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

Business Aspect of Software Reusability

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

In today's saturated process and product market where time, money and productivity is very crucial, software reuse is considered to be one of the most promising approaches for increasing productivity [1]. A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. By re-using existing software, in addition not having to re-implement it, one can avoid downstream costs of maintaining additional code, and if the re-used artifacts has been thoroughly tested and increase the overall quality of the software product. Several industrial and governmental initiatives are underway to increase the reuse of software, involving both adjustments to process, and the adoption of new technologies. Reusability is not always fruitful because some time reusability requires more effort than building new so careful study should be carried out when to reuse and when to build. In this paper effort are made to clear financial evidence of the benefits of reuse. This paper involves an exhaustive study on comparison of economic models of software reusability, their benefits and drawbacks.

General Terms

Business, Computer Science, Software Development, Software System, Reusability

Keywords

Software, Reusability, Economic Cost Models, Productivity, and Reliability

1. INTRODUCTION

Software reuse is the process of implementing or updating software systems using existing software assets. Although first reaction may lead you to believe that a "software asset" is simply another term for source code, this is not the case. Software assets, or components, include all software products, from requirements and proposals, to specifications and designs, to user manuals and test suites. Anything that is produced from a software development effort can potentially be reused [2,3,4, 6, and 8].

An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses. In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects, and ultimately reducing the risk of new projects that are based on repository knowledge [3, 8, 11, and 14].

Software component reuse does not just indicate the reuse of application code. It is possible to reuse specification and designs. The potential gains from reusing abstract product of development process such as specifications may be greater than those from reusing code components .Application system reuse, subsystem reuse, module or object reuse and function reuse are number of levels in which the software is divided. Sub-system and module reuse are less usable [10].

Reusable program involves more overhead in accordance while designing a new one time system. The cost shall involve related technical, organizational, process, tools and associated training for the people of the organization [13, 14].

A well-defined procedure, tools and a library should be created and maintained to achieve good quality and productivity of the system that is under development. Asset management tools like designs, architectures etc are required in full scale development that will help in the integration and speed up modifications, maintenance [9].

The domain area specialists shall decompose the domain into smaller partitions, these partitions can be developed independently and can be used for future changes [4].In order to understand the concept of long term benefits of productivity and reusability, it is necessary for the existing staff to be motivated for the importance of the same [13].

In most engineering disciplines the developed process is based on components reuse. Software system design usually consider that all component to be designed especially for the system being developed. There is no common base apart from libraries such as windows system libraries of reusable software components. By using widespread and systematic software reuse, demands for lower software design and maintenance costs, along with increased quality can be met [20, 24].

2. SOFTWARE RELIABILITY

It is impossible to achieve absolute reliability specification but reusable components may have an associated quality explanation [15]. Software development with reuse is an approach which tries to maximize the reuse of existing software components [11]. Benefit of this approach is that overall development costs of the software are decreased. Cost reduction is only one potential benefit of software reuse. Systematic reuse in the development offers further advantages:

Due to repeated use and test, high quality product are produced. Every successful reuse of an asset increases it reliability level, increases its usefulness in the reuse repository, and decreases the risk of failure [18].If we use a function which is already exists, there is less uncertainty in the cost of reusing that component than in the costs of development. Instead of doing the same work on different project environment, the application specialists can develop reusable components which encapsulate their knowledge [17].Software system or product, that time is minimal in comparison to development time for a new module [16]. Reusing software components speeds up system production because both development and validation time should be reduced [14].

3. OBJECT ORINATATION

One of the reasons that object-oriented programming is becoming more popular is that software reuse is becoming more important [21,23]. It is observed that perfective maintenance accounts for 60 percent of all maintenance, while adaptive and corrective maintenance each account for about 20 percent of maintenance. Since 60% of maintenance activity is perfective, an evolutionary phase is an important part of the lifecycle of a successful software product [20].

Object-oriented programming languages encourage software reuse in a number of ways. Class definitions provide modularity and information hiding. Late-binding of procedure calls means that objects require less information about each other, so objects need only to have the right protocol [22].

