A Pragmatic Based Approach to Software Reuse

Ganesh Patidar, Nilesh Parmar, Kamlesh Patidar
Department of Computer Science & Engineering
Jawaharlal Institute of Technology, Borawan (MP) 451228, India
ganeshpatidar22@gmail.com, nilesh.14@hotmail.com, patidar_kamlesh2003@yahoo.com

Abstract - Software reuse has become a topic of much interest in the software community due to its potential benefits, which include increased product quality and decreased product cost and schedule. The most substantial benefits derive from a product line approach, where a common set of reusable software assets act as a base for subsequent similar products in a given functional domain. In order to decrease the time and effort of the software development process and increase the quality of the software product significantly, software engineering required new technologies. Nowadays, most software engineering design is based on reuse of existing system or components. Also, it is become a main development approach for business and commercial systems [12]. The concept of reusability is widely used in order to reduce cost, effort, and time of software development. Reusability also increases the productivity, maintainability, portability, and reliability of the software products. That is the Re usable software components are evaluated several times in other systems before [12].

Keywords: reuse, components, product cost, quality.

I. INTRODUCTION

1.1 What is Software Reuse?

Software reuse is the process of creating software systems from existing software rather than building them from scratch [1]. Many different products for reuse range from ideas and algorithms to any documents that are created during the software life cycle. Source code is most commonly reused; thus many people misconceive software reuse as the reuse of source code alone. Recently source code and design reuse have become popular with (object-oriented) class libraries, application frameworks, and design patterns [2].

1.2 Why Reuse Software?

A good software reuse process facilitates 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, minimizing the amount of development work required for future projects and ultimately reducing the risk of new projects that are based on repository knowledge.

What is Software Component Reuse?

Software component reuse is the software engineering practice of creating new software applications from existing components, rather than designing and building them from scratch. Reusable components can be requirements specifications, design documents, source code, user interfaces, user documentation, or any other items associated with software. All products resulting from the software development life cycle have the potential for reuse. [13]

1.3 Types of Reuse

1.3.1 Systematic software reuse

Systematic software reuse and the reuse of components influence almost the whole software engineering process (independent of what a component is) [2]. Software process models were developed to provide guidance in the creation of high-quality software systems by teams at predictable costs. The original models were based on the (mis)conception that systems are built from scratch according to stable requirements.

1.3.2 Horizontal reuse
A Pragmatic Based Approach to Software 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.

1.3.2 Vertical reuse

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 [2]. 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”[3].

II. PREREQUISITES TO CREATING REUSABLE SOFTWARE

Unfortunately, software reuse doesn't just happen [4]. Ad hoc reuse, (i.e., reusing a function here, a function there, often times with modifications), also known as opportunistic reuse, doesn't reap the same large-scale benefits as a domain engineering approach. And it's not just a technical issue; it is highly managerial in nature. As much as libraries of reusable code and other assets are important, they will not be fully utilized without management and process support of reuse.

2.1 Organization and Process

The classical software development process does not support reuse.[4] Reusable assets should be designed and built in a clearly defined, open way, with concise interface specifications, understandable documentation, and an eye towards future use. Typically, customer, client, and contract projects are built as “one-time only,” without reuse in mind, and tend to be tightly bound within themselves, without the more robust open interfaces which ease the reuse process. Therefore, in order to make the most of software reuse, the software development process must evolve to include reuse activities.

A strong organizational foundation must exist for reuse to succeed, since domain engineering involves a different way of looking at software products, called a product line approach. A product line is a family of similar products addressing a particular market segment, or domain, and provides a massive opportunity for reuse.

2.2 Technical Expertise

Transferring to a product line approach requires some different technical skills than traditional software development processes, along with many of the current familiar techniques, such as layered architectures, object-oriented programming, information hiding, and abstract interfaces, to name a few. One “new” addition, an aspect of domain engineering, is domain analysis, which involves producing a domain model of the product line that identifies common members and allowable variations for each. Product line software architecture is built based on the domain model, the backbone for all current and future product line family members.

