Software Reuse and Reengineering: With A Case Study

Prabhot Kaur Chahal, Amritpal Singh

ABSTRACT: ***Reuse of existing system has been regarded as a feasible solution to solve the problem of software Productivity and Quality. In this paper, the reference paradigms for setting up of reuse reengineering processes, has been explained. Approaches to reengineering and reuse are also discussed.***

In Product development an important step is to have a clear and correct set of systems requirements. When a product is produced by a variety of models with different set of features, it is desirable to make the requirements Reusable. But this imposes certain restrictions on the Requirements development that are described here.

KEYWORDS: System Requirements, Re-engineering, Reuse, Salvaging, Restructuring

I. INTRODUCTION

When software has been developed our long periods of time, it is often difficult and costly to ensure that it meets the best industry practice and standard at all times. So it is better re-engineer software to improve functionality, efficiency and maintainability of the code. This is a cost effective way of upgrading customers, resources and improving productivity. Existing applications may be adapted or converted to improve their functionality or user image, and to take advantage of current best practice techniques. The result is that the customer's initial investment in the software is preserved fully. And the applications are made more reliable, robust, efficient, and easier to use.

What is software re-engineering?

There is no universally accepted definition of software re-engineering.

1) The IBM user Group GUIDE[1] (GUIDE 1989) defines software re-engineering as "the process of modifying the internal mechanism of a system or program or the data structures of the system or program without its functionality"
2) CHIKOFSKY AND CROSS[2] defines software re-engineering (chikofsky,1990) as: "The examination and alteration of a subject system to reconstitute it in a new form and subsequent implementation of that form"
3) ARNOLD[3] defines software re- engineering (Arnold 1990, Arnold 1993) as: "Any activity that improves one's understanding of software / improves the software itself"

In this definition, the interpretation of "software" is quite broad.

It includes source code, design records, and other sources of documentation.

This definition partitions software re-engineering into two sets of activities.

1) The first set consists of activities supporting program understanding, such as browsing, measurement, and reverse re-engineering
2) The second set includes activities geared towards software evolution, such as re-documentation, re-structuring and re-modularization.

Strategic reengineering refers to the process during which software systems are re-developed in order to meet company's long-term strategic plans. Strategic reengineering lifecycle involves four phases, namely a preliminary phase of business and Information System (IS) planning, reengineering planning, building a reusability framework and reengineering of software. The structure of the strategic reengineering process is shown:

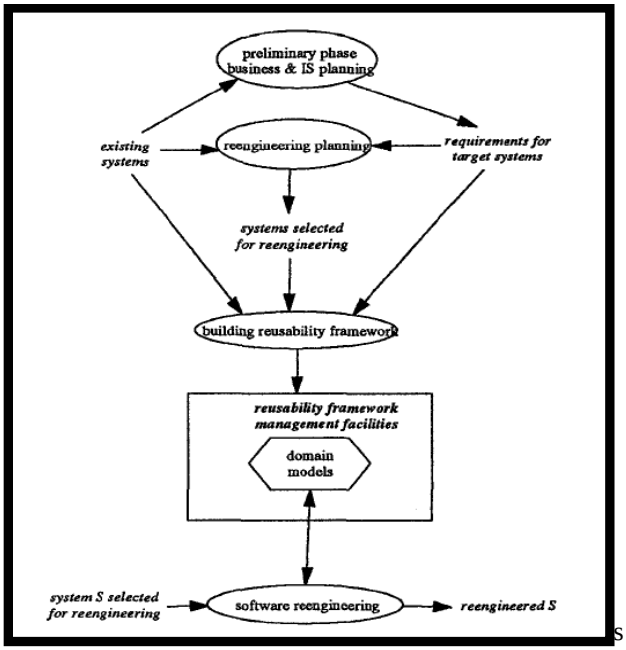

Figure 1: phases of strategic reengineering [4]

Definition of software reuse

Software Reuse can be defined as [5,6]:

1) "the process of creating software systems from existing software systems, rather than building software systems from scratch"
2) "the systematic process of developing software from a stock of building blocks, so that similarities in requirements and/or architecture between applications can be exploited to achieve substantial benefits in productivity, quality and business performance".
3) "the ability to use software routines over again in new applications". This is one of the benefits of other technology.

Manuscript received January 15, 2014.

Prabhjot Kaur Chahal:- Graduated from BCET, Gurdaspur in stream IT in the year 2012 and pursuing my MTech in CSE from GNDU, Amritsar, India.