A polymorphic procedure is easier to reuse than one that is not polymorphic, because it will work with a wider range of arguments. Class inheritance permits a class to be reused in a modified form by making subclasses from it. Class inheritance also helps form the families of standard protocols that are so important for reuse. These features are also useful during maintenance. Modularity makes it easier to understand the effect of changes to a program. Polymorphism reduces the number of procedures, and thus the size of the program that has to be understood by the maintainer. Class inheritance permits a new version of a program to be built without affecting the old [22].

4. SOFTWARE REUSABILITY

Software reuse is the use of existing software in the development of new software. Two types of decisions are involved in software reuse. The first is whether to acquire the software to reuse or not. In fact, this decision is unnecessary if the software to be reused is already possessed as a result of some other activity (for example, code that is cut-and-pasted is usually not developed with its later reuse in the mind).

The second decision is whether to reuse the software in particular instances or not. Because the reuse process involves finding the software, understanding how to reuse it, and perhaps modifying it before it is actually reused, it can be more attractive to redevelop[19,22]. In economics, software reuse is an investment.

Acquiring reusable software is an initial cost. The act of reusing the software should only go ahead if the cost of reusing is less than it would cost to create the software afresh. Economic models of reuse can help make decisions concerning reuse investment. Their main use is to present the estimated net benefits of a potential reuse investment, but because reuse savings can be difficult to determine even after reuse has taken place, another use of economic models is to estimate the net benefit due to reuse after the event[8,11].

5. SOFTWARE REUABLE MODEL

The assistance provided by reuse models is twofold:

1) They enumerate costs and benefits.

2) They break down some of these costs and benefits into combination of parameters for which values are more easily obtained.

DB = $\sum_{s=1}^{\#system}$ [(average Normal Code unit Cost – average Reused Code unit Cost)$\times$ # Reused Code units] . (1)

There is non-linear Relationship between system size and system costs.

Reused Cost = Normal Cost $\times$ (1-RCR). (2)

(RCR) = Relative cost of Reuse.

Reuse with modification

DB = (Normal Cost – Reused Cost unmodified) + (Normal Cost - Reused Cost modified). (3)

Reused Cost modified = DB (Reused Cost modified)

DB = Normal Cost – (Reused Cost - modification unmodified + modification cost). (4)

5.1 Reusable Model

Things that are reused are code units and components reused If reuse is not component based, the part which has been reused rather than developed can be considered one component.

DB=

$$\sum_{s=1}^{\#systems}\sum_{c=1}^{\#components}\sum_{r=1}^{\#reuses,c}\sum_{l=1}^{\#codeunits,c}\left(NormalCost_{s,c,r,l} - \text{Re}usedCosts,c,r,l\right)$$

(5)

Instead of summing we can take averaging for component reuses. $\sum_{i=1}^{n} x_i = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)\times n$ = average x $\times$n.DB =

$\sum_{c=1}^{\#components}$ [(average normal cost ^{c} - average reused cost ^{c}) # reuses ^{c}]. (6)

Average normal costs =

$$\frac{1}{\#components}\left(\sum_{c=1}^{\#components} av.normal\cos t_c\right).$$

(7)

Reused cost = F ^{b} + U ^{b} +I ^{b} +N ^{b} +P ^{b} +O ^{b} +F ^{w} + U ^{w} +M ^{w} +I ^{w} +N ^{w} +P ^{w} +O ^{w} . (8)

Where b = black box reuse (without modification).

w = white box reuse (with modification).

F = cost to find reusable software (location cost).

U = cost to understand the reusable software.

I = cost to integrate reusable software.

M = cost to modify the reusable software (white box only).

N = cost to develop new software if the reuse attempt fails.

P = an incentive payment to the reusable software producer.

O = other reused costs not mentioned above.

