# Adopting Domain based Reuse for Large-scale Company

**Gowtham Gajala[1], Nagavarapu Sateesh[2]**

[1]Department of Information Technology, Kakatiya Institute of Technology & Science, Warangal, Andhra Pradesh, India

[2]Department of CSEIT, Malla Reddy Institute of Technology, Secunderabad, India

**Abstract:** ***Software is rarely built completely from scratch. To a great extent, existing software documents (source code, design documents, etc.) are copied and adapted to fit new requirements. Yet we are far from the goal of making reuse the standard approach to software development. Software reuse demands that existing components must be readily incorporated into new products. To be able to reuse software components, it is necessary to locate the component that can be reused. Locating components, or even realizing that they exist, can be quite difficult in a large collection of components. These components need to be suitably classified and stored in a repository to enable efficient retrieval. Adopting domain based reuse requires sufficient company size to maintain specialized groups. Component groups are responsible for developing reusable components.***

**Keywords:** Reuse, Component, Software, Source code, Application group, Domain group.

## 1. Introduction

Software reuse is the process of creating software systems from existing software rather than building them from scratch. Software reuse is still an emerging discipline. It appears in many different forms from ad-hoc reuse to systematic reuse, and from white-box reuse to black-box reuse. 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. Software components provide a vehicle for planned and systematic reuse. The software community does not yet agree on what a software component is exactly. Nowadays, the term component is used as a synonym for object most of the time, but it also stands for module or function. Recently the term component-based or component-oriented software development has become popular.

Software reuse has many technical and nontechnical aspects, for example, ad-hoc reuse, institutionalized reuse, black-box reuse, white-box reuse, source code reuse, design reuse. As software companies increase their commitment to reuse, they will pass from ad-hoc reuse with application groups only through domain based reuse with domain groups and application groups. Adopting domain based reuse requires sufficient company size to maintain specialized groups. Component groups are responsible for developing reusable components. Domain groups are also responsible for the development of reusable components; in addition, they have to gain knowledge about their specific domain. Application groups are obligated to develop applications by using components created by these specialized groups.

## 2. Existing System

Large-scale reuse in an organization cannot be adopted without organizational changes. Technology is an important pre requisite for reuse, but people make it work. Reuse in an organization can only be achieved when people cooperate. Producing valuable and reusable software components is not enough. We must ensure their transfer to the consumers. Various models for organizations that support software reuse exist. These models influence development practices and are an indicator of reuse maturity in a software company. In practice, a company might reflect some combination of these models.

### 2.1 Ad-hoc reuse among application groups

Frequently companies use organizations based on projects. If there is no explicit commitment to reuse then reuse can happen in an informal and haphazard way at best. Most of the reuse, if any, will occur within projects (see Fig. 1). The reuse of components from different projects may occur but is the exception.

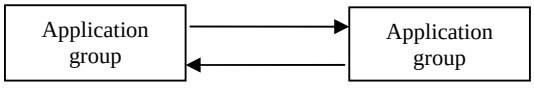


**Figure 1**: Ad-hoc reuse

### 2.2 Repository-based reuse among application groups

The situation slightly improves when a component repository is used and can be accessed by various application groups (see Fig. 2). However, no explicit mechanism exists for putting components into the repository and no one is responsible for the quality of the components in this repository. This can lead to many problems and hamper software reuse. The repository-based reuse approach is based on quantity because any components can be put into the repository and there is no control over their quality and usefulness. If no effort had been made to make the components reusable, then re-users must be cautious. And as

there is no control over the input and no maintenance of the components, re-users have to be cautious even when the components had been prepared with high quality and reusability in mind.

