RESEARCH ARTICLE OPEN ACCESS

Software Testing Using CMM Level 5

Dhananjay Singh
M.Tech (SE)
Gautam Buddha University
Greater Noida - India

ABSTRACT

The Capability Maturity Model is a software development processes with the goal of developing high quality software within budget and planned cycle time. There are two major Things. One is identify the key project factor such as software size that determine software project development outcome for CMM level 5 projects, Another one is benchmark for effort, quality and cycle time based on CMM level 5 project data. Cycle time may be compressed at the cost of quality. Quality may be achieved at the cost of increases testing effort. To find the software quality, effort and cycle time we are using three constraints such as (1) Literature review, (2) Research model, design and methodology, (3) Data analysis and results. The cycle time for software development, which depends on two factors one is planned development time another one is actual development time. Software development effort as the total effort beginning with the end of the requirements specification stage until the end of customer acceptance testing.

Keywords:-Line Of Code, Effort Calculation, Function Point Count, Processing Complexity, Development Time.

I. INTRODUCTION

Developing software to meet functional needs with acceptable levels of quality, within budget, and on schedule, is a goal pursued by every software development organization. CMM has been one of the most popular efforts in enhancing software quality and reducing development costs [1]. There is a need to reexamine relationships between software project development outcomes and various factors identified from prior literature. Conventional wisdom suggests that there are conflicting influences on software development effort, quality, and cycle time. Cycle time may be compressed at the cost of quality, experienced professionals may improve quality but at increased costs, quality may be achieved at the cost of increased testing effort, larger team size may reduce development time while raising total costs, process maturity may improve quality but at high cost, and so forth. One of the most important consequences of improved processes is superior conformance quality. The reduction in variability is likely to be most pronounced in development organizations at CMM level 5, which is the highest level of process maturity. Its results, therefore, may not be generalizable outside the environment where they were calibrated. Valuable insights can be gained from a study that focuses exclusively on CMM level 5 software development projects [2]. We make two major contributions [3]. First, we identify key project factors such as software size that determine software project development outcomes for CMM level 5 projects. Second, we provide benchmarks for effort, quality, and cycle time based on CMM level 5 project data. Our results suggest that estimation models based on CMM level 5 data are portable across multiple CMM level 5 organizations [4] [5].

II. CAPABILITY METURITY MODEL

Capability Maturity Model (CMM). Broadly refers to a process improvement approach that is based on a process model. CMM also refers specifically to the first such model, developed by the Software Engineering Institute (SEI) in the mid-1980s, as well as the family of process models that followed. A process model is a structured collection of practices that describe the characteristics of effective processes; the practices included are those proven by experience to be effective. CMM can be used to assess an organization against a scale of five process maturity levels. Each level ranks the organization according to its standardization of processes in the subject area being assessed. The subject areas can be as diverse as software engineering, systems engineering, project management, risk management, system acquisition, information technology (IT) services and personnel management.

2.1 Maturity Model

A maturity model can be viewed as a set of structured levels that describe how well the behaviors, practices and processes of an organization can reliably and sustainably produce required outcomes. A maturity model can be used as a benchmark for comparison and as an aid to understanding - for example, for comparative assessment of different organizations where there is something in common that can be used as a basis for comparison. In the case of the CMM, for example the basis for comparison would be the organizations' software development processes.

2.2 Structure

The model involves five aspects:

- **Maturity Levels:** a 5-level process maturity continuum - where the uppermost (5th) level is a notional ideal state where processes would be systematically managed by a combination of process optimization and continuous process improvement.
- **Key Process Areas***:* a Key Process Area identifies a cluster of related activities that, when performed together, achieve a set of goals considered important.
- **Goals:** the goals of a key process area summarize the states that must exist for that key process. As to have been implemented in an effective and lasting way. The extent to which the goals have been accomplished is an indicator of how much capability the organization has established at that maturity level. The goals signify the scope, boundaries, and intent of each key process area.
- **Common Features:** common features include practices that implement and institutionalize a key process area.

2.3 CMM LEVEL 5 AND KEY PROCESS AREAS

There are five types of common features: commitment to perform, ability to perform, activities performed, measurement and analysis, and verifying implementation.

