IJCTA, 9(40), 2016, pp. 7-17

Component Based Software Development using Refactoring with Form Method

S. Manimekalai*

Abstract : In industrial, for the enhancement and maintenance of software the Software Product Lines and Components Based Software Engineering (CBSE) is to increase the reuse level significantly. The Software Product Lines required to locate the problem of feature unpredictability that is a one feature may need different implementations for various customers. The refactoring method is used to reuse level of software in business components. During the planning stages, the domain analysis can be conducted by software product line engineering to manage the variability. However, high load planning processes are not well arranged with the minimal values of the practices. Refactoring techniques have been applied to develop the software quality attribute, but the refactoring effect on exacting quality attribute is still indefinite. Therefore, in this paper, the refactoring with Feature Oriented Reuse Method (FORM) is proposed to manage the variability of software development process. The CBSE is employed for the development of software that depends on reuse. CBSE is arising from the failure of object-oriented improvement to sustain reuse efficiently. Here the refactoring method is applied to improve the design of existing code for the Component Based Software Development. In FORM, for the development of reusable business components and architectures and the software applications are developed using the domain artifacts generated from the domain engineering. The feature model is an assessment space for software expansion and it is a high-quality starting point for discovering candidate reusable components. The refactoring is performed based on the software quality attributes. The software quality may be reusability, security, supportability, testability and maintainability. These methods can provide more efficient, reusable components for the software development in industries.

Keywords : CBSE, Software Product Lines, Refactoring, Feature Oriented Reuse Method, software quality attributes, reusable components

1. INTRODUCTION

A Software Product Line (SPL) is a set of software demanding process that contribute to a common, fulfilling the exact requirements of a particular segment of the market or assignment by managing the set of features and that are improved from a frequent set of core assets [7][12][17]. The Software product line engineering permits organizations to handle products' families that are parallel but not equal. A product-line share a common set of necessities and also demonstrates important variability in necessities. An enhancing number of organizations that recognize the business consequence of maintaining associated products as members of a product-line [7][19][20]. The product-lines can provide extra efficient component reuse. Here, common components are reused many times and fault correctness and improvements to one product is quickly disseminated to other product-line members. From ideas of product line, the Component-Based Software Engineering (CBSE) stands to expand considerably. Within a given domain it is probable that component-based methods or made by a given organization and it will share a lot of resemblances and in exacting, it can utilize several of the similar components. By creating obtainable mechanisms

* Assistant Professor, Department of Computer Science, Theivanai Ammal College for Women (Autonomous), Villupuram
Email: mamekaroshni@gmail.com

the beginning of component-based software engineering modifies this situation that allows elements of software and it is to be quickly and capably gathered into novel applications. This permits the fundamental principle of product line expansion is applied at every phases and software development levels [3][19].

Refactoring is the process of enhancing the existing code design by modifying its internal structure without disturbing its behavior of external. The badly designed code is complex to sustain, analysis and execution and therefore the software reduces quality [8][9][13]. We can discover the work balance modifies with refactoring. In this process, during development Locate, design, slightly than occurring each upfront, happens continuously. Making the scheme that how to progress the design [2]. The necessary goal of refactoring is the secure transformation of the program to develop the quality. The advantage of responsibility refactoring contains, development of external software quality attributes. The interaction of resulting directs to a plan with intend that remains good quality as expansion keeps. To assuring security and refactoring process efficiency previously needs automation. Where difficulty enhances due to require are handling a high number of variants, this support becomes still further necessary in the SPL framework. In this paper, we illustrate FORM and refactoring tool that executes code transformations for extracting variations of the product.

FORM method is an efficient technique that is viewed for and takes similarities and variations of applications in a domain in terms of features and utilizing the examination results to expand components and architectures of domain [16] [21]. FORM product line engineering contains two major methods, component improvement and product improvement. First one is component improvement contains recognizing a product line that is marketing and product plan (MPP) improvement and alteration, feature modeling, and analysis of requirement and improving architectures and reusable components depends on the results of the analysis. Second one is a product improvement contains recognizing necessities, selecting features, adapting components and making code, selecting and adopting architecture for the product [3] [18]. In this paper, we insert related information to the FORM's assets to refactoring process and we describe a reusable database of business components in order to develop the reusability of the components. In this process, we revealed the implementation refactoring with the FORM method in component based software development. We mostly focus on four major topics: CBSE, Refactoring, Feature Oriented Reuse Method and Software Quality Attributes.

