ISSN No. 0976-5697

Volume 5, No. 6, July-August 2014

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Software Reuse – Changes in Process and Organizations

Dr. B V Ramana Murthy
Department of CSE,
Jyotishmathi College of Technology and Science,
Shamirpet, Hyderabad, India

Mr. Vuppu Padmakar
Department of CSE
Guru Nanak Institutions Technical Campus
Ibrahimpatnam, Hyderabad, India

Ms. A Vasavi
Department of CSE
Jyotishmathi College of Technology and Science,
Shamirper, Hyderabad, India

Abstract: The transition form no reuse to informal code reuse in which chunks of code are copied adapted slightly, and then incorporated the new systems occurs when developers are familiar with each other's code and trust each other, later on the trend has changed and the programming paradigms' has changed there is evolution of object orientations, which formally " Never rewrite the code but reuse the code" reusability can be achieves by inheritance at the programming level but the context here is at the product level, which can be achieved by using component technologies which is order of the day, component is a readily available software module which can be used directly or can be tailor made according to the specification. For better reusability the company must have vision towards the market to find out which companies they create, the organizational strength must emphasize on domain engineering and application engineering. The company has to maintain reuse manager for the management of these components in a better way. Using component technologies the company can able to save development cost, time and can deliver product with quality.

Keywords: Software Reuse, Domain Engineering, Software Application Engineering, Software Cost.

I. INTRODUCTION

The basic concept of systematic software reuse is simple. Develop systems of components of a reasonable size and reuse them. Then extend the idea of "Component Systems" beyond code alone to requirements, analysis models, design, and test. All the stages of the software development process are subject to "reuse".

Developers can save problem-solving effort all along the development chain.[5] They can minimize redundant work. They can enhance the reliability of their work because each reused component system has already been reviewed and inspected in the course of its original developments. Code components have passed unit and system test elsewhere and often have stood the test of use in the field. By these means developers can reduce development time from years to months or to weeks instead of months.

II. REUSE INVOLVE CONCURRENT PROCESSES

The reuse community has come to understand on the basis of its experience that making systematic reuse effective requires major changes in the way organizations develop software. In the past the software process has focused on developing each application from scratch. At most, individual developers have shared code on an ad hoc basis.

The new way links many application development projects with processes that identify and create reusable assets. To do so, they must overhaul their business and organizational structures. We have come to understand that this significant organizational change can be thought of in terms of business process reengineering. It is rethinking of everything pertaining to software from there stand point of those who ultimately benefit from good software obtained quickly reliably and inexpensively.

Substantial reuse requires, first of all, that reusable assets be identified in terms of a system architecture. Then the assets must be created and appropriately packaged and stocked. Potential users must have confidence in the components integrity, secondly an organization must refashion its systems engineering process so that developer can identify opportunities for reuse and work selected components into the process.

Systematic software reuse is thus the purposeful creation, management, support, and reuse of assets. As illustrated in figure below this can be expressed in terms of four concurrent processes. We call the people in the reusable asset processes, creators, and those in the development projects, reusers.