- **Key Practices:** The key practices describe the elements of infrastructure and practice that contribute most effectively to the implementation and institutionalization of the area.

Level	Focus	Key Process Areas
1 Initial	Ad hoc software development and an Unstable development environment	Non
2 Repeatable	Basic software management controls and disciplined process adherence.	Software Project Planning. Software Project Planning and Oversight. Software Subcontract Management. Software Quality Assurance Software Configuration Management. Requirements Management.
3 Defined	Engineering Process Standard and consistent development process.	Organization Process Focus Organization Process Definition Peer Reviews Training Program Intergroup Coordination Software Product Engineering Integrated Software Management
4 Managed	Product and Process Quality	Software Quality Management Quantitative Process Management

5 Optimizing	Continuously improving process.	Process Change Management Technology Change Management Defect Prevention

Table-1: The five levels of the CMM and their key process areas [6]

Organizing the CMM into the five levels shown in Table-1 prioritizes improvement actions for increasing software process maturity [10], [11], [12].

III. STUDY AND REVIEW OF LITERATURE

According to **Saleem Basha el.at,** Reliable effort estimation remains an ongoing challenge to software engineers. Accurate effort estimation is the state of art of software engineering, effort estimation of software is the preliminary phase between the client and the business enterprise. The relationship between the client and the business enterprise begins with the estimation of the software. The credibility of the client to the business enterprise increases with the accurate estimation. Effort estimation often requires generalizing from a small number of historical projects. Generalization from such limited experience is an inherently under constrained problem. Accurate estimation is a complex process because it can be visualized as software effort prediction, as the term indicates prediction never becomes an actual. This work follows the basics of the empirical software effort estimation models. The goal of this paper is to study the empirical software effort estimation. The primary conclusion is that no single technique is best for all situations, and that a careful comparison of the results of several approaches is most likely to produce realistic estimates.[7]

According to **Martin Höggerl et.at,** A CMM is a process model of mature practices in a certain discipline. CMMI tries to integrate multiple CMMs. The old Software CMM is totally absorbed in CMMI. CMMI identifies 25 process areas in the software development process, each specifying a set of goals and practices, and it offers a continuous and a staged representation for each of its models. The continuous representation assigns capability levels to process areas; the staged representation assigns an overall maturity level to an organization's development process [8]. CMMI is often said to favor large, bureaucratic organizations, and it is also criticized for its exclusive focus on the process. CMMI is similar to but not easily comparable with the ISO/IEC 15504 (often referred to as SPICE). The teams assessing an organization for CMMI compliance have to meet various requirements, such as special training and experience. We present two examples of a CMMI assessment for illustration purposes.

According to **Jyoti G. Borade el.at,** the main goal of software project cost and effort estimation is to scientifically estimate the required workload and its corresponding costs in the life cycle of software system. Software cost estimation is a complex activity that requires knowledge of a number of key attributes that affect the outcomes of software projects, both individually and in concert. The most critical problem is the lot of data is needed, which is often impossible to get in needed quantities. Hence, Software cost and effort estimation has become a challenge for IT industries. In this paper, several existing methods for software project effort, cost estimation are illustrated and their aspects are discussed. Also, it describes software metrics used for software project cost estimation. This paper summarizes existing literature on software project cost estimation. The paper includes comment on the performance of the estimation models and description of research trends in software cost estimation [9].

IV. PROBLEM IDENTIFICATION

In existing system there are some drawbacks in that quality of software. Those are in quality, cycle time, effort, product size, and product complexity. These factors are not producing satisfactory result because the planed activity is not able to predict actual effort cycle time, effort, product size, and product complexity. The literature review in the existing system is not covering each and every quality factors.

V. OBJECTIVE

In existing system they are find the some drawbacks in that quality of software. Those are in quality, cycle time, effort, product size, and product complexity. One is identify the key project factor such as software size that determine software project development outcome for CMM level 5 projects, Another one is benchmark for effort, quality and cycle time based on CMM level 5 project data. Cycle time may be compressed at the cost of quality; Quality may be achieved at the cost of increases testing effort [13]. Software development effort as the total effort beginning with the end of the requirements specification stage until the end of customer acceptance testing. In proposed system the quality of software will be found based on the quality, cycle time, effort, product size, product complexity and schedule pressure.