2. LITERATURE REVIEW

Martin L. Griss discussed the process on agent-based product-line CBSE for flexible e-commerce methods. Many technologies are integrated for analysis of product-line and design of component, realization and customization to generate a foundation for systematic product-line improvement. Mainly independent task on reuse and object-oriented method has developed to the degree that integration of the methods undertaking a coherent approach. A practical improvement process is outlined that structures a set of request and changeable features sustaining a product-line and to create reusable components that is combined into adapted components and frameworks to sustain the product-line.

Marcel FoudaNdjodo et.al extended the semantic of FORM assets in order to identify theoretical reusable business elements in FORM. The model is utilized by Ramadour, here the notion of context is introduced to direct conception and reuse of business elements. The Z notation is employed to give a framework for exact analysis of reusable business components generated. The Assets of the technique are formalized. This allows to obviously describe how an activity generates an objective asset from an input one and an improvement of assets through a concept steps. The tool improvement sustaining the technique is also a concern.

Vander Alves et.al extended the conventional refactoring notion to an SPL framework. Besides refactoring processes, Feature Models may also be refectories. A set of sound refactoring is presented for FMs. This comprehensive refactoring description is evaluated for supplying the mobile games area. Besides conventional program refactoring, feature representations are refactored. Not only the program

quality is enhanced and also the feature model quality by sustaining or enhancing its configurability. In the mobile domain, a set of refactoring of the sound feature model and estimated them in a genuine case study is shown.

Colin Atkinson et.al discussed a technique, KobrA, which plainly united the two paradigms into a methodical, united approach to maintenance and software improvement. From this integration Key synergy resulting contains sustain for the fast and flexible instantiation of system variables and the procedural support provision for component based software improvement. In an industrial the technique itself is at present being validated setting on a case study in the Enterprise Resource Planning area. It is also being utilized in the improvement of the workbench of KobrA.

Karim O. Elish et.al initial step is taken towards a refactoring classification to pattern methods based on their quantifiable result on software quality attributes. This classification process assists software designers in choosing the suitable refactoring to pattern methods that can develop their design quality based on their design goals. By utilizing exact refactoring it permits them to forecast the quality flow caused to pattern methods.

Yaser Ghanam et.al introduced a test-driven and bottom-up approach to initiate inconsistency to systems by reactively refactoring existing code. This approach with an eclipse plug-in to mechanize the process of refactoring is maintained. This approach is evaluated by a case study to decide the possibility and sensibleness of the approach. An inadequate estimation of the probability and practicality of the approach was introduced. Systematic refactoring is utilized in order to insert difference points and variants in the scheme, when required. An Eclipse plug-in is contributed to mechanize this procedure. By the plug-in the approach is supported and it was discovered to be reasonable and practical, but suffered a number of limitations that presently trying to address.

3. METHODOLOGY

A. Component Based Software Engineering

Component-Based Software Engineering (CBSE) is one method for development of software that depends on reuse of software. It is emerging from the failure of improvement of object-oriented to maintain efficient reuse. Component-based software engineering also called Component-Based Development (CBD) and it is a software engineering branch that emphasizes the division of regards in respect of the wide-ranging functionality obtainable throughout a given software scheme. A single object classes are too complete and detailed. Components are furthermore conceptual than object classes and it is considered to be stand-alone service providers. These practices, goals to take about an evenly wide-ranging degree of benefits in both the long-term and short-term for the software itself and for organizations. Software engineers look upon components as starting platform part of service-orientation.