Amritpal Singh:- Graduated from BCET, Gurdaspur in stream IT in year 2010 and pursuing MTech from BCET, Gurdaspur,India.

Why Reuse Software?

A good software reuse process facilities the increase of productivity, quality, and reliability and the decrease of costs and implementation time. 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, minimising the amount of development work required for future projects and ultimately reducing the risk of new projects that are based on repository knowledge.

Software salvaging

Software salvaging is representation specialty for recovering software assets for reuse. Software salvaging is part of the reengineering.

Software Salvaging Process:

Raw Material → prospecting → transformation → classify

Prospecting means to determine what software parts are worth further attention.

Transformation means modified and certified so that it meets style and quality criteria for insertion into the repository.

Software salvaging refers to a reengineering activity for recovering software assets for reuse. It can be done in the

- large extent i.e allowing the entire system to be reused.
- Small extent i.e obtaining software building blocks for reuse,
- Populating a repository with parts and relationships
- Recovering object oriented objects and classes from non-object-oriented software.

Three Salvaging Approaches:

Domain- Independent Software Salvaging(using software metrics to find redundant code, use of plagiarism detection program, use of McCabe cyclomatic complexity to find control flow reducdancies)

1) Domain-Dependent software salvaging
   (uses information about the software applications or design history to find parts from code)
2) Object Salvaging tries to find OO objects from non-OO code.
3) E.g. creation of " C++" classes and object instances from software written in "C"

II. A PARADIGM FOR REUSE REENGINEERING PROCESSES

The importance of high-level organizational paradigms in setting up software processes (production, evolution, certification) is well known [7]. In our case, this model has the function not only of guiding the creation of new processes (by means of suitable tailoring and instantiation operations) but also and above all of learning from these. In particular, for identification, delimitation, classification of theoretical, methodological and technological problems, for which solutions have not yet been proposed, so as to locate in the process and thus experiment the limitations and qualities of any solutions adopted the paradigm must be useful for the identification, delimitation and classification. The paradigm defined for RE^2 is shown in figure 2.

In order to model the process, the paradigm is characterized in sequential phases, each of which includes a set of homogeneous activities aiming to produce objects that are and therefore characteristic of the process.

The phase are:

- Candidature phase
- Election phase
- Qualification phase
- Classification and Storage phase
- Search and Display phase

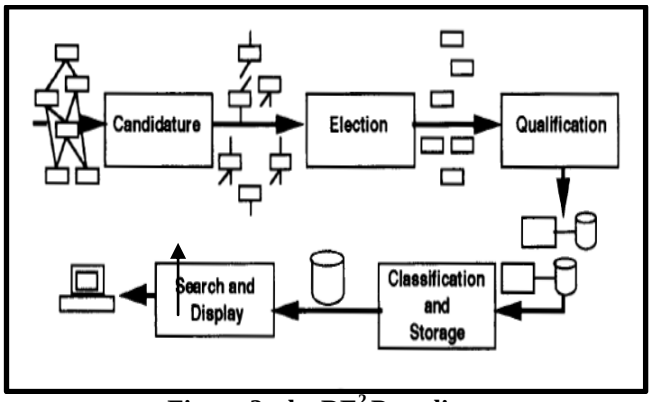

Figure 2: the RE^2 Paradigm

1) The Candidature phase includes the set of activities that start from source code analysis and a set of software components is produced, each making up a module.
2) Election phase includes the set of activities that start from the set of candidates to make up a module and produce a set of reusable modules.
3) The Qualification phase includes the set of activities that produce the functional and interface specifications of each reusable module.
4) The Classification and Storage phase includes the set of activities for classifying the reusable module on the basis a reference taxonomy and for organizing and populating a repository or a system of repositories for the produced modules.
5) The Search and Display phase includes the set of activities concerned with the organization and creation of the "front end user" in order to navigate through the repositories with the help of visual supports.

III. PRINCIPLES OF REENGINEERING AND REUSE [8]

The three views that should be considered when developing requirements and that should also be considered important when the requirements are reengineered

1. **The view of the stakeholders**: the requirements are specified by the stakeholders. Stakeholders are the one who show interest in the on the several possible points. Stakeholders can be customers, consumers, management. One party that show interest, its interest is winded up and the requirements are worked accordingly.
2. **The view of the development team**: the responsibility of the development team is to fulfil the demand of requirements in the form of a product. The requirements are mapped accordingly into the product based on it. The logic of the system behaviour is depicted by the system requirements in total. It is not necessary in most

cases to specify the logical behaviour of the system in the sequences of algorithmic steps. Rather, the system requirements are structured in the break down into modules, tasks, algorithms. It happens that some of the details are left for the implementation but still they can be derived from the system logic behaviour.