Development cost DC =

$$\sum_{c=1}^{\#components}\sum_{i=1}^{\#codeunits}\left(Normal\cos t_{c,l} - \mathrm{Re}used\cos ts_{c,l}\right) . \quad (9)$$

The cost of maintenance with reuse us the same as that without reuse. High quality reusable software results in consumer benefits. Quality can increase directly through extra testing by the producer and also indirectly through feedback (bug report) from the consumers of the software

MB =

$$\sum_{s=1}^{\#system}\sum_{c=1}^{\#component}\sum_{r=1}^{\#reuse}\sum_{l=1}^{\#codeunits}\left(Costswithout\,\mathrm{Re}use_{s,c,r,l} - Costwith\,\mathrm{Re}use_{s,c,r,l}\right). \quad (10)$$

$$\mathrm{MC} = \sum_{c=1}^{\#components}\sum_{l=1}^{\#codeunits} \text{Producer's maintenance Cost}^{c.l} . \quad (11)$$

Software Reusability metrics, models and analysis carried out suggests quite strongly that Reuse Software in the circumstances. where there are economic and financial benefits to be gained and this we can save the clients money and can have better customer Relationship and can compete in the market with competitors[28]. Software industry treats reuse in a financially desirable way. In the software industry, an investment should pay back. In the software industry the accuracy of the results of reusability are directly related to the quality of the data that is fed into the model. Accuracy is nothing but closeness to reality. Results may not give always accurate results so for this sensitivity analysis is done.

NVP Analysis for Software Reusability

$$\mathrm{NVP} = \sum_{y=1}^{n}\frac{CFY}{(1+d)y} . \quad (12)$$

$$\mathrm{PI} = \frac{TotalBenefits}{TotalCost} \Rightarrow \mathrm{PI} = \frac{Costwithout\,\mathrm{Re}use}{Costwith\,\mathrm{Re}use} . \quad (13)$$

Mathematical modeling is proved to be very useful for validation and verification of the software reusability metrics [10, 20].The other benefits of the software metrics are:- Development Benefits, Maintenance, Quantification of the benefits and costing validation, Use of economic models for validation, Economic models of reuse can help in taking decision concerning reuse investment, Economic models tell about financial property of reuse, cost saving and profitability ratio. Reuse metric emphasize on quantity of reuse in a system and value addition through reusability [31,32,34,35]. Software metric for reusability will supply models with values for their parameters.

6. COMPARISION OF MODELS

There are numerous economic models present. They are not discussed in detail here as they are available in many texts. But in this section, an effort has been made to differentiate these economic models in a tabular form on the basis of parameters they use, benefits and disadvantages [3,4, 7, 8, 17]. Comparison of models Table I. is given below:

7. CONCLUSION

As the saying goes, "no pain, no gain," and the reuse of software is no exception. The product line approach to software reuse requires substantial upfront investment with substantial, but not immediate, benefits. Much commitment, planning, and effort are required to begin a reuse program. Reuse processes and procedures must be incorporated into the existing software development process. Repositories of software assets must be created and maintained. Despite the initial overhead, there are high benefits to software reuse, if appropriate processes are invoked and the requisite planning takes place. Product quality and reliability can increase. Project development time can decrease, along with associated project costs.

Models	Parameters	Benefits	Disadvantages
1. Schimsky	Avg. normal code unit cost, avg. reused code unit cost, no. of reused code units, avg. new reusable code unit cost, library overhead.	i) Most simplest model. ii) Library related cost is also included as library overhead in development cost.	i) Mainly concentrated with code price and units. ii) Maintenance Benefits and Maintenance costs are not included.
2. Gaffney and Durek	Cost of development with reuse relative to without reuse, proportion of reused code, relative cost of incorporating reused code, relative cost of creating reusable code, no. of	i) Perhaps the best known model.	i) Maintenance Benefit and Maintenance cost