VI. MODULES AND ITS DESCRIPTION

a. Effort
b. Quality
c. Cycle Time
d. Testing

6.1 Effort

Effort is often regarded as a surrogate for software development cost since personnel cost is the dominant cost in software development. In this module we have used COCOMO algorithm. This algorithm use the number of source lines of code (SLOC) as the basis for effort estimation. Effort is a function of system size combined with production rate that is how much work someone can complete in a given time.

6.2 Quality

Product size (SIZE) can be measured using lines of codes or using FPs .In this step we are using Function Point count to measure the quality of the software. A function point is a measure of program size that is based on the system's number and complexity of inputs, outputs, queries, files, and program interfaces. We List major elements of system and then Determine the total number of each element. Specify complexity index of each component (low, med., high).

6.3 Cycle Time

Cycle time is an important outcome variable because software projects are often carried out under strict delivery schedules. When planned schedules are longer than the minimum cost-effective schedule, they do not raise development costs. This is because, under ideal circumstances, projects can be completed using fewer developers than the optimal staffing strength. The number of lines of code has been reported to be a significant predictor of construction time.

6.4 Testing

This module is involving both Software validation and verification.

Software Verification and Validation:

Verification	**Validation**
Are you building it right?	Are you building the right thing?
Ensure that the software system meets all the functionality.	Ensure that functionalities meet the intended behavior.
Verification takes place first and includes the checking for documentation, code etc.	Validation occurs after verification and mainly involves the checking of the overall product.
Done by developers.	Done by Testers.
Have static activities as it includes the reviews, walkthroughs, and inspections to verify that software is correct or not.	Have dynamic activities as it includes executing the software against the requirements.

Table 2-Software Verification and Validation

VII. FUNCTION POINT

Function Points are the output of the software development process. Function points are the unit of software. It is very important to understand that Function Points remain constant regardless who develops the software or what language the software is developed in. Unit costs need to be examined very closely. To calculate average unit cost all items (units) are combined and divided by the total cost. On the other hand, to accurately estimate the cost of an application each component cost needs to be estimated.

1. Determine type of function point count and determine the application boundary,

2. Identify and rate transactional function types to determine their contribution to the unadjusted function point count.

3. Identify and rate data function types to determine their contribution to the unadjusted function point count.

4. Determine the value adjustment factor (VAF).

5. Calculate the adjusted function point count.

To complete function point count knowledge of function point rules and application documentation is needed. Access to an application expert can improve the quality of the count. Once the application boundary has been established, FPA can be broken into three major parts

1. FPA for transactional function types
2. FPA for data function types
3. FPA for GSCs

Rating of transactions is dependent on both information contained in the transactions and the number of files referenced, it is recommended that transactions are counted first. At the same time a tally should be kept of all FTR's (file types referenced) that the transactions reference. Every FTR must have at least one or more transactions. Each transaction must be an elementary process. An elementary process is the smallest unit of activity that is meaningful to the end user in the business. It must be self-contained and leave the business in consistent state.

7.1 FUNCTION POINT AL CULATIONS

The function point method was originally developed by Bij Albrecht. A function point is a rough estimate of a unit of delivered functionality of a software project. Function points (FP) measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an unadjusted function point count (UFC). Counts are made for the following categories [23].

7.2 Number of User Inputs-
Each user input that provides distinct application oriented data to the software is counted.

7.3 Number of User Outputs:
Each user output that provides application oriented information to the user is counted. In this context "output" refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.

7.4 Number of User Inquiries:
An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

7.5 Number of Files:
Each logical master file is counted.

7.6 Number of External Interfaces: All machine-readable interfaces that are used to transmit information to another system are counted.

Once this data has been collected, a complexity rating is associated with each count according to Table-3 [20].

7.7 FUNCTION POINT COMPLEXITY WEIGHTS

Measurement parameter	Weighting factor		
	Simple	Average	Complex
Number of user inputs	3	4	6
Number of user outputs	4	5	7

Number of user inquiries	3	4	6
Number of files	7	10	15
Number of external interfaces	5	7	10

Table-3: Function point complexity weights.