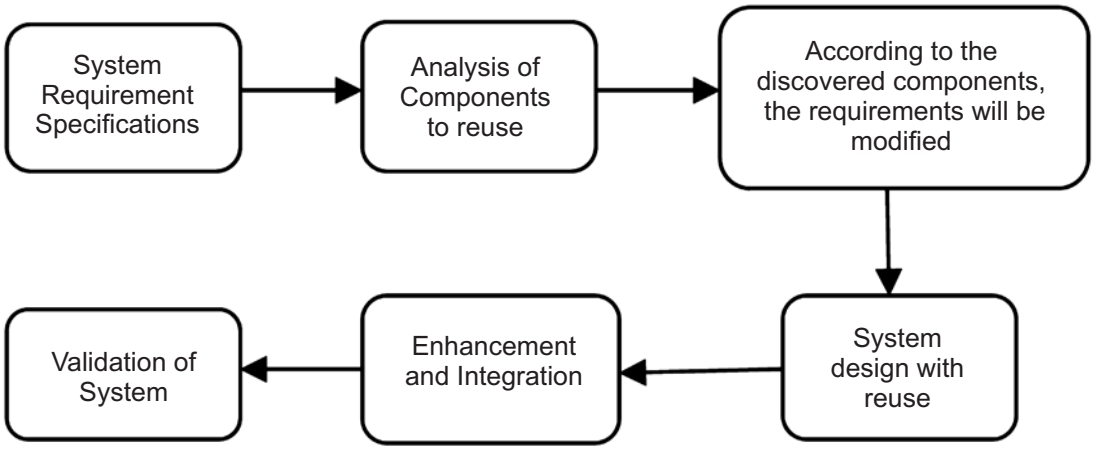

Figure 1: CBSE process

For instance, in Web Services the components play this role and more lately, in Service-Oriented Architecture (SOA) - whereby an element is converted into a service and consequently inherits further characteristics ahead of that of a normal component. Components can generate events or use events and it is utilized for Event Driven Architecture (EDA).

Figure 1 shows the CBSE process, system requirement specifications are noted to reuse the business components and the components are analyzed. The system requirements are modified according to the found components that are reusable purpose after the component analyzed. Design the architectural with reuse process after requirements modifications. To develop and integrate the modified requirements based on the architectural design. Finally, validate the system for reuse.

The main objectives of CBSE are :

1. **Decrease of cost and time for construction, large and complex systems:** Main objective of Component based approach is to construct complex software systems utilizing off the projection component so that the time to construct the software reduce significantly. Using function point or other techniques the cost efficiency of the present technique is recognized.
2. **Enhancing the software quality:** By enhancing the component quality the software quality is enhanced. Although the model is not true in common. Sometimes the assembled systems' quality might not be straightforwardly interrelated to component quality in the sense that enhancing the component quality does not essentially involve the enhancement of the systems.

By promoting the utilization of software components built, CBSE of Product-line offers assurance of large-scale software reuse by profitable vendors or in-house developers. A product-line is constructed approximately the set of reusable components by recognizing the products to decide the ordinary and variable features utilizing a method is known as analysis of the domain. Then a structure of product and strategy of implementation approximately a set of reusable components is developed that is composed to execute numerous different products. CBSE of product-line subsequently becomes of considered significance, and management will pay concentration.

CBSE of product-line has the probable to :

1. By permitting the systems to be constructed by assembling reusable components rather than from scrap, considerably the cost and time-to-market of enterprise software systems is reduced.
2. By permitting novel (higher) quality components to restore old ones, the maintainability of enterprise software systems is enhanced.
3. Improve the consistency of enterprise software systems because every reusable component has gone through numerous evaluation and examination stages in the course of its unique improvement and earlier utilization; and because CBSE depends on explicitly described system design and interfaces.
4. By improving only a only some novel components, and reusing the rest, the ability of organizations is enhanced in meeting demands of a changing market by permitting novel products to be rapidly constructed.
5. In component-based software improvement, improve the enterprise software systems quality by permitting application-domain experts to expand components and software engineers particular to accumulate the components and construct enterprise software systems.

A feature is a characteristic of a product that users and customer analysis as significant in defining and differentiating product-line members. A feature is

1. A definite necessity
2. An assortment among elective or alternative requirements associated to definite characteristics of the product, such as performance, usability and functionality.
3. Associated with characteristics of accomplishment, such as size, computer or operating system.
4. Compatibility with definite standards such as HL7, CORBA, or TCP/IP.

In this paper, initially we employ the CBSE for reuse the business components, the CBSE method is applied before refactoring system. Reusability is a significant characteristic of a high- software component quality. Programmers should plan and execute software components in such a way that several various programs are reusing them. In addition, testing of component-based usability is measured as software components directly cooperate with users. It takes important attempt and responsiveness to write a software component that is efficiently reusable. The component requires be completed documenting and systematic testing, designing with an alert that it will be put to unexpected utilizes CBSE combines fundamentals of software architecture, software verification, configuration, modular software design and exploitation. To promote exchange and teamwork with the community of software architecture, with the Quality of Software Architectures Conference (QoSA) CBSE is co-located and the International Symposium on Architecting Critical Systems (ISARCS) as an element of the event of federated CompArch.

B. Feature Oriented Reuse Method (FORM)

FORM is FODA (Feature Oriented Domain Analysis) extension which goals to cover analysis of the domain and development of core assets and which goals to maintain a view of business on the improvement of software product line. The concept of utilizing a feature model for requirements engineering was introduced in FODA. FORM expands FODA to the software plan and execution phases, and dictates how the feature model is employed to expand domain components and architectures for reuse. In a specified domain FORM starts with a commonality analysis amongst applications. During the analysis the model built is known as feature model.

