Estimating Cost of Software Maintenance Using Component Based 4th GL Approach

Mohammad Islam[1], Dr. Vinodani Katiyar[2], Prof. (Dr.) S. Qamar Abbas[3]

[1]*Research Scholar, Shri Venkateshwara University, Gajraula, UP., India*
[2]*Professor, SRM, University, Lucknow, UP., India,* [3]*Director, AIMT, Lucknow, UP., India*

Abstract- **All software products require maintenance and support, depending upon the abilities of project term in the overall software development environment. A good software maintenance process would reduce the cost involved in terms of money, manpower, resources and time. In recent years, software development turned into engineering through introduction of component-based software development and maintenance (CBSDM). There are various models to estimates the maintenance cost of traditional software like COCOMO, SLIM, Function Point etc., but still there is no such a model to estimate the cost of maintenance using component-based 4th GL tools. This paper presents a new approach and direction for estimating cost of software maintenance using component-based 4thGL tools at the basis of COCOMO II model and its existing parameters. The model is calibrated using the empirical data collected from 12 software 4th GL projects. The efficiency of the model is also compared with our model used for such environment. The favorable results closely matching and it can be achieved better predictive accuracy through model implementation.**

Key words: Software Cost Estimation, Maintenance cost estimation model, Component based 4th GL tools, ACT, Existing weights of Factors.

I. INTRODUCTION

As software development has become an essential investment for many organizations, accurate software cost estimation models are needed to effectively predict, monitor, control and assess software development. The organization software maintenance system has to fulfill the needs like technical measure of the domain as well as optimum quality service with maximizes strategic impact and minimum cost of maintenance activities. One of the greatest challenges facing software engineers in the management of change control. Software engineers have hoped that new languages and new process would greatly reduce these numbers. However, this has not been the case. This is fundamentally, because software is still delivered with a significant number of defects. As new tools and technologies are emerging for the development of software, these issues have become even more important. Component based fourth generation languages (4thGL) software development provides one such difficulty. These languages are based on reusable components which are neither suitable to be calculated by FP analysis technique nor classical effort estimation methods can be applied that are specifically developed for procedural languages. The limitations of traditional models are problematic for effort estimation in component based 4thGL environment. There exist some software effort estimation models for 4thGL environment like van Koten, but these are developed for data-centre application using database. Smith developed a model to identify parameters for effort estimation in component based software systems. COCOMO is a well-studied and accepted effort estimation model. By augmenting the COCOMO model with the proposed a new model has been built upon the experience inherent in the COCOMO techniques. This model can be used for all types of software applications developed in any component based 4thGL environment.

II. RELATED RESEARCH STUDIES

A number of studies have been published to address cost estimation models for software development and maintenance. Existing studies are investigated and their contents and limitations are as follows:

- Van Koten also developed Bayesian statistical software effort prediction models for database-oriented software systems, which are developed using a specific 4GL tool suite. It is actually an extension of the previous model using statistical approach. Riquelme *et. al*. compared different effort estimation methods and presented a model for 4GL applications that analyses the relationship between a set of metrics for 4GL programs and the maintenance time for such programs which uses SQL statements. This model is based on the following parameters: NS= Total number of Select instructions in the considered program. NI= Total number of Insert instructions in the considered program. ND= Total number of Delete instructions in the considered program. NU= Total number of Update instructions in the considered program. NT= Total number of used Tables in the considered program. NN= Total number of Nesting in the considered program. This model is also suitable only for database applications. Morgan Peeples developed a model, where he calculates level of effort of a project as: $LOE_P = a + b_1 \times$ forms $+ b_2 \times$ reports $+ b_3 \times$ tables $+ b_4 \times$ modules, where LOE_P is the number of person days of effort it takes to develop a software application project *p*; *a* is the *y* intercept; b_1 is the number of person days for one form F; b_2 is the number of person days for one report R; b_3 is the number of person days for one table T; b_4 is the number of person day for one module M.