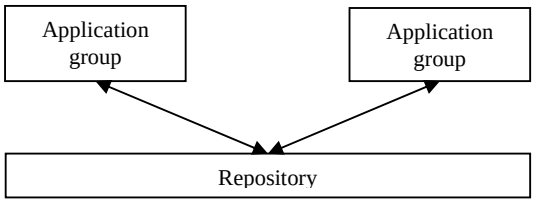


**Figure 2**: Repository-based reuse

### 2.3 Centralized reuse with a component group

In this scenario a component group is explicitly responsible for the repository (see Fig. 3). The group determines which components are to be stored in the repository, ensures the quality of these components and the availability of necessary documentation, and helps in retrieving suitable components in a particular reuse scenario. This amounts to centralized production and management of reusable software components. Application groups are separated from the component group, which acts as a kind of subcontractor to each application group. An objective of the component group is to minimize redundancy.

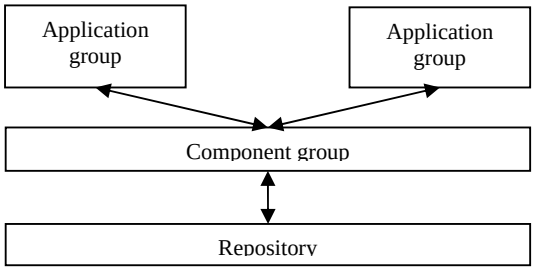


**Figure 3**: Centralized reuse

### 2.4 Domain-based reuse

The specialization of component groups amounts to domain-based reuse (see in Fig. 4). Each domain group is responsible for components in its domain, e.g., network components, user interface components, database components. Application groups may build their applications by integrating components from different domains. This organization yields to the acquisition of specific skills and knowledge of specific software domains. One possible drawback may be an overhead in communication between project and domain groups.

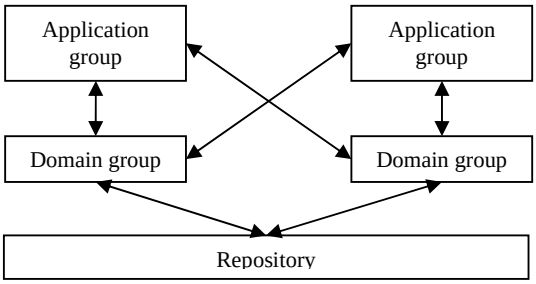


**Figure 4**: Domain-based reuse

## 3. Proposed System

Systematic software reuse and the reuse of components influence almost the whole software engineering process (independent of what a component is). 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 conception that systems are built from scratch according to stable requirements.

When considering either with Centralized reuse with component group or Domain-based reuse, the classification is done without prior consideration of some constraints like versions, input/output, time complexity, etc. If we try to apply any of these constraints for Centralized reuse with component group or Domain-based reuse, i.e., maintaining versions for the components at Centralized reuse for a component group reduces redundancy for component identification. Or considering input constraints and wishing for the desired/expected output constraints component makes the user/developer of the component with reduced time in reusability of the software. In this regard any large-scale software company must consider certain constraints to improve the efficiency of retrieval of the software component for reuse.

## 4. Conclusion

In this paper we proposed the constraint based retrieval for the software component reuse to improve the efficiency of retrieval of the software component for reuse.

The software classification goes beyond source code components and also covers aspects from the area of distributed computing and emphasizes the importance of open systems and standards. There is more to software components than functions and classes. Like software reuse, software components go beyond source code. Components cover a broader range than frameworks and patterns do. We give examples of successful component reuse and evaluate them by using the suggested classification scheme.

Domain groups are also responsible for the development of reusable components; in addition, they have to gain knowledge about their specific domain. This approach serves as an effective means to categorize components and to retrieve the relevant components efficiently to improve retrieval efficiency.

In future this proposal can be improved more by pertaining the domain based reuse with considering time complexity or even by the developer along with specific versions. In addition with considering time complexity or even by the developer along with specific versions, we can still work on applying more multimedia effects like adding video output for the searched output so as to make the registered user more comfortable in selecting and downloading the searched component.

## References


[1] Gowtham Gajala, Implementation of attribute value & faceted value classification scheme for constructing Reuse Repository, in International Journal of Computer Trends and Technology (IJCTT) - Volume4 Issue1 - 2013, ISSN: 2231-2803.

