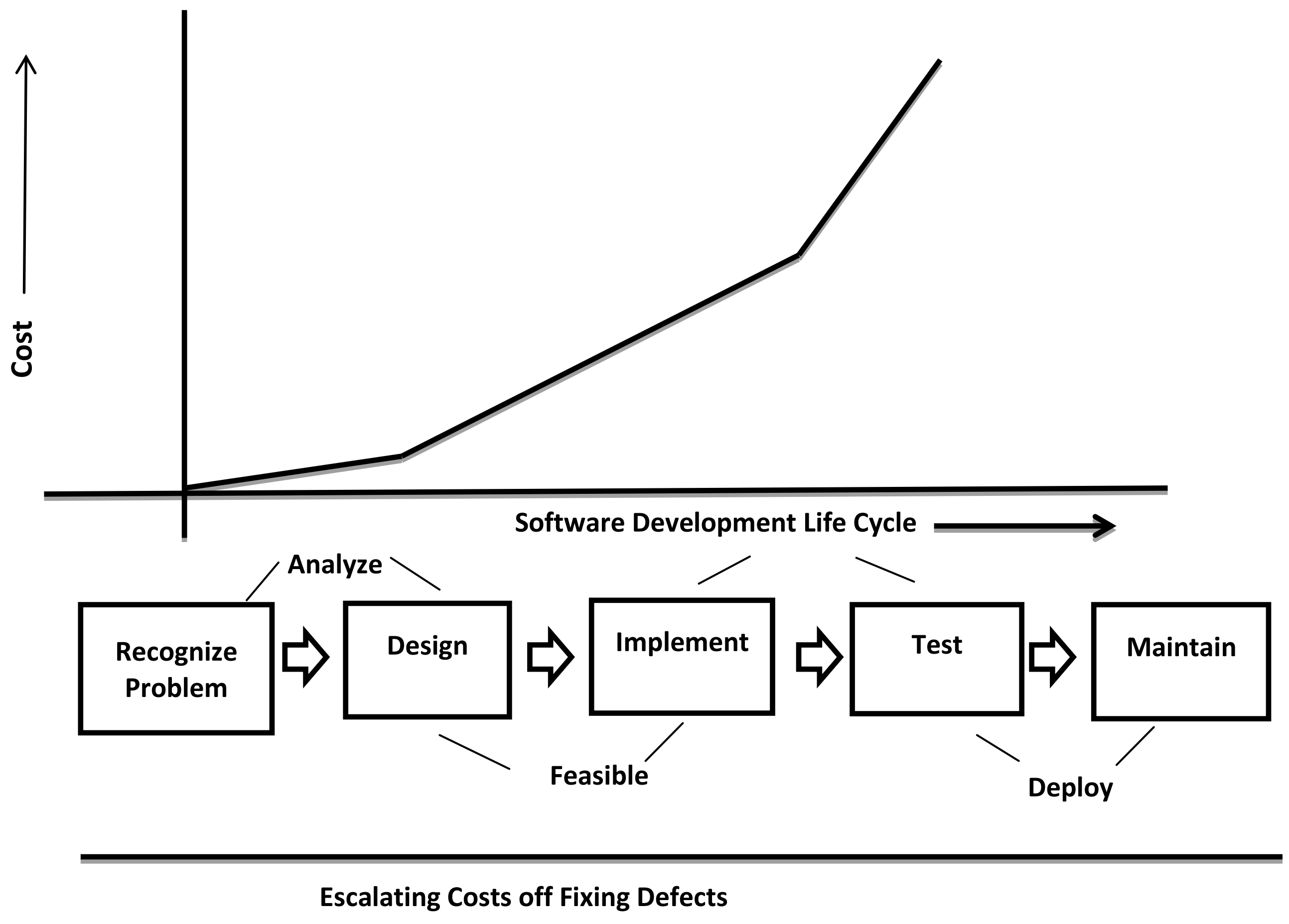

Figure1: Cost Vs. Defect

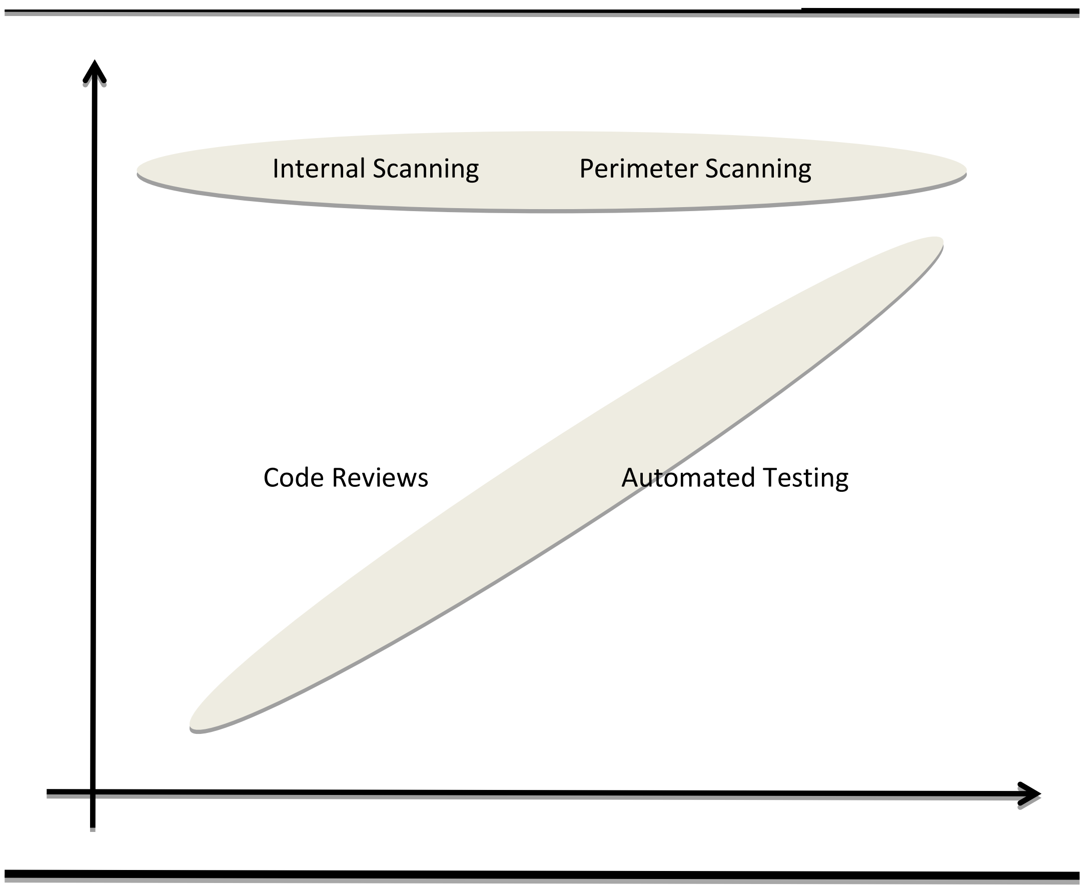

Figure3: Lifecycle of a Bug

Figure 3 shows how security testing can be distributed among in-house and outsource and manual versus automated axes. Penetration testing is generally preferred done in-house and manually, but vulnerability scanning; including perimeter scanning can be automated and outsourced. Co-sourcing can be applied to either case because, unlike pure outsourcing, internal resources are available.

In addition to the direct benefits of co-sourcing, there are several intangible benefits as discussed below:

- Proven methodologies — established co-sourcing QA and test companies have streamlined operations and test
- practices. Documentation, test plan templates, methodologies, and processes of a good Software Quality and Test ROI co-sourcing partner can help QA managers effectively plan, manage, and deliver results.
- Resource strengths — most good co-sourcing QA and test companies have highly trained and enthusiastic engineers who are familiar with a gamut of tools and technologies. This provides significant resource strength and team motivation required for the success of projects.
- Faster time-to-market — co-sourcing enables 24/7 test and QA cycles and reduces the time required for conceptualization to the final product. Ramping up of a large pool of resources becomes possible.