Each count is multiplied by its corresponding complexity weight and the results are summed to provide the UFC. The adjusted function point count (FP) is calculated by multiplying the UFC by a technical complexity factor (TCF) also referred to as Value Adjustment Factor VAF). Components of the TCF are listed in Table 3.

.

7.8 COMPONENTS OF THE TECHNICAL COMPLEXITY FACTOR

F1	Reliable back-up and recovery	F2	Data communications
F3	Distributed functions	F4	Performance
F5	Heavily used configuration	F6	Online data entry
F7	Operational ease	F8	Online update
F9	Complex interface	F10	Complex processing
F11	Reusability	F12	Installation ease
F13	Multiple sites	F14	Facilitate change

Table-4: Components of the complexity factor.

Each of the general system characteristics will be assigned a value from 0 to 5 to show its degree of influence. [21],[22],[23].

7.9 The Values of The Degree of Influence Represent:

0 = Not present, or no influence when present

1 = Insignificant influence

2 = Moderate influence

3 = Average influence

4 = Significant influence

5 = Strong influence at all development stages

The adjustment factor is then calculated as a percentage of the sum of the degree of influence from standard solution (values 65) and the total degree of influence of the system. The factor will range from 0.65 to 1.35. [24].

Each component is rated from 0 to 5, where 0 means the component has no influence on the system and 5 means the component is essential. The VAF can then be calculated as: [25],[26].

7.10 VALUE ADJUSTMENT FACTOR CALCULATION

VAF = 0.65 + (Sum of GSCs x 0.01) Where Sum of GSCs = $\sum$ (Fi)

The factor varies from 0.65 (if each *Fi* is set to 0) to 1.35 (if each *Fi* is set to 5).

The final function point calculation is: [16] [30].

Final Adjusted FP = UFC x VAF

Convert AFP into SLOC using appropriate conversion factor.

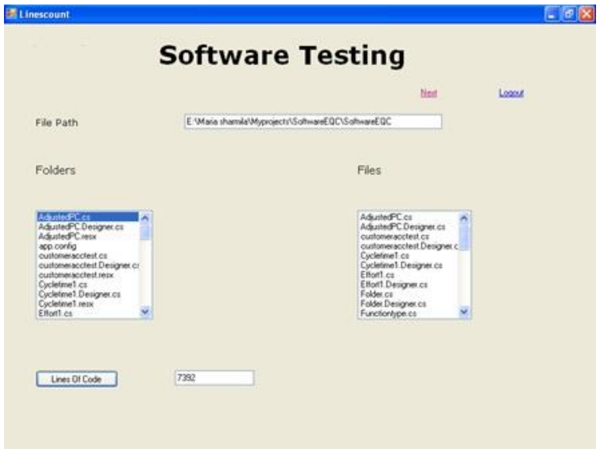
SLOC = 16 x SLOC/AFP [NOTE: 16 is the conversion factor]
EFFORT = EAF x A x $(SLOC)^{EX}$
EAF = CPLX x TOOL
A = 3.2= Constant based on the development mode.
EX = 0.38= Constant based on the development mode.
CPLX = 1.3 = Constant based on the development language.
TOOL = 1.1 = Constant based on the development Tool.
TDEV = 2.5 x $(EFFORT)^{EX}$ in months.
Once all the 14 GSC's have been answered, they should be tabulated using the IFPUG Value Adjustment Equation (VAF) --
14 where: Ci = degree of influence for each General System Characteristic [17] [18].

VAF = 0.65 + [(Ci) / 100] .i = is from 1 to 14 representing each GSC.

The final Function Point Count is obtained by multiplying the VAF times the Unadjusted Function Point (UAF).