3. **The view of the tester**: the testers are the one who checks and maps the views given by the stakeholder and the views given by developer. The requirements given by the stakeholder are moulded in to the product by developer. The requirements verification is done and the input-output relation is tested along with behavioural response.

With consideration for these views, a set of principles or guidelines for reengineering and reusing requirements were developed.

IV. CASE STUDIES

Case studies based on the effects of reuse on the quality, productivity and economics:

Reuse is not the new concept, but it has been used for improving the quality and productivity of the software now a days.

To prove this concept true that reuse increase quality and productivity case studies were performed at Hewlette-Parkard. Hewlette- Parckard [7] found that reuse significantly play a positive effect on the software development. In the case studies he presented two metrices from two HP reuse programs that showed that with the quality improvement the productivity is also increased.

Here, the terms used in this case study are defined as follows:

- Work products are the software- development process products or by-products: like code, design and test plans.
- Reuse is defined as the use of these work products without modification in some other software.
- Leverage reuse is modifying existing work products to meet specific system requirements.
- Producer is the one who creates the reusable products.
- Consumer is the one who uses that reusable product for creating other software.
- Time-to- market is defined as the time taken for a software to get completed and delivered into the market for customers to use it.

Software reuse is not free as it requires resources to create and maintain reusable work products, a reuse library or tools. An economic analysis method has been developed to evaluate the costs and benefits of the reuse, which is also applied to multiple reuse programs at HP.

Case study 1:

The first case study was done in the Manufacturing Productivity section of HP's software technology division. The MP section produces large-application software for manufacturing resource planning. The study was started in 1983.

Originally the motive of this was just to increase the engineering productivity. But the MP section has discovered reuse for maintaining the burden and support product enhancement.

Technical aspect

- MP engineers practiced reuse by using generated code and other work products such as applications and architecture utilities and files.
- The data reported only the use of reusable work-products, not the generated code.
- Total code size for the 685 reusable work products was 55,000 lines of non-comment source statements.
- The reusable work products were written in Pascal and SPL, system programming language for the HP 3000 computer system.
- The development and target operating system was MYPEXL, the multiprogramming Environment.

Case study 2:

- The second case program is within the San Diego Technical Graphics Division, which develops, enhances, and maintains firmware for plotters and printer.
- The STG reuse program began in 1987. Among the program's goals was the same to reduce the development cost and see how the productivity quality gets effected.
- This could be done by reducing duplication and providing consistent functionality across products.
- The reusable work product analyzed here is 20,000 non comment source statements written in C.
- The development operating system was HPUX and the target operating systems were PSOS and an internal one.

Observations of these studies.:

At HP, data was collected from these use rese programs and conducted a REUSE ASSESSMENT[7], which is an analytical and diagnostic method used to evaluate both qualitative and quantitative aspects of a reuse program. And then part of this assessment includes the data on the improved quality,productivity, and economics attributes to reuse is analyzed and documentaed.

The graphs will show the effects of each attribute on the two programs. A comparative study of these to programs along with the attributes is also depicted.

The first attribute is"

- **Code Quality:**

The first thing which come in the mind of every consumer or user is the quality of the product. The quality is the main aspect on which the developer has to focus. Sometimes the imbalance between the attributes of quality, time etc may occur. This may due the reason that to fulfil one attribute's aspect one has to give up the other.

In gerenal if we talk about our day to day life, we shop alot of househol things and the first thing which we try to achieve is the best quality in less price. That same concept is applicable in software world also, the reason is that the same humans are dealing and the same thinking is mostly found.