- The authors (Boehm and Valerdi, 2008) proposed evaluation criteria for the validity of the process models and they provided effective results. This article also explained the strengths and weaknesses of various cost estimation techniques for the period of 1965 to 2005 (40 years). Cocomo-II (Boehm, 1999) was an excellent model up to 2005 but it did not enfold the new requirement and development styles for the reuseness or estimation of cost. Cocomo-II directed the software experts to create and designed new models such as the Chinese government version of Cocomo (Cogomo) and the Constructive Commercial-off-the-Shelf Cost Model (Cocots) etc. Different future challenges were discussed for the invention of new model/methods and tools.
- An extension of UML (Unified Modeling Language) to RE-UML (Requirements Engineering - UML) is presented by the author (Mahmood and Lai, 2009). RE-UML enabled a system analyst to find accurate candidate components those fulfilled the stakeholders' requirements. One of the main reasons of this research was the lack of Component- Based System (CBS) development phases in the UML particularly requirements analysis and component selection. According to them, RE-UML removed the need for a system analyst to learn the new notations to model CBS requirements and component selection process.
- Reusability of components in Component Based Development (CBD) is illustrated in (Qureshi and Hussain, 2008). The author also discussed and compared different architectures of CBD. It may be mentioned that a detail explanation of advantages and disadvantages of CBD elaborated very nicely. The authors in this paper (Qureshi, 2006) presented the comparison of component based development (CBD) with other traditional software development practices. This paper evaluated object oriented process model and author emphasized to get full benefits of reuse. The role of repository for CBD has also been discussed.
- The problem of crosscutting which is produced during component development is elaborated (Clemente and Hernández-2001). They solved this problem by the extension with Aspect oriented methodology. It was mentioned by an example that how new business rules resulted in the more adaptable and reusable components. According to them, this Aspect Component Based Software Engineering has been developed with success in the CORBA Component Model domain (Frakes and Kang, 2005).

III. SOFTWARE MAINTENANCE

Maintenance activities include all work carried out post-delivery and should be distinguished from block modification which represent significant design, development effort and supersede a previously released software package. Formally, we may define software maintenance as "It is the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes or to a changed environment". In addition to the undiscovered flaws, it is common that some number of known defects pass from the development organization to the maintenance group. This bow wave of unclosed bugs is exacerbated when multiple versions of the same deliverable exist simultaneously. Accurate estimation of the effort required to maintenance delivered software is aided by the decomposition of the overall effort into the various activities that make up the whole process. The definition includes the following types of activity of software maintenance:

- Redesign and redevelopment of smaller portions (less than 50% new code) of an existing software product.
- Design and development of smaller interfacing software packages which require some redesign (less than 20%) of the existing software product.
- Modification of the software products code, documentation or database structure.
- Design and development of a sizeable (more than 20% of the source instructions comprising the existing product) interfacing software package which requires relatively little redesign of the existing product.
- Data processing system operations, data entry and modification of values in the database.

According to the ISO/IEC-14764 standard, software maintenance falls into one of four categories: corrective, preventative, adaptive, or perfective which are defined in the terms of (a) the goal of the change (correction or enhancement) and (b) the timing of the change (proactive or reactive). Corrective and preventative maintenance are grouped more generally as corrections, while adaptive and perfective maintenance are considered enhancements.

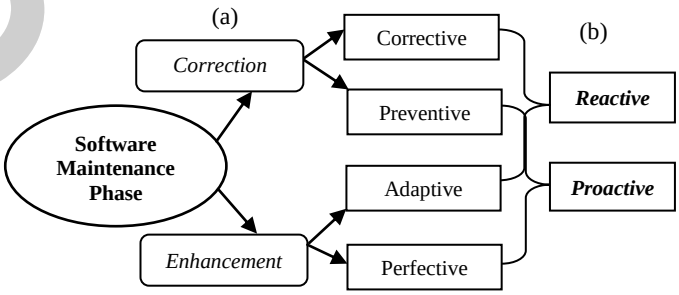

Fig-1: The taxonomy of Maintenance Categories.

Software maintenance cost is derived from the changes made to software after it has been delivered to the end user. From the given figure-2 below, it is obvious that maintenance related to enhancement or perfection of a software product is the largest single cost driver.

