IOSR Journal of Computer Engineering (IOSRJCE)
ISSN : 2278-0661 Volume 1, Issue 3 (May-June 2012), PP 01-08
www.iosrjournals.org

A Cost Estimation of Maintenance Phase for Component Based Software

Pragya Siddhi[1], Varun Kumar Rajpoot[2]
[1](*Software Engineering, Gautam Buddha University, Greater Noida, India)*
[2](*Software Engineering, Gautam Buddha University, Greater Noida, India)*

ABSTRACT : *Cost estimation of maintenance phase is necessary to predict the reliability, improve the productivity, project planning, controlling and adaptability of the software. Accurate estimation makes good understanding between the customer and user. Maintenance plays important role in various software developments like Iterative development, Agile development and Component Based Software Development (CBSD). In the recent year, software development turned into engineering through the introduction of Component Based Software Development. Though there are various models to estimate the maintenance cost of traditional software like COCOMO model, SLIM, Function Point (FP) model but till now there is no such model to estimate the maintenance cost of Component Based Software Development. In this kind of situation there is needed to develop a model to estimate the maintenance cost of Component-Based Software (CBS). This paper presents proposed basic maintenance cost estimation model for Component Based Software on the basis of COCOMO. This model is based on three parameters: Development cost of Component Based Software, ACT (Annual Change Traffic), Technical and Non-Technical factors which affect the maintenance cost of Component Based Software.*

Keywords: *ACT (Annual Change Traffic), Basic maintenance cost estimation model, Component-Based Software (CBS), Cost Estimation, Software Maintenance, Technical and Non-Technical Factors.*

I. INTRODUCTION

I.1 Software Maintenance:

Maintenance is the process of changing system software after it has been delivered to the customer [1]. In other words, Maintenance is about actions taken when a product does not function properly. Software maintenance workload is very large, although in different applications of its maintenance costs vary widely, but averagely, the maintenance costs of large software development costs as high as 4 times. Maintenance plays important role in various software developments like Iterative development, Agile development and Component Based Software Development. In the recent year, software development turned into engineering through the introduction of Component Based Software Development.

I.2 Component Based Software Development:

In the recent year, software development turned into engineering through the introduction of Component Based Software Development [6]. Component Based Software Development (CBSD) was a shift of paradigm from traditional software development to facilitate the software development in effective, faster and economical way by ensuring the reuse of software packages known as component or COTS(Commercial off the-shelf). Component Based Software Development provides a method of building the software system that makes use of reusable components. It also increases the reliability of the software when it is up and running. There are two main components to CBSD: The component architecture and component based development procedure. Component architecture is used as a standard for reuse software component. Some example of component models are EJB (Enterprise Java Beans) from Sun Microsystems, COM and COM+ from Microsoft and CORBA (Common Object Request Broker Architecture).Component Based Software Development moves the focus onmaintaining interaction between components rather than at the source code level, as the maintenance of the specific component spills to their developer. Component Based Software often consists of a set of self-contained and loosely coupled component that allow plug and play. The component may be implemented by using different programming languages and can execute in various operational platforms distributed across geographic distance. Some component may be developed in-house while other may be third party off- the-shelf components and source code may not be available to user and developers. When the source code is not available, the component is called implementation transparent [7].

For example, this is a .NET component:

```
public class MyClass
{
public string GetMessage( )
{
return "Hello";
}
}
```

When components are incorporated into a system, maintenance becomes much harder because source code is either partially or completely invisible. For example, most applications built for Windows NT use Microsoft's Foundation Classes. When you build a Windows NT application, you have effectively teamed with hundreds of Microsoft's developers. However, when the application needs maintenance, you become a one-person team. Clearly, component-based development forces us to rethink our maintenance technologies. If we are component vendors, for example, we must think not just of maintaining a block of source code in a specific application, but of maintaining code that is reused in numerous customer applications. Because each application may have slightly different requirements, component modifications may not work for all applications.
If we are component integrators, we must think in terms of technologies that will let us maintain the entire system. If the components are "black boxes," visibility is limited to documentation that describes the component's operation and functionality. Although maintaining unfamiliar code is a common maintenance dilemma, maintaining systems filled with black boxes adds a new level of difficulty.

Maintenance plays the important role in CBSD:According to SEI, maintenance of CBSD is different from the maintenance of custom built system in the following ways [2]:

- System developers do not have access to the source code.
- Maintenance and development is controlled by a third party.
- Maintenance is done at component level rather than the source code level.

