Volume 4, Issue 3, March 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Three R's of Software Engineering: Reuse, Reengineering, Reverse Engineering

Vikshant Khanna*
Faculty, Ramgarhia Institutions
Phagwara, India

Parul Mohindru
Faculty, CT Institutions
Jalandhar, India

Abstract— With an initial aim of modernizing legacy systems, often written in old programming languages, software reverse engineering has now extended its applicability to virtually every kind of software system. Existing system's modification or replacement often applies reverse engineering for understanding of that system. Reverse engineering is nowadays used for more than one purpose that is sometimes illegal also. This paper presents an insight on this exciting and swiftly growing technology in the computer world.

Keywords— Software engineering, Reuse, Reengineering, Reverse Engineering.

I. INTRODUCTION

Engineering is the profession involved in designing, manufacturing, constructing, and maintaining of products, systems, and structures. Software Engineering is concerned with putting computer knowledge to practical use. It is application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, i.e. the application of engineering to software. Software is the engine that makes run everyday life, such as in business, industry, administration, research, etc. As enterprise software systems are continually growing in complexity, IT industry is forced to find ways to streamline development process. Software engineering is believed to be one such approach as most effective way to significantly improve the software process, shorten time-to-market, improve software quality and application consistency, and reduce development and maintenance costs.

II. SOFTWARE REUSE

Code reuse is called Software reuse. Software reuse is the process of creating new software systems from existing software components. Reuse has an enormous impact on productivity. The following elements of software reused are software specifications, designs, tests cases, data, prototypes, plans, documentation, frameworks and templates. Reusability is the degree to which the artefacts can be reused. It is ability to use some of parts or the greater part of the same programming code or system design in another application. Reusability is the segment of source code that can be used to add new functionalities with slight or localizes code modifications when a change in implementation is required[2]. Reusability is the extent to which a software component is able to be reused. Reusable modules and classes increase the prior testing reduce implementation time & use has eliminated bugs & no modification. Reusability is the degree to which the artefacts can be reused. A reusable component may be code, but the bigger benefits of reuse come from a broader and higher-level view of what can be reused. Software reuse can cut software development time and costs. 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[8].

Classification of Software Reuse

Transformational vs. compositional reuse. Transformational systems are obtained via transformations of high-level specification of the desired system whereas in the second approach systems are obtained from combining components by the choice of the developers.
Black box vs. white box reuse. In the first approach products are reused as-is whereas in the second approach products can be modified to the specific application.
Abstraction reuse. Reuse applied at the level of requirements, code, design, tests, etc.
Development of reusable assets vs. application reuse.
Vertical vs. horizontal reuse. The former takes place in the same domain for example, financial object models, algorithms, frameworks; the latter is related to the assets which are created for on domain but are reused in different one. Examples of them include GUI objects, database access libraries, authentication service, and network communication libraries.

Procedures reuse. It means reusing skills and know-how. This has received significant attention from the expert-systems community while project managers tend to reuse skills informally when they reassign personnel. To encapsulate knowledge funds are needed[4].

Software Reuse Methods

Software development is divided into stages such as requirements analysis and specification, design, coding, testing and maintenance. To manage difficulties of the development process different models are proposed. Reuse methods can be divided in two groups:

Generative methods. The idea is very similar to automatic programming, however while automatic programming tries to automate the whole process of software development, the generative approach tries either to automate the sequences of transformations of the process development or narrows the application domain.

Compositional methods. It is the most common form of reuse and it is based on reusing components stored in libraries as potential assets for new software developments. One of the most effective ways to significantly improve the software process, shorten time-to-market, improve software quality and application consistency, and reduce development and maintenance costs is the systematic application of software reuse. Software reuse can be opportunistic or ad-hoc and planned[3][8].