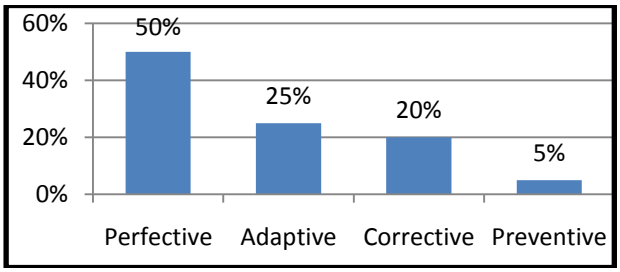

Fig-2: Percentage of total maintenance effort by repair categories.

IV. COMPONENT BASED 4th GL APPROACH

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). CBSD 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. EJB (Enterprise Java Beans) is the example of CBSD from Sun Microsystems. 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:

- 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.

V. PROPOSED SOFTWARE MAINTENANCE COST ESTIMATION MODEL

COCOMO (Constructive Cost Model) is used as a base model to estimate the cost of software project. This model was developed by Barry W. Boehm and published in 1981 using data collected from 63 projects. COCOMO II is an extended version used to estimate the cost when planning new software development. It is a good guide to estimate the software maintenance cost. It is actually an extension of the previous proposed model with new statistical approaches for estimating cost of software maintenance using component based 4th GL approach.

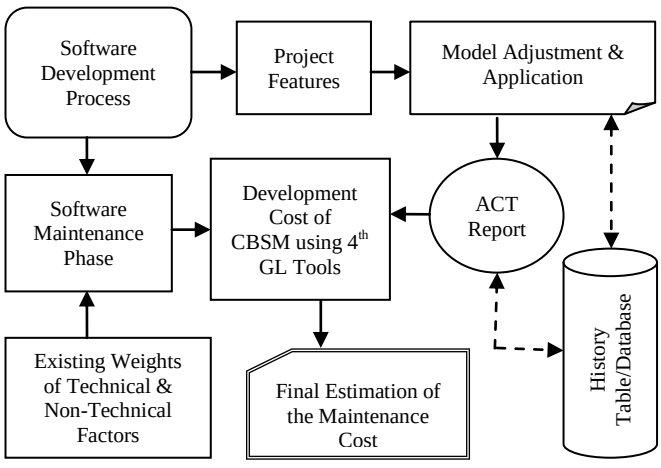

Fig-3: Maintenance Cost Modeling Process.

If we are familiar with the model and development process of subjects, software development process can be processed in two ways. Firstly software maintenance phase and other is project features. Software maintenance works on the CBSM cost using 4th GL tools with the help of existing weights of technical and non-technical factors, which will be able to estimate the cost of software maintenance phase. Project features includes selection of model adjustment and application with its characteristics; annual change traffic could be estimated using the history table which included database. CBSM cost using 4th GL project could be estimated the maintenance and development cost with the help of maintenance phase and result of ACT report. There are three specific existing parameters used:

(a) Development cost of CBSM using 4th GL approach.
(b) Existing weights of factors (Technical & Non-Technical)
(c) Annual Change Traffic (ACT).

(a) Development cost of CBSM using 4th GL approach:

CBSD includes the overall cost of Component Based Software Development. Although it may be somewhat controversial but modern software development environments are better understood as aggregates of forms, reports, tables and screens, rather than LOC. Fourth generation language are component based languages which provide a rich set of components. There are languages like VB, Java, EJB and C#, where user can develop an application without writing single line of code. When there is no line of code, SLOC-based models cannot be used for cost estimation of these applications. This model is developed to predict the software effort only for that software where data is accessed from database to forms, reports and graphs. For non-database applications, this model is not suitable. Fourth-generation languages are not used only to develop data-centered application. These systems can be used to develop various type of software, including database applications, scientific applications, generic software, computer games, mobile applications, the list is endless.

One solution to this problem has been the use of fourth generation languages which allow software to be developed more quickly than would otherwise be the case. This change has led to an increase in the amount of software to be maintained.

