www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 5 may, 2014 Page No. 6114-6117

An Overview of Reusability of Software Components

Ramandeep Kaur[1], Navjot Kaur[2], Rajwinder Kaur[3]

[1]CTIEMT, CSE Department, Jalandhar, India
ramangrewalg@gmail.com

[2]Aryabhatta Group of institutes, CSE Department, Barnala, India
[2]navkaurtoor@gmail.com

[3]LCET, CSE Department, Ludhiana, India
[3]rajgill1989@email.com

Abstract: ***Reuse of software is required to improve the quality and productivity. If their need some changes in the existing software than software can be reused rather than creating software from scratch. Software reuse is the process of creating software systems from existing software rather than building them from scratch. This paper gives details about various concepts of reuse. For making some changes in software no need to change the whole software working particular changes can be done with reuse in existing software. This paper explains about the things in software that can be reused, various techniques of reuse and advantages and drawbacks of reuse. A component reuse is famous for reuse. The quality of software depends on the quality of the reusable component.***

Keywords: Software component, reuse intentions, reuse artifacts, reusable function, package, module.

1. Introduction

To improve software quality and productivity their need some changes in the existing software. The method of enhancing the quality and power of a software by making some changes in the existing software is called software reuse. It is more appropriate to do the changes in the existing software rather than creating new software from scratch. Software reuse is the process of creating software systems from existing software rather than building them from scratch.
Next concept used in reuse is reusability. *Reusability* defines the degree to which software can be reused. It depends upon the new features of software that are going to be edited in the software and on the existing features of the software.

2. Need for reuse

The concept of reuse comes from the manufacturing companies. Let's take an example of manufacturing company of vehicles. Every vehicle needs to be repaired after some time. For repairing of vehicles there is the need of spare parts. Every manufacturing company should provide the spare parts of every vehicle. Similar to spare parts of vehicles software reuse is required for proper working of software. Software companies use same concept for developing the software in parts those are called software components. Reuse process increase the productivity, quality, and reliability, and the decrease the costs and implementation time.

3. Artifacts that can be reused

The software can be reused based on various factors. In today's world software is reused on the basis of many different artifacts produced during the software development life cycle. Some typical examples of reusable artifacts include:

- Source code
- Analysis and design specification
- Plans
- Data (testing)
- Documentation
- Expertise and experience (life cycle model reuse, quality assurance)
- And any information used to create software and software documentation.

However, while all of these items are useful, the most often reused artifacts are software components.
Reuse of software components is becoming more and more important in a variety of aspects of software engineering. *Software component* can be any part of the software that is required for functioning of software it can be module or and function. Components can be seen as some part of a software system that is identifiable and reusable. Components can be reusable functions, e.g., statistics libraries, numerical libraries, or packages, modules, subsystems and classes. Reuse success depends upon quality of components. A component is fit for reuse, if it provides the required behavior in the proposed circumstance. If we want high levels of reuse, then various circumstances for the usage of the components should be considered. A software component may be used for many different applications, in different business and technical environments, by different developers using different methods and tools, for different users in different organizations

4. Reuse methodologies

As the development costs of software systems increase, the role of reuse becomes more important in software engineering. For this reason, a software engineering methodology should support the notion of developing and leveraging reusable software resources developing a software methodology that supports reuse is an active focus of current research. Here, reuse methodologies is the process of following steps that might be performed by a software development group in traditional software development methodologies (such as the software development life cycle or prototyping). Six processes are involved in developing a target software system.

The first process involves *classifying the existing software resources* to be reused in the future. This has to be performed at the initiation of the reuse program to develop a library of software resources. It must also be performed each time a new reusable resource is to be cataloged. The remaining processes form part of the software development life cycle proper.

The second process is for *specifying requirements* for the new system. This has to be performed regardless of whether the software component is to be developed from scratch or not.

The third process is for *identifying and selecting appropriate target software resources* from reusable software resources based on the requirements specification.

The fourth process *modifying software resources* is necessary when the library resources retrieved do not exactly match the requirements specification.

The fifth process *build new component* is necessary when there is no similar software resource in the existing reusable software resources for some of the requirements.

Finally the sixth process is required to *combine the new and reused software resources* into the target software system.

