Reusability and Comparison of Economic Models in Software Development Environment

Dinesh Kumar Saini
Faculty of Computing and IT, Sohar University, Oman
Research Fellow University of Queensland, Australia
dinesh@soharuni.edu.om
Nebras Altar
Faculty of Computing and IT, Sohar University, Oman
nebras@soharuni.edu.om
Samir K Amin
Faculty of Computing and IT, Sohar University, Oman
samir@soharuni.edu.om

ABSTRACT

Software development is tedious and complicated process. The process and product market in the software development costs lot of time, money and productivity is very crucial. Software development is following object oriented approach where software components can be reused. Software components that can be reused will effects the productivity of the development process. Software reuse is promising approaches for increasing productivity. Productivity, quality and reliability these are the main criteria in evaluating the software which get improved via reusability. Software reusability reduces the development cost, testing cost and implementation cost. Cost component in downstream process get reduced via reusability and quality of the software product and process get improved. Reusability involves adjustments to process, and the adoption of new technologies which may emerge after the existing software. Reusability is very critical decision in the software because some time reusability requires more effort than building new product or process. It is very critical decision to make when to reuse and when to build. In this paper effort are made to identify the 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

Computer Science, Software Systems, Software Development.

Keywords

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

1. INTRODUCTION

Software reuse is very systematics approach in the software development process which includes using existing software assets and components. Software development process starts with architecture selection, design, code, test and implementation. While taking a decision for software component reuse it starts with architecture selection. Most of the software architecture is like COM, DCOM, CORBA, MVC etc.

Software development includes assets, or components that can be reused that can result in good product. Reusability decision starts from requirement specification, planning and proposal preparation including design components. Anything that is produced from a software development effort can potentially be reused [1, 2, 6, 8].

While taking a decision of software reuse and comparing the economic benefit of reuse, investigation is needed for initial investment and economic benefit in long term. 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 [3,8,11,14].

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].

As 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].

Software engineering is quite young branch of knowledge and it is trillion dollar industry. In this branch of knowledge development supports component reuse like other branches of knowledge.

Software is broadly classified in systems and application software in development process. The reusability can be planned like libraries, applications, or components. By using widespread and systematic software reusability, demands for lower software design and maintenance costs, along with increased quality can be met [20, 24].

Software reliability and quality get improved through reusable components [15]. Software development with reuse is an approach which tries to maximize the reuse of existing software components [11].The reusable components reduce development time and testing time overall development costs of the software development 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 is 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 then it will reduce the efforts needed to write that function in the software component. 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 [12].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].

II. Software Development Environment

Most of the domains are becoming very complex and software development for these domains need bottom up design approach and object oriented design preference. Object orientation makes software reusability popular and preferred in the development environment [21,23].

Perfective maintenance, adaptive maintenance and corrective maintenance get improved while using reusable component. Perfective maintenance accounts for 60% adaptive and corrective maintenance accounts 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].

In bottom up approach of software development object oriented programming language supports reusability. UML is used for fixing the requirement specification and design. Use cases can be reused. Concept of class which is modular have responsibility and to implement responsibility have methods [22].

Polymorphism, encapsulation, abstraction and inheritance make object orientation approach more reusable. Range of argument is supported polymorphic procedures.

Super class and sub class concept with inheritance property permits a class to be reused. Inheritance forms standard protocols which can be reused in the class.

Code size can be maintained and modularity support changes that are needed in reused software component.

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].

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].

III. Software Reusability 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.

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

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

$$\text{Reused Cost} = \text{Normal Cost} \times (1\text{-RCR}). \quad (2)$$

(RCR) = Relative cost of Reuse.

Reuse with modification

$$DB = (\text{Normal Cost} - \text{Reused Cost unmodified}) + (\text{Normal Cost} - \text{Reused Cost modified}). \quad (3)$$

Reused Cost modified = DB (Reused Cost modified)

$$DB = \text{Normal Cost} - (\text{Reused Cost} - \text{modification unmodified} + \text{modification cost}). \quad (4)$$

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.

$$DB = \sum_{s=1}^{\#systems} \sum_{c=1}^{\#components} \sum_{r=1}^{\#reuses,c} \sum_{l=1}^{\#codeunits,c} \left(NormalCost_{s,c,r,l} - \text{Re}usedCosts, c, r, l \right) \quad (5)$$

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 =

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

$$\text{Average normal costs} = \frac{1}{\#components} \left(\sum_{c=1}^{\#components} av.normal\cos t_c \right) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (7)$$

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

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.

$$\text{Development cost DC} = \sum_{c=1}^{\#components} \sum_{i=1}^{\#codeunits} \left(Normal\cos t_{c,l} - \text{Re}used\cos ts_{c,l} \right).$$

(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

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

(10)

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

(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[18]. 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.

iv. NVP Analysis for Software Reusability

$$VP = \sum_{y=1}^{n} \frac{CFY}{(1+d)y} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (12)$$

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

Mathematical modelling 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 [18]. Software metric for reusability will supply models with values for their parameters.

V. Comparison 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:

VI. Conclusion

Software development process is tedious complex and time consuming. Some of the efforts can be minimized using reusable component in the software development process. In this paper we tried to model the software reusability cost and NVP of the reusability. Software reusability needs careful decision because all time reusability will leads to profit is not garneted. There is upfront investment in the product line approach to software development. Reusability of these software components requires investment and immediate, benefits are not certain. 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; it will improve reliability and quality of the software. Project development time can decrease, along with associated project costs. Efforts are made to formulate the software reusability model. Table 1 shows the comparison of the reusability models.

TABLE I COMPARISION OF MODELS

Models	Parameters	Benefits	Disadvantages
1. Schimsky	Avg. normal code unit cost, avg. reused code unit cost, no. of reused code units, avg.	i) Most simplest model.	i) Mainly concentrated with code price and units.

	new reusable code unit cost, library overhead.	ii) Library related cost is also included as library overhead in development cost.	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 uses.	i) Perhaps the best known model. ii) Takes into consideration no. of systems and no. of reuses.	i) Maintenance Benefit and Maintenance cost 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.	i) One of the best software cost-estimation model.	i) Reuse without modification is not considered.

	normal code unit cost, RCR, no. of modified code units.	ii) Also gives weight age to the design modified (not only the code part).	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 Development cost.	i) Maintenance Benefit and Maintenance cost are not included.
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.	i) Maintenance cost is not included.

		iii) Overhead contribution can for different for each system.	
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 software cost, library overhead.	i) It considers that many components can be reused. ii) Maintenance cost is included.	i) Maintenance benefit is not 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.

VII. Acknowledgement

Authors are thankful for Faculty of Computing and Information Technology Sohar University. Authors would like to thank RIC unit of Sohar University for financial support to attend the conference.

V111. References

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

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

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

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

[5] 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.

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

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

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

[9] 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.

[10] 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.

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

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

[13] 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.

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

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

[16] 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.

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

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

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

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

[21] 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.

[22] 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.

[23] 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.

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