Study of Perfective Maintainability for Component-based software systems using Aspect-Oriented-Programming Techniques

JyothiR and Dr. V. K. Agrawal

Abstract—As Maintainability plays a very important role in the software development life cycle, we propose to use Aspect-oriented-programming technique “AspectJ” to improve the maintainability of component based software system in this paper. This method helps in dealing with the issue of separation of concerns. Once the software product is developed and put into use, major effort for maintenance goes to modifications/enhancement of the different components in a system. Perfective maintenance deals with the changes resulting from enhancements of the software systems. For better understanding of the perfective maintenance, we are proposing a process model which gives step-by-step procedure to identify all the components and sub-components whenever any modifications/enhancements are made to the components.

Keywords—Aspect-oriented programming, Component-based Software Systems, Maintainability, Separation of Concerns.

I. INTRODUCTION

MAINTAINABILITY occupies the last phase in the software development life cycle. Since it is the longest phase in the software development life cycle, maintainability plays an important role in the software development life cycle [1]. Maintainability can be categorized into three different view-points: they are set of activities necessary to accomplish the maintenance phase, costs associated with maintainability and also different problems that may encounter when software maintenance is undertaken. Majority of the software development cost goes to maintenance phase. Maintainability requires a different approach to deal with component-based-software systems, because same components may be used by several other components in the system. Thus, if we make changes to any one component, impact of that should be analyzed on all the other components which are depending on this changed - component in the system. In addition to the above, maintainability has a different dimension, such as: changes required correcting the bugs, changes required enhancing the components, and changes required to update the existing components. Each of these requires specific treatment.

Aspect-oriented-programming technologies aim to improve system modularity by modularizing crosscutting concerns. These are the concerns that span across multiple modules in a program. In several programs, global properties of design issues lead to crosscutting concerns. This problem can be overcome by using a separation of concerns through concepts of Aspect-oriented-programming (AOP) [3]. Architectural slicing is a static analysis-slicing technique presented in the literature [2]. This technique can be used to find all the objects affected by changes. However, this technique suffers from common problems of Object-Oriented programming such as crosscutting concerns [3]. In AOP we have found some improvements over object-oriented programming like encapsulation of data with its associated behavior in a class. AOP introduces langugemechanism for identifying and capturing crosscutting concerns and it is considered as a good candidature for modularizing the different aspects of concern in a system. It provides a mechanism for encapsulating crosscutting concerns into modular units. It also eases with the situations that involve code- tangling [3].

Maintenance consists of four parts: i) Corrective maintenance ii) Adaptive maintenance iii) Perfective maintenance and iv) Preventive maintenance [1]. Corrective maintenance deals with the repair of faults or to fix bugs in the code. Other 3 maintenance types can be viewed as a software evolution; this evolution is required for adjusting to the new requirements. Examples of such evolution are the updation of the software to cope with the changes to user requirements, such as hardware or operating system changes in the environment. Perfective maintenance deals with the accommodation for new or changed user requirements. It focuses mainly on the functional enhancements to the system and also the activities to increase the system’s performance. Among various maintenance categories mentioned above, major effort goes to perfective maintenance. Hence improvements in the perfective maintenance result into a remarkable progress in the overall maintainability of the system. Existing process models provide a single general process model for all the categories of maintenance, they do not provide required performance and may also lead to many inconsistencies [11]. These models do not provide enough information towards the deep understanding of each of the maintenance category. To overcome those problems, we are proposing a process model for perfective maintenance based on aspect-oriented-programming: AspectJ. AspectJ is an
aspect-oriented extension to java. As we discussed in the above section AspectJ attempts to aid programmers in the separation of concerns i.e., the breaking down of a program into distinct parts that overlap in functionality. Aspect-oriented programming AspectJ focuses on modularization, and encapsulation of cross-cutting concerns. AspectJ works both as a language specification and language implementation. As a language specification, it defines varied constructs and their semantics to support aspect-oriented concepts. As a language implementation it offers tools for compiling, de-bugging, and documenting code. Aspect-oriented programming is advantageous over object-oriented programming, such as, it enhances the performance of component-based software systems. Process model for perfective maintainability using AOP AspectJ concepts gives better throughput. It provides reliability and also it enhances the modularity. Using aspect-oriented programming AspectJ, we can improve the design, source code, modularity of crosscutting concerns and also the development process.