Grindley [IDPM86] reported that some companies with experience of fourth generation languages found it economically sensible to consider rewriting their systems rather than maintaining and patching existing software. There are several types of effect which this move to fourth generation languages can have on software maintenance:

- Simple hidden errors can be avoided, a fourth generation language can deal with certain aspects of the system automatically, and for example it can determine the first and last records.
- Many fourth generation languages are linked to data management systems with built in data dictionaries. The programmer cannot misrepresent the data or fail to declare variables.
- Many fourth generation languages are self documenting. Poor documentation is likely to be a

cause of maintenance difficulties with third generation languages.
- Fourth generation language make the understandability of a program clearer, and therefore easier for maintenance by the third person.
- Many fourth generation languages disallow ill-structured program constructs which can cause trouble later.

(b) Existing weights of factors:

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

Technical Factors:

- *Module Independence*: It should be possible to modify one component of a system without affecting other system components.
- *Programming Languages*: Programs written in high-level programming languages are usually easier to understand and hence maintain, then programs written in a low-level language.
- *Programming style*: The way in which a program is written contributes to its understandability and hence the ease with which it can be modified.
- *Program Validation*: Generally the more time and effort spent on design validation and program testing the fewer errors in the program. Consequently, corrective maintenance costs are minimized.
- *Documentation*: If a program is supported by clear, complete yet concise documentation, the task of understanding the program can be relatively straightforward. Program maintenance costs tend to be less for well-documented systems then for systems supplied with poor or incomplete documentation.
- *Configuration management*: It is used one of the most significant costs of maintenance is keeping track of all system documents and ensuring that these are kept consistent. Effective configuration management can help control this cost.

Non-Technical Factors:

- *Application Domain*: If the application domain is clearly defined and well understood, the system requirements are likely to be complete. Relatively little perfective maintenance may be necessary. If the application is in a new domain, it is likely that the initial requirements will be modified frequently, as users gain a better understanding of their real needs.
- *Staff stability*: Maintenance costs are reduced if system developers are responsible for maintaining their own programs. There is no need for other engineers to spend time understanding the system. In practice, however, it is very unusual for developers to maintain a program throughout its useful life.
- *Program Age*: As a program is maintained, its structure is degraded. The older program, the more maintenance it receives and the more expensive this maintenance becomes.
- *External Environment*: The dependent of the program on its external environment. If a program is dependent on its external environment it must be modified as the environment changes.
- *Hardware Stability*: If a program is designed for a particular hardware configuration that does not change during the program's lifetime, no maintenance due to hardware changes will be required. However, this situation is rare. Programs must often be modified to use new hardware which replaces obsolete equipment.

(c) Estimation of ACT (Annual Change Traffic):

In a survey of 63 products in various application areas, Boehm [B0EHM81] developed a formula for estimating software maintenance costs. The estimation is calculated in terms of the Annual Change Traffic (ACT), defined as "The fraction of a software product's source instructions which undergo change during a (typical) year, either through addition or modification". The ACT quantity is used, in conjunction with the actual or estimated development effort in person months, to derive the annual effort for software maintenance. ACT 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.

(VII.1) PROPOSED MODEL IMPLEMENTATION:

Enterprise Java Beans component consists all of three types of user interface components- forms, reports and graphs. Once the related table(s) in the database is defined, the tool automatically generates code that ensures the connectivity between them. This implies that developer's effort would be primarily spent performing the two tasks: creating each component by using various readymade graphical user interface items such as text boxes, combo boxes and adding code to the items. Modern software development environments are aggregates of forms, reports, tables and modules. In order to adjust the nominal effort, same effort multipliers will be used as described in COCOMO II model definition. According to COCOMO II, the nominal effort for a given size project and expressed as PM is given by:

$$PM_{nominal} = A \times (Size)^{B}$$

The inputs are the Size of software development, a constant A and a scale factor B. The size is in units of thousands of source lines of code (KSLOC).