Within the architecture, standard interfaces must exist, so that if a particular base component needs to be specialized for a specific customer, a specialized version will use the standard interfaces and be able to plug right into the global architecture. The biggest new technical challenge on a product line approach is the initial design of the software architecture for robustness towards potential future expansions, and its subsequent maintenance to deal with technology changes. The domain analysis and the design of the software architecture should be carried out by domain experts, people with experience and a solid understanding of the product line base. In order to build quality reusable software and achieve the most gain from reuse, standard coding practices and code documentation must exist across the organization. These standards help developers understand each asset quickly, since each developer is familiar with the standard, and know exactly what to expect and look for in each new module he or she encounters.
III. REUSE COSTS - THE INVESTMENT

There is no denying the large cost associated with starting a reuse program. It is an extra cost on top of the traditional development costs, since designing reusable assets takes more time and care than designing a one-time specific system.

3.1 Process

The software development process must be enhanced to include reuse activities. A reuse library or repository must be created and maintained, and tools must be acquired or developed to access the assets, and many new procedures must be specified:

• Procedures for developing reusable assets and inclusion of assets in the repository
• Procedures for domain analysis and architecture design and modification
• Procedures for configuration management and control of reusable assets Project planning should include extra time for designing, implementing, and testing robust reusable assets as opposed to system-specific functionality, since their quality is important not just to one system, but potentially many future systems.
• The software reuse activities are maintained, managed, and controlled as part of the organizations and project's defined software process.

IV. REUSE ADVANTAGES

Reused software, that has been tried and tested in working systems, should be more dependable than new software. The initial use of the software reveals any design and implementation faults. These are then fixed, thus reducing the number of failures when the software is reused.

Savings in costs and time - As a developer uses already pre-defined components, hence, the activities associated with components specification, design and implementation are now replaced with finding components, their adaptation to suit new requirements, and their integration. Experience shows (also from other fields, like electronic engineering) that the latter set of activities takes less times and therefore costs less. It should be noted, though, that development of components for reuse will certainly attract additional effort, time and cost. This cost, however, can be offset by savings in a number of different software projects.

Increase in productivity. - A set of reusable artifacts can frequently be viewed as a high-level language of concepts drawn from a given problem domain. Hence, a developer is given an opportunity to work with more abstract notions related directly to the problem at hand and to ignore low-level, implementation details. It has been shown that working at a higher level of abstraction leads to an increase in development productivity.

Increase in reliability – Reuse library can be viewed as a software product itself; therefore, its development follows a normal cycle of requirements specification, design, implementation, testing, documentation and maintenance. This also leads to an improved reliability of systems built of reusable components rather than of those built entirely from scratch.

Increase in ease of maintenance.

Systems constructed of reusable parts are usually simpler, smaller, and more abstract. Their design is closer to the problem domain and their reliability is greater. This of course has a very positive impact on the quality of such systems maintenance.

Improvement in documentation and testing. Reusable components are normally accompanied by high quality documentation and by previously developed tests plans and cases. Whenever a new system is created by simple selection and altering of such components, then, their documentation and tests will have to be much easier to develop as well.

High speed and low cost replacement of aging systems. As the reuse-based systems share a very large collection of
A Pragmatic Based Approach to Software Reuse

program logic via the reuse library, thus, they are significantly less complex and much smaller in size than those developed from scratch. Such systems will therefore need less effort during porting or adaptation to new hardware and software environments. It should also be noted that it would normally be the reusable components of the system that is technology intensive, and thus, very expensive to develop, e.g. graphical user interfaces, databases, communications, device control, etc. Sharing that cost across several systems would certainly reduce it when a global replacement of computing resources is called for. [14]

V. CONCLUSION

The product line approach to software reuse requires substantial upfront investment with substantial, but not immediate, benefits. Much commitment, planning, and effort are required to begin a reuse program. Reuse processes and procedures must be incorporated into the existing software development process. Repositories of software assets must be created and maintained. Reusable assets must be designed for reusability. People must be trained in the skills of software reuse. Despite the initial overhead, there are high benefits to software reuse, if appropriate processes are invoked and the requisite planning takes place [11]. Product quality and reliability can increase. Project development time can decrease, along with associated project costs. Project scheduling can become another standard calculation instead of a guessimate. All these benefits, in the long term, can dramatically increase productivity in an organization, and decrease the overall risk of project development by supplying a solid foundation from which all subsequent product family members are derived.

REFERENCE

[15] Introduction to Software Reuse Jacob L. Cybulski Department of Information Systems the University of MelbourneParkville, Vic 3052, Australia