Prospective of Software Reusability

VIVEK BHATNAGAR and ASHOK KUMAR

ABSTRACT
The idea of software reuse was projected in late sixties. Software reusability is the use of engineering knowledge or artifacts from existing software components to build a new system. Reusability is the key paradigm for increasing software quality in the software development. Reusability implies some explicit management of build, packaging, distribution, installation, configuration, deployment, maintenance and upgrade issues. If these issues are not considered, software may appear to be reusable from design point of view, but will not be reused in practice. This paper presents an empirical study of the software reuse activity by expert designers in the context of object-oriented design. Our study focuses on the following aspects of reuse: the interaction between some design processes, e.g. constructing a problem representation, searching for and evaluating solutions, and reuse processes, i.e. retrieving and using previous solutions. How to retrieve the process which depicts bottom-up versus top-down expanding of the solution, that uses the concept of reusability.

Keyword: Re-engineering, Reusability, Software, Functions, Metrics.

1. INTRODUCTION
Generally speaking, the benefits of software reuse far are more important than the costs. But in the chaotic world of real-world application development, the challenge can be complex. Many managers value rapid prototypes over “best practice engineers, not understanding that building on somewhat sloppy early results will typically and dramatically increase the need for this extra work are not increasing commensurably [1]. There are two approaches for reuse of code: develop the reusable code from scratch or identify and extract the reusable code from already developed code. Software, but not yet used the software reuse concept, there exists extra cost to develop the reusable components from scratch to build and strengthen their reusable software reservoir [2]. The cost of developing the software from scratch can be saved by identifying and extracting the reusable components.

2. Re-engineering Process
A. Reverse Engineering
In the first stage (reverse engineering) software is analyzed and modules are separated to its underlying technology. Reverse engineering involves in extracting design artifacts and making abstraction at higher levels that are easy to understand. It is a process of inspection to know the underlying technology of the software system. In this stage, software engineers move from code level to higher level of abstraction. Some objects are added and deleted according to changed requirements and changed business process. It is on left side upward step as shown in Figure 1.1.

B. Architecture Transformation
In the second stage, new architecture of the software is designed according to the changes required. Output of the first stage (reverse engineering) will be used as input for the second stage. Recovered design from reverse engineering is modified according to a new set of requirements. It is architecture transformation stage. This depends upon the changed business process and other factors relating to the change in environment or technology. It is on the top horizontally right
ward step as shown in fig 1.1

C. Forward Engineering
In this stage we move from higher level of abstraction of code level and software is integrated according to new design. It is vertically downward step as shown in fig 1.1. Forward engineering means developing the system form design specifications recovered after reverse engineering. In forward engineering software engineers come down from higher level of abstraction to code level of development. The new target system is created by moving downward through the levels of abstractions, a gradual decrease in the abstraction levels of the system representation by successive replacement of existing system information with more detailed information.

Re-engineering = Reverse engineering + Enhancement and design improvement + Forward Engineering

In reengineering we go from code level to higher level of abstraction where all the changes are made and made again. We come down from higher level abstraction to the code level (implementation). In re-engineering process; software goes through three forms of system as shown below in the Figure 1.2.

![Abstract System Diagram](image)

Figure 1.2: Abstract System
Re-engineered software system works with reduced execution time, better service in quality and quantity and with less maintenance cost. Advance software technology plays very important role in re-engineering object oriented software systems. The role of software technology is to make software tools and techniques to speed up and easy re-engineering process. Software tools are available for understanding software, improving business process, improving software architecture and developing software. in the coming future more sophisticated tools will be available for re-engineering. 100% automation in re-engineering is not possible, but stepwise automation can be achieved up to great extent.

3. Types of Reuse
Reusability tools are based upon software testing development. Software reuse can apply to any life cycle product, not only to fragments of source code [3, 4]. Process is nothing but Method of developing Software, so one must follow a process which is strong process, if process is weak, end product will also be weak and project may fail, whereas product is what is delivered to the customer/client.

A. Opportunistic reuse
While getting ready to begin a project, the team realizes that there are existing components that they can reuse. Planned reuse - A team strategically designs components so that they'll be reusable in future projects. Opportunistic reuse can be categorized further:

B. Internal reuse
A team reuses its own components since the team may want to control a component critical to the project. Because a project has fix span of time, it has start date and end date. Output of the project is measured.