	uses.	ii) Takes into consideration no. of systems and no. of reuses.	are not included. ii) Assumes that code is reused in each system and all code written for reuse is actually used.
3. Gaffney and Cruickshank	Unit cost of the system, unit cost of reusing code, unit cost of new code developed, unit cost of creation of reusable code, total size of system(in code units), amount. of new code developed, amount. of reused code incorporated, available reuse functionality, expected no. of systems.	i) Generalization of model 2. ii) Does not imply that same code is used in each system. iii) Costs are shared equally by all the systems.	i) Maintenance Benefit and Maintenance costs are not included.
4. Raymond and Hollis	Avg. normal unit cost, avg. modified unit cost, no of modified code units (in each system), new reusable software cost, no. of systems.	i) General form of ROI (Return-On-Investment). ii) Reuse without modification is free.	i) Maintenance Benefit and Maintenance cost are not included.
5. Poulin and Caruso	Avg. normal code unit cost, RCR, avg. cost per error, avg. no. of errors per code unit, no. of reused code units, COTS, startup costs and overhead, RCWR.	i) Maintenance benefit is included. ii) Startup costs, overhead and COTS are taken into account.	i) Maintenance cost is not included. ii) Assumes that investment payback within a year.
6. Poulin	Avg. normal code unit cost, RCR, no. of reused code units, avg. cost per error, avg. no. of errors per code unit, RCWR	i) Both MB and MC are included ii) Takes into account the full cost of creating reusable software.	i) Startup costs, overhead and COTS are not included. ii) Assumes that all reusable code is maintained even if it is not used.
7. COCOMO	No. of person months, delivered source code instructions, amt. of design modified, amt. of code modified, integration required, avg. normal code unit cost, RCR, no. of modified code units.	i) One of the best software cost-estimation model. ii) Also gives weight age to the design modified (not only the code part).	i) Reuse without modification is not considered. ii) α, β are empirical constants. iii) Maintenance Benefit and Maintenance cost are not included.
8. COCOMO II	Avg. normal code unit cost, RCR, no. of reused code units, assessment and assimilation, software understanding, unfamiliarity, amt. of design modified, amt. of code modified, integration required, effort modifier.	i) Update on the earlier version. ii) Tries to determine how easily understood is the reused software. Treatment of RCR is a improvement over COCOMO.	i) The effort modifier is less accurate than RCWR. ii) Maintenance Benefit and Maintenance cost are not included.
9. Balda and Gustafson	Avg. normal code unit cost, avg. reused code unit cost, no. of reused code units, avg. modified code unit cost, no. of modified code units, avg. new reusable code unit cost.	i) Calculates the effort to develop new software. ii) separate α term for modify	i) α's and β are empirical constants. ii) cost are not included.
10. Defense Information Systems Agency	Avg. normal code unit cost, avg. reused code unit cost, no. of reused code units, avg. modified code unit cost, no. of modified code units, COTS, avg. new reusable code unit cost, no. of reusable code units.	i) Same as Model 9 but β term is omitted. ii) COTS is included in	i) Maintenance Benefit and Maintenance cost are not included.

		Development cost.	
11. Malan and Wentzel	Normal cost, reused cost, consumer's upgrade development cost, consumer's upgrade integration cost, profit increase, new reusable software cost, producer's upgrade development cost, startup costs and overhead.	i) Most sophisticated model of reuse. ii) All the costs are discounted individually. iii) All costs and benefits are included.	i) Assumes that every upgrade is appropriate for each system and so must be included in each system.
12. Frazier	Normal Cost, reused cost, no. of systems, COTS, overhead, additional reusability cost.	i) A straightforward and simple model. ii) COTS and overhead are included in the Development cost.	i) Maintenance Benefit and Maintenance cost are not included.
13. Bowes (Model B)	Normal Cost, reused cost, no. of systems, overhead, new reusable cost, and additional reusability cost.	i) MB is included. ii) Compares the total system costs with and without reuse. iii) Overhead contribution can for different for each system.	i) Maintenance cost is not included.
14. Henderson-Sellers	No. of modified classes, normal cost, avg. location cost per component, avg. modification cost per component, no. of reused components, no. of systems, no. of components, additional reusability cost.	Model is good for object-oriented systems. ii) Cost to modify is calculated and used for each single component.	i) Maintenance Benefit and Maintenance cost are not included.
15. Kang and Levy	No. of components, avg. normal cost, avg. reused cost, no. of reuses, additional reusability cost.	i) First Model to consider components (modules) and sum over them.	i) Maintenance Benefit and Maintenance cost are not included. ii) Assumes that producer is a consumer components, so only cost of additional reusability is counted.
16. Mayobre	No. of components, avg. normal cost, avg. location cost, avg. modification cost, no. of reuses, COTS, library overhead, RCWR, maintenance cost.	i) Finds cost as a summation and also considers components. ii) Overhead, COTS and Maintenance cost are included.	i) Maintenance benefit is not included. ii) There is some ambiguity in MC(it can be a cost to producer without benefiting the consumer that reuses it)
17. NATO	No. of components, avg. reused cost, no. of reuses, library overhead, new reusable software cost.	i) All benefits and costs are per component. ii) First Model to include DCF analysis. iii) Cost parameters are adjusted by the cost of overcoming risks.	i) Maintenance Benefit and Maintenance cost are not included. ii) Inclusion of COTS is ambiguous.
18. Bowes	No. of components, avg. normal cost, avg. reused cost, producer's incentive, no. of reuses, avg. new reusable	i) It considers that many components can be reused.	i) Maintenance benefit is not