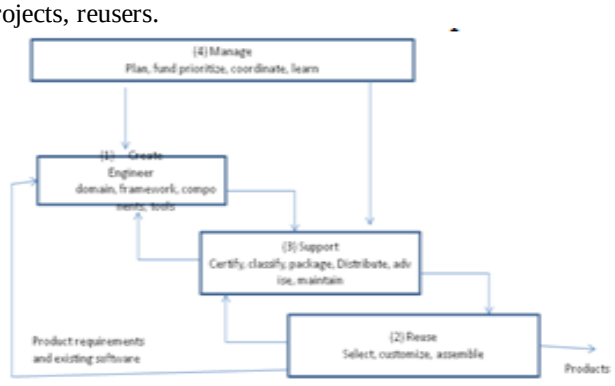

Figure: 1

- ***a. Create:*** This process identifies and provides reusable assets appropriate to the needs of the reusers. These assets may be new, reengineered, or purchased of various kinds such as code, interfaces, architectures, tests, tools and so on. This process may include activities such as inventory and analysis of existing applications and assets, domain analysis, architecture, definition, assessment of reusers needs technology evolution reusable asset testing and packaging.[1]
- ***b. Reuse*** This process uses the reusable assets to produce applications or products. Activities include the examination of domain models and reusable assets, the collection and analysis of end-user needs the design and implementation of additional components adaptation of provided assets, and the construction and testing of complete applications.
- ***c. Support:*** This process supports the overall set of processes and manages and maintains the reusable asset collection. Activities may include the certification of submitted reusable assets. Classification and indexing in some library, announcing and distributing the asset, providing additional documentation, collecting feedback and defect reports from reusers.
- ***d. Manage:*** This process plans, initiates, resources, tracks and coordinates the other processes. Activities include setting priorities and schedules for new asset construction, analyzing the impact and resolving conflict concerning alternative routes when a needed asset is not available, establishing training and setting direction.

A. Domain engineering:

In most reuse programs to date, a key activity associated with the create process is a fairly systematic way of identifying potentially reusable assets, and an architecture to enable their reuse.[2] This activity is called domain engineering in the systematic reuse community. The development of reuse process is also sometimes called application system engineering. The essence of systematic software reuse is that initial investment by the creator to identify and carefully structure reusable assets will enable reusers to build application rapidly and cost effectively.

Domain engineering reflects the idea that sharing between related applications occurs in one or more application domain or problem domain or solution domains. Reuse of the assets then occurs during a subsequent application system engineering phase.

Sometimes domain engineering has been loosely described as just lke ordinary systems engineering such as structure analysis structured design or object oriented analysis object oriented design except that it applies to a family of systems rather than just one.[4] It is like systems engineering but it is also more that one of kind systems engineering. It seeks the family of similar systems that can inhabit a domain. As a result domain engineering is more complex that established systems engineering. Therefore management should not turn to it without forethought and should establish domain engineering only when it foresees a business benefit in reuse.

B. Application System Engineering:

This activity has long existed in the form of building applications from scratch, possibly with the aid of a few back pocket programs. The goal now is to make use of the extensive set of reusable assets that have been provided. The intent is to build the application much more rapidly and cost effectively.[3]

Application system engineering specializes and assembles these components into application. These applications are largely constrained to fit the architecture and the components. Typical applications usually consist of components from several different sets of components.

Starting from the models of the architecture and reusable components, the reusers puts together available reusable assets to meet at least the bulk of the new set of requirements. This is sometimes called a delta implementation because it is an outgrowth of what already exists.

The reusers have to find and specialize components by exploiting a variability mechanisms provided. If it is not possible to meet all the new requirements with the available reusable components additional programming will be needed. This programming may be done by the creator, producing new reusable components or by the reusers.

Finally the components are integrated and the application tested.

Table: 1

Domain engineering	**Application system Engineering**
Define and scope domain	Do delta analysis and design relative to domain model and architecture
Analysis examples needs trends	Use component systems as starting point
Develop domain model and architecture	Find specialize and integrate components
Structure commonality and variability	Exploit variability mechanism language generators.
Engineer reusable component systems languages and tools	

III. REUSE REQUIRES CHANGED IN ORGANIZATION

The traditional software organization was a senior manager over a number of project managers. The senior manager allocated resources, such as people coming off projects that were completing, to projects that were building up. Each project manager ran his or her own project. There was no organized source of reusable components. An organization geared for reuse is different.[4]

A systematic reuse process is different because it involves two primary functions, which usually find expression as two organizations. One is the creator or domain engineering organization. The second is the reuser, or application engineering organization.[8] Companies with experience in systematic use generally find that a third function evolves that of support it in turn finds expression in organizational form as shown fig.