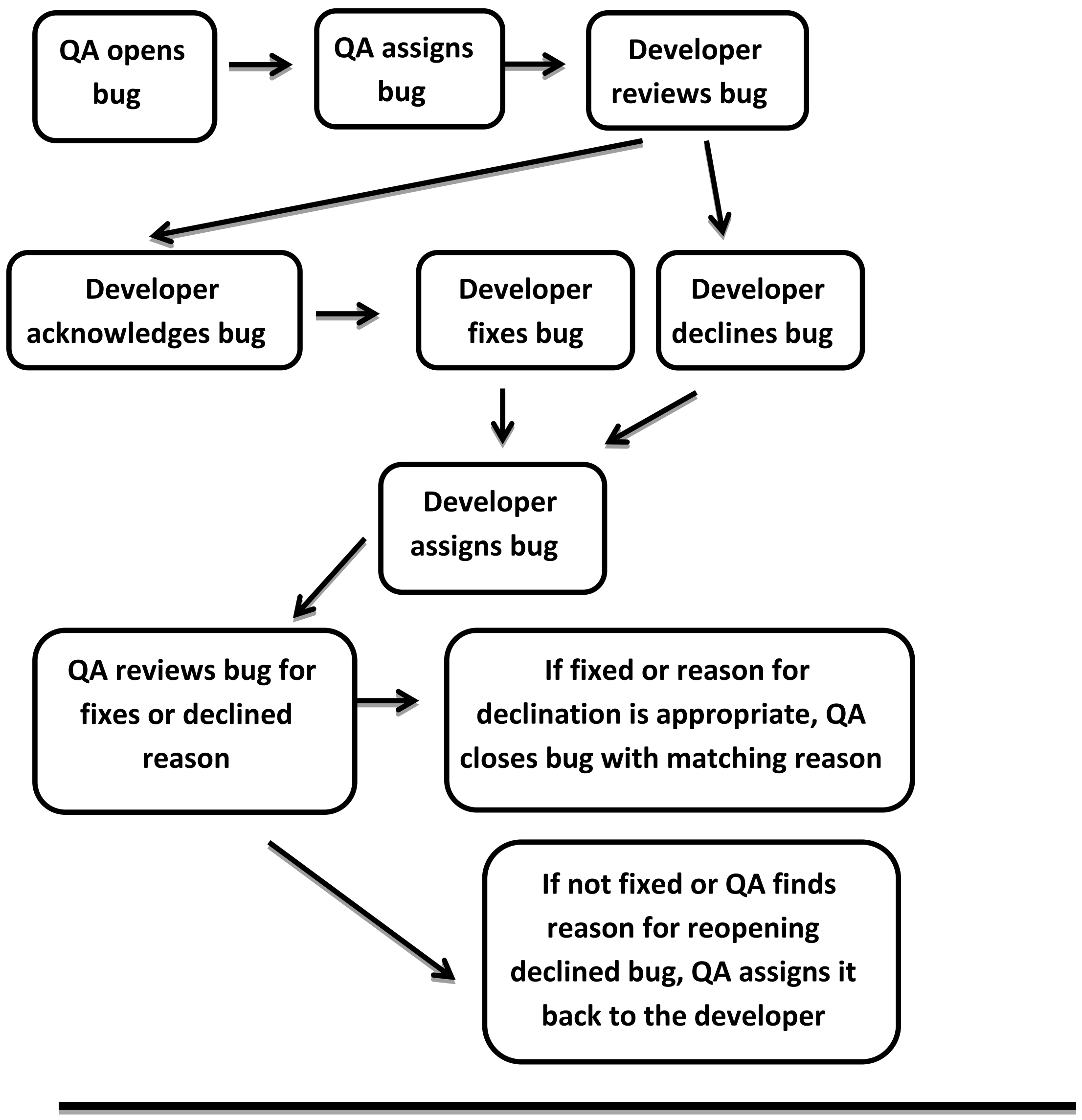
QA opens bug
QA assigns bug
Developer reviews bug
Developer acknowledges bug
Developer fixes bug
Developer declines bug
Developer assigns bug
QA reviews bug for fixes or declined reason
If fixed or reason for declination is appropriate, QA closes bug with matching reason
If not fixed or QA finds reason for reopening declined bug, QA assigns it back to the developer

Fig.4. Life cycle of a Bug in Software Life Cycle

Figure 3 discuss the life cycle of the bug and how the bugs are handled for the quality improvement.