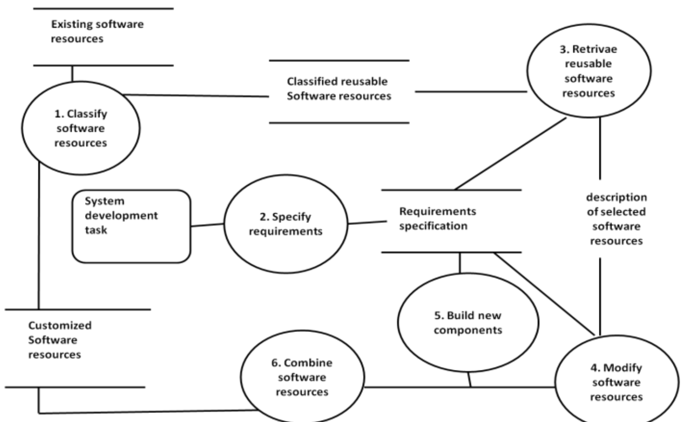
Existing software resources
Classified reusable Software resources
3. Retrivae reusable software resources
1. Classify software resources
System development task
2. Specify requirements
Requirements specification
description of selected software resources
Customized Software resources
5. Build new components
6. Combine software resources
4. Modify software resources

Fig 1: Reuse methodology process

5. Reuse intentions

Depending on whether the internals of a software component are visible to reusers it can be defined as black-box or white-box reuse. If a component is a black box we cannot modify its internals: we use it as is. White-box components are usually modified, even though this is not necessarily the case. They offer both as-is reuse and reuse by adaptation. The term glass-box reuse means white-box visibility but black-box reuse.

5.1 Black-Box Reuse

Reusing a component as a black box means using it without seeing, knowing or modifying any of its internals. The component provides an interface that contains all the information necessary for its utilization. The implementation is hidden and cannot be modified by the reuser. Thus reusers get the information about what a component is doing, but they do not have to worry about how this is achieved.

5.2 White-Box Reuse

White-box reuse means reuse of components of which internals are changed for the purpose of reuse. White boxes are typically not reused as is, but by adaptation. They create more opportunities for reusers due to the ease of making arbitrary changes. On the negative side of white-box reuse, it requires additional testing and costlier maintenance. Unlike black boxes, a new component derived by modifications to an existing component must be regarded as a new component and thoroughly tested. Additionally, the new component requires separate maintenance.

5.3 Glass-Box Reuse

The term glass-box reuse is used when components are used as-is like black boxes, but their internals can be seen from outside. This gives the reuser information about how the component works without the ability to change it. But this

information may be crucial for understanding how certain tasks are carried out. It may also give the reuser some confidence from being able to see inside the component and capture how it works. Additionally, getting internal information provides some kind of knowledge transfer and, for example, can help in building new components.

5.4 Generative Reuse

Generative reuse is itself a reuse technique, but it can be seen as kind of black-box reuse. Instead of picking one of several existing black boxes, a component's specification is created and its implementation automatically generated by a program generator. The program generator is a black box, its internals are of no interest to the reuser. Also, the generated implementation will not be modified. If changes are necessary, they will be made in the specification and the implementation is recreated.

6. Types of software reuse

6.1 Opportunistic reuse

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

6.1.1 Internal reuse

A team reuses its own components. This may be a business decision, since the team may want to control a component critical to the project.

6.1.2 External reuse

A team may choose to license a third party component. Licensing a third party component typically costs the team 1 to 20 percent of what it would cost to develop internally. The team must also consider the time it takes to find, learn and integrate the component.

6.2 Systematic software reuse

Independent of what a component is, Systematic Software Reuse influences almost whole software engineering process. For providing guidance in the creation of high quality software systems at low-cost the software process models were developed. The original models were based on the misconception that systems are built from scratch according to stable requirements. Software process models have been adapted since based on experience and several changes and improvements have been suggested since the classic waterfall model. With increasing reuse of software, new models for software engineering are emerging. New models are based on systematic reuse of well-defined components that have been developed in various projects. These trends are particularly evident in markets, such as electronic commerce and data networking, where reducing development cycle time is crucial to business success.

6.3 Horizontal 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 email 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.