Reuse Metrics and Models

According to Frakes, reuse models and metrics are categorized as following as shown in Fig 1:

1. Cost Benefit Analysis
2. Maturity Assessment
3. Amount of Reuse
4. Failure Model Analysis
5. Reusability Assessment
6. Reuse Library Metrics

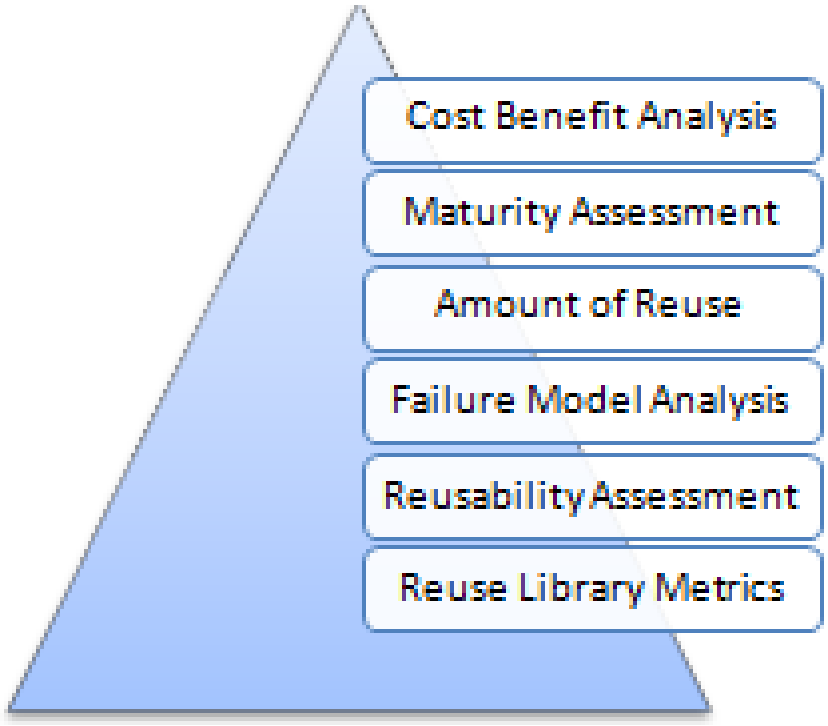

Figure 1: Reuse Metrics and Models

III. SOFTWARE REENGINEERING

Software Re-engineering is the examination, analysis and alteration of an existing software system to reconstitute it in a new form, and the subsequent implementation of the new form. The process typically encompasses a combination of other processes such as reverse engineering, re-documentation, restructuring, translation, and forward engineering. The goal is to understand the existing software (specification, design, implementation) and then to re-implement it to improve the system's functionality, performance or implementation[1].

There are four re-engineering objectives, they are: Preparation for functional enhancement, Improve maintainability, Migration, and Improve reliability. Figure refers to the general model of software re-engineering and illustrates the reverse engineering and forwarded engineering.

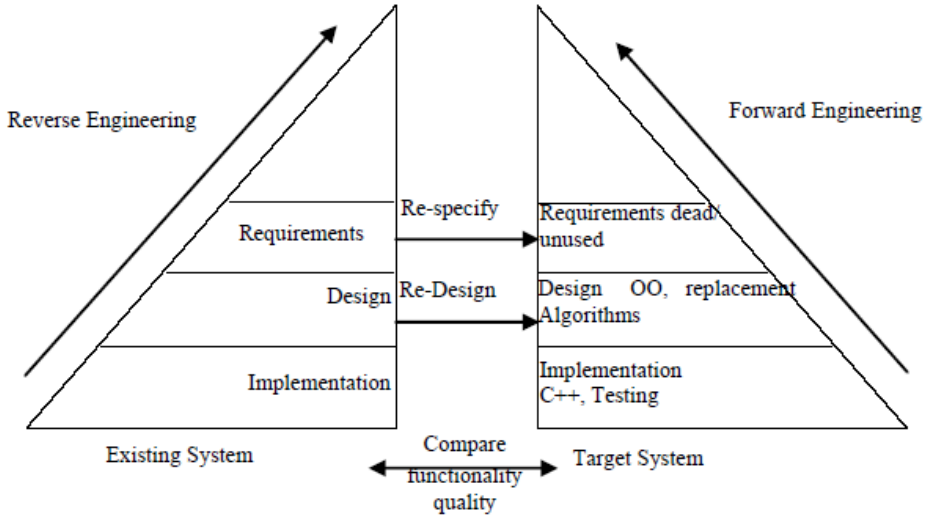

Figure2: Forward and Reverse Engineering

For examples when one would wish to re-engineer code could be the modification of a date field (Y2K) or possibly re-modularization of code to facilitate later maintenance. In a typical Y2K scenario, one would expect to find transformation of items such as the size of the date field in a record, and other transformations that are concerned with code that is used for outputting dates[5].

