ijecs
open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume1 Issue 3 Dec 2012 Page No. 137-142

Maintainability of Software using Aspect Oriented Programming for Process quality Analysis

Dr.V.Khanaa. Krishna Mohanta

Dean Bharath University

Sri Sai Ram Engg.College

Corresponding Author: Dr.V.Khanaa Dean Bharath University

Abstract

Among all the phases in software development cycle, maintainability forms the key phase. Once the software is engineered, Software maintenance is the most effort and cost consuming part. A quality software must be adaptable to any real time working conditions. In a component-based system, different components are integrated, enrichment/improvement of a component to make it adaptable to prevailing conditions require more cost. This research presents the modeling work and prototyping techniques, which highlights the importance of project quality analysis for perspective maintainability. Here we are proposing a mathematical approach for time computation which is the sum of response time and time for solution generation. For efficient analysis we require high execution speed for handling complex algorithms and huge data volumes. For this we are providing aspect oriented programming techniques which increase the development speed, modularity that outputs quality products

Keywords— Perfective Maintainability, software maintenance, Aspect-Oriented Programming Technique, component-based System

I. Introduction

This document is a template. An electronic copy can be downloaded from the conference website. For questions on paper guidelines, please contact the conference publications committee as indicated on the conference website. Information about final paper submission is available from the conference website.

Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes. Software Development has many phases. These phases include Requirements Engineering, Architecting, Design, Implementation, Testing, Software Deployment, and Maintenance. Maintenance is the last stage of the software life cycle. After the product has been released, the maintenance phase keeps the software up to date with environment changes and changing user requirements.

Among four different types of maintainability namely corrective maintainability, adoptive maintainability, perfective and preventive maintainability, major costs go to the enhancement/modifications of components in component-based software systems [1].

1. Perfective maintainability establishes the change control procedure to initiate enhancement/modification request, to evaluate modification/enhancement request, to approve and implement changes to a baseline [3].

2. Process quality analysis of perfective maintainability is based on three parameters they are Time, Quality, and Efficiency

The earlier phases should be done so that the product is easily maintainable. The design phase should plan the structure in a way that can be easily altered. Similarly, the implementation phase should create code that can be easily read, understood, and changed. Maintenance can only happen efficiently if the earlier phases are done properly. There are four major problems that can slow down the maintenance process: unstructured code, maintenance programmers having insufficient knowledge of the system, documentation being absent, out of date, or at best insufficient, and software maintenance having a bad image. The success of the maintenance phase relies on these problems being fixed earlier in the life cycle.

Maintenance consists of four parts. Corrective maintenance deals with fixing bugs in the code. Adaptive maintenance deals with adapting the software to new environments. Perfective maintenance deals with updating the software according to changes in user requirements. Finally, preventive maintenance deals with updating documentation and making the software more maintainable. All changes to the system can be characterized by these four types of maintenance. Corrective maintenance is 'traditional maintenance' while the other types are considered as 'software evolution.'

As products age it becomes more difficult to keep them updated with new user requirements. Maintenance costs developers time, effort, and money. This requires that the maintenance phase be as efficient as possible. There are several steps in the software maintenance phase. The first is to try to understand the design that already exists. The next step of maintenance is reverse engineering in which the design of the product is re-examined and restructured. The final step is to test and debug the product to make the new changes work properly.

This paper will discuss what maintenance is, its role in the software development process, how it is carried out, and its role in iterative development, agile development, component-based development, and open source development.

II. Description of the Nature of the Phase

This section will cover what the software maintenance phase is about. As briefly seen in the introduction, software maintenance is not limited to the correction of latent faults. The term software maintenance usually refers to changes that must be made to software after they have been delivered to the customer or user. The definition of software maintenance by IEEE [1993] is as follows:

The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment. The following subsections will discuss different types of software maintenance, the significance and the characteristics of software maintenance.

A. *Four types of software maintenance*

There are four types of maintenance according to Lientz and Swanson: corrective, adaptive, perfective, and preventive [1980].