- Opens new bug.
- Assigns bug.
- Reviews bug for fixes or declined reason.
- If fixed or reason for declination is appropriate, the testing and QA team closes bug with matching reason.
- If not fixed or testing and QA team finds reason for reopening the declined bug, it is assigned back to the developer

6. Statistical Modeling of Effort Expenditure

Expression of manpower distribution on a software project over time is the main

concern these days. Rayleigh curve play an important role in studying these cases (Figure.1). We can model the curve by

$$\frac{ds}{dt} = 2M\alpha t e^{-\alpha t^2} \qquad (1)$$

Where ‘ds/dt’ is the staff build-up rate, ‘t’ is the time interval between the start of design and product replacement. ‘α’ is the physics of the curve supposed to be constant and *M* is the area under the curve, which represents the total life-cycle effort including maintenance.

By the theoretical definition of productivity in software,

$$P_s = \frac{S_s}{D_e} \qquad (2)$$

Where 'S_s' is the size of the software product and 'D_e' is the development effort.

Productivity in software can be linked to Rayleigh manpower distribution model. Suppose in the Rayleigh model, the top of the curve corresponds to the development time 'd_τ'. Then the area under the curve which is approximately 48% of *M*, represents the development effort D_e. On the basis of the available project date, it is found that more productive projects had an initial slower staff building and the less productive projects had an initial faster staff build-up. Assume 'P_δ' be the difficulty of the projects which corresponds to initial staff build-up of a project. At t=0, the slope of the Rayleigh curve represents 'P_δ' (Figure.5)

Thus, $P_\partial = \dfrac{M}{d_\tau^{\,2}}$ (3)

Which is obtained by taking the derivative of (1) and setting t=0.Equation (3) defines difficulty.

Also, $P_s = \gamma P_\partial^{\,-2/3}$ (4)

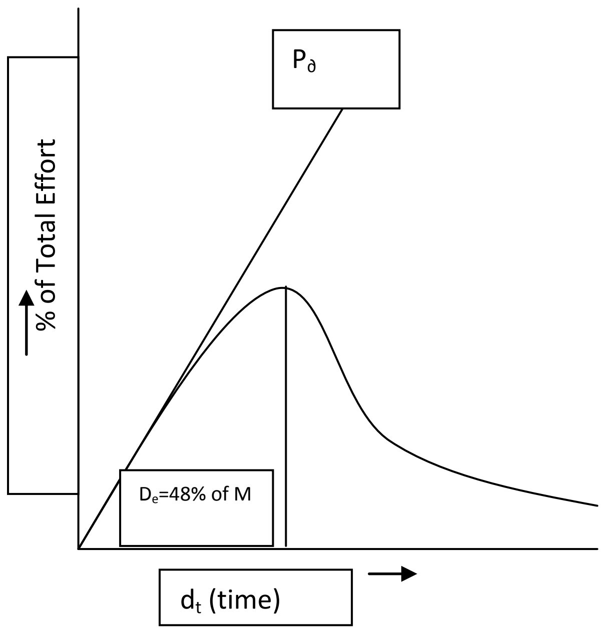

Figure 4 : Effort Expenditure

Equation (4) defines a relation between difficulty and productivity, where γ is proportionality constant. Also, it was defined and assumed earlier.

$$D_e = 48\% of M \qquad (5)$$

Thus from equation (2), (3), (4) and (5) we have,

$$\frac{S_s}{48M} = \gamma \left[\frac{M}{d_\tau^{\,2}} \right]^{-2/3} \qquad (6)$$

Or,

$$S_s = 0.48 \gamma M^{1/3} d_\tau^{\,-4/3} \qquad (7)$$

Thus, the total life-cycle effort is given as

$$M^{1/3} = \frac{S_s}{0.48 \gamma d_t^{\,4/3}} \qquad (8)$$

Let 0.48xr = T_f = Technology factor, which may differences among projects such as programming skills/environment, hardware conditions and individual expertise.

Thus,

$$M = \frac{S_s^{\,3}}{T_f^{\,3} (d_\tau)^4} \qquad (9)$$

Hence,

$$D_e = 0.48 \left(\frac{S_s}{T_f} \right)^3 \left(\frac{1}{d_\tau^{\,4}} \right) \qquad (10)$$

7 CONCLUSIONS

The development effort increases as the cube of the size of the software product, if the schedule remains constant. Also, the effort increases as the inverse of the fourth power of development time for a fixed program size. It is found that a static model of the software development life cycle that treats all modules similarly is becoming inadequate to the task. In this paper efforts are made to quantify the concept of return on investment and factors responsible for improving the return on investment. we are trying to justify the effort expenditure and how to optimize the effort expenditure in the software development environment.