Building a component for reuse purpose or changing an existing component to make it reusable may also add additional maintenance cost. In case study at NASA was found that a few change (12%) on single component increased the reuse cost by 55% compared to the development of particular component built from scratch. This cost is related not only with the physical changes, but also with the cost of understanding of requirements of component and cost of technical and non-technical factor which affect maintenance of Component Based Software. The component maintenance cost can be very high since the component must respond to the different requirements of different applications running in different environments, with different reliability requirement and require different level support. Maintenance is really the most important for reusable components, developers and service people. In enhancement, perfective and adaptive enhancement is also requiring in reusable component. In perfective, component may get user enhancement and improved documentation and in adaptive, changes to data input and file system is require if the reusable component is used in software which is again time consuming and increase the cost of software. Due to this reason toestimate the cost of Component Based Software Development is difficult.

I.3 Cost Estimation:

In recent years, software has become the most expensive component of computer system projects. The bulk of the cost of software development is due to the human effort, and most maintenance cost estimation methods focus on this aspect and give estimates in terms of person-months. Accurate software maintenance cost estimates are critical to both developers and customers. They can be used for generating request for proposals, contract negotiations, scheduling, monitoring and control. Underestimating the costs may result in management approving proposed systems that then exceed their budgets, with underdeveloped functions and poor quality, and failure to complete on time. Overestimating may result in too many resources committed to the project, or, during contract bidding, result in not winning the contract, which can lead to loss of jobs. Accurate maintenance cost estimation is important because [5]:

- It can help to classify and prioritize development projects with respect to an overall business plan.
- It can be used to determine what resources to commit to the project and how well these resources will be used.
- Projects can be easier to manage and control when resources are better matched to real needs.
- Customers expect actual development costs to be in line with estimated costs.

But there is no such model for estimating the maintenance cost of CBS.

II. PROPOSED MAINTENANCE COST ESTIMATION MODEL OF COMPONENT BAESD SOFTWARE

COCOMO (Constructive Cost Model) is used to estimate the cost of software projects. This model was developed by Barry W. Boehm and published in 1981 using data collected from 63 projects. We proposed a maintenance cost estimation model on the basis of COCOMO model to estimate the maintenance cost of component based software. The model which we have proposed is shown in Fig 1.

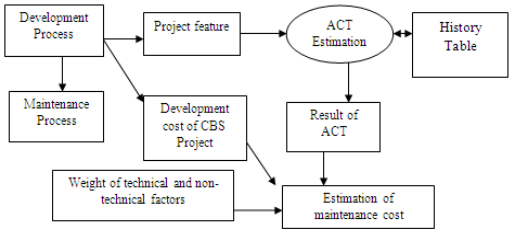

Fig.1: .Maintenance cost estimation modeling process

In this model there are two main elements: Development Process and Maintenance Process. Development Process is further divided into two elements: Project feature and Development Cost of CBS. Project feature includes the project characteristic. Annual Change Traffic (ACT) could be estimated using the both History Table which includes the data base and project feature. Maintenance cost of CBS could be estimated using the result of ACT, weight of technical and non-technical factors and development cost of CBS.

To estimate the maintenance cost of CBS three main parameters are used:

2.1) Development cost of CBS Project
2.2) Technical and Non-Technical factors
2.3) ACT(Annual Change Traffic)

II.1 Development Cost of CBS:

Component Based Software Development provides a method of building the software system that makes use of reusable components. CBSD includes the various phases: Requirement Analysis, Software Architecture AnalysisSelection and Evaluation, Component identification and Customization, System Integration, System Testing and System Maintenance. This element includes the overall cost of Component Based Software Development

II.2 Technical and Non-Technical Factors:

These are the various technical and non-technical factors which affect maintenance cost Component Based Software:

Technical Factors: These are the various technical factors are [3]:

- **Component Performance:** The performance of component based software depends on the performance of individual software. When the component is composed in to a system or component with some adaptation and integration efforts, it is necessary to count on minimal performance. A weight value 0 indicates not mandatory to have very efficient components, weigh value 1 indicates little change, weight value 2 indicates overall system performance is necessary, weight value 3 indicates component performance important but can be compromised for availability for example Web-based application, weight value 4 indicates very much performance based system or component and weight value 5 indicates zero-tolerance system like real time system.
- **End –user efficiency:** On the basis of end user can operate the efficiency of the system very well. This efficiency based on complexity of the system. A weight value 0 means the system is easy to use and no efficiency required from user, weight value little knowledge is required from end-user, weight value 2 means good knowledge is required to operate efficiency, weight value 3 means professionally trained user is required, weight value 4 means an experienced and trained professional is required and weight value 5 shows good technical skills required operating the system.
- **Internal Complexity:** It defines how much the internal working of component or system is complex. A weight value 0 means easy- to- integrate components are available that has very simple interface as well as simple implementation, weight value 1 indicates few complex components but simple implementation, weight value 2 indicates complex system, weight value 3 indicates a good complexity with overall easy-to-

integrate components, weight value 4 indicates Sophisticated system, weight value 5 indicates highly complex system which is not easy to integrate.