Corrective maintenance deals with the repair of faults or defects found. A defect can result from design errors, logic errors and coding errors (Takang and Grubb [1996]).

Design errors occur when, for example, changes made to the software are incorrect, incomplete, wrongly communicated or the change request is misunderstood. Logic errors result from invalid tests and conclusions, incorrect implementation of design specifications, faulty logic flow or incomplete test of data. Coding errors are caused by incorrect implementation of detailed logic design and incorrect use of the source code logic. Defects are also caused by data processing errors and system performance errors. All these errors, sometimes called 'residual errors' or 'bugs', prevent the software from conforming to its agreed specification. The need for corrective maintenance is usually initiated by bug reports drawn up by the end users (Coenen and Bench-Capon [1993]).

Examples of corrective maintenance include correcting a failure to test for all possible conditions or a failure to process the last record in a file (Martin and McClure [1983]).

Adaptive maintenance consists of adapting software to changes in the environment, such as the hardware or the operating system. The term environment in this context refers to the totality of all conditions and influences which act from outside upon the system, for example, business rule, government policies, work patterns, software and hardware operating platforms (Takang and Grubb [1996]). The need for adaptive maintenance can only be recognized by monitoring the environment (Coenen and Bench-Capon [1993]).

An example of a government policy that can have an effect on a software system is the proposal to have a ‘single European currency’, the ECU. An acceptance of this change will require that banks in the various member states, for example, make significant changes to their software systems to accommodate this currency (Takang and Grubb [1996]). Other examples are an implementation of a database management system for an existing application system and an adjustment of two programs to make them use the same record structures (Martin and McClure [1983]). A case study on the adaptive maintenance of an Internet application ‘B4Ucall’ is another example (Bergin and Keating [2003]). B4Ucall is an Internet application that helps compare mobile phone packages offered by different service providers. In their study on B4Ucall, Bergin and Keating discuss that adding or removing a complete new service provider to the Internet application requires adaptive maintenance on the system.

III. PROCESS MAINTAINABILITY ASPECT ORIENTED PROGRAMMING

Maintainability plays a very important role in the software development life cycle. Since majority of the software development costs goes to maintenance phase. Among four different types of maintainability namely corrective maintainability, adoptive maintainability, perfective and preventive maintainability, major costs go to the enhancement/modifications of components in component-based software systems [1]. Perfective maintainability establishes the change control procedure to initiate enhancement/modification request, to evaluate modification/enhancement request, to approve and implement changes to a baseline [3]. Process quality analysis of perfective maintainability is based on three parameters they are Time, Quality, and Efficiency. Aspect-oriented-programming technologies aim to improve system efficiency and 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) [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 [2]. In AOP we have found some improvements over object-oriented programming like modularization of data enhances the quality of software product. AOP introduces language mechanism 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; this mechanism provides an easy approach during the evaluation of the quality of perfective maintainability.

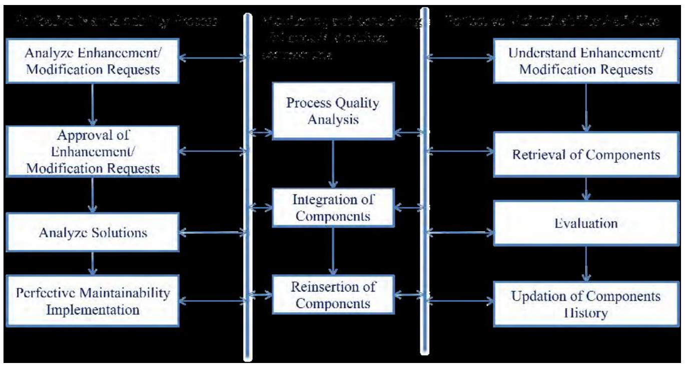
Analyze Enhancement/ Modification Requests
Approval of Enhancement/ Modification Requests
Analyze Solutions
Perfective Maintainability Implementation
Process Quality Analysis
Integration of Components
Reinsertion of Components
Understand Enhancement/ Modification Requests
Retrieval of Components
Evaluation
Updation of Components History

Fig. 1 image title

IV. ACTIVITIES FOR PERFECTIVE MAINTAINABILITY

Process activities included in perfective maintainability are

i) Understanding of the requested modification/enhancement.

ii) Determination of which software components should be retrieved to meet the modification/enhancement requirements.

iii) Evaluation of the requested requirements.
iv) Reinsertion of updated component.
v) Observing the workflow.