FP = UAF * VAF

VIII. RESULTS


Linescount

Software Testing

Next   Logout

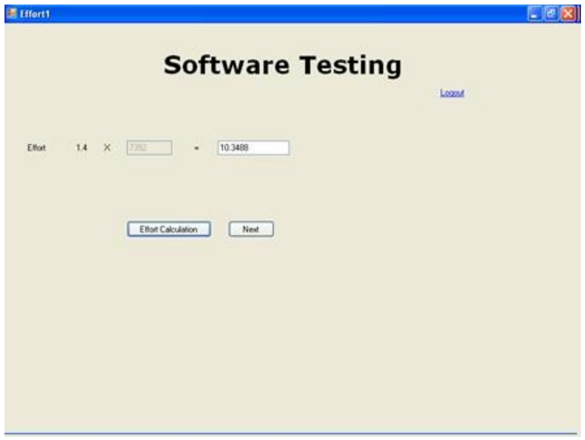
File Path: E:\Maria shamila\Myprojects\SoftwareEQC\SoftwareEQC

Folders:
AdjustedPC.cs
AdjustedPC.Designer.cs
AdjustedPC.resx
app.config
customeracctest.cs
customeracctest.Designer.c
customeracctest.resx
Cycletime1.cs
Cycletime1.Designer.cs
Cycletime1.resx
Effort1.cs

Files:
AdjustedPC.cs
AdjustedPC.Designer.cs
customeracctest.cs
customeracctest.Designer.c
Cycletime1.cs
Cycletime1.Designer.cs
Effort1.cs
Effort1.Designer.cs
Folder.cs
Folder.Designer.cs
Functiontype.cs

Lines Of Code: 7392

Figure-1: Line of code


Effort1

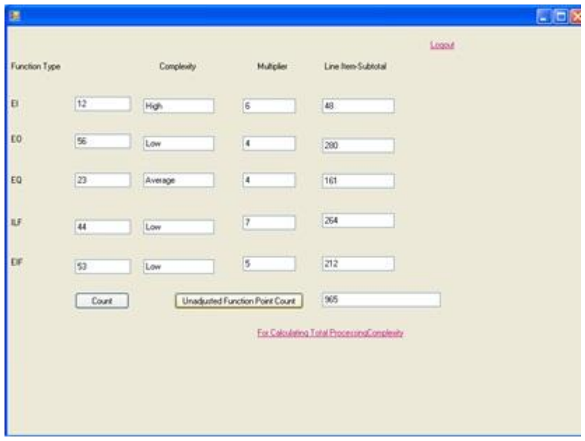
Software Testing

Logout

Effort 1.4 X 7392 = 10.3488

Effort Calculation   Next

Figure-2: Effort Calculation

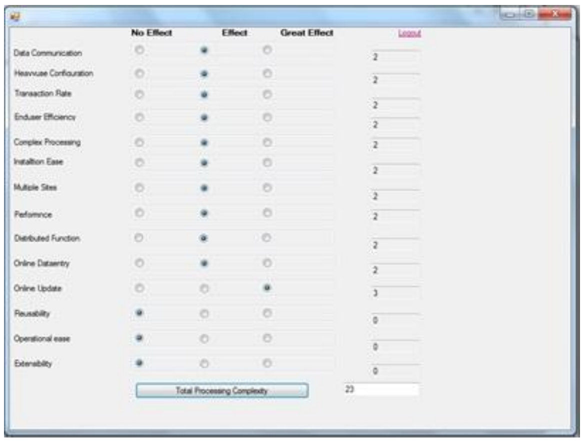

Logout

Function Type		Complexity	Multiplier	Line Item-Subtotal
EI	12	High	6	48
EO	56	Low	4	280
EQ	23	Average	4	161
ILF	44	Low	7	264
EIF	53	Low	5	212

Count   Unadjusted Function Point Count: 965

For Calculating Total ProcessingComplexity

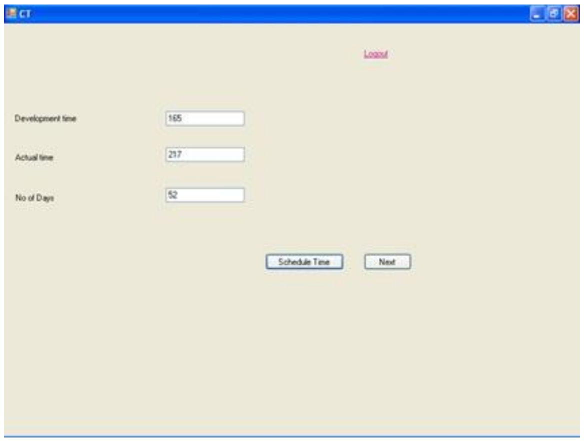
Figure-3: Unadjusted Function Point Count