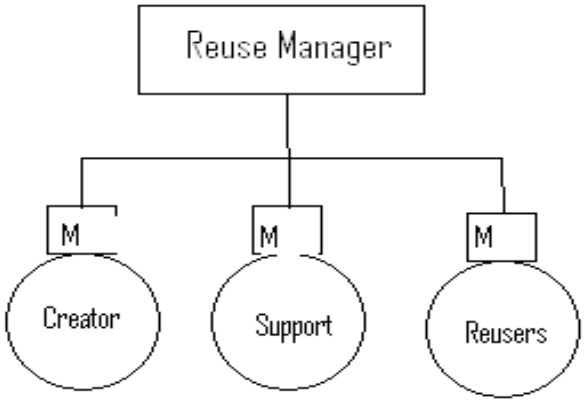

Figure: 2

Experience shows some problems with other organizational structures. For example if creators are put into the project organizations under project managers focused on getting something out the door, that objective often results in delaying, or even forgetting the creator objectives.[7] This pressures explains why setting goals on how much software is to be contributed to a library and establishing financial rewards for reuse have not worked.

However, if creator and reusers functions are totally separated, as happens when a reuse repository is set up in a geographically separate area, the creators tend to be working in a vacuum. That is the reusable components do not meet the practical needs of the reusers or they may appear too late to meet schedule needs.

The creators must be close enough to the reusers to keep reusable components practical. At the same time they must be insulated from daily project pressures if they are to get reusable components designed and built. The result is the three-function organization diagrammed in figure above.

Even with this organization structure, the pressures are still present. The creator and reusers functions have distinct goals. Creators need to build high quality assets that will serve the needs of many reusers over years of product cycles. Reusers have the usual business goals more faster, cheaper for example a project manager facing tight deadlines and a high challenging problem might kill to get a high qualified creator on his or her team. That would interfere with the more long-range goals of the creator organization. There is no right answer to issues such as this one. That is why the diagram shows a senior manager over all three functions. He or she has to adjudicate the interests of creators and resuers. Some organizations have labeled this post the "reuse manager". That title has the advantage of focusing attention on the overall goal.

IV. INCREMENTAL ADOPTION OF REUSE

A company faces two conflicting pressures. On the one hand, it must keep the existing operations going. They are the activities that bring in the funds which keep the company going. Line managers are keenly aware of this need. On the other hand it must keep updating it practices because as it often seems, competition never sleeps. Unfortunately Line managers find it difficult to inaugurate new practice while keeping the old practices in motion.[10] Obviously a successful company has to do both. The adoption path common to several reuse strategies is to start with focused pilot projects. As these pilots meet a degree of success, expand them incrementally, increasing reuse coverage and penetration into the organization. Observation of the introduction of business process reengineering and indeed change management in general further reinforces this stepwise approach.