IV. SOFTWARE REVERSE ENGINEERING

At a higher level, there are two types of engineering: forward engineering and reverse engineering. Both are equally important for developing software. Both play important role in software development life cycle.

Forward Engineering

Forward engineering is the traditional process of moving from high-level abstractions and logical designs to the physical implementation of a system. In some situations, there may be a physical part without any technical details, such as drawings, bills-of-material, or without engineering data, such as thermal and electrical properties. In case of software development life cycle forward engineering is process of moving from requirements and design view to coding process. Design of software is developed with the help of various tools like flowcharts and unified modelling language diagrams. After designing, coding is done and software is tested against the required specifications.

Reverse Engineering

Reverse engineering is taking apart an object to see how it works in order to duplicate or enhance the object. Reverse engineering is very common in such diverse fields as software engineering, entertainment, automotive, consumer products, microchips, chemicals, electronics, and mechanical designs. For example, when a new machine comes to market, competing manufacturers may buy one machine and disassemble it to learn how it was built and how it works. A chemical company may use reverse engineering to defeat a patent on a competitor's manufacturing process. In civil engineering, bridge and building designs are copied from past successes so there will be less chance of catastrophic failure. It often involves taking something e.g. electronic component or software program apart and analyzing its workings in detail, usually to try to make a new device or program that does the same thing without copying anything from the original. The notion of computers automatically finding useful information is an exciting and promising aspect of just about any application intended to be of practical use. So in case of computer system, reverse engineering is the way of analyzing a system to identify its current components and their dependencies, and to extract and create system abstractions and design information. Reverse engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction. It can also be seen as "going backwards through the development cycle. In software engineering, good source code is often a variation of other good source code. Frequently, engineers use reverse engineering to obtain quick solutions to design and maintenance. Reverse engineering can be viewed as the process of analyzing a system to identify the system's components and their interrelationships and to create the representations of the system in another form or a higher level of abstraction and to create the physical representation of that system. Reverse engineering is a truly exciting field of research that is ready to be taught in computer science, computer engineering, and software engineering curricula. There is need to integrate forward and reverse engineering processes for large evolving software systems and achieve the same appreciation for product and process improvement for long-term evolution as for the initial development phases. It is a multistage process aimed at using capital resources efficiently and increasing productivity[6].

Two types of reverse engineering are software and hardware reverse engineering.

Software Reverse Engineering

It is the decompilation of any application, regardless of the programming language that was used to create it, so that one can acquire its source code or any part of it. Design for change for instance, focuses on the aspect of developing software

due to the fact that each software system will age. Reverse engineering techniques provide the means for recovering lost information and developing alternative representations of a system, such as generation of structure charts, dataflow diagrams, entity-relationship diagrams, etc. Software reverse engineering involves reversing a program's machine code back into the source code that it was written in, using program language statements. Software reverse engineering is done in order to retrieve the source code of a program because the source code is lost. It is used to study how the program performs certain operations. Software Reverse Engineering is done to improve the performance of a program and fix a bug. It is adapted to identify malicious content in a program such as a virus. It is used to adapt a program written for use with one microprocessor for use with another and it is done to understand how a product works more comprehensively than by merely observing it. It is used to Investigate and correct errors and limitations in existing programs. The purpose of Software Reverse Engineering is to study the design principles of a product as part of an education in engineering and make products and systems compatible so they can work together or share data. It is done to evaluate one's own product to understand its limitations and determine whether someone else has literally copied elements of one's own technology. The common use of it is to transform obsolete products into useful ones by adapting them to new systems and platforms.
The reverse engineer can reuse code in his own programs or modify an existing program to perform in other ways. He can use the knowledge gained from RE to correct application programs, also known as bugs. But the most important is that one can get extremely useful ideas by observing how other programmers work and think, thus improve his skills and knowledge.