II. INTER & INTRA LEVEL COMPONENTS INTERACTION AND DEPENDENCY AMONG DIFFERENT COMPONENTS IN A COMPONENT-BASED SYSTEM.

Perfective maintainability is a process of adding new features to the existing software systems for improving performance [11]. This can be achieved at 3 different phases: i) In the first phase, good understanding of the existing software components design. ii) During second phase we concentrate on the reverse engineering process, in which software design of the software product is re-examined and restructured. iii) In the third phase, testing and debugging must be done to check the validity of the software product. As we mentioned above in the first phase, good understanding of the existing software components design includes having the complete knowledge of what the component-based software system does. Identifying where the changes are to be effected in the system, and in-depth knowledge about how the different correlated components can be modified according to the user requirements. In order to make modifications or enhancements to the system successfully, the following points must be understood completely i.e., the limitations of the existing components, its effects on execution, relation of cause effect, relation of product environment and features of decisions-support. During reverse engineering process, analysis of a subject system to identify the system components, their interrelations and creation of representation for the system in another form or at higher levels of abstractions. In the next phase we can do acceptance testing and debugging of the software product to make new modifications/enhancements.

For maintainability of a component-based software system, understanding of an existing system is a must, such as how many components and sub-components are there in the system, what are the functionalities of each of these components, how they are interacting with other components that is inter and intra level interaction as shown in Figure 1. These sub-components interaction can be within the same component or different components of the system. Whenever any changes are made to the component, they must be identified, and the aspects affected by these changes must be found and should be updated during implementation, for that, it is better to identify dependency among different components. To identify the problems of code dependency among different subcomponents, we propose to use the concept of dependency digraph.

![Fig.1: Intra and Inter level components interaction.](image)

Dependency digraph is a digraph with set of vertices and edges [10], which shows the inter and intra dependency among different subcomponents. To find the dependency among different components, component-based software systems can be represented as a program digraph; component can be represented as program block. Each component has a set of inputs and a set of outputs. A program digraph is an abstraction, in which vertices represent the program block, and edges represent the dependency among the program blocks. Here we are representing set of component $C_1 = \{SC_1, SC_2, SC_3, SC_4\}$, where $SC_1, SC_2$, and $SC_3$ are subcomponents of component $C_1$. Figure 2 shows the dependency among different components i.e. program blocks. Subcomponents such as $SC_1$, $SC_3$ and $SC_4$ in component $C_2$ are dependent on the subcomponents $SC_2$, $SC_1$ and $SC_4$ respectively in component $C_3$. Subcomponents $SC_1$ and $SC_2$ depend on SC1 and SC3 respectively. Concerns are spanned across different components, which result into an interdependency among the components.

63
III. PROCESS MODEL FOR PERFECTIVE MAINTAINABILITY

This Process model represents the progress taken by the model of the process, and also gives the abstract representation of the process. Using the benefits of AOP AspectJ programming technique, the proposed process model has been constructed with the following 9 functional entities:

- Receiving Modification/Enhancement requirements
- Prioritization of requirements
- Procedure to Analysis/Isolation
- Identify Aspects
- Redesign of aspects
- Update modifications. Or Create new aspects
- Verification and validation
- Generation of report
- Repeat the procedure