	software cost, library overhead.	ii) Maintenance cost is included.	included. ii) Maintenance cost is actually library overhead.
19. Margono and Rhoads	Avg. normal unit cost, no. of components, RCR, no. of reused code units, no. of component reuses, RCWR, no. of reusable code units.	i) Sums entirely over all the components.	i) Maintenance Benefit and Maintenance cost are not included.
20. Bott	Avg. normal component cost, RCR, no. of reuses, maintenance cost, startup costs and overhead, RCWR, no. of components.	i) Maintenance benefit is included and calculated with the help of maintenance constant. ii) No. of components is also included.	i) Maintenance cost is not included. ii) There is a problem with averaging.
21. Lim	Development Benefit, Maintenance Benefit, avoided cost, profit increase, Development Cost, COTS, Maintenance Cost, startup cost and overhead.	i) Maintenance Benefit and Maintenance cost are included. ii) All costs are discounted per year.	i) There is no specification on how to calculate these parameters.
22. Reifer	Development Benefit, Maintenance Benefit, profit increase, Development Cost, COTS, Maintenance Cost, startup cost and overhead.	i) Similar to Lim's Model. ii) Maintenance Benefit and Maintenance cost are included.	i) There is no specification on how to calculate these parameters.
23. Bollinger and Pfleeger	No. of development activities, activity cost with reuse, activity cost without reuse, the reuse investment.	i) This model concentrates on reuse process rather than products. ii) It tries to enumerate the number of activities.	i) Maintenance Benefit and Maintenance cost are not included. ii) Enumeration of activities is somewhat ambiguous.

Table 1: Comparison of Various Software Reusability Models

8. FUTURE RESEARCH WORK

The study carried out in the paper will help the software architectures and designers to decide whether to go for reuse or built the new. This study will help in working with complex software systems that need reusable components in certain places in the software system. The reusability concept is not exploited in the many areas of software development like in testing and maintenance so future research direction should how to use reusable component in software testing.

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. REFERENCES

[1]. DK Saini "Sense the Future" Campus 1 (11), 2011.

[3] G.W. arnold and M..C.Floyd,"Reengineering the New Product Introduction Process,"AT&T Technical Journal, Vol. 71, November/December 1992.

[4] B. Balfour, S. Adams, and D.M. Wade, "Developing Software for Large-Scale Reuse," ACM SIGPLAN Notices, Vol. 28, No. 10, October 1993.

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

[6] B. Barnes and T.Bollinger,"Making Reuse Cost-Effective," IEEE Software, Vol. 8, No. 1, January 1991.

[7] R. P. Beck, S. R. Desai, and D.R. Ryan, "Architectures for Large-Scale Reuse," AT&T Technical Journal, Vol. 71, November/December 1992.

[8] D. Balda and D. A. Gustafson, "Cost Estimation Models for Reuse and Prototype SW Development Life-Cycles," Software Engineering Notes, Vol. 15, No. 3, July 1990.

[9] T.J. Biggerstaff, "Design Recovery for Maintenance and Reuse," Computer, Vol. 22, July 1989.

[10] B. Boehm, "Software Risk Management: Principles and Practice," IEEE Software, Vol. 8. No. 1, January 1991.

[11] T. B. Bollinger and S. L. Pfleeger, "Economics of Reuse: Issues and Alternatives," Information and Software Technology, Vol. 32, No. 10, December 1990.

[12] Y.-T.Chen, I. Bayraktar, and M. M. Tanik, "Techniques of Software Reuse in Design and Specification," Control and Dynamic Systems; Advances in Theory, Vol. 61, No. 2, 1994.