A feature model contains

1. A further illustration, which is a graphical AND/OR features hierarchy that takes logical structural relationships, for example composition and generalization between features. There are three kinds of relationships are represented in this illustration: generalization / specialization, composed-off, and implemented-by.
2. Rules of Composition that enhancement the feature illustration with mutual dependence and shared exclusion relationships.
3. Issues and decisions that record different trade-offs, justifications and rationale for selection of feature.

The utilization of features is provoked by the information that customers and engineers frequently speak of the characteristics of the product in terms of features the product contains delivers of AND/OR. They communicate functions or requirements in terms of features and features are characteristically particular functional abstractions that must be executed, tested, delivered, and maintained. This information, conversely, has not been maintained or exploited entirely by the majority of software engineering techniques so far.

There are various kinds of features that are of concern relying on the interest one might have in system improvement. Users, system analysts, and developers are all frequently involved in system improvement and have various interests. Users are regarded more about the functions or services offered by the system (*i.e.*, service features) and system analysts and designers are regarded about domain methodologies (*e.g.*, navigation techniques in the avionics domain, techniques of finance transmit in the banking domain) and developers are regarded about implementation methods (*e.g.*, databases, sorting algorithms). Applications are not constructing unless a decision of sound is prepared among them that is create an executable and reliable set of features. The chosen set of features is to restrain the feasible architectures space.

FORM Engineering Process

Figure 2 shows the FORM engineering process. The FORM engineering process has two engineering process that processes are domain and application engineering process. Domain engineering purpose is to

increase domain artifacts that might be utilized in improving applications for a given domain. The domain engineering contains operations for assembling and describing information on systems that distribute a common set of data and capabilities. In a further complete manner the domain knowledge is engineered and organized and it gives the user recognizable and selectable frequent features and reference software system designs of the aim domain in which reusable component roles are clearly defined in terms of their places in FORM.