Upon receiving the modification/enhancement requirements from the clients or users, these requirements are based on an addition of a new component, modification of an existing component. Process model prioritize the requirements, and based on this priority, it handles one modification/enhancement requirement at a time. Priorities are judged on the following parameters: Immediate requirement cost and time taken to fulfill the requirements, Importance, etc. During analysis/isolation procedure, process model analyses the cost benefit of the changed system, it also measures the time taken to understand the proposed enhancement to the system. The process model identifies the specific component required to modify/enhance, next, it locates the specific aspect in which it is necessary to change the code. During redesign, aspects are rewritten based on the requirements, in this step. With respect to the concepts of separation of concerns specific aspects are identified, since it has information about all the concerns which need to be modified. Once identified, the aspects modifications done to this will affect all the concerns related to the aspect. Here we changed only one aspect without spending much time in identifying and spending time in analyzing each and every class related to that concern. This process drastically reduces the time compared to OOP techniques. Design consists of redesigning the system based on the understanding of the modifications/enhancements necessary for the components using AOP AspectJ programming techniques. Outcome of this design results into new aspects with required modifications/enhancements or an updated version of the same aspects. Documents are generated for new requirements or updated version once the requirements are fulfilled. If the requirements are not fulfilled completely then it is required to repeat the procedure.

Following diagram shows the process model for perfective maintenance.

IV. SIMULATION OF PROCESS MODEL FOR PERFECTIVE MAINTENANCE

The simulation of Process Model for Perfective Maintenance is done using Petrinets. Petrinets are a Mathematical and graphical modeling tools first introduced in Carl Adam Petri’s dissertation [11] submitted in 1962. Major application areas for petrinets are Performance Evaluation, communication protocols and other interesting applications which include Modeling and analysis of distributed software systems. Petrinet is a type of directed, weighted, bipartite graph, consisting of 2 kinds of nodes called places and transitions [11], where interaction between places and transitions is shown by arcs. Graphically places are represented as circles and transitions as boxes, total number of places are represented as m, where initial marking is represented by Mo. with respect to software modeling in terms
of conditions and events, places represents conditions and transitions represents events. A transition has a certain number of input and output places representing the pre-conditions and post-conditions of the event.

As shown in figure 4, initially we need to identify the exact location in the components to change the code using component identification techniques. Then we need to find the aspects needed to be changed. Corresponding action must be taken to change the identified aspects, once we change the code we need to update the aspects. After this, we need to verify the model to check whether aspects are working properly. If model is working properly we can update the existing component with new changes, or we can create a new component with new version. If the model is not working properly we need to repeat the procedure. Time taken for overall maintainability has been reduced by using AOP AspectJ techniques of programming. Instead of spending much time in identifying all modules in a system, single aspect identification is sufficient. This aspect holds the information about all the concerns required to be modified/ enhanced. This reduces the time during identification of the specific modules in a component. Even during designing and implementation phase, specific aspect needs to be modified. With the concepts of aspects, time for these maintenance activities has been reduced and results in increase of overall performance of the maintainability. With the AOP AspectJ the maintainability becomes more modularized.

V. CONCLUSION AND FUTUREWORK

To provide better maintainability for component-based software systems, we have designed a process model for perfective maintainability using AOP AspectJ programming techniques. This model describes the benefits of aspects when any modifications/enhancements required to specific modules in component based systems are done. This technique reduces the time taken for maintainability and increases the performance of the component-based systems. We will use this description in our future work for maintainability of component-based software systems and architecture development. In future we will concentrate on generation of maintainability tool for all the categories of maintenance.

ACKNOWLEDGMENT

Our sincere thanks to Prof. K N B Murthy, Principal, Prof. Nitin V Pujari, HOD, Department of Computer Science and Engineering for their constant encouragement to carry out my research work.

REFERENCES

[1] Software Maintenance, Gerardo Canfora and Aniello Cimitile gerardo.canfora@unisannio.it, cimitile@unisannio.it University of Sannio, Faculty of Engineering at Benevento Palazzo Bosco Lucarelli, Piazza Roma 82100, Benevento Italy 29 November, 2000.
[10] Graph Theory with Applications to engineering and computer science, Narsingh Deo.
Fig. 4: Simulation of Process Model for Perfective Maintenance.