[13].DK Saini, LA Hadimani, N Gupta "Software Testing Approach for Detection and Correction of Design Defects in Object Oriented Software" Journal of Computing 3 (4), Page No.44-50, 2011.

[14] D. J. Chen and D.T. K. Chen, "An Experimental Study of Using Reusable Software Design Frameworks to Achieve Software Reuse," Journal of Object-Oriented Programming, Vol. 7, No. 2, May 1994.

[15] R.T.Due, "The Economics of Reuse," Information Systems Management, Vol. 12, No. 1, Winter 1995.

[16] K. Franchi and R. A. Fleck, Jr., "Ergonomic Improvements in the Office Environment," Business Horizons, Vol. 37, No. 2, March 1994.

[17] J. E. Gaffney and T. A. Durek, "Software Reuse-Key to Enhance Productivity: Some Quantitative Models," Information and Software Technology, Vol 31, No. 5, June 1989.

[18] P. A. V. Hall, "Overview of Reverse Engineering and Reuse Research," Information and Software Technology, Vol. 34, No. 4, April 1992.

[19] E.Henry and B. Faller, "Large-Scale Industrial Reuse to Reduce Cost and Cycle Time," IEEE Software, Vol. 12, No. 5, September 1995.

[20] N. A. M. Maiden, "Saving Reuse from the Noose: Reuse of Analogous Specifications through Human Involvement in Reuse Process," Information and Software Technology, Vol. 33, December 1991.

[21] J. S. Poulin, J. M. Caruso, and D. R. Hancock, "The Business Case for Software Reuse," IBM Systems Journal, Vol. 32, No. 4, 1993.

[22]G.Stark, R. C. Durst, and C. W. Vowell, "Using Metrics in Management Decision Making," Computer, September 1994, pp. 42-48.

[23] V. Seppanen, "Acquisition, Organization and Reuse of Software Design Knowledge," Software Engineering Journal, Vol. 7, No. 4, July 1992.

[24] WEILI, "An Empirical Study of Software Reuse in Reconstructive Maintenance," Software Maintenance: Research and Practice, VOL. 9, 69–83 (1997).

[25] J. van Gurp and J. Bosch," Design, implementation and evolution of object oriented frameworks: concepts and guidelines," SOFTWARE—PRACTICE AND EXPERIENCE, Softw. Pract. Exper. 277-300, 2001.

[26] V. Khusidman, D. M. Bridgeland: "A Classification Framework for Software Reuse," in Journal of Object Technology, vol. 5, no. 6, July -, pp. 43-61,2006.

[27] P. Lahire, L. Quintian: "New Perspective to Improve Reusability in Object-Oriented Languages," in Journal of Object Technology, vol. 5, no. 1, January–, pages 117–138, 2006.

[28] 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.

[29]. H Saini, DK Saini "Malicious Object dynamics in the presence of Anti Malicious Software" European Journal of Scientific Research ISSN, 491-499, 2005.

[30]. DK Saini, JH Yousif, WM Omar "Enhanced inquiry method for malicious object identification" ACM SIGSOFT Software Engineering Notes 34 (3), 1-5, 2009.

[31]. LS Prakash, DK Saini, NS Kutti " Integrating EduLearn learning content management system (LCMS) with cooperating learning object repositories (LORs) in a peer to peer (P2P) architectural framework" ACM SIGSOFT Software Engineering Notes 34 (3), 1-7, 2009.

[32].Bimal Kumar Mishra and Dinesh Kumar Saini "Mathematical Models on Computer viruses" Elsevier International Journal of Applied Mathematics and Computation, Volume 187, Issue 2, 15 April 2007, Pages 929-936.

[33].Bimal Kumar Mishra and Dinesh Kumar Saini "SEIRS epidemic model of transmission of malicious objects in computer network" Elsevier International Journal of Applied Mathematics and Computation, Volume 188, Issue 2, 15 May 2007, Pages 1476-1482.

[34].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.

[35].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

[36].Dinesh Kumar Saini and Hemraj Saini "Proactive Cyber Defense and Reconfigurable Framework for Cyber Security" International Review on computer and Software (IRCOS) Vol.2. No.2. March 2007, pages 89-98.