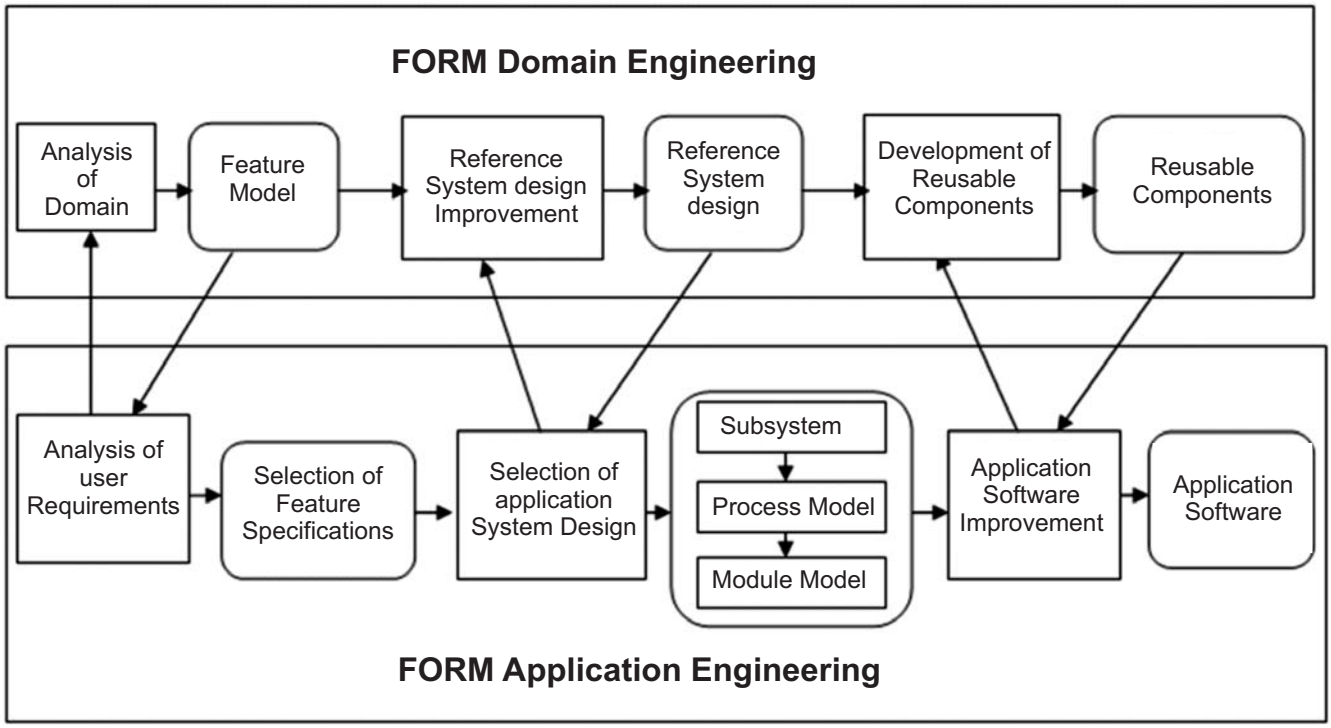

Figure 2: FORM Engineering Process

An application engineering is a process of improving a precise application creating utilization of the domain knowledge obtained through domain engineering (i.e., through discovering an accurate reference system design and plugging in reusable software components) in FORM. By first recognizing requirements of user, application engineering progresses and choosing suitable and valid domain features from the feature model, recognizing the corresponding reference model, and by reusing software components, finishing the application improvement in a bottom-up fashion.

C. Refactoring

Refactoring is the process of enhancing the existing code design by modifying its internal structure without disturbing its behavior of external. The badly designed code is complex to sustain, analysis and execution and therefore the software reduces quality. That's an abnormal turn of expression. In our present perception of software improvement, we consider that we design and then we have to write to code. An excellent design comes primarily and the coding comes next. Over time the code is altered and the system integrity, its structure according to that design and regularly fades. The code gradually goes down from engineering to chopping.

It is the differing of this practice. We can take a bad design, chaos even and rewrite it into elegant code in the refactoring process. Every step is easy, even unsophisticated. We move a field from one class to another class and drag a number of codes out of a technique to create into its own technique and drive a number of codes up or down a hierarchy. However the collective result of these small modifications is fundamentally developing the design. It is the precise normal notion reverse of software decompose. We can discover the balance of work alters with refactoring. We can discover that design, rather than occurring every up front, happens continuously during the improvement of software. In this process, we

can study from constructing the scheme how to develop the design. The interaction of resulting is directed to a program with a design that remains good as improvement continues.

Example for code refactoring based on feature to Extract Super-class

Figure 3 shows the code refactoring using features. In this process, when we discover two or more classes that distribute common features, regard as abstracting those distributed features into a super-class. Yet again, this creates it simpler to bind classes to an abstraction, and eliminates duplicate code from the unique classes.

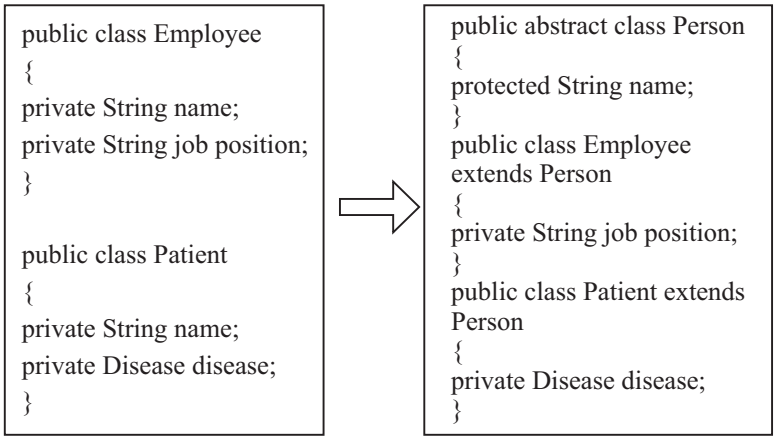

Figure 3: Code refactoring using features

In this paper, not only a program, we also focus on the Feature model refactoring with a feature oriented reuse method to reuse the business components. SPL refactoring have not only program refactoring, but also Feature Model refactoring. Based on this description, we propose a FORM method with Feature Model refactoring. A feature model refactoring is a transformation that develops the feature model quality by developing its configurability.

Figure 4 represents two small Feature Models. It defines the mobile colors. A mobile can be white or black in the left-hand side (LHS) Feature Model. Assume that we would like to refactor the LHS model to the RHS model by adding a new substitute color. Therefore, we can have an additional color blue mobile in the resultant model, as still sustaining the earlier configurations. We show that the resultant Feature Model develops the primary Feature Model configurability for assuring accuracy of the refactoring described in Figure 4.

