Modeling of Reusability of Software Systems

Priyanka Kakkar, Meenakshi Sharma, Dr. Parvinder Sandhu

Abstract—In this paper, we have done the modeling of the components of the software for the better reusability. The reusability is the quality of a piece of software, that enables it to be used again, be it partial, modified or complete. Software professionals have recognized reuse as a powerful means to potentially overcome the situation called as software crisis. Software Reuse promises significant improvements in software productivity and quality. According to Gomes, the idea of software reuse appeared in 1968, opening new horizons for the software design and development. Reusable software components have been promoted in recent years. The software development community is gradually drifting toward the promise of widespread software reuse, in which any new software system can be derived virtually from the existing systems. As a result, an increasing number of organizations are using software not just as all-inclusive applications, as in the past, but also as component parts of larger applications. In this new role, acquired software must integrate with other software functionality.

Keywords—Software Reusability, Metric, Clustering, Modeling.

I. INTRODUCTION

SOFTWARE reuse [12] is the process of implementing or updating software systems using existing software assets. Software assets or components include all software products, from requirements and proposals, to specifications and designs, to user manuals and test suites. Anything that is produced from a software development effort can potentially be reused. The reusability is the quality of a piece of software, that enables it to be used again, be it partial, modified or complete. Software reusability is an attribute that refers to the expected reuse potential of a software component. Software reuse not only improves productivity but also has a positive impact on the quality and maintainability of software products. Given the attractive payoff of reusing software, there have been several efforts undertaken to discuss the topic of reusability, including overviews of software reusability research directions and software reusability in practice. Developers are adopting many of these reuse approaches, including reuse in product lines, design patterns templates, reference architectures and advanced searching, matching, and modeling tools. Many other reuse approaches, such as product lines, design patterns, and context-independent techniques, address reuse in different ways and have also demonstrated benefits. The software industry is moving toward large-scale reuse, resulting in savings of time and money. To develop a new system from scratch is very costly. This has made custom software development very expensive. It is generally assumed that the reuse of existing software will enhance the reliability of a new software application. This concept is almost universally accepted because of the obvious fact that a product will work properly if it has already worked before. A component can be considered an independent replaceable part of the application that provides a clear distinct function. A component can be a coherent package of software that can be independently developed and delivered as a unit, and that offers interfaces by which it can be connected unchanged with other components to compose a larger system.

There are two approaches for reuse of code: develop the code from scratch or identify and extract the reusable code from already developed code. For the organization that has experience in developing software, but has 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. The cost of developing the software from scratch can be saved by identifying and extracting the reusable components from already developed software systems or legacy systems.

There are two approaches for reuse of code: develop the code from scratch or identify and extract the reusable code from already developed code. For the organization that has experience in developing software, but has 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. The cost of developing the software from scratch can be saved by identifying and extracting the reusable components from already developed software systems or legacy systems.

The contribution of metrics to the overall objective of the software quality is very well understood and recognized. But how these metrics collectively determine reusability of a software component is still at its naïve stage.

There are two forms of reuse and they are as:
• Horizontal Reuse.
• Vertical Reuse.

Horizontal reuse refers to software components used across a wide variety of applications. In terms of code assets, this includes the typically envisioned library of components, such as a linked list class, string manipulation routines, or graphical user interface (GUI) functions. Horizontal reuse can also refer
to the use of a commercial off-the-shelf (COTS) or third-party application within a larger system, such as an e-mail package or a word processing program. A variety of software libraries and repositories containing this type of code and documentation exist today at various locations on the Internet.

Vertical reuse, significantly untapped by the software community at large, but potentially very useful, has far reaching implications for current and future software development efforts. The basic idea is the reuse of system functional areas, or domains that can be used by a family of systems with similar functionality [5]. The study and application of this idea has spawned another engineering discipline, called domain engineering. Domain engineering is "a comprehensive, iterative, life-cycle process that an organization uses to pursue strategic business objectives. It increases the productivity of application engineering projects through the standardization of a product family and an associated production process." [4] Which brings us to application engineering, the domain engineering counterpart: "Application engineering is the means by which a project creates a product to meet a customer's requirements. The form and structure of the application engineering activity are crafted by domain engineering so that each project working in a business area can leverage common knowledge and assets to deliver a high-quality product, tailored to the needs of its customer, with reduced cost and risk." [4] Domain engineering focuses on the creation and maintenance of reuse repositories of functional areas, while application engineering makes use of those repositories to implement new products.

II. RELATED WORK

Poulin [25] presents a set of metrics used by IBM to estimate the efforts saved by reuse. The study suggests the potential benefits against the expenditures of time and resources required to identify and integrate reusable software into a product. Study assumes the cost as the set of data elements like Shipped Source Instructions (SSI), Changed Source Instructions (CSI), Reused source Instructions (RSI) etc.

Boetticher, G., et.al, [6] assimilated knowledge about object oriented concepts, analysis and design. Explanation of various object-oriented metrics was also given such as class oriented metrics (CK metric), metrics for source code, testing, analysis model and design model. Advantages and disadvantages of each object-oriented metric was explained. Information regarding theoretical background of the reusability of software and Object oriented metrics for measuring size, complexity are also given in this reference.

Boetticher, et.al, [6] discussed various approaches for measuring software reusability, to build reusable components and to identify useful modules in existing programs. Taxonomy of reusability metrics was given which provide the attributes of reusable software. Two main methods are there to measure the reusability. One is Empirical methods stress objective, numerical, and repeatable metrics, such as those obtained by observing the module complexity or size. Other is Qualitative methods included (or even emphasized) subjective criteria, such as how well a module complies with a set of style, certification, quality guidelines, or simply agrees with the opinions of “experts.” These are further divided into two categories module oriented and component oriented.