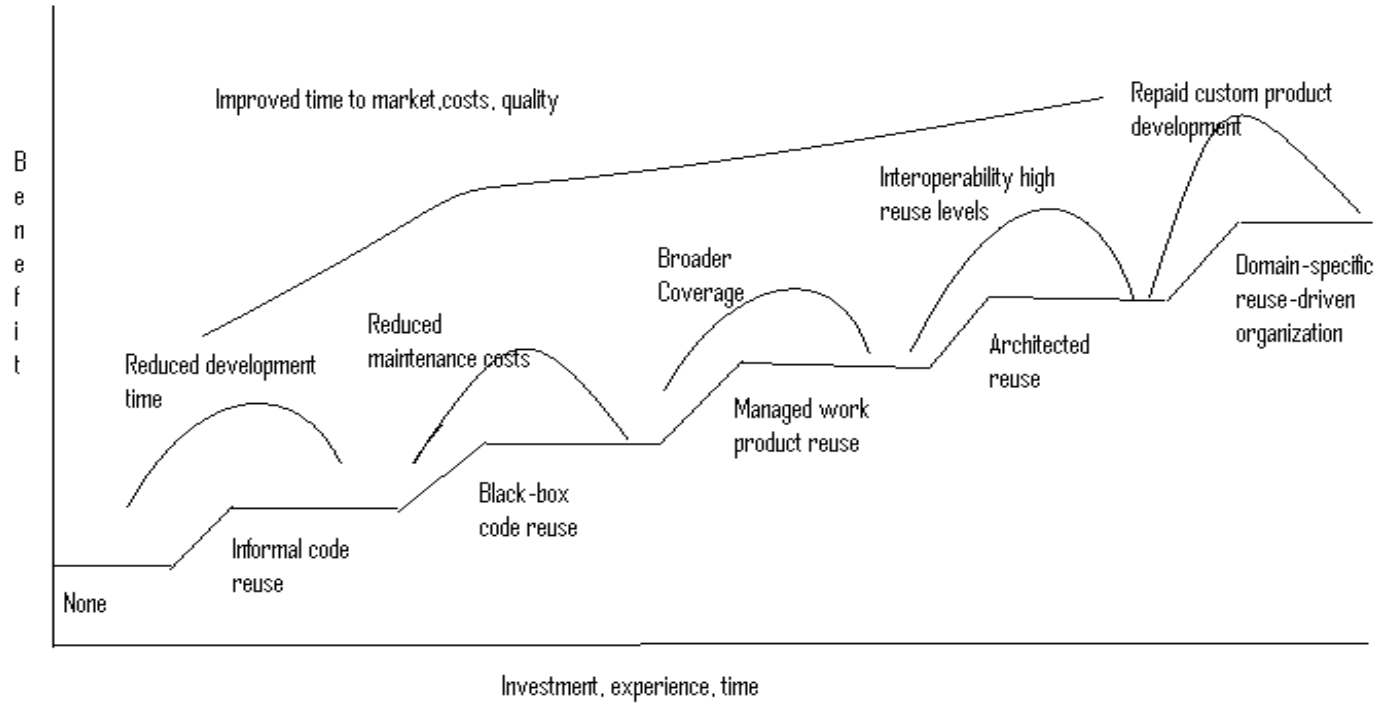

Figure: 3

The above figure illustrates some of the steps that a typical organization may progress through. This figure is based on synthesis of experience at several organizations.

What we observe is that the benefits due to reuse, such as improved time to market (TTM), or higher-quality systems, or lower overall development costs, increase as the levels of reuse and the sophistication of the reuse program

increase.[9] Furthermore it takes time investment and experience with reuse in each organization to get to those levels of reuse. So what seems to happen is that the organization moves from stage to stage as it consolidates it gain and feels the pressure for more improvement.

For example, the transition from no reuse to informal code reuse in which chunks of code are copied adapted slightly, and then incorporated into the new system occurs when developers.

a. Are familiar with each other's code and trust each other
b. Feel the need to reduce time to market, even though they would prefer to rewrite the software.

a) This strategy works – for a while. Development time is reduced, and testing is often less tedious than with totally new code. But as more products are developed using this approach, maintenance problems increase. Multiple copies of the software, each slightly different have to be managed. Defects found in one copy have to be found and fixed multiple times. This often leads to a black box code reuse strategy, in which a carefully chosen instance of code is reengineered, tested and documented for reuse.

b) Measure progress: Management will use a variety of measurements to gauge the progress of reuse programs. Fundamentally, they appear to fall into three categories. [6]

The first category measures levels of reuse within an application area for example each completed system employs 75% of reused components.

A second measures the properties of reusable components for the purpose of assessing their intrinsic reusability. These include measures of size, complexity, cohesion, and coupling to other components.

The third category measures process efficiency and savings in development time and cost.

An survey has taken out, the following are the metrics taken out by discussion from IT Professionals.

Table: 2

SNo	Item	With Reuse	Without Reuse
1	Time to delivery	Reduced by 40%	At delivery time
2	Cost	Reduced by 60%	Total product cost
3	Quality	Increments due to components usage 100%	Depends on the development up to 80%
4	Maintains	Easy & 100%	Has to work due to authentication engineering.

Scaling of reuse effect on product development. The following is the data compiled after having hands on experience on reuse factor of the software product development.

Scaling from 1- 5

Table: 3

Scale	Intensity	Remarks
1	Trivial	Not much impact on product.
2	Small	Less impact up to 10%
3	Medium	Impact on application developer
4	High	Good impact on application developer with respect domain engineering
5	Very High	Recommended to go for reuse

We have taken parameter of the product development and the results as follows.