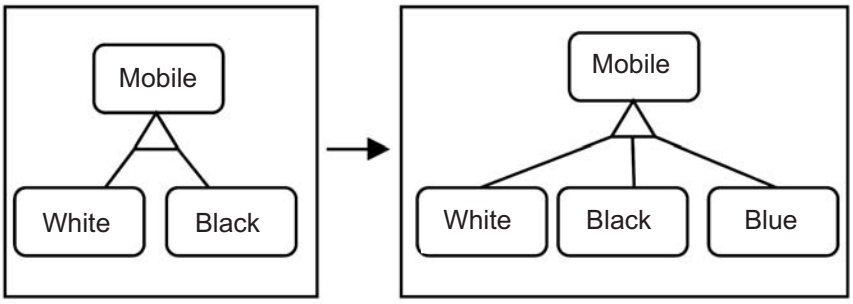

Figure 4: Example of Feature Model Refactoring

The LHS Feature Model has two applicable configurations: {mobile; White} and {mobile; Black}. The RHS Feature Model contains similar configurations of the LHS Feature Model and additionally it has the configuration {mobile; Blue}. As the RHS model has every valid configuration of the LHS Feature Model, it is an applicable Feature Model refactoring. Subsequent to a related approach to establish Feature Model refactorings having significantly extra features, relations and formulas might be complicated, error-prone and time-consuming. In order to avoid that, we FORM with refactorings are presented.

D. Software quality attributes

In this paper, we mainly focus on five software quality attributes of the refactoring process to reuse the business components. The five software quality attributes are

1. Reusability
2. Security
3. Supportability
4. Testability
5. Maintainability

Software quality is the level to which software has a preferred combination of attributes (*e.g*. reusability and adaptability). This means that describing the quality of software for a system is equal to describing a list of software quality attributes of that organization. ISO/EIC 9126 standard defines the characteristics of software quality as a set of attributes of a software product by which its quality is defined and estimated. The factors that concern quality of software is categorized into two groups (*i*) factors that is directly calculated *i.e*., inner quality attributes (*e.g*. program length as lines of code) and (*ii*) factors that is evaluated only not directly, *i.e*., outer quality attributes (*e.g*. reliability and maintainability). The inner and the outer software quality attributes utilized in our classification and define how external quality attributes is evaluated utilizing inner quality attributes.

Reusability

Reusability is the possibility that a component is utilized in other components or scenarios to add novel functionality with slight or no modifications. Reusability reduces the components duplication and the execution time. Analyzing the frequent attributes between different components is the first step in constructing little reusable components for utilize in a superior system. There are many key issues:

1. The utilization of various codes or components to attain the similar result in different places; for instance, replication of related logic in many components, and similar logic duplication in many layers or subsystems. Observe the design of application to recognize the common functionality and execute this functionality in separate components that we can reuse. Observe the design of application to recognize crosscutting regards such as logging, validation and authentication and execute these functions as divided components.
2. The utilization of multiple same techniques to execute works that contains only a small difference. Instead, utilize parameters to differ the performance of a single technique.
3. Utilizing many systems to execute the similar feature or function instead of distribution or reusing functionality in an additional system, across several systems or across various subsystems within an application. Through service interfaces, consider disclosing functionality from components, layers, and subsystems that other layers and systems can utilize. On various platforms employ platform agnostic data types and structures that is accessed and understood.

Security

Security is the system's ability to decrease the malicious chance or unintended actions outside of the designed procedure disturbing the system and avoid revelation or information loss. Enhancing security also enhance the system reliability by minimizing the chances of an attack following and impairing system process. Secure a system should have protected assets and avoid accessing of unauthorized to or information alteration. The factors disturbing security of the system are integrity, confidentiality and availability. The features utilized in protected systems are authenticated, logging, encryption and auditing.

Supportability

Supportability is the capability of the system to give information, supportive for recognizing and resolving issues as it fails in effort properly.

Testability

Testability is evaluated on how well system or components permit us to make examination criterion and implement tests to establish if the criterion are met. Testability permits faults in a system to be separated in a timely and efficient manner.

Maintainability

Maintainability is the capability of the system to undertake modifications with an amount of effortlessness. These modifications can impact features, services, components and interfaces as adding or altering the functionality of the application in order to fix bugs or to assemble novel business necessities. Maintainability also affects the time it gets to reinstate the system to its prepared status subsequent a failure or removal from operation for improving. Enhancing system maintainability is to enhance availability and reduce the effects of run-time imperfection.