- **Component Reusability:**Component reusability means amount of code developed from scratch to be included in to the system and defines the extent of code reusability. A weight value 0 indicates no influence of reusability, weight value 1 indicates minimal influence of reusability, weight value 2 indicates overall design should be reusable, weight value 3 indicates few components should be adhere to it, weight value 4 indicates relatively high reusability is desired, weight value 5 indicates highly cohesive components which requires maximum reusability.
- **Third Party Integration:**It defines how many third party components are participating in system? COTS component reduce the development cost efforts but increases adaptation and integration cost. This factor depends how many components are gathered from in-house or shared libraries. A weight value 0 means no third- party components are used, weight value 1 indicates maximum in-house components, weight value 2 indicates hybrid of maximum third-party components and in-house components, weight value 4 indicates few component are integrated and mostly are developed and weight value 5 indicates components are integrated with few third-party components.
- **Usability**:Usability means ease of use. How much usability is required means how many GUI (Graphical User Interface) and features are used in the system. A weight value 0 no usability is required, weight value 1 indicates usability is required according to the requirements of the user, weight value 2 indicates usable which can be aided by documentation only, weight value 3indicates fairly usable, weight value 4 indicates higher user friendly for example web application and weight value 5 indicates very good user friendly environment with all features and graphics like Operating System.
- **Maintainability**: Maintainability means we can change implementation in system. A highly maintenance system accounts for overall high development effort. Frequency of change required after the system is deployed it should account for high maintenance and it also describes volatility of component i.e. how frequently component is changed. A weight value 0 indicates highly coupled system, weight value 1 indicates reconfigurable low coupling system, weight value 2 indicates reconfigurable, documented and standardized interfacing, weight value 3 indicates changes would not be largely localized, weight value 4 strong coupling between components and weight value 5 indicates high maintenance is required.
- **User Education:**Does the user require to be educated to operate the system? How complex is the system for user perspective is the key for this factor.A weight value 0 means no user education required, weight value 1 indicates no used education required only documentation is sufficient, weight value 2 indicates documentation with embedded help file is sufficient weight value 3 indicates an orientation program with practical demonstration, weight value 4 indicates user training is needed and weight value 5 indicates user training and also online help is required.
- **CASE Tools:**CASE (Computer Aided Software Engineering) tools are software programs that are designed to assist human programmers with the complexity of the processes and the artifacts of software engineering. CASE stands for a large number of applications reaching from simple editing tools to environments supporting the whole life cycle. CASE attacks software productivity problem at both ends of the life cycle by automating many analysis and design task, as well as program implementation and maintenance tasks. A weight value 0 indicates no CASE tools is required, weight value 1 indicates IDE used, weight value 2 indicates IDE with documentation software, weight value 3 indicates IDE with documentation and diagram tools, weight value 4 indicates CASE tools for design, development and testing and weight value 5 indicates maximum use of CASE tools for configuration management & project management .
- **Interface Complexity:** How much complex is interfacing of the components? If interface complexity is high then the maintenance cost of Component Based Software will be high. A weight value 1 indicates easy interfacing, weight value 3 indicates complex interfacing and weight value 5 indicates highly complex interfacing.

  These are the various technical factors with weighting value which affect the maintenance of Component Based Software [3] shown in Table 1.

Table 1: Technical factors

Technical Factors	**Weight**
Component Performance	0.5
End-User Efficiency	1
Internal Complexity	1.5
Component Reusability	1
Third Party Integration	1.5
Usability	1
Maintainability	2
User Education	1
CASE Tools	-0.5
Interface Complexity	1

Non-Technical Factors: These are the various non-technical factors [3]:

- **Domain Experience:** It defines how much people working in the project are familiar with domain and technical details of the project? Technologies change so fast that product managers must be skilled at quickly learning new technologies and solving problems in new domains. A weight value 0 indicates no experience is required, weight value 1 indicates simple awareness of the idea, weight value 2 indicates little knowledge of domain, weight value 3 indicates knowledge of domain and few are experienced, weight value 4 indicates most of the team has either experience or knowledge of the domain and weight value 5 indicates all are experienced.
- **Stable Requirements:** This factor describes clear requirement expectations of client. Client with highly volatile nature so this factor is to be given highest weight. A weight value 0 indicates quite stable, weight value 1 indicates very few changes in the requirements, weight value 1 very few changes in the requirements, weight value 2 indicates little changes 20% in meeting with customer, weight value 3 indicates Changes in requirement amounts up to 50%, weight value 4 indicates highly volatile requirements, changes around 60-70% and weight value 5 indicates no stability.
- **Part Time Workers:** Part time workers like consultants etc. add the value into development process. They input expert knowledge for the shortest duration they stay into the process. A weight value 0 indicates no part-time staff, weight value 1 indicates Very few part-time consultants; weight value 2 indicates approximately one-third of team is composed of part-time members, weight value 3 indicates approximately half of the team is of part-time experts, weight value 4 indicates mostly are part-time experts and weight value 5 indicates all are part-time staffers.
- **Technology Newness:**A potential cause to software risk is the newness of the technology being implemented. It is also affected by its volatility, which implies the frequency with which it keeps changing.A weight value 0 indicates easy technology and one week is required to pick up, weight value 1 indicates at least two weeks required to be familiar with the technology, weight value 2 indicates a month may be required to be familiar, weight value 3 indicates special training for the technology is required, weight value 4 indicates special training along with help during the project is required and weight value 5 indicate needs only skilled and experienced people.
- **Component Model:** In the absence of a defined criterion for component certification, the quality of a component can be determined by the quality of component model it follows. The standards, practices and procedures defined in a component model warranties the quality of component development. Lowest value indicates no component model followed and highest means a detailed, standardized and accepted component model. A weight value 1 indicate no component model, weight value 3 indicate old but

established component models like COM etc. and weight value 5 indicates latest and sophisticated component models are followed.

- **Internal Library Resources:**A successful component based development largely depends upon availability of suitable components. Presence of internal or shared library resources definitely lowers the efforts required in searching, retrieving and selecting components with respect to the given requirement set. A 0 value indicates no such resources whereas the maximum value is indicative of a detailed repository. A weight value 0 indicates no shared resources; weight value 1 indicates internally managed collection of components, weight value 2 indicates documented collection of reusable components, weight value 3 indicates small repository with shared resources and/or libraries, weight value 4 indicates management initiative towards repository management and weight value 5 indicates dedicated infrastructure for repository management.
- **Organization Maturity:** Organization maturity in terms of its quality certification and/or CMM level defines a minimum guarantee level for quality of development process. A highly placed organization is assumed to have quality skilled staff, defined and repeatable processes along with procedures for defect prevention and continuous improvement. This surely reduces the dependability on experts, reduced effort requirement for quality software development. A weight value 0 indicates no certification, weight value 3 indicates any CMM level with ISO certification, and weight value 5 indicates CMM level 5 certified organizations. Non-TechnicalFactors**:** These are the various Non-Technical factors with weighting value which affect the maintenance of Component Based Software [3] shown in Table 2.

Table 2: Non-Technical Factors.

Non-Technical Factors	Weight
Domain Experience	1
Stable Requirements	2
Part Time workers	-0.5
Technology newness	1
Component model	1
Internal Library Resources	0.5
Organization Maturity	0.5

II.3 Estimation of ACT:

ACT (Annual Change Traffic) is another parameter that is used to estimate the maintenance cost. It includes the proportion of original instruction that undergo a change during a year by addition or modification, if ACT is given. For estimating the ACT of future software project we start with the existence of a series of given characteristics of a software project. The characteristics must be believed to important influences upon ACT. The characteristics should be evaluated and revised periodically by the company's experts to identify critical characteristics of the projects covered in the HT (history table). Based upon the data in the HT, each characteristic will be assigned a weight pj which permits us to give appropriate recognition to every characteristic based upon accumulated ACT data. Each project will only have two possibilities for every characteristic that is, to have it or not. So, for a table formed by n projects, we will have for each project the data shown by four matrices. Matrices A and T includes the historical data; matrices C and B includes the data about the current project of concern [4]. If ACT is not given then we can calculate the for any future software project by this equation1 :

$$ACT= (B*T^T) / (B^T*B) \text{ ……. (1)}$$

Where, $B= A*C^T$

Suffix T = Transpose of Matrix

A= Matrix of n*m elements that shows the feature of each project i.e. based on CBS in HT (History Table).