The graph given below shows the code of quality. The quality of the code, which means the correctness and errorness of the source code for a program.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

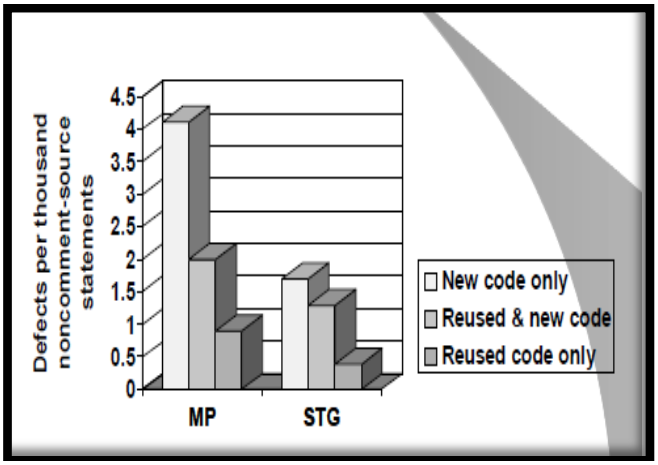

Figure 3: Code Quality Graph

Here the defects per thousand non-comment source statements is calculated of each program. Because quality works products are used multiple times, the effect fixes from each reuse accumulate, resulting in higher quality. More important, reuse provides incentives to prevent and remove defects earlier in the life cycle because the cost of prevention and debugging can be amortized over greater number of uses.

Figure 3 above tells the quality results. The MP section data shows a defect-density rate for reused code of about 0.9 defects per thousand non-comment source statements (KNCSS). Compared to the new code with 4.1 defects/ KNCSS. Using reused code in combination with new code .in which 68% of the product was from reused work products)

Resulted in 2.0 defects/KNCSS, a 51% reduction in defect density compared to new code. If the effects of generated code is taken into account, the achievement is of a total defect-destiny reduction of 76% compared to new code.

The same thing was observed in STG, a positive feedback is observed in case of reuse. They estimated the actual defect-density rate for reused code to be 0.4 defects/KNCSS, compared to 1.7 defects/KNCSS for new code. A product that incorporated the STG reusable work product had a 31% reuse level and a defect-density rate of about 1.3 defects/KNCSS, a 24% reduction in defect-density.

The second attribute is:

- **Code of Productivity**

Reuse improves productivity because the life cycle now require less input to obtain the same output. For example, reuse can reduce labour costs by encouraging specialization in areas such as user interfaces. Because of their experience, specialists usually accomplish tasks more efficiently than non-specialists. Or productivity may increase simply because fewer work products are created from scratch. For example, if the reused work products are already documented and tested, the new products require less work in these areas. A product's maintainability and reliability is improved, thereby reducing maintenance labour costs.

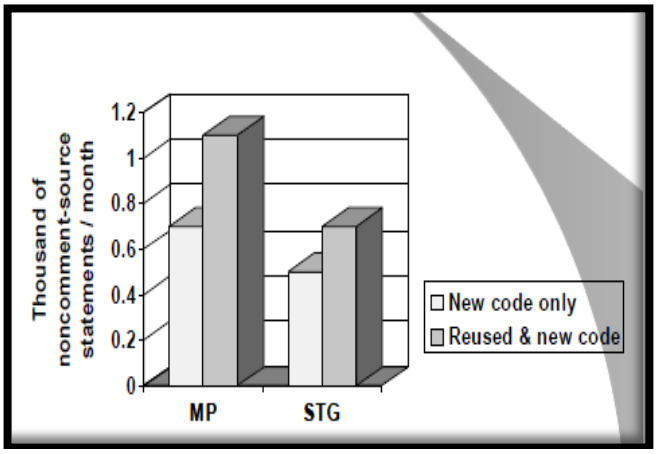

Figure 4: Code of Productivity

Productivity is improved by reuse, by reducing the amount of time and labour needed to develop and maintain a software product. Figure 4 shows another similar project in MP section reported a productivity rate of 0.7 KNCSS/engineering month for new code. Its product , which was composed of 385 reused code, had a productivity rate of 1.1 KNCSS/engineering month, a 57% increase in productivity over development from scratch.

The STG division estimates a productivity rate of 0.5 KNCSS/engineering month for new code. in contrast, its released product comprising 31% reused code had a productivity rate of 0.7 KNCSS/engineering month, a 40% improvement.

Additional effort in creating reusable code in STG[7]

Figure 5 shows percent increase in engineering months by life-cycle phase (except maintenance) in creating a reusable software work product in the STG division.. the data shows that the most significant increase were in the investigation and external design phases. This is because the producer of the work product required a greater amount of time to understand the multiple contexts in which the work product will be reused.