(1) ***Forms*** are objects created by a developer for interaction or navigation by the user. The 4th GL tool provides templates for the developer to use. The form templates can be used as menus that move the user

through the application, or for use as data entry, querying screens or other forms required for the application. A form is actually a collection of different components.

(2) ***Reports*** are objects that a developer uses to retrieve data from tables, to format and present that data to the user. The 4th GL tool provides an easy way to use interface and readymade products for the developer to tailor and create specialized reports. Some reports may be simple but some may be more complicated as they need extensive coding as well as higher level complicated database queries and stored procedures.

(3) ***Tables*** are objects created to store data. The 4th GL tool provides features for the developer to define and generate data definition language and create tables automatically form design models. The level of complexity also varies as some table may require only a few fields, whereas other will require hundreds of fields with integrity constraints.

(4) ***Modules*** represent that portion of a software development application that cannot otherwise be delivered, except to be created. These might be computational algorithms, transaction handling, processes and code written for different events.

For 4th GL application, the Size of the software will be calculated as:

Size = $size_c$ + $Size_m$, where $Size_c$

is the size of components (Forms, Reports and Tables) converted into KLOC. $Size_c$ will be calculates as:

$Size_c$= $Size_{Form}$+$Size_{Reprot}$+$Size_{Table}$.

It will be calculates as:

$Size_F = \sum_{i=1}^{n}(SFi)$,

where SF is the size of forms. The size of form will be calculates as sum of size of its component. It will be calculated as:

$SF = \sum_{i=1}^{n}(SFCi)$,

where SFC is the size of component used in the form. In order to determine the nominal PMs, these components have to be converted to SLOC.

In COCOMO II model, a table for converting FPs, written in different languages to SLOC has been provided. But unfortunately, there is no such type of conversion available anywhere to convert 4th GL components to equivalent LOC. The recommended method converting these components to their equivalent LOC will be expertise based.

$Size_R$ = the sum of size of all reports size, used in the project. It will be calculates as:

$Size_R = \sum_{i=1}^{n}(SRi)$, where SR is the size of report.

As the nature of every report is not the same, therefore the size of each report also depends on its complexity. Some reports may be very simple and can be prepared using report generator wizards, like Crystal Report. But there will be many reports that will require complicated cross-tab queries, stored procedure, selection formula, run-time parameters and even some coding.

Tab-1: (Predicted Effort calculation of the selected 12 projects).

P#	Tools	Size (KLOC)	$Size_{Tables}$ (KLOC)	$Size_{Reports}$ (KLOC)	$Size_{modules}$ (KLOC)	$Size_{Forms}$ (KLOC)	Predicted Effort (PM)
1	VB.NET	6.263	1.5992	1.0902	0.788	3.7802	23.59
2	VB.NET	5.080	1.4607	0.7911	0.658	3.1651	19.22
3	VB.NET	5.692	1.4738	1.4713	0.725	3.0157	21.48
4	VB.NET	6.260	1.5986	1.0907	0.796	3.7788	23.58
5	VB	4.503	1.1462	0.7814	0.561	2.7095	16.01
6	VB	5.691	1.3102	0.8543	0.615	2.9056	17.79
7	VB	4.987	1.2483	0.7489	0.537	2.4469	15.22
8	VB	7.906	1.7825	1.2152	0.879	4.2134	26.69
9	C#	5.532	1.2184	0.8306	0.597	2.8802	17.21
10	C#	8.760	1.9284	1.3157	0.952	4.5584	29.17
11	C#	10.65	2.3480	2.0278	1.162	5.1233	36.35
12	C#	8.541	1.6602	1.2828	0.818	3.7734	24.62

$Size_T$ = the sum of size of all tables in database used with the project. It can be calculated as:

$Size_T = \sum_{i=1}^{n}(STi)$, where ST is the size of table. The table size can be easily converted to LOC. The size of table is directly related to number of fields. $Size_m$ = the size of modules in KSLOC.