C. Artifacts may be used
There are many different type of artifact that can be used during software development life cycle process which includes: source code, analysis and design specification, plans (project management), data (testing), documentation, expertise and experience. However, while all of these items are useful, the most often reused artifacts are software components.

4. Process of reuse
i). Domain analysis phase: This phase is to come certain whether deserve to reuse the infrastructure for the domain development
   mainly through the definition and analysis of application domain.
   i. Property obtaining phase: This phase includes development of reuse, may also include some exterior adopt of reuse property.
ii. Property categorizing phase: This mission is a database management mission actually, including categorizing and saving reuse property.

iii. Property maintaining phase: This mission is a maintenance mission actually.

5. Tools of reuse

Software programming is a hard design task, mainly due to the complexity involved in the process. Reuse deals with the ability to combine independent software components to form a larger unit of software. To incorporate reusable components into a software system, programmers must be able to find and understand them. Thus Software reuse is software design, where previous components are the building blocks for the generation of new systems. These are the three or four specific tools by Reusability.

• White Box Reusability
• Black Box Reusability
• Glass Box Reusability

Reusability tools are based upon software testing development. Software reuse can apply to any life cycle product, not only to fragments of source code [5 & 6]. In White-box reusability is verification technique software engineers can use to examine if their code works as expected and a box can share its internal structure or implementation with another box through inheritance. One of the most common complaints of designers or print service providers when previewing and printing transparency from In Design is that a transparency effect like a drop shadow doesn’t display or print correctly. Instead, a white box appears behind the transparency effect [7 & 8]. It discusses several benefits of component characterization, which includes improved cataloguing, improved usage, and improved retrieval and improved understanding eventually for better reuse [9 & 10]. In Black box reusability, the reuse sees the interface, not the implementation of the component. [11 & 12] If a programmer were to change the code of a black box component, compiling and linking the component would propagate the change to the applications that reuse the component. As the users of the component trust its interface, changes should not affect the logical behavior of the component. In Glass box reusability the inside of the box can be seen as well as the outside, but it is not possible to touch the inside to obtain the digital displays.

6. SOFTWARE REUSE BENEFITS

A. Reduced process risk

If software exists, there is less uncertainty in the costs of reusing that software than in the costs of development. This is an important factor for project management as it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as sub-systems are reused.

B. Effective use of specialists

Instead of application specialists doing the same work on different projects [13 & 14], these specialists can develop reusable software that encapsulate their knowledge.

C. Reliability and Safety

Better system reliability is one of the goals of software reuse. It is argued that reusable components, because of more careful design and testing and broader and more extensive usage, can be more reliable that one use equivalents. If so, then it is further argued that using these more reliable components in system architecture can increase the reliability of the system as a whole. Higher system reliability via generative reuse is based on the idea that replacing error prone human processes in software development by automation can produce a more reliable system [15].

7. Conclusion

The above article describes how software reuse play important role in developing software in spite of having number of hurdles since early days of programming. The full benefit of software reuse can only be achieved by systematic reuse that is conducted formally as an integral part of the software development cycle. In this paper, we have reviewed the history of software reliability engineering, the current trends and existing problems, and specific difficulties. Possible future directions and promising research problems in software reliability engineering have also been addressed. Various tools and technique have been developed to achieve the reusability. Hence, it is concluded that software reuse makes our development cheap and effective which is the main goal of software reuse.

References:


Vivek Bhatnagar received his MCA & M. Phil. (Computer Sc. & Technology) degrees and is currently working as Assistant Professor at Maharishi Markandeshwar Institute of Computer Technology & Business Management, M. M. University, Mullana (Ambala). His current areas of interest include Software Engineering and Software Reusability. He is an Associate Life Member of Universal Association of Computer and Electronics Engineers (UACCE), Australia, Member of International Association of Engineers (IAENG), Hongkong and Member of the Society of Digital Information and Wireless Communication (SDIWC). He has attended four short term courses/workshops. He has also written a text book titled “Text book of Computers”.

Dr. Ashok Kumar received his M.Sc. & Ph.D. degrees from the University of Agra. He chaired the Department of Computer Sc. & Applications, Kurukshetra University, Kurukshetra for a very long period and is currently working as a Professor in Department of Computer Engineering at Maharishi Markandeshwar University, Mullana (Ambala). He has over 105 publications in refereed journals, book chapters, and conferences. He has also written books on software Engineering. At present he is conducting collaborative research in the area of Optimization techniques, numerical methods, software engineering, biometrics & pattern recognition and also, 13 scholars awarded Ph.D. under his supervision.