8. LIMITATION

The concept of return on investment is to be verified on the certain number of software development projects so that formulated concept can be supported by the data. The data support of the model is to be implemented.

9. ACKNOWLEDGEMENT

The authors would like to thanks Sohar University and University of Queensland for the research support and environment. Authors would like to thank faculty members and students, especially Prof Lance Bode for his continuous support for the research. We would also like to thank our family members

10. REFRENCES

[1] Software Errors Cost U.S. Economy $59.5 Billion Annually. NIST Assesses Technical Needs of Industry to Improve Software-Testing, June 28, 2002.

[2] DK Saini "Testing polymorphism in object oriented systems for improving software quality" ACM SIGSOFT Software Engineering Notes 34 (2), 1-5, 2009.

[3] Sikka, V.,2003. Maximizing Outsourced Software Quality and ROI: Selecting and managing an Outsourcer, Systems Development Management, Boca Raton, FL: CRC Press.

[4] Zarate, A. et al. 2003.A Portable Natural Language Interface for Diverse Databases Using Ontologies. Computational Linguistics and Intelligent Text Processing, Lecture Notes in Computer Science, Vol. 2588, pp. 494–505, New York: Springer-Verlag, 2003.

[5] Rosario, S. and Robinson, H., 2000.Applying Models in Your Testing Process, Information and Software Technology, Vol. 42, No. 12, September 1, 2000.

[6] Aissi, S., 2002.Test Vector Generation: Current Status and Future Trends. Software Quality Professional, Vol. 4, No. 2, March 2002.

[7] Dinesh Kumar Saini, Lingaraj A. Hadimani and Nirmal Gupta, 2011.Software Testing Approach for Detection and Correction of Design Defects in Object Oriented Software. Journal of Computing, Volume 3, Issue 4, April 2011, ISSN 2151-9617, pp. 44-50.

[8] Dinesh Kumar Saini and Bimal Kumar Mishra, 2007. Design Patterns and their effect on Software Quality. ACCST Research Journal, INDIA .Vol.5, No.1, January 2007, pp.356-365.

[9] Dinesh Kumar Saini and Nirmal Gupta, 2007. 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.

[10] Dinesh Kumar Saini and Nirmal Gupta, 2008. 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. USA.

[11] Dinesh Kumar Saini, 2009. Testing Polymorphism In Object Oriented Systems For Improving Software Quality. ACM SIGSOFT Vol. 34 Number 2 March 2009, ISSN: 0163-5948, USA.

[12] Wail M.Omar, Dinesh K. Saini and Mustafa Hassan. 2010, Credibility Of Digital Content in a Healthcare Collaborative Community. Software Tools and Algorithms for Biological Systems in book series- Advances in Experimental Medicine and Biology, AEMB. Springer, Vol.696, Part 8, pp. 717-24, DOI: 10.1007/978-1-4419-7046-6_73.

[13] Dinesh Kumar Saini, 2011.Sense the Future. Campus. Vol. 1- Issue 11, pp.14-17,February 2011.

[14] Dinesh Kumar Saini and Moinuddin Ahmad, 2011. 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.

[15] Dinesh Kumar Saini and Moinuddin Ahmad, 2011. 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.

[16] Dinesh Kumar Saini, Lingaraj A Hadimani, Poonam V Vaidya and Sanad Al Maskari, 2011. Software Quality Model Six Sigma Initiatives. The 2011 International Conference of Computer Science and Engineering (ICCSE-2011) World Congress in Engineering, July 6-9th 2011, London UK, pp. 1226-1231.

[17] Lingaraj A. Hadimani, Dinesh Kumar Saini, Vaishali P Khoche and Sanad Al Maskari, 2011. Comparison of Software and Hardware Design Tools (CASE vs. Simulators). The 2011 International Conference of Manufacturing Engineering and Engineering Management, (ICMEEM-2011), World Congress in Engineering, July 6-9th, 2011. London, UK.

[18] Dinesh Kumar Saini, Sanad Al Maskari and Lingaraj Hadimani ,2011. 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.

[19] R..E.fairle, "The influence of COCOMO on software engineering education and training",- The Journal of Systems and Software. New York: Vol. 80, Issue. 8; pg. 1201,2007.