[2] Swathy Vodithala, P. Niranjan Reddy and M. Preethi, A Resolved Retrieval Technique For Software Components, in International Journal of Advanced Research in Computer Engineering & Technology Volume 1, Issue 4, June 2012, ISSN: 2278 – 1323.

[3] S. Henninger, "An Evolutionary Approach to Constructing Effective Software Reuse Repositories", ACM Transactions on Software Engineering Methodology, no 2, 1997, pp. 111-150

[4] Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse", Communication of the ACM, Vol. 34, No.5, May 1991

[5] Gerald Kotonya, Ian Sommerville and Steve Hall, "Towards A Classification Model for Component Based Software Engineering Research", Proceeding of the of the 29th EUROMICRO Conference © 2003 IEEE

[6] William B. Frakes and Thomas. P. Pole, "An Empirical Study of Representation Methods for Reusable Software Components", IEEE Transactions on Software Engineering vol.20, no.8, Aug. 1994, pp.617-630.

[7] Lars Sivert Sorumgard Guttorm Sindre and Frode Stokke, "Experiences from Application of a Faceted Classification Scheme" © 1993 IEEE, pp 116-124.

[8] Jeffrey S. Poulin and Kathryn P. Yglesias "Experiences with a faceted Classification Scheme in a Large Reusable Software Library (RSL)", In The Seventh Annual International Computer Software and Applications Conference (COMPSAC'93), 1993, pp.90-99

[9] Vicente Ferreira de Lucena Jr., "Facet-Based Classification Scheme for Industrial Automation Software Components"

[10] Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse" © 1990 IEEE, pp.300-304

[11] Klement J. Fellner and Klaus Turowski, "Classification Framework for Business Components", Proceedings of the 33rd Hawaii International conference on system Sciences- 2000, 0-7695-0493-0/00 © 2000 IEEE

[12] Vitharana, Fatemeh, Jain, "Knowledge based repository scheme for storing and retrieving business components: a theoretical design and an empirical analysis", IEEE Transactions on Software Engineering, vol 29, no. 7, pp, 649-664.

[13] William B. Frakes and Kyo Knag, "Software Reuse Research: Status and Future", IEEE transactions on Software Engineering, VOL.31 NO.7, JULY 2005

[14] R. Prieto-Diaz and P. Freeman, "Classifying Software for Reuse", IEEE Software, 1987, Vol.4, No.1, pp.6-16.

[15] Rym Mili, Ali Mili, and Roland T. Mittermeir, "Storing and Retrieving Software Components a Refinement Based System", IEEE Transactions of Software Engineering, 1997, Vol.23, No.7, pp. 445-460

[16] Hafedh Mili, Estelle Ah-Ki, Robert Godin, and Hamid Mcheick, "Another nail to the coffin of faceted controlled vocabulary component classification and retrieval", Proceedings of the 1997 symposium on software reusability (SSR'97), May 1997, Boston USA, pp.89-98.

[17] Hafedh Mili, Fatma Mili, and Ali Mili, "Reusing Software: Issues and Research Directions", IEEE Transactions on Software Engineering, Vol.21 No.6, June 1995.

[18] Gerald Jones and Ruben Prieto-Diaz, "Building and Managing Software Libraries",© 1998 IEEE, pp.228-236.

[19] Prieto-Diaz, Freeman, "Classifying Software for Reuse", IEEE Software, vol.4, mo.1, pp.6-16, 1997

[20] Nancy G. Leveson, Kathryn Anne Weis, "Making Embedded Software Reuse Practical and Safe "12 th ACM SIGSOFT, October, 2004


## Author Profile

**Gowtham Gajala** received the M.Tech degree in Software Engineering from Kakatiya Institute of Technology & Science in 2010. Since then he started his journey as Assistant Professor at KITS. His area of interest is Software Engineering.

**Nagavarapu Sateesh** received the M.Tech degree in Software Engineering from Kakatiya Institute of Technology & Science in 2010. Since then he started his journey as Assistant Professor at MRIT. His area of interest is Software Engineering.