Logout

	No Effect	Effect	Great Effect	
Data Communication		●		2
Heavvuse Configuration		●		2
Transaction Rate		●		2
Enduser Efficiency		●		2
Complex Processing		●		2
Installtion Ease		●		2
Multiple Sites		●		2
Peformnce		●		2
Distributed Function		●		2
Online Dataentry		●		2
Online Update			●	3
Reusability	●			0
Operational ease	●			0
Extensibility	●			0

Total Processing Complexity: 23

Figure-4: Total Processing Complexity



Development time 165
Actual time 217
No of Days 52

Figure-5: Development Time

IX. SCOPE OF RESEARCH

The main scope of this research is measuring the effectiveness of these techniques for reducing rate of software defects produce failures. The quality of software will be found based on the quality, cycle time, effort, product size, product complexity and schedule pressure. Developing software to meet functional needs with acceptable levels of quality, within budget, and on schedule, is a goal pursued by every software development organization. Many organizations are adopting the best practices in software development, such as those based on Capability Maturity Model. CMM has been one of the most popular efforts in enhancing software quality and reducing development costs. The reduction in variability is likely to be most pronounced in development organizations at CMM level 5, which is the highest level of process maturity. Our results suggest that estimation models based on CMM level 5 data are portable across multiple CMM level 5 organizations.

X. CONCLUSIONS

In this study, we used data set of some projects from different organizations that were at CMM level 5 to investigate the impact of various factors on software development outcomes. We found that software size was the most significant factor that affected development effort, quality, and cycle time. The models, although parsimonious, achieved an MMRE of about 12 percent in predicting effort and cycle time and about 49 percent in predicting the number of defects in holdout samples. This compared extremely favorably to the widely used software estimation models that achieved MMREs in effort estimation ranging from 100 percent for FPs to 600-700 percent for COCOMO [14] [15]. Our results showed that the potential benefit of achieving high process maturity was a step reduction in variance in effort, quality, and cycle time that led to relative uniformity in effort, cycle time, and quality [19]. Our models for effort and cycle time appeared portable across organizations based on good predictions for effort and cycle time, whereas our model for quality appeared less portable. Our results found that productivity differences among organizations were extremely important in estimating effort and that software estimation models were not portable across organizations. Overall, our results indicated that the adoption of highly mature software development processes during software development reduced the significance of many factors such as personnel capability, requirements Specifications, requirements volatility, and so forth. Discussions with our principal indicated some reasons for the reduced significance of requirements-related factors: 1) increased adoption of best practices by client organizations that were to a great degree influenced by the software development organizations, thereby leading to well-defined requirements, and 2) software development organizations leveraging their expertise from prior engagements in assisting clients in requirements gathering and specification.

XI. FUTURE DIRECTION

The project has covered almost all the requirements. Further requirements and improvements can easily be done since the coding is mainly structured or modular in nature. Improvements can be appended by changing the existing modules or adding new modules. One important development that can be added to the project in future is file level backup, which is presently done for folder level.

REFERENCES

[1] D.E. Harter, M.S. Krishnan, and S.A. Slaughter, “Effects of Process Maturity on Quality, Cycle Time and Effort in Software Product Development,” *Management Science, vol. 46, pp. 451-466*, 2000.

[2] H. Wohlwend and S. Rosenbaum, "Schlumberger's Software Improvement Program," *IEEE Trans. Software Eng., vol. 20, no. 11, pp. 833-839*, Nov. 1994.

[3] M. Diaz and J. Sligo, "How Software Process Improvement Helped Motorola," *IEEE Software, vol. 14, no. 5, pp. 75-81*, Sept.- Oct., 1997.

[4] K. Chari and M. Agrawal, "Software Effort, Quality and Cycle Time: A Study," *Proc. INFORMS Conf. Information Systems and Technology*, 2005.

[5] Paulk, Mark C., Bill Curtis, Mary Beth Chrissis, Charles V. Weber, Capability Maturity Model for *Software, v1.1*, 1993.