To perform the above activities perfective maintainability has the following responsibilities i.e., once the understanding of requested modification/enhancement is over, determination of particular components for the identified request should be retrieved. This retrieval of components consists of using a search method; search method is based on the classification of components and keywords. Once the retrieval of identified components over, next step is to evaluate the component. Evaluation of components is based on the some sub-activities i.e., well understanding of retrieved component version, investigation of different alternatives to fit into the required enhancement/modifications, approval for the requested enhancement/modifications are based on the requirements of the process quality. Once the approval of an identified request is done, propagation of changes must be made to the respective components. Again, we need to perform adaptation of enhanced components with other components in the component-based software systems. Process quality analysis must be made once the adaptation of components has been done. The following figure shows the set of activities involved in the administration of perfective maintainability.

V. PROCEDURE FOR PROCESS QUALITY ANALYSIS OF PERFECTIVE MAINTAINABILITY

Evaluation of process quality would be derived from the following parameters.

I. Time
II. Quality
III. Efficiency

I. Time: Measure of time comprises of two parameters; response time and average answering time. The analysis of complete source code efficiency and scalability provides a clear picture of the response time. This again depends on the type of enhancement/ modification request and also the associated components.

Average answering time is the time interval that elapses from the arrival of a request into the maintenance request buffer until the required enhancement/ modification has done to fulfill the user requirements. The time taken to fulfill the user requirements based on the type of enhancement/modification requirement. This can be calculated as follows.

$Tcr = \Sigma \; P(Ai) * T(Ai)$

Tcr-> is time taken to complete the enhancement/ modification requirements.

P (Ai) ->is the probability of receiving a good acceptable modification/enhancement request.

T (Ai) -> is the Mean Time for answering i.e., fulfilling the identified request. This is different for different requests.

This time taken includes the time taken for reconstruction of components.

II. Quality: Quality of perfective maintainability is the extent to which software system possesses desirable characteristics. Quality analysis is performed through qualitative and quantitative approaches [3]. The qualitative approach is based on the identification of critical programming errors that may encountered whenever any enhancement/modification request has been fulfilled. This approach involves proper analysis for correcting the errors in each of the aspects or components affected by that request. This also includes identification of aspects of all affected components. Identification of aspects is done by the concepts of separation of concerns. Design consists of redesigning the system based on the understanding of the modifications/enhancements necessary for the components using AOP Aspect J programming techniques. During redesign, identified aspects are rewritten based on the requirements. Once the aspects are redesigned, the modifications done to this affects all the other concerns related to the aspect. Outcome of this redesign results into new aspects with required modifications/enhancements and an updated version of the same aspects. Documents are generated for new aspects and also for updated version once the requirements are fulfilled. If the requirements are not fulfilled completely then it is required to repeat the procedure. In this redesigning process we changed only one aspect without spending much time in analyzing and identifying each and every class related to that concern, and this will drastically reduces the time compared to Object-oriented programming techniques.

III. Efficiency: Efficiency of component-based software systems requires analysis of some of the software engineering best practices and technical attributes, they are

1. Component-based software system architectural practices
2. Interactions among different components in the system
3. Coding practices for the software development
4. Compliance with Aspect-oriented programming and Object-oriented programming techniques.