6.4 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. The study and application of this idea has spawned another engineering discipline, called domain engineering. This brings us to application engineering 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. Domain engineering focuses on the creation and maintenance of reuse repositories of functional areas, while application engineering makes use of those repositories to implements new products.

7. Reuse benefits

Various benefits of software reuse are the following:-

7.1 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.

7.2 Increase in productivity

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.

7.3 Increase in reliability

This also leads to an improved reliability of systems built of reusable components rather than of those built entirely from scratch.

7.4 Increase in ease of maintenance

This of course has a very positive impact on the quality of such systems maintenance.

7.5 Improvement in documentation and testing

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.

7.6 High speed and low cost replacement of aging systems

As the reuse-based systems share a very large collection of 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.

8. Reuse drawbacks

Despite the benefits of software reuse, it is not as widely practiced as one might assume. There are many factors that directly or indirectly influence the success or failure of reuse. These factors can be of conceptual, technical, managerial, organizational, psychological, economic or legal nature. Some of these are:-

- Reusing code, as compared with development of entirely new systems, is boring.
- Locally developed code is better than that developed elsewhere.
- It is easier to rewrite complex programs from scratch rather than to maintain it
- There are no tools to assist programmers in finding reusable artifacts.
- In majority of cases, developed programs are too specialized for reuse.
- Adopted software development methodology does not support software reuse.
- Reuse is often ad-hoc and is unplanned.
- There is no formal training in reusing code and designs effectively.
- Useful reusable artifacts are not supported on the preferred development platform.
- The reuse process is too slow.
- Interfaces of reusable artifacts are too awkward to use.
- Code with reusable components is often too big or too inefficient.
- Programs built of reusable components are not readily transportable.
- Reusable components do not conform to adopted standards.

- Reuse techniques do not scale up to large software projects.
- There are no incentives to reuse software.

9. Conclusion

This paper reviews the concepts of software reuse, reusability, reuse artifacts, intentions, advantages and disadvantages, types and methodology of reuse. With reuse productivity and efficiency of software can be improve. Reuse is the best way for making new changes in the existing software rather than making efforts for change in the software from the scratch. Reuse have various advantages, it increases the quality, productivity, reusability and decreases the development cost. There exist some situations where software reuse is not appropriate depend upon the requirements of software. This paper gives the information related the reuse artifacts that can be reused. Most of engineers work with reuse of coding. But in now days software component reuse is popular. A component of software can be an object, class, function or anything related to the software that is responsible for the smooth working of the software. The methodology followed for reuse is similar to the software development life cycle. There are various types of software reuse types opportunistic reuse, systematic reuse, horizontal reuse and vertical reuse on the basis of these technologies software can be reused.

References

[1] Kuljit Kaur, Parminder Kaur, Dr. Hardeep Singh, 2005, Quality Constraints on Reusable Components.

[2] Neha Budhija and Satinder Pal Ahuja, 2011, Review of Software Reusability.

[3] Yongbeom Kim, Edward A. Stohr, 1991, software reuse: issued and research directions.

[4] Johannes Sametinger, 1997, software engineering with reusable components.

[5] Bruce W. Weide, William F. Ogden, and Stuart H. Zweben, 1991, Reusable Software Components.

[6] B.Jalender, Dr A.Govardhan, Dr P.Premchand, 2011, Breaking the Boundaries for Software Component Reuse Technology

[7] Jacob L. Cybulski, Parkville, Introduction to Software Reuse

Author Profile

Ramandeep Kaur received the B.Tech. and M.Tech. degrees in Information technology and Computer Scinence and Engineering from Guru Nanak Dev Engineering College, Ludhiana and Guru Nanak Dev University, Amritsar in 2010 and 2012, respectively. Working as Assistant Professor at CT Group of Institutions, Jalandhar, India.

Navjot Kaur received the B.Tech. and M.Tech. degrees in Information technology and Computer Scinence and Engineering from Guru Nanak Dev Engineering College, Ludhiana and Punjabi University, Patiala in 2010 and 2012, respectively. Working as Assistant Professor at Aryabhatta Group of institutes, CSE Department, Barnala,India.

Rajwinder Kaur received the B.Tech. degrees in Information technology and Computer Scinence and Engineering from Guru Nanak Dev Engineering College, Ludhiana in 2010. Working as Assistant Professor at CT Group of Institutions, Jalandhar, India.