[6] Paulk, Mark C.; Weber, Charles V.; and Chrissis, Mary B., "The Capability Maturity Model: A Summary" (1999). *Institute for Software Research.1999*

[7] Saleem Basha, Dhavachelvan P.," Analysis of Empirical Software Effort Estimation Models" *(IJCSIS) International Journal of Computer Science and Information Security, Vol. 7, No. 3*, 2010**.**

[8] Martin Höggerl, Bernhard Sehorz, "An Introduction to CMMI and its Assessment Procedure", *Department of Computer Science University of Salzburg Seminar for Computer Science*, February 2006.

[9] Jyoti G. Borade, Vikas R. Khalkar "Software Project Effort and Cost Estimation Techniques" *International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue* 8, August 2013.

[10] Mark C. Paulk, Charles V. Weber, Mary B. Chrissis,"The Capability Maturity Model: A Summary", *Institute for Software Research*.1999.

[11] Robert S. Oshana , Richard C. Linger, " Capability Maturity Model Software DevelopmentUsing Cleanroom Software Engineering Principles - Results of an Industry Project", *Proceedings of the 32nd Hawaii International Conference on System Sciences –* 1999.

[12] Majed Alyahya, Rodina Ahmad, and Sai Peck Lee, "Impact of CMMI-Based ProcessMaturity Levels on Effort, Productivity and Diseconomy of Scale", *The International Arab Journal of Information Technology*, July 2012.

[13] Manish Agrawal and Kaushal Chari "Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects*", IEEE Transactions on Software Engineering*, March 2007.

[14] Kevin Crowston , Jian Qin " A Capability Maturity Model for Scientific Data Management:Evidence from the Literature" *National Conference on Postgraduate Research (NCON-PGR)* 2009.

[15] Saša Baškarada, Andy Koronios, Jing Gao, "Towards a Capability Maturity Model for information quality management", *Journal of Theoretical and Applied Information Technology* .2010.

[16] D.E. Harter, M.S. Krishnan, and S.A. Slaughter, "Effects of Process Maturity on Quality, Cycle Time and Effort in Software Product Development," Management Science, vol. 46, pp. 451-466, 2000.

[17] H.Wohlwend and S. Rosenbaum, "Schlumberger's Software mprovement Program," IEEE Trans. Software Eng., vol. 20,no. 11, pp. 833-839, Nov. 1994.

[18] M. Diaz and J. Sligo, "How Software Process Improvement Helped Motorola," IEEE Software, vol. 14, no. 5, pp. 75-81, Sept.-Oct., 1997.

[19] D.E. Harter, M.S. Krishnan, and S.A. Slaughter, "Effects of Process Maturity on Quality, Cycle Time and Effort in Software Product Development," Management Science, vol. 46, pp. 451-466, 2000.

[20] M.C. Paulk et al., "Capability Maturity Model, Version 1.1," *IEEE Software, vol. 10, no. 4, pp. 18-27*, July 1993.

[21] W.S. Humphrey, "Characterizing the Software Process: A Maturity Framework," *IEEE Software, vol. 5, no. 3, pp. 73-79*, Mar. 1988.

[22] W.S. Humphrey, "Characterizing the Software Process: A Maturity Framework," *IEEE Software, vol. 5, no. 3, pp. 73-79*, Mar. 1988.

[23] Paulk, Mark C., Bill Curtis, Mary Beth Chrissis, Charles V. Weber, Capability Maturity Model *for Software, v1.1*, 1993

[24] [24] Linger, R.C. and Carmen J. Trammell, "The SEI Cleanroom Software Engineering

Reference Model and its Implementation for the CMM for Software, *Proceedings of the 3rd Annual International Conference on Cleanroom Software Engineering Practices*, October 1996.

[25] K.K.Aggarwal, Yogesh Singh, A.Kaur, O.P.Sangwan "A Neural Net Based Approach to Test Oracle" *ACM Software Engineering Notes Vol. 29, No. 4, pp 1-6*, 2004.

[26] Linger, R.C., Mark C. Paulk, and Carmen J. Trammell, "Cleanroom Software Engineering Implementation of the Capability Maturity Model for Software", *Software Engineering Institute*, December 1996.