Kartalopoulos, S. V. [15] discussed REBOOT (reusability based on object oriented technique) that develop a taxonomy of reusability attributes. It provided reusability factors, a list of criteria for factor and a list of metrics for each criteria. Various object oriented concepts were defined in this paper, which are useful for finding the reusability. Information about software reuse, types of software reuse, requirements for building software reuse and management issues toward reusable software were also given. This also shows the advantages of this technique over other methods.

The metric based approach is used successfully in designing the framework for evaluation of reusability of object oriented systems and Levenberg – Marguardt algorithm is proved to be best as compared to the other algorithms considered in this work. The performance of this neural network is better than the results of the Neuro-Fuzzy Technique as used in the literature. The Quasi-Newton BFGS Algorithm come out to be the second best algorithm for modeling of reusability data and the Self Organizing Network are showing the worst results among fourteen neural network algorithms under study. Hence, it is concluded that for non linear and complex Engineering applications involving control, inference and analysis by and large Neural Network is an efficient technique.

Jang, J-S. R. et al [13] stated that basic reusability attributes depend on qualities of Correctness, readability, testability but it is not possible to directly measure the most of these attributes of reusable software. Attributes of reusable software were clearly defined in their work.

Selby, R. W., [23] discussed neural network approach to generate the metrics for measuring the reusability and it was proved better over other object oriented metrics. Neural network approach is used to find the reusability index, which give information about software reusability. There were two principle criteria determining which neural network to use. First, a supervised neural network was required. Second, the network needed to be able to classify. This paper reports on an empirical research based on two software products. The research goal is to ascertain the impact of the adoption of a reuse policy on customer satisfaction. The results show that when a systematic reuse policy is implemented, such as the adoption of a domain specific library, 1) reuse is significantly positively correlated with customer satisfaction and 2) there is a significant increase in customer satisfaction. The results have been extended to the underlying populations, supposed normal.

Selby, R. W. [23] tried to identify a number of characteristics of those components, from existing systems, that are been reused at NASA laboratory and reported that the
developers there has achieved a 32 percent reusability index.

Dunn and Knight, [10] also experimented and reported the usefulness of reusable code scavenging.

Chen, Nishimoto and Ramamoorthy, [9] briefly discussed the idea of subsystem extraction by using code information stored in a relational database. They also described a tool called the C Information Abstraction System to support this process.

Esteva and Reynolds, [11] Inductive Learning techniques based on software metrics are used to identify reusable modules.

Caldiera and Basili, [8] A tool, called Care, is used to identify reusable components according to a set of “reusability attributes” based on software metrics.

Mayobre [17] These techniques can be extended and used to help in identifying data communication components.

Arnold, [2],[3] mentioned a number of heuristics that can be used for locating reusable components in the Ada source code. The heuristics are counting the number of references to a particular procedure, identifying the loosely coupled modules and identifying modules that carry high cohesion.

Cho et al [26] proposes a set of metrics for measuring various aspects of software components like complexity, customizability and reusability. The work considers two approaches to measure the reusability of a component. The first is a metric that measures how a component has reusability and may be used at design phase in a component development process. This metric is calculated by dividing sum of interface methods providing commonality functions in a domain to the sum of total interface methods. In the second approach particular component’s reuse level per application in a component based software development is measured.

Washizaki et al [27] discusses the importance of reusability of components in order to realize the reuse of components effectively and propose a Component Reusability Model for black-box components from the viewpoint of component users.

Selby’s recent experimental study Richard W. Selby [28] identified factors that characterize module reuse without revision are low coupling, high cohesion, few input-output parameters, and many comments. The module implementation factors that characterize module reuse without revision were small size in source lines and many assignment statements (i.e. low Cyclometric complexity). Reformat et al have used decision tree based approach to the problems of identification of good or bad software based on Java and C++ objects. In the study fifteen metrics have been used and 55 to 72% accuracy has been reported.

Prieto-Diaz and Freeman encouraged white-box reuse and identified five program attributes for evaluating reusability. The attributes used are:

- Program Size
- Program Structure
- Program Documentation
- Programming Language
- Reuse Experience

Chen and Lee developed about 130 reusable C++ components and used these components in a controlled experiment to relate the level of reuse in a program to software productivity and quality. In contrast to Selby, who worked with professional programmers, Chen and Lee’s experiment involved a team of 19 students, who had to design and implement small database system. The software metrics collected included the Halstead size, program volume, program level, estimated difficulty and effort. They found that lower the value of the software complexity metrics, the higher the programmer productivity.

Richard W. Selby [28] discussed that CK metric suit is able to target all the essential attributes of OO-based software.

Parvinder [20] used Tuned and Refined values of the following Metric suit for the reusability data Modeling:

- Weighted methods per class (WMC)
- Depth of inheritance tree (DIT)
- Number of Children (NOC)
- Coupling Between Object Classes (CBO)
- Lack of Cohesion in Methods (LCOM)

III. CONCLUSION

Software reusability is the process of implementing or updating software systems using existing software assets. Software assets or components include all software products, from requirements and proposals, to specifications and designs, to user manuals and test suites. The requirement to improve software productivity has promoted the research on software metric technology. The requirement to improve software productivity has promoted the research on software metric technology. There are metrics for identifying the quality of reusable components but the function that makes use of these metrics to find reusability of software components is still not clear. These metrics if identified in the design phase or even in the coding phase can help us to reduce the rework by improving quality of reuse of the component and hence improve the productivity due to probabilistic increase in the reuse level.

REFERENCES


