Design a New Optimal Process for Selection of Software Components with Selection Factors

Arvind Kumar
Research Scholar
Computer Science Engineering Department
JJT University, Jhunjhunu, Rajasthan, India
k.arvind33@gmail.com

Dr. Surender
Dept. of Information Technology, HCTM Technical Campus, Ambala Road, Kaithal, Haryana, India
ssjangra20@rediffmail.com

1. Introduction
Component-Based Software Engineering (CBSE) is a technology used to develop a complex software system using reusable software components according to user requirements which aims to satisfy the user satisfactions. CBSE is based on reusable software component that can be replaced and updated easily. CBSE provides to maintain a high level of quality and trust in system. CBSE offers a promising way to promote software reuse, decrease the complexity and time of the development process, via proper information encapsulation and separation of concerns at various levels. The basic unit in CBSE generally has a large granularity. Therefore, CBSE enhances the modularity of a software system at a much higher level. Software components often take the form of objects or collections of classes and objects from object-oriented programming, in some binary form. For large and complex application, some components need to be developed separately specifically tailored to the need of the application and some
components are selected from the third party repositories. So CBSE is an approach which is used to enhance the reusability with the development of component-based software from the pre-existing software components or with the components which is developed from the scratch. But when reuse a pre existing software component, components selection play an important role to enhance the reusability. Component selection is a crucial problem in CBSE. This paper presents a new optimal process to select a subset of components for specific application domain or optimal components which fulfill the requirements of client.

2. Component Selection Problems

In component selection, a number of software components selected from a subset of components or from component repository in such a way that their composition satisfies a set of objectives. Components selection plays an important role in CBSD. Researchers and practitioners generally use the simple component selection problem and criteria based component selection problems.

2.1 Simple Component Selection Problem (SCSP)

Simple Component Selection Problem (SCSP) is the problem of choosing a number of components from a set of components such that their composition satisfies a set of objectives [1].

Consider SR is the set of the System Requirements as
\[
\text{SR} = \{r_1, r_2, \ldots, r_n\},
\]

SC is the Set of Components available for selection as
\[
\text{SC} = \{c_1, c_2, \ldots, c_m\}.
\]
Each component $c_i$ can satisfy a subset of the requirements from $SR$, $SR_{c_i} = \{r_{i1}, r_{i2}, ..., r_{ik}\}$.

The goal is to find a subset of components $Sol$ (obtained solution) in such a way that to every requirement $r_j$ from the set $SR$ can be assigned a component $c_i$ from $Sol$ where $r_j$ is in $SR_{c_i}$.

### 2.2 Criteria-Based Component Selection Problem (CCSP)

Criteria-Based Component Selection Problem (CCSP) is the problem of choosing a number of components from a set of components such that their composition satisfies a set of objectives and using various criteria. The goal is to find a set of components $Sol$ in such a way that to every requirement $r_j$ from the set $SR$ can be assigned a component $c_i$ from $Sol$ where $r_j$ is in $SR_{c_i}$, while minimizing the number of components in the solution $Sol$ and/or while minimizing $c_i \in Sol \text{ cost (ci)}$. Another criterion for selection of components in CCSP is dependencies involved between the components [1].

### 3. New Proposed Optimal Process for Component Selection

This paper proposes a new selection process with the following phases as shown in Figure 1 for software component which helps in selecting the subset of components from repositories and optimal selection of software components from subset of components of specific application domain.

#### 3.1 Domain Engineering

Domain engineering is a mechanism used to identify and develop a subset of software components that have applicability to existing and future software in a specific application domain [2]. The main aim is to develop a mechanism which helps in identification of software components and to reuse them for CBSD.
Domain engineering includes the domain analysis, design and implementation process which helps in identification and selection of specific application domain of component-based software. Domain analysis is the identification, analysis, and specification of common, reusable components capabilities within a specific application domain. The main aim of domain analysis is to find those components that are broadly applicable, so that may be reused and produce the domain model. Domain design aims to produce architectural patterns which solve a problem common across the systems within the domain, despite differing requirement configurations [3]. Domain implementation is simply generated a process and tools for efficiently generating a customized program in the domain.