Main aim of efficiency is to ensure centralization of clients' enhancement/modification requests and reduction of data flow among different modules in intra and inter workings of systems. Aspect-oriented programming (AOP) techniques impacts on code quality improvements. This technique enhances a system feature such as modularity, readability and simplicity. AOP techniques would provides a better modularization of crosscutting concerns, which leads to an implementation of a component-based software systems as a less complex and easily readable software [11]. This in turn increases the software development efficiency, so that the system would be maintained efficiently than object-oriented programming techniques. AOP affects software development efficiency in terms of the time needed to develop the system. Programming paradigms of AOP would simplify the development process, so that the software product may be created faster. Development Time is the metric, we identified here to measure the efficiency. Here we are defining active time as the time needed for software development by writing code and passive time as the time spends on the other activities concerned in the development of the software project. AOP affects efficiency of perfective maintainability in terms of Active Time (TA), and Passive Time (TP).Whenever any enhancement/ modification requests comes, if programmer have prior knowledge about following activities such as, how to handle the requests, what are the activities necessary to ful fill the requests, which parts of the source code needs to be changed and also about different modules and components where the changes have to be made, then the fraction of passive time in total time will become smaller. This will drastically increases the efficiency of component based software systems.

VI. SIMULATION USING PETRINETS

Petrinets are a Mathematical and graphical modelling tool, it was first introduced in Carl Adam Petri's dissertation in 1962[10]. This tool helps in modelling of concurrent systems, for that reason we are using this tool for concurrent evaluation of process quality activities of perfective maintainability for component-based software systems. Major application areas for petrinets are Performance Evaluation, communication protocols and other interesting applications which include Modelling and analysis of distributed software systems. Figure 2. Shows the simulation of well planned process quality analysis activities of perfective maintainability. This shows set of activities included in our proposed approach for process quality analysis of perfective maintainability. It also shows how effectively evaluation of process quality analysis of perfective maintainability has been done. As we discussed above evaluation of process quality analysis is based on the parameters namely Time, Quality, and Efficiency. When there is a request of enhancement/ modifications for an existing software product, initially we need to identify the exact location of the components in the component-based software systems. Once, locations of components are identified we need to retrieve those components to modify them according to the requirements. This identification procedure is based on component identification techniques such as search methods by several keywords. Next, we need to find the aspects which are needed to be changed within each of the retrieved components. Then evaluation has to be done to perform the process analysis based on Time, Quality, and efficiency. Corresponding results obtained must be updated in the history of components. In the next step it is necessary to monitor and control each of the components which may affected by these enhanced/ modified components. Again, we need to check with process quality analysis for each of those components. Then, possible measures have to be taken to integration and reinsertion of the enhanced/ modified component in to the system. Further, one can check with the entire process of perfective maintainability solutions, so that these maintainability implementations can be done. Usage of this aspect-oriented programming technique would likely to improve efficiency of perfective maintainability for component-based software systems.

VII. CONCLUSIONS

To provide better process quality of perfective maintainability for component-based software systems, we have proposed a set of activities necessary for a process quality analysis of perfective maintainability, by using three important parameter such as time, quality, and efficiency. The proposed approach helps in achieving better efficiency of perfective maintainability with the concepts of aspect-oriented programming, when any modifications/enhancements have been done In future performance evaluation of perfective maintainability for component based software systems could be a future area of research.

REFERENCES

[1] cimitile@unisannio.it University of Sannio, Faculty of Engineering at Benevento Italy 29 November, 2000.

[2] NTNU Empirical study of Component Based Software Engineering with Aspect Oriented Programming by Axel Anders Kvale Trondheim, 1. November 2004.

[3] Kiczales, G., Lamping, J., endhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.-M., and Irwin, J.: 'Aspect- oriented programming'.

[4] Proc. European Conf. Object-Oriented Programming (ECOOP 1997) vol. 1241 of Lecture Notes in Computer Science, Jyva¨skyla¨, Finland, June 1997, pp. 220–242.

[5] Lientz B., Swanson E., 1980: Software Maintenance Management. Addison Wesley, Reading, MA

[6] Lehman M. M., 1980: Program, Life-Cycles and the Laws of Software Evolution. In Proceedings of IEEE, 68, 9,1060-1076.

[7] Penny Grubb, Armstrong A. Takang, 2003: Software Maintenance: Concepts and Practice. World Scientific Publishing Company