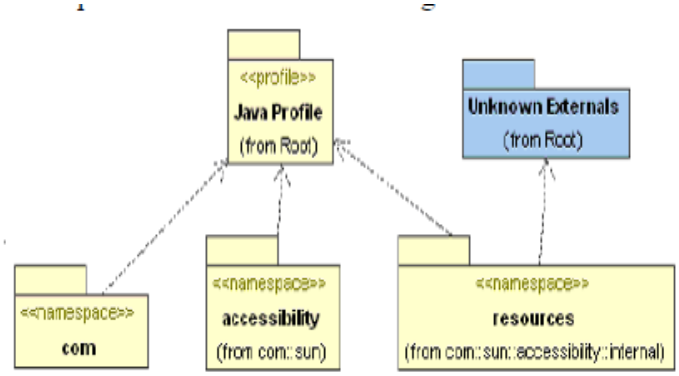

Figure 3: Example of Software Reverse Engineering

In the figure, java source binaries are imported in the UML model. With the help of reverse engineering, existing source code can be imported into the current model. This helps in utilizing existing code and developing new design without copy the old code. One group of reverse engineers are users of old products, who want to maintain them, but find that the original supplier may no longer exist, or may have dropped support for the product. Reverse engineering for the purpose of copying or duplicating programs may constitute a copyright violation. In some cases, the licensed use of software specifically prohibits reverse engineering[7][8].

Hardware Reverse Engineering

It involves taking apart a device to see how it works. For example, if a processor manufacturer wants to see how a competitor's processor works, they can purchase a competitor's processor, disassemble it, and then make a processor similar to it. Computer vision has been widely used to scan PCBs for quality control and inspection purposes, and based on this, there are a number of machine vision for analyzing and reverse engineering PCBs. However, this process is illegal in many countries. In general, hardware reverse engineering requires a great deal of expertise and is quite expensive..

V. CONCLUSION

Investing in program understanding technology is critical for the software and information technology industry to control the inherent high costs and risks of legacy system evolution. There is also a desire, amongst some software users, to reengineer old software, to make it more modular, reusable, accessible or reliable. Thus, Reverse engineering is a truly exciting field of research that is ready to be taught and used as a beneficial technology in the industry. Infrastructures for tool integration have evolved in recent years. It is expected that control, data, and presentation integration technology will continue to advance at amazing rates.

REFERENCES

[1]. Ahmed Saleem Abbas, W. Jeberson, V.V. Klinsega, "The Need of Re-engineering in Software Engineering", International Journal of Engineering and Technology Volume 2 No. 2, pp 292-295, February, 2012.

[2]. Ashwin B. Tomar, V. M. Thakare, " A Study of Sofware Reuse and Models", National Conference on Innovative Paradigms in 'Engineering & Technology (NCIPET-2012) Proceedings published by International Journal of Computer Applications® (IJCA), PP: 14-17, 2012.

[3]. R. Kamalraj, Dr. A. Rajiv Kannan, P. Ranjani, "Stability-based Component Clustering for Designing Software Reuse Repository", International Journal of Computer Applications (0975 – 8887), Volume 27, No.3, PP: 33-36, 2011.

[4]. B.Jalender, N.Gowham, K.Praveen Kumar, K.Murahari, K.Sampath, "Technical Impedimnts To Software Reuse", B. Jalender et. al. / International Journal of Engineering Science and Technology, Vol. 2, No. 11, ISSN: 0975-5462, PP: 6136-6139, 2010.

[5]. Harry M. Sneed, Anecon GmbH, Wien, " Re-engineering for Testability", Universities of Regensburg and Passau, May, 2006

[6]. Chikofsky, E.J. and J.H. Cross, "Reverse Engineering and Design Recovery: Taxonomy in IEEE Software". IEEE Computer Society: 13–17, 1990.

[7] Spencer Rugaber and Richard Clayton , "The representation problem in reverse engineering", In Proceedings of the Working Conference in Reverse Engineering, Baltimore, Maryland, pp. 8–16, IEEE Computer Society, 1993.

[8] Warden R., "Software Reuse and Reverse Engineering in Practice". London, England: Chapman & Hall, pp. 283–305, 1992.