Step 1

A=

$$\begin{pmatrix} C11 & C12 & C1n \\ C21 & C22 & C2n \\ \ldots.. & \ldots\ldots & \ldots.. \\ Cm1 & Cm2 & Cmn \end{pmatrix}$$

Cij= feature j for the project i
Two possible values are used:
1: The project has the feature Cj
0: Otherwise
If we take positive value 1 i.e. greater than zero then company's experts could assign relative weights to the ACT values on a project-independent basis Values of 1carry the unweight ACT values forward from the HT into the estimates of the future ACT.

Step 2:

$T= (ACT_{1,} ACT_2 \ldots\ldots\ldots\ldots ACT_i)^T$

T = matrix of n*1 elements that shows the annual change traffic for project i.

Step 3:

$C= (C_{1,} C_{2\ldots\ldots\ldots\ldots} C_n)$

C = matrix of 1*m elements that shows the features of current project.
$C_{j=}$ Characteristic jfor the current project
Two possible values are used:
1: The project has the characteristic
0: Otherwise
Based on these factors, development cost of Component Based Software Project and ACT (Annual Change Conflict), Technical and Non-Technical factors the basic Maintenance Cost Estimation Model based on COCOMO for Component Based Software is:

$$AME_{CBS} = ACT * CDT \prod_{i=1}^{n} W_i * F_i$$

Where
AME_{CBS} = Actual maintenance cost of Component Based Software in Person-Month
CDT= Component Development Cost of Project expressed in man-month
ACT = Annual changes in conflict for any Project
$W_{i=}$ i^{th} weighting maintenance load, Fi = Factors value

III. CONCLUSION & FUTURE WORK

Software maintenance needed modification, to correct the faults, improve performances or the other attributes or to adapt the changed environment of system or component. Maintenance plays important role in various software developments like Iterative development, Agile Development and Component Based Software Development. In the recent year, software development turned into engineering through the introduction of Component Based Software Development. Component Based Software Development provides a method of building the software system that makes use of reusable components. It also increases the reliability of the software when it is up and running. When components are incorporated into a system, maintenance becomes much harder because source code is either partially or completely invisible. Building a component for reuse purpose or changing an existing component to make it reusable may also add additional maintenance cost. In case study at NASA was found that a few change (12%) on single component increased the reuse cost by 55% compared to the development of particular component built from scratch Software Component Maintenance

Cost plays an important role when it comes to maintaining legacy software. Legacy software's generally requires lot more maintenance than the other software's as they with time modified by the software developers. Now in such a situation it is required to find out the cost of the maintenance cost so that exact estimation of the amount spent for software can be received.

This paper involves development of Software Component Maintenance Cost Estimation Model. The specialty of this model is this model is used for Component Based Software's and is based on basic COCOMO model. The various parameters considered for this maintenance model are as follows:

- Development of CBS Project:
- ACT (Annual Change Traffic)
- Factors affecting maintenance cost of Component Based Software

All of the three parameters used to device the maintenance cost of the CBS. Through this model one could not only find out the maintenance cost rather one can also predict the Annual Change Traffic of any future projects hence used for estimation purpose also. Apart from the model presented this research work includes the validation of the model which is done with the help of historical data. This model is applicable for providing the accurate estimates, making the project planning, improving the adaptability of Component Based Software, and developing an understanding between the user, customer and third party. For the future perspective this model can be enhanced to calculate the maintenance cost of large data and maintenance cost of Aspect Oriented Software.

REFERENCES

[1] Roger S. Pressman, Software Engineering: A Practitioner's Approach Seventh Edition, McGraw-Hill Higher Education, 2010.
[2] Vingder, Building Maintainable Component Based Systemhttp://www.sei.cmu.edu/icse99/papers/38/38.htm., 1990.
[3] Rodrigo B.M., Vergilio S.R., Software Effort Estimation Based on Use Cases , proceeding of 30th annual International Conference on ComputerSoftware and Application,2006,221-228.
[4] Jaun Carlos Granja Alvarez and Manvel Jose Barranco Garcia, A Method for Estimating Maintenance Cost in a Software Project : A Case Study, JSoftw Main Evol-9, 1997,161-175.
[5] Hareton Leung and Zhang Fan, Software Cost Estimation,ftp://cs.pitt.edu/chang/handbook/42b.pdf.
[6] T. Wijayasiriwardhane, R. Lai, K. C. Kang, Effort estimation of component- based software development- a surveyIET Softw., vol. 5,2011, 216-228.
[7] K. Aggarwal, Yogesh Singh, Pravin Chandra and Manimala Puri, Measurement of software maintainability using a fuzzy Model,Journal of Computer Science, ISSN 1549-3636, 2005 Science Publications, 538-542.