Different GUI-based database environments provide database designers like SQL Server. According to our own experience, one simple field designing counts for three LOC, whereas one integrity constraint implementation counts for two LOC. Depending on the complexity of table, the tables will be converted to LOC. Finally, $Size_m$ will be calculated as: $Size_m$ = Size of Modules in KSLOC, whereas $Size_m$ is the total number of statements, written for different computational algorithms, transaction handling, processes and events.

Nominal effort for the selected projects was calculated by calculating the equivalent size of forms, reports, tables and modules. To determine the nominal PMs, all converted to SLOC. In order to translate them into equivalent SLOC different equivalence tables are constructed for different categories of components. The SLOC value assigned to different categories of component was set on the basis of experience of the developers. As it has been discussed earlier that these conversion tables can be maintained by developers themselves, therefore these tables are not going to be included here. These calculations are presented in table 1 & 2 present empirical validation results.

We applied van Koten model to these projects. There were considerable difficulties in applying this model as these applications also had many non-database controls. However their relative costs were calculated in best possible manner. The results of van Koten's model are presented in table 3.

This model was developed to predict the software effort only for that software where data is accessed from database to forms, reports, modules & graphs. For non-database applications, this model is not suitable. These systems can be used to develop every type of software, including database applications, scientific applications, generic software, computer games, mobile application, this model is not suitable. These systems can be used to develop every type of software, including database applications, scientific applications, generic software, computer games, mobile applications etc.

(VII.2) VALIDATION OF THE PROPOSED MODEL

SoftTech is a private software company, located in G. Noida, is providing effective computing solutions within public and private sector. Major development tools used in this software company are VB.Net, Java, VB & C#. This company was facing many problems in software cost estimation. FP metrics were used to estimate the cost but result were always unsatisfactory. We have applied the proposed model on 12 projects using component based tools that Enterprise Java Bean (EJB) is used instead of VB.Net, which gave satisfactory results. All of these projects were database projects, where SQL Server was used as backend and Crystal Report as development tool. However, different tools are used as frontend.

P#	Tools	Actual Effort (PM)	Predicted Effort (PM)	MRE, %
1	EJB	21.54	19.69	14.53
2	EJB	22.38	17.32	11.32
3	EJB	18.61	16.58	19.31
4	EJB	21.43	15.33	06.97
5	VB	17.24	15.11	10.63
6	VB	16.11	14.89	9.13
7	VB	18.24	14.32	12.71
8	VB	28.30	24.79	19.21
9	C#	21.54	23.87	24.24
10	C#	26.24	29.27	27.24
11	C#	30.69	36.45	26.16
12	C#	21.56	24.72	28.21

Tab-2: Empirical validation results of our model:

MMRE of all 12 projects = 15.41%
EBJ = 09.42%, VB = 14.57%, C# = 27.29%

The conversion tables can be maintained by developers themselves. These calculations are in table 2 presents empirical validation results and table 3 presents the results of van Koten's model. We applied van Koten model to these projects.

P #	Tools	Actual Effort (PM)	Predicted Effort (PM)	MRE, %
1	VB.Net	23.52	27.34	16.24
2	VB.Net	24.48	24.89	14.67
3	VB.Net	20.71	27.40	22.30
4	VB.Net	23.53	24.58	13.46
5	VB	16.34	19.34	18.36
6	VB	18.21	20.23	11.09
7	VB	15.34	18.45	20.27
8	VB	31.40	32.40	13.85
9	C#	23.64	26.39	26.63
10	C#	28.34	37.56	32.53
11	C#	32.79	45.82	39.74
12	C#	22.66	28.45	25.55

Tab-3: The Results of van Koten's Model

MMRE of all 12 projects= 17.81%
VB.Net = 10.70%, VB = 16.28%, C# = 33.33%

There were considerable difficulties in applying this model as these applications also had many database controls. However, their relative costs were calculated in best possible manner. We have also applied Function Point metrics on these projects but mean magnitude of relative error (MMRE) was very high (26%). Best MRE in FP metrics was 19%, whereas worst was 34%. Although the personnel, who did the estimation work were competent enough in applying FP metrics, the major problem was the nature of 4th GL tools as it is not suitable for FP metrics. Therefore it is suggested that the developer should maintain a conversion table for these components developed on the basis of their expert and own experience.