4. OVERALL PROPOSED SCHEME PROCESS

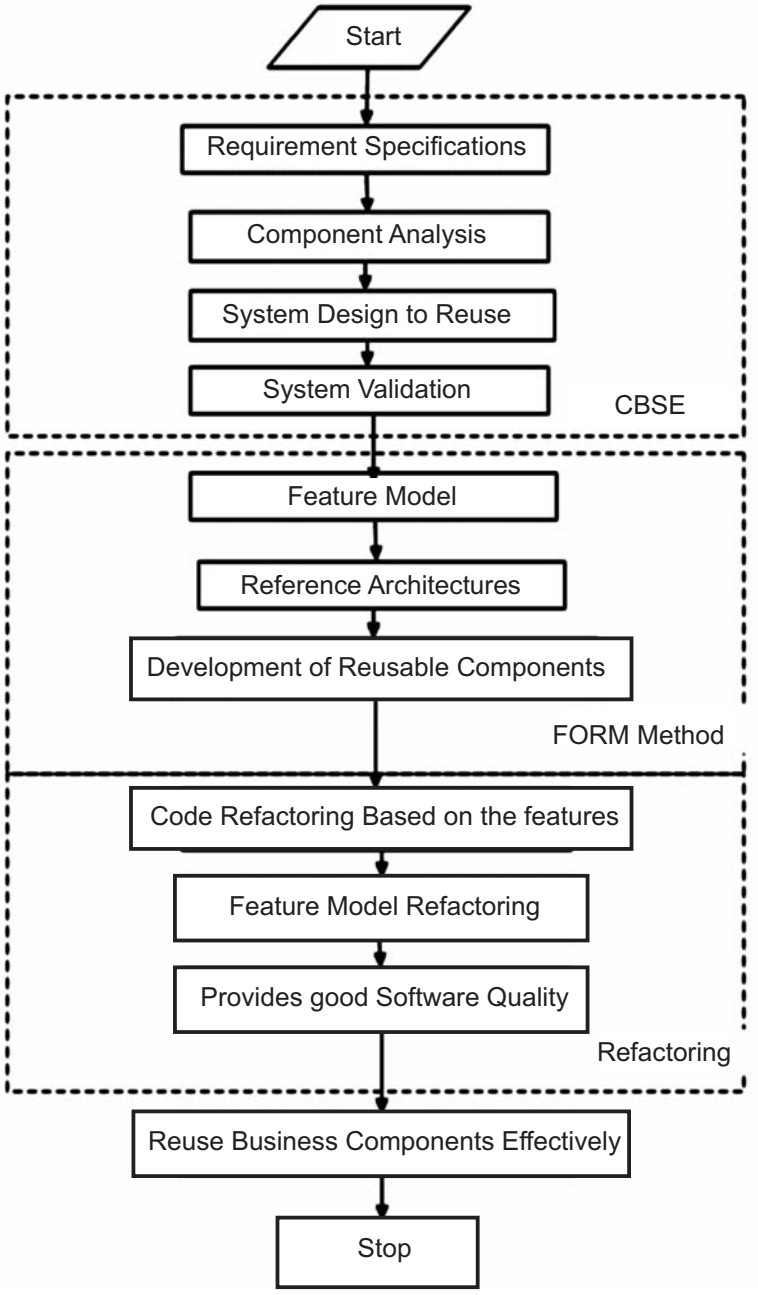

Figure 5: Overall Process Scheme

Figure 5 shows the overall process of our proposed scheme. In this process, initially specify the requirement specifications for reuse purpose. The components are analyzed in Component Based Software Engineering. To construct the architecture or system design to reuse business components after components are analyzed. The System is validated in CBSE.

The Feature Oriented Reuse Method is performed after CBSE process. In FORM method, the feature model is utilized and reference architectures are used for the development of reusable components. These reference architectures are described from three viewpoints such as subsystem, process and module and it is close involvement with the features. The domain analysis is performed in FORM method in the refactoring process. The Feature Oriented Reuse Method is applied before the refactoring process. In this scheme, the components are reused based on their features. Refactoring is applied after the FORM method is used. Refactoring is the process of modifying the internal structure of code, with changes of external behaviors. To refactor the code based on their features to provide good quality and to reduce the complexity. And the feature model has also refactors. Finally, we will get reusable business components. Our proposed scheme will give good software quality attributes like reusability, testability, security, supportability and maintainability.

5. CONCLUSION

In this paper, we focus on efficient refactoring method and this method for single software systems contains well-established meaning. Refactoring techniques are applied to develop the software quality attribute, but the refactoring effect on exacting quality attribute is still indefinite. In this scheme, CBSE with refactoring is performed using feature oriented reuse model is presented to provide the best software quality attributes to component based software improvement. The CBSE is used to reuse business components. CBSE is a reuse-based approach to describing and executing freely united components into systems. In the CBSE, reusable components are analyzed and architectures are designed to reuse approach. The FORM method is applied to reuse the feature effectively after CBSE process is completed. With a commonality analysis amongst applications the FORM method is started. During the analysis the model built is known as feature model and it takes unity as an AND/OR graph. In this FORM feature model, AND graph specifies mandatory features and OR graph specify alternative features selectable for various applications. After that, this model is utilized to describe parameterized reference architectures and suitable reusable components during actual software improvement. The reusable components are developed during the feature model is improved using the FORM method. Finally, refactoring method is applied to refactor the code effectively to give the quality of the program and also feature models are refactored. This scheme can provide good software quality attributes. In this scheme, not only is the program quality and also the quality of the feature model is improved to reuse the business components efficiently.