3.2 Software Analysis and Specification

The software analysis is the activity that feeds information to the specification activity. During analysis a complete and consistent set of specifications emerge for the software development through answer to questions, questionnaires, information from documentation. The software analysis phase emphasizes identification of high-level components in a real-world application and decomposition of the software system. The software analysis phase demands the systems analyst to study the application and its constraints, understand the essential features of the system, understand the requirements expected to be satisfied by the software system and create an abstract model of the application in which these requirements are met. The main product of the software analysis phase is a graphical or textual description of an abstract model of the application which helps in figure out its subsystems and major components. This phase is not concerned with the details of the components when we compare functional decomposition. This abstract model of the application comprising high-level abstractions of software components is
better understood for software specification. The software specification is instead to establish what services are required from system, for system operation and development [4]. Software specification is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirement which help in design of components and selection of components from third party repositories for development of CBSD [5].

3.3 Search Algorithm for Set of Components

In this phase, this paper presents following steps and try to convert these steps in the form of algorithm for selecting the subset of components in future work. These steps help in selection of pre-existing subset of components from third party repositories with three following sublevels.

- According to domain engineering, software analysis and specification, this process helps in selecting components from repositories of specific application domain.

- Then apply the approximation algorithm on repositories to select all those subset of components which are available in repository according to software specification and also according the user requirements.

- In subset of components, if component which satisfy the user requirements are not available in repositories according to the specific domain than try to develop all those components from scratch.
Figure 1: New Component Selection Process
3.4 Component Repository

Component repository is a collection (store) of software components and contains the related information about components. These information are name, hardware platform, and required operation system and programming language (high level language, low level language) etc. The main aim of repository is to reduce the development time, cost, risk and improves the quality by storing the reusable software component with good quality.

3.5 Selected Set of Components

Steps of how to select the component from repository has already been described in paragraph 3.3 which helps in selection of pre-existing subset of components from third party repositories. Now this study helps in selection of optimal components from the subset of components of specific application domain with the two processes or algorithms namely optimal component selection and Greedy algorithm.

3.5.1 Optimal Component Selection

This optimal selection process helps in developing an algorithm to select the optimal component according to the user demand even after completion of domain engineering, software analysis and specification from the subset of components. In this phase, select the software components according to the user specific demands. These demands may be used to provide the better performance, high security, reliability and reusability with less time and cost in component selection. This paper summarizes few factors which help in selection of optimal components selection after completing the subset of component selection.
• **Performance**

The degree to which a system or component accomplish its designed function within given constrains such as accuracy, availability, efficiency, response time, recovery time, resource usage, speed etc [6]. Performance can be increased by selecting those components which contains high cohesion, less coupling and less number of interfaces of components according to the following equation.

\[
\text{Performance} \propto \frac{1}{\text{Interface}} \propto \frac{1}{\text{Cohesion}} \propto \frac{1}{\text{Coupling}}
\]

• **Time**

Software development activities require more time for best software quality. But users want to reduce the development time. So this study proposes to use maximum number of COTS component because COTS components save the development and testing time and improve quality according to the following equation time and COTS are inversely propositional to each other.

\[
\text{Time} \propto \frac{1}{\text{COTS}}
\]

• **Size**

Components size depends on programming language and components may be in high level or low level language. User wants to reduce the size of system so try to select those components which use high level language. It means, select those components which are developed in high level language according to the following equation size and high level language are inversely propositional to each other, because high level language try to reduce the size of software.

\[
\text{Size} \propto \frac{1}{\text{Programming Language (High Level Language)}}
\]
• Fault Tolerance

The ability of system or component to work for long time continuously without any hardware or software faults. Fault tolerance can be increase if mean time to failure will be increased according to the following equation fault tolerance and mean time to failure is directly propositional to each other.

\[ \text{Fault Tolerance} \propto \text{Mean Time to Failure} \]

3.5.2 Greedy Algorithm

Greedy algorithm is playing main role in component selection. Greedy algorithm looks for locally optimal solution and assumes it as best but they do not always yield the optimal solution because it never backtracks or changes past choices. Greedy techniques are used to find optimum components and use some heuristic to generate a sequence of sub-optimums that hopefully converge to the optimum value. Once a sub-optimum is picked, it is never changed nor is it re-examined [7], [8], [9].