Table: 4

S.No	Item	Scale
1	Time	4
2	Cost	5
3	Quality	4

Time, cost and quality are the basic principles of software engineering and the above table represents the success of reuse. The graphical representation of the above table

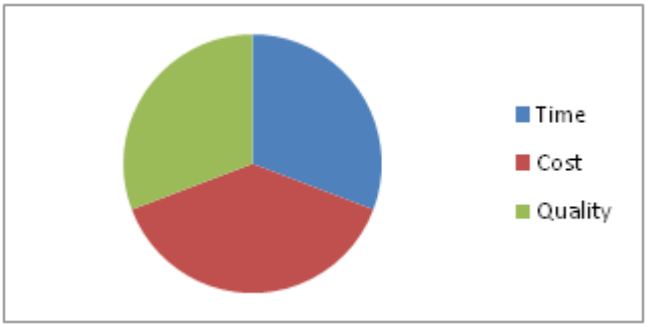

Figure: 4

A graph has been considered to show the effectiveness of reuse in the companies.

Table: 5

Reuse	%
With components	60%
Without components	0

Scaling from 1 – 10

Table: 6

SNo	Item	With Component Technology	Without Component Technology
1	Time to delivery	40%	More 10
2	Cost	60%	Max 10
3	Quality	100%	60%
4	Maintains	easy	40% product development cost.

V. CONCLUSION

a. A component has to be used three to five times in application projects to recover the initial cost of creating it and the ongoing cost of supporting it.
b. It costs 1.5 to 3.0 times as much to create and support a reusable component as it does to implement a similar component for a single application.
c. It costs only one quarter as much to utilize a reusable component as it does to develop a new one from scratch.
d. It takes two or three product cycles, usually about three years, before the benefits of reuse become significant. It takes time for the accumulating benefits to pay off the start-up cost.

VI. ACKNOWLEDGEMENTS

The authors would like to thank Sardar Tavinder Singh, Chairman and Mr. Gagandeep Singh Kohli, Vice Chairman and Dr. H S Saini, Managing Director, Guru Nanak Institutions for their encouragement and support for this research paper work. The authors would like to express their gratitude to the reviewers for their valuable suggestions and comments.

VII. REFERENCE

[1]. Powell,T.A., Website Engineering Prentice Hall,1999.

[2]. Pressman, R. S.,"Can Internet Based Applications be Engineered?" IEEE Software, September 1998, pp. 104-110.

[3]. The Agile Alliiance Home Page, hhtp//www.agilealliance.org/home

[4]. Ambler ,S, "what is Agile Modelling http://www.agilemodeling.com/index.htm.

[5]. Cockburn A.,Agile Software Development: Addison Weley

[6]. Cockburn and J HighSmith ., What is Agile Software Development The People Factor "IEEE computing Vol 34 pp 131-133

[7]. DeMarco ,.T,and T Listener .Peopleware second edition

[8]. DeMarco ,.T and Boehm,"The Agile Methods fray "IEEE Computer Vol 35 pp 90-92.

[9]. HighSmith, J. Agile Software Ecosystem Addision–Wesley.

[10]. Highsmith J., "The Methodology Debate" Part -1 Vol 14.

Short Bio Data for the Authors

Dr. B. V.Ramana Murthy has done his PhD from Osmania University, presently he working as Professor in Computer Science and Engineering, has 18 years of experience in Teaching and R&D. His primary area of interest is Software Engineering & Web Engineering.

Mr. V Padmakar is pursuing PhD in CSE and has done his M Tech (CSE) from JNTUH, presently working as Professor in Computer Science and Engineering has 17 years of experience in Teaching and Industry. His primary area of interests is Software Engineering, Network Security and Data mining

Mrs. A.Vasavi has done her M.Tech (CSE) from JNTUH, presently She is working as Associate Professor in Computer Science and Engineering department, has 10 years of experience in Teaching. Her area of interest is Network Security and Formal Languages.