6. REFERENCES

1. Alshayeb M. (2009), “Empirical Investigation of Refactoring Effect on Software Quality”, Information and Software Technology Journal, vol. 51, PP.1319-1326.
2. Alshayeb M. (2011), “The Impact of Refactoring to Patterns on Software Quality Attributes”, The Arabian Journal for Science and Engineering, vol. 36, PP. 1241-1251.
3. Apel S., Batory D., Kastner C., and Saake G. (2013). “Feature-Oriented Software Product Lines Concepts and Implementation” .Springer publications.
4. Apel S., Kastner C., and Lengauer C. (2013), “Language-independent and automated software composition”, The Feature House experience. IEEE Trans. Software Engineering,Vol.39, No.1, PP.63-79.
5. Batory D (2005), “Feature models, grammars, and propositional formulas”, 9th International Conference of Software Product Lines, Vol.3714 of Lecture Notes in Computer Science, PP. 7-20.
6. Benavides D., Ruiz-Cortes A., and Trinidad P. (2005). “Automated reasoning on feature models”, Advanced Information Systems Engineering (CAiSE), Vol.3520, PP. 491-503.

7. Berger C., Rendel H. , and Rumpe B. (2010),"Measuring the ability to form a product line from existing products", VaMoS, PP. 151-154.

8. Bryton S. and Abreu F. (2009), "Strengthening refactoring: towards software evolution with quantitative and experimental grounds", 4th International Conference on Software Engineering Advances, PP. 570-575.

9. Elish K. and AlshayebM. (2011), "A Classification of Refactoring Methods Based on Software Quality Attributes," The Arabian Journal for Science and Engineering, vol. 36, PP.1253-1267.

10. Francisco ZigmundSokal, Mauricio FinavaroAniche and Marco Aurelio Gerosa, (2013), "Does The Act Of Refactoring Really Make Code Simpler?, A Preliminary Study

11. Gheyi R., AlvesV.,MassoniT., KuleszaU., BorbaP., and Lucena C. (2006), "Theory and proofs for feature model refactorings", Technical ReportTR-UFPE-CIN-200608027, Federal University of Pernambuco.

12. Gunter Bockle, Klaus PohlanFrank van der Linden (2005), "Software Product Engineering Foundations, Principles and Techniques", Springer publications. PP.19-37.

13. Karim O. Elish and Mohammad Alshayeb (2012), "Using Software Quality Attributes to Classify Refactoring to Patterns", Journal of Software, Vol. 7, No. 2, PP.408-419.

14. Kayarvizhy, N. and Kanmani, S., (2011) "Analysis of quality of object oriented systems using object oriented metrics," Electronics Computer Technology (ICECT), vol.5, PP. 203-206.

15. Liu J., Batory D., and Lengauer C. (2006), "Feature oriented refactoring of legacy applications", In Proceedings of the 28th International Conference on Software Engineering, PP.112-121.

16. Marcel FoudaNdjodo and AmougouNgoumou (2009), "The Feature Oriented Reuse Method with Business Component Semantics", International Journal of Computer Science and Applications, Vol. 6, No, 4, PP. 63 – 83.

17. Manimekalai S, (2016), "State of the Art with Cooperative Approach for Software Product Lines in IC Reengineering", Middle-East Journal of Scientific Research 24 (2):271-278.

18. Sven Apel and Christian Kastner (2009), "An Overview of Feature-Oriented Software Development", Journal of Object Technology, Vol.8, No.4, PP.1-36.

19. Wolfram Fenske, Thomas Thum and Gunter Saake (2014), "A Taxonomy of Software Product Line Reengineering", Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems, No.4.

20. YaserGhanam and Frank Maurer (2010), "Extreme Product Line Engineering – Refactoring for Variability: A Test-Driven Approach", Agile Processes in Software Engineering and Extreme Programming, Vol.48, PP.43-57.

21. Stefan Ferber, Jurgen Haag and JuhaSavolainen (2002), "Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line", Springer publications, PP. 235-256.