VII. CONCLUSION & FUTURE RESEARCH

The COCOMO II model is a good guide to estimate the maintenance cost of software projects. There were 12 software projects, which developed in 4GL tools. We have applied our model on these projects using component based tools that Enterprise Java Bean (EJB) instead of VB.Net, which gave satisfactory results. The approach tested on these cases proved robust and stable through cross-validation and verification trials on software development applications. EJB gave a significant level of accuracy and measured in terms of MRE and MMRE. Best MRE among these projects cases was 6.97% for EJB projects, whereas worst was 28.21% for C# projects. We also compared it with van Koten's model, which resulted in very high MMRE. Although MMRE for EJB projects was slightly better for our approach, it was far worse for C# projects. Further research with larger projects is still required. It is also required to identify more components as only four structural components that is Forms, Reports, Tables and Modules have been identified that contribute to modern software development using 4th GL software tools and their associated structural components. Calibration is likely to be essential in order to improve the level of accuracy. Finally, this research will be a new approach & direction for estimating cost of software maintenance using component based 4th GL (EJB) tools.

REFERENCES

[1] Byoung-Chol Lee and Sung Yul Rhew "An Empirical Study on Adjustment Factors to Estimate Maintenance Cost of Applications Developed Using Components", SoongSil University, Seoul, 156-030, Republic of Korea, *Lecture Notes on Software Engineering, Vol. 2, No. 1, February 2014.*

[2] Marounek P. "Simplified Approach to effort estimation in software maintenance", University of economic, Prague, Faculty of information and statistics, Journal of systems integration, 2012: 51-63.

[3] Marounek P. "SW Support and maintenance: Extension of ontology about COE concept", simplification of effort estimation, thesis, Prague, VSE-FIS, 2012.
[4] T. Wijayasiriwardhane, R. Lai, K. C. Kang, "Effort Estimation of Component based software development", a survey IET Software, vol. 5, pp. 216-228, 2011.
[5] Roger S. Pressman, Software Engineering: A Practitioner's Approach Seventh Edition, McGraw-Hill Higher Education, 2010.
[6] Deutsche Post DHL, "Deutsche Post DHL investors' MAIL-facts and figures", 2010.
[7] Nguyen Vu. "Improved Size and Effort Estimation Models for Software Maintenance", University of Southern California, 2010.
[8] Nguyen V., Boehm B.W., Danphitsanuphan P. (2010), "A Controlled Experiment in Assessing and Estimating Software Maintenance Tasks", APSEC Special Issue, Information and Software Technology Journal, 2010.
[9] Van Koten C., Grey A., "Bayesian statistical effort prediction models for data-centred 4GL software development", Discussion paper 2005/2009, department of information science, university of Otage, Dunedin, NewZealand.
[10] V. Nguyen, B. Boehm, and P. Danphitsanuphan, "Assessing and estimating corrective, enhancive and reductive maintenance tasks: A controlled experiment", IEEE, 2009, pp. 381-388.
[11] Shukla, R and Mishra, A. K. 2009, "AI Based Framework for Dynamic Modeling of Software Maintenance Effort Estimation", Proceedings of International Conference on Computer and Automation Engineering, 313-317.
[12] Boehm B.W., Valerdi R. (2008), "Achievements and Challenges in COCOMO-Based Software Resource Estimation," IEEE Software, pp. 74-83, September/October.
[13] Nguyen V., Steece B., Boehm B.W. (2008), "A constrained regression technique for COCOMO calibration", Proceedings of the 2nd ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM), pp.213-222.
[14] Riquelme J.C., Polo M., Aguilar Ruiz J.S., Piattini M., Ferrer-Troyano F.J., Ruiz F "A comparison of effort estimation methods for 4GL progeams: Experiences with statisticals & data mining", 2006, 16, (1), pp.127-140.
[15] Smith R.K. "Effort estimation in component-based software development: Identifying paramenters". The twenty-ninth ACM SIGCSE Technical Symp., Atlanta, GA, 25 Feb-2005.