3.6 Development of Components from Scratch

When subset of components and optimal components are not available according to user demands, than researchers and developers think of to start development of software component from the scratch. A software component is built to be reused and reusability implies generality and flexibility, and these requirements may significantly change the component characteristics. The development process of building components can follow an arbitrary development process model as shown in Figure 1. The generality requirements imply often more functionality and require more design and development efforts and more qualified developers. The component development will require more efforts in testing and specification of
the components. The components should be tested in isolation, but also in different configurations. Finally the documentation and the delivery will require more efforts since the extended documentation is very important for increasing understanding of the component [5].

3.7 Component Composition

Component integrated into an architecture style and interconnected with an appropriate infrastructure that allows the component to be coordinated and managed effectively as shown in Figure 1 from repository. It focuses on selecting a collection of reusable components or frameworks from specific application domains. There are differences in the mechanisms used to achieve reusability when different kinds of reusable components are involved. The most basic software components are often reused by composition, which can be seen as a process of building a piece of software from elementary self-contained components; although reusability is naturally accomplished by reusing classes through inheritance during object-oriented development. This phase is usually akin to sifting through a junkyard of books rather than visiting a library. Testable component assembled together to build a component-based software for particular application domain by using all testable components according to domain engineering, software analysis and specification of user. [5]

3.8 Software Testing

Software Testing is the Process of exercising or evaluating a system by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results.
3.8.1 Integration Testing

Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them [9]. Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design [10].

3.8.2 System Testing

System testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing as shown in figure 1 is the process of verifying that the system components work together as described in the design specification. After a collection of components has been unit-tested, the next step is ensuring that the interfaces among the components are well defined. CBST is an important activity of CBSD cycle and is based on two main perspectives. First one is component providers that refer to the testing of components by their developers on the basis of the source code and the second one is the component user that refers to the testing of components by their users, without access to their source code [5].

3.9 Deployment

The deployment phase is characterized by the implanting the component together to build a complete software at user site. The CBS must be easily configurable or adaptable at user site. Some components picked out during the implementation phase should undergo further refinements, e.g. treatment of exceptional conditions and verification, until they become generic and robust enough to be placed in a
reusable library. This surely adds an overhead to software construction, which is more than compensated for by the long term savings when such components are reused in future projects. Implementation and deployment is almost the end of system development, now the system is ready to be presented to the customer. [4]

3.10 Maintenance

The process of modifying a software system or component after delivery to correct faults, improves performance or other attributes, or adapt to a changed environment. Software is normally subject to continuing changes after it is built, when it is operational. Thus the efforts turn now to the challenge of maintaining a continually evolving system. During software maintenance, as shown in figure 1 change are introduced to a software system. Such changes are not meant only for correcting errors occurred in the operational software; these changes may be also for improving, updating the system to anticipate Future errors or adapting the system in response to a modification in the environment. Therefore, during the maintenance phase, Maintenance of software system does not only allow the software to evolve but also the reusable library concerning the existing systems expands during the maintenance of a legacy system. When the system is implemented and deployed at user site than the maintenance phase start. [5]

4. Conclusion

Optimal selection process of components not only improves selection process and productivity but also has a positive impact on the quality and maintainability of software products. The new optimal process supports quality software development with algorithms to select the optimal component according to the user demand even after completion of domain engineering, software analysis and
specification from the subset of components. Initially, the software engineer identifies potentially optimal components from existing reusable component repository. In addition, a repository should address the problem of conceptual closeness to retrieve components that are similar to but not exactly the same as the desired one, therefore this process addresses the concerns of developing family of software systems by using the new optimal selection process for optimal components, thus it has great applicability in CBSD.

5. References