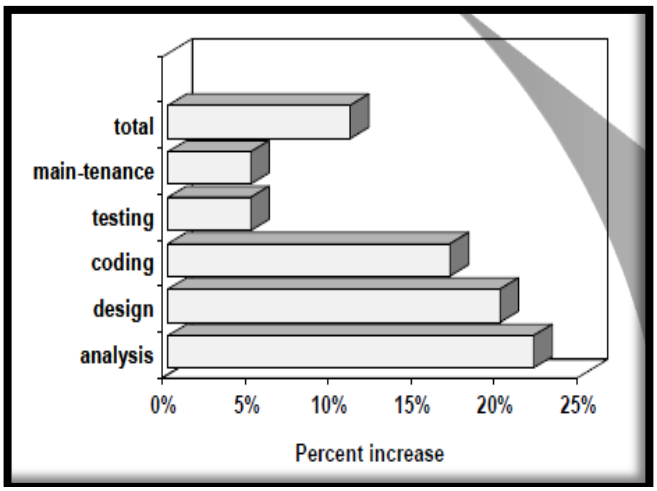

Figure **5: Additional Effort in creating reusable code in STG**

The third attribute is:

- **Reusable case study economics**

Reuse effort if not carefully planned and properly carried out, oftentimes becomes an inhibitor rather than a catalyst to software productivity and quality. Despite numerous articles in the areas of domain analysis, component classification, automated component storage/retrieval tools, reuse metrics, etc., only a handful have managed to address the economics of software reuse. In order to be successful, not only must a reuse program be technically sound, it must also be economically worthwhile. After all, reducing costs and increasing quality were the two main factors that drove software reuse into the software mainstream.

The comparision of the two programs MP & STG in economics attribute is depicted below.

It is based on the various factors like: time horizon, gross cost, gross savings, return on investment

	MP	STG
time horizon	1983-1992	1987-1994
start-up resources required	26 engineering months $0.3 million	107 engineering months $1.4 million
Ongoing resources required	54 engineering months $0.7 million	99 engineering months $1.2 million
Gross cost	80 engineering months $1.0 million	206 engineering months $2.6 million
Gross savings	328 engineering months $4.1 million	446 engineering months $5.6 million
Return on investment	410%	216%
Break even year	2nd year	6th year

Figure 6: Reuse Case Study Economics [8]

III. CONCLUSION:

A software that is reengineered for the bases of reuse, is found to be far more successful as it shows a remarkable diversification in terms of quality, productivity and economics as compared to the new software development.

By applying the principles of the reengineering the requirements for reuse defined herein enable a more effective and efficient evolution of models within a product family during development life cycle. Domain-specific software development offers a comprehensive framework for a reuse and reengineering based on revitalizing the existing systems and resulting with a new more maintainable systems.

REFERENCES

1. CHIKOFSKY, E. and Cross, J. H. (1990) “Reverse Engineering and Design Recovery: A Taxonomy,” IEEE Software, vol. 7, no. 1, pp. 13-17.
2. GUIDE, (1989) Application Reengineering, GUIDE Pub. GPP-208, GUIDE International Corp., Chicago
3. ARNOLD, R. S. (1993). “A Road Map Guide to Software Reengineering Technology,” in Software Reengineering, R. S. Arnold (ed.), IEEE Computer Society Press.
4. Stan Jarzabek, “Strategic Reengineering of Software: Lifecycle Approach”, 1993, IEEE
5. W. Tracz ed. "Tutorial: Software Reuse: Emerging Technology" IEEE Computer Soc. Press. 1988
6. P. Freeman ed. "Tutorial: Software Reusability" Computer Soc. Press, 1987.
7. Engineering, Portici (Naples), Italy, Dec. 1991. V. R. Basili "Viewing Maintenance as Reuse- Oriented Software Development" IEEE Software.
8. Dr. Larisa Melikhova, Albert Elcock, Andrey A. Dovzhikov, Georgii Bulatov, Dr. Dmitry 0. Vavilov, “Reengineering for System Requirements Reuse: Methodology and Use-Case”, 2007, IEEE
9. Wayne C. Lim, “article of managed reuse organisational and economics assessment” 1994, IEEE software
10. Software Engineering Course Given by: Arnon Netzer (ppt)

AUTHORS PROFILE

Prabhjot Kaur Chahal:- Graduated from BCET, Gurdaspur in stream IT in the year 2012 and pursuing my MTech in CSE from GNDU, Amritsar

Amritpal Singh:- Graduated from BCET, Gurdaspur in stream IT in year 2010 and pursuing MTech from BCET, Gurdaspur.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication