Proceedings of the IASTED International Conference
Software Engineering and Applications (SEA 2010)
November 8 - 10, 2010 Marina Del Rey, USA

REQUIREMENTS CHANGES REWORK EFFECTS: A CASE STUDY

Bee Bee Chua
Faculty of Engineering and Information Technology,
University of Technology, Sydney

Sydney, Australia
bbchua@it.uts.edu.au

ABSTRACT

Although software managers are generally good at estimation, their experience of scheduling reworks is poor. Inconsistent or incorrect effort estimation in turn increases the risk that the completion time for a project will ultimately become problematic. To continually alter software maintenance schedules while maintaining software projects is, in fact, a daunting task. Our proposed framework, validated in a case study, confirms that variables in requirements change suffer from weaknesses in coding, user involvement and user documentation. Our results clearly show that there is significant impact on rework as a result of unexpected errors found to correlate to 1) weak characteristics and attributes as described in the source lines of code, especially in data declaration and data statement, 2) lack of communication between developers and users on a change effect, and 3) unavailability of user documentation. To keep rework under control, new criteria in change request forms are proposed. These criteria are shown in the framework to need refining; thus, the more case studies that are validated, the more reliable the result will be in determining outcomes of effort rework effects.

KEYWORDS

Requirements change, effort rework, unexpected errors, weak characteristics and attributes, change request forms

1. INTRODUCTION

Software maintenance is becoming a core focus into today's Information Technology business contexts. More companies are focusing on upgrading existing applications than on implementing new projects. The global economic downturn has unfortunately pressured many companies into withholding new projects and deferring their implementation.

Maintaining and keeping project budget cost and time aligned to project delivery for an existing software project is never an easy task for a project manager. Much of the project funding is spent on requirement analysis, design, coding and testing. Eventually, the remaining funding that left may not be enough to provide support for software maintenance issues. Sometimes, such funding may run out completely because the risk of effort rework on the cost of a requirement change has not been anticipated.

Because requirements change is expensive, estimating the cost of effort rework on each change is consequently also costly. To know what requirements changes are needed for correction or modification, over-estimate and under-estimate lessons learned from previous effort reworks on requirements change need to be applied by IT practitioners. A requirements change can cause a ripple effect on other changes, thus, an investigation into the effect of rework is necessary.

The motivation for this paper is to provide an overview of a conceptual change management framework validated in a case study, in which a new development is found to have impacted significantly on the effect of effort rework. Further to this finding, an insight is provided into evaluating the criteria in current change request forms and the introduction of new criteria on which change should be based. In section 2, related works of software maintenance are discussed. A research method is discussed in section 3. Validation results from the case study are updated in sections 4 and 5. In section 6, the new criteria in change request forms are introduced. Section 7 presents a brief discussion of our framework refinement, and a conclusion and outline of future work is discussed in section 8.

2. RELATED WORKS

The definition of a requirements change has been made across a multi-disciplinary base. More specifically, its definition and concept have originated from software maintenance and change management, and each of requirements change identifies the type of change, the functionality of change, and the effect and impact of each change.

Requirements changes that are documented in change control forms provided limited information for software practitioners to approve and implement the change. Edelstein [1] quotes a definition of software maintenance based on an IEEE standard 1219 report published in 1993 which states that it is the modification of a software product after delivery, to correct faults, to improve performance or other attributes or to adapt the product to a modified

DOI: 10.2316/P.2010.725-073

environment. This definition is similar to the definition of why requirements change is needed. Other software maintainers [2,3,4] define software maintenance by focusing their views specifically on bug-fixing, user support and system adaptation.

It is said that views vary according to individual perspective. Nonetheless, the reasons for requirements change mostly relate to error detection and correction, modification of original requirements change for operational purposes and the support of user requests. Requirements change can be categorized by type, by volume, by case study context, by domain, by change management and by its own characteristics and attributes [5,6,7,8,9,10,11,12] .

According to both small and large software maintenance environments and change management, user change and software change are the most significant reported kinds of requirements change. At the user level, user change is a typical type of requirements change. Most user requirements changes concern the analysis and design of a system. After the prototyping stage and requirement elicitation stage, users' original requirements are modified to suit their needs and to accommodate their demands for enhanced functionality and design change. These changes can be large, small, simple or complex, important or trivial, depending on the demands made by users.

The other commonly known kind of requirements change type is software change. A study conducted by Weiss and Basili [13] of software development changes made in projects at the NASA Software Engineering Laboratory reported that the most frequent type of software change is the unplanned designed modification. They found reasons for software changes are related to 1) improve program optimization, 2) to improve the services offers to its users and 3) to clarify and improve the maintainability of the software products.

From a wide range of systematic literature [2, 3,4,5,7,8,13,14,15] that were reviewed, many fallible factors of requirements change found in project management, change management, software maintenance, information systems and software engineering disciplines which are perceived to be of, and related to mostly 1) environmental-issues and 2) learning-issues. No doubt there are many other factors influencing to requirements change, however, these two groups are known as two largest groups that critical to projects and frequently mentioned in literature repeatedly.

Software maintenance consists of four kinds: Corrective, Adaptive, Perfective and Preventive [16]. They have been classified by various authors with their taxonomies presented in accordance to their studies showed significant data found [2, 17,18,19]. The definition of a corrective change refers to modification initiated by defects in the software. A defect can be an error found in design, logic and coding. Adaptive change, however, drives by the need to accommodate modifications in the environment of the software system. Perfective change describes changes undertaken to expand the existing requirements of a system. Preventive change is undertaken to prevent malfunctions or to improve maintainability of the software [17].

Requirements change that fall into any of these categories without a cost on software maintenance effort made known, almost no one change can be successfully implemented. When assessing on the cost of each requirement change, effort is taking into account. The conventional approach drives most project managers and software maintainers undertake estimation is by applying an analogy-based approach where they assign value for effort reworks. Modern-based estimating tools like parametric ones for examples, SLIM [20] and COOCOMO 2.0 [21], they provide good software development effort and the support cost of effort. Unfortunately, these tools do not elaborate on maintenance effort rework in respect to requirements change.

Project size estimates can base on component, the count of function points, expertise and other non-parametric methods and non-algorithmic models so does a requirements change type. For many years, human-based estimation or expert judgment appeared to be the most frequently used effort estimation as the predominant choice of methodology [22].

Earlier work done by [23] found that the most accurate estimates were based on analogy and expert opinion. Molokken and Jorgensen [24] makes similar claims based on a review conducted on a number of surveys carried out on software effort estimation, in which it was found that expert-based estimation incorporated with fuzzy logic outperforms other types of models-based techniques. Hoch et al [25] in their study on decision support systems also suggest that experts perform better than models in a predictive environment.

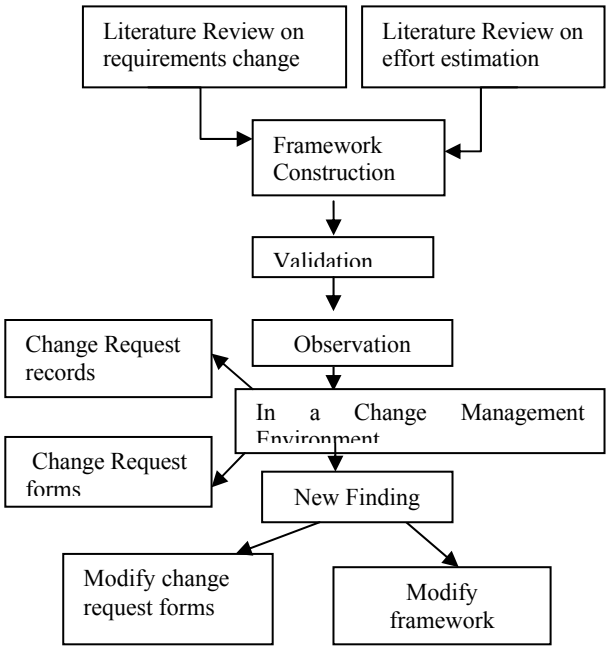
There are problems with estimation. According to Putman [20] and Boehm [21], there are several drawbacks. Firstly, there is no good logic or rationale used to develop estimates. Secondly, there is no stable of requirements, design, coding and process and no realistic interpretation of original requirements and resources estimates from which to develop the system. Thirdly, a large number of faults have been discovered in the software productivity rate estimates. Fourthly, estimation done early is less knowledgeable of the software to be developed and end up with more estimation errors [26].

The objectives of the paper are as follows: Firstly, the aim is to address a significant attribute that affects estimate rework inappropriately. Secondly, to propose new criteria to be included in change request forms for mandatory checks and reviews for the change management committee or configuration team to reduce rework impacts. Thirdly, to highlight the steps modified in our framework in order that tests can be successfully conducted on other case studies.

3. A RESEARCH METHOD FOR FRAMEWORK VALIDATION

Case studies are specifically designed for investigating research in field studies of phenomena when 1) a large variety of factors and relationships are included 2) no basic laws exist to determine which factors and relationships are important, and 3) when the factors and relationships can be directly observed [28].

The main emphasis of this research is a case that concerns understanding the relationship between change requirements and effort rework in a Change Management environment. It is neither a research case about organizational change nor is it a personal change. It is also not an interview case, because it does not rely heavily on interviewing for data collection. It is, however, an instrumental and observational case study in which the author is the primary data gather to observe and to collect data in a component of an organization, that is, IT Change Management. The case is instrumental because it emphasis on to gain understanding of something else. In other words, the whole of the focus is not solely on the case but also on the other issues related to the case. An overview of our research method applied in a case study context is shown as follows:

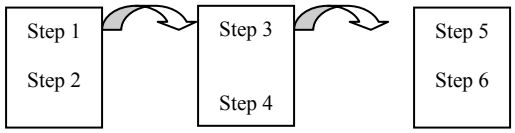


The subject of the observational case study is IT practitioners responsible for estimating requirements change. Because Change Management is only found in the software industry environment, to gain an access of the company data is private and difficult. The authorization of the organization is required.

Records of requirements change that have been documented and updated in change management databases for this organization mostly report maintenance support on existing applications. The examination of IT change request forms is typical of the kind of information needed for our data collection.

4. FRAMEWORK TESTED IN A CASE STUDY

The aim of developing a change management framework is to guide practitioners to control or avoid the cost of requirements change rework effort. There are other change management frameworks; however, their focus is mainly on procedures, process and people. In addition, current parametric estimation models do not provide a feature on the cost of estimating a requirements change and or the effort rework. Our conceptual framework focuses on six steps, which are reasonably easy to understand, to guide practitioners. Previous works [12, 27] outline the six steps of the framework (see below).



Step 1: Categorize requirements change (RC) into first order change and second order change.
Step 2: Note the reasons for the RC.
Step 3: Understand the factors relating to, and impacting on the RC.
Step 4: Distinguish vertical and horizontal dimensions of RC relationships**.**
Steps 5 and 6: Identify the relationship between effort and various change types, and estimate the amount of person effort required. (To be incorporated with COCOMO 2.0)

The preliminary framework was validated in a medium-size software organization specializing in maintaining large in-house embedded systems. A change management system was developed for capturing maintenance issues in respect of requirements change for large applications that are old; for example, having an application age of more than 8 years.

Incomplete and inconsistent change management records are common in any change management database so it is necessary to conduct a record filtration process. It took two person-days to complete the filtration process for 132 records. We successfully managed to filter 13 unwanted records that were duplicated, incomplete, or were change records that did not belong to the four maintenance types.

5. RESULTS

We applied our framework to test 106 data sets. Table 1 shows that adaptive change and corrective change are responsible for the submission of a higher number of change request forms than perfective and preventive changes.

Clearly, 73 out of 106 change records show that adaptive and corrective changes are the most frequent changes reported by users, and interestingly these changes perform more updates than insertion or deletion; i.e. 56 defects (21 from adaptive, 35 from corrective). Step 1 of our framework validated successfully and the following table generates the findings.

Framework step 1						
First order change		Second order of change				
Change Types	Total number of Change Request Records on application size ranging from 12,000 to 600,000	Insertion	Updates	Deletion	Insert and update	Insert and delete
Adaptive	32	10	21	1	-	-
Corrective	41	4	35	1	1	-
Perfective	8	4	2	2	-	-
Preventive	6	3	2	1	-	-
Functional Change	19	14	4	0	-	1

Table 1 shows statistics of change types based on the number of change request forms submitted.

To determine change type and the reasons for change, we validated step 2 of our framework and grouped the records into five kinds of change types. We followed the literature to define each of the four maintenance types: adaptive, corrective, perfective, preventive and the addition to functional change. The following diagram (A) shows implicit reasons based on
The change type

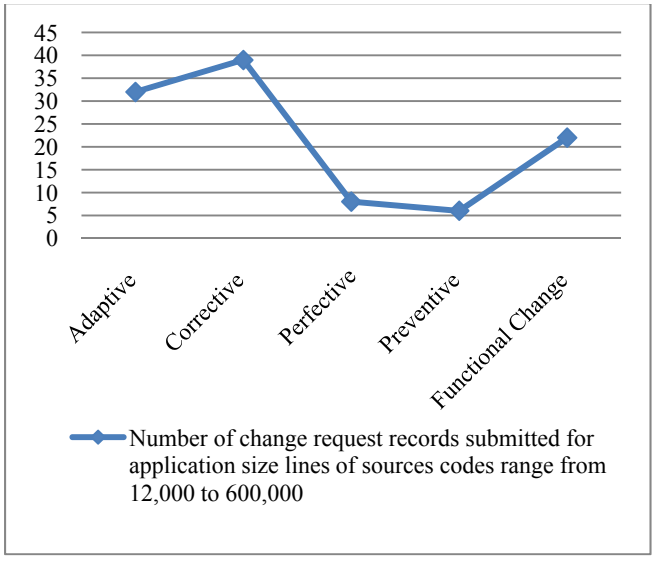


Diagram A shows the categorization of change types by the number of change forms submitted for application size source of lines of codes ranging from 12,000 to 600,000

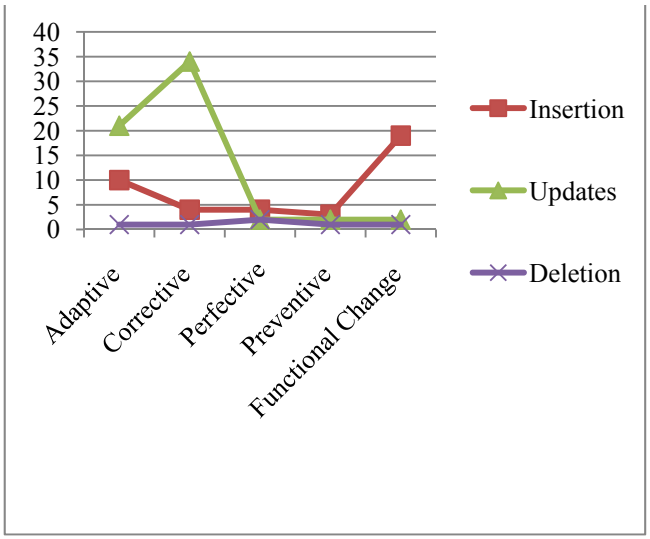


Diagram B shows the mode of function for each change type

Diagram B illustrates the mode of function for each change type. Typically, corrective change has a high number of updates and this correlates to users' requirement from their applications that defects are fixed on updating the lines of source code in the program. Ironically, a high number of change requests submitted by users in regard to functional change relate to insertion. In other words, a new feature needs to be added whereby it is possible to insert a new module or modify an existing module.

From test results confirmed no refinement of step 1 and 2 is required. We proceeded to validate the remaining steps showed in the framework. In validating the process of step 3 and 4 of the framework, we need to understand the factors relating to, and affecting the requirements change; the other step is to distinguish the vertical and horizontal relationship between these dimensions.

Identifying factors that are commonly discussed in literature is not our primary goal for the validation, but it is reasonable to know whether they exist. Many authors discuss how business, technology, environment, people and organization policy factors impact on a change. We were less concerned with such information, because there has been very little update on this. Our approach is to investigate implicit factors not discussed widely in the context of the literature, from the perspective of change effort.

In order to conduct this validation successfully, a scenario described below helped to address the implicit impact factors that can affect a requirements change and effort reworks. Updates constituted 35 corrective changes and 3 were especially selected based on the filtration sequence that we conducted. It is important that no assumption should be allowed in carrying out this test. We have information on the features of software maintainers regarding their years of experience, their involvement in projects, the number of maintainers, project characteristics and change characteristics. As we want to focus closely on impacts, all change records must fulfill the basic criteria. Unfortunately, only three projects meet the stated conditions.

Table 2 shows three projects named A, B and C. Each has the same number of total lines of source code but different effort reworks apply on updates. In particular, project B appears to have involved more rework effort than projects A and C. Because of unexpected errors correlated 1) errors (inconsistent, incorrect, incomplete or missing) of attributes residing on data statement or declaration, 2) the lack of user and developers communication of the change effort, and 3) the lack of user documentation that can trigger rework.

Project A has good data statement, but the lack of user involvement or user documentation can also generate unexpected errors. In exploring the effects of unexpected errors that cause enormous rework, the belief is that reference should be made to the paper documentation of a requirement change. Table 2 shows that unexpected errors occurred due to weak characteristics and attributes in the lines of sources codes, the lack of user involvement and user documentation.

Scenario 1 for corrective changes

Criteria Checks	
People characteristics	
Software maintainers' skills and experience	Same
Number of software maintainers involved	Same
Project Characteristics	
Project size	Same
Change Characteristics	
Update	Yes
Number of lines of source code to be updated	Same
Effort	Varies

	Lines of codes	'000		
	Project	A	B	C
Criteria	Project size	50	50	50
	Effort	1	10	3
	Lines of source code needing update	500	500	500
	New module	X	X	X
	Interface	X	√	X
	Code declaration	√	X	X
	Statement declaration	√	√	√
	Communication with users and developers	X	X	√
	User documentation	X	X	√
	Unexpected errors	√	√	X

Table 2 shows three projects of corrective changes

6. DESIGNING NEW CRITERIA IN CHANGE REQUEST FORMS

We reviewed 41 change request forms of corrective changes and addressing the criteria on the forms provided little convincing information. The change request form was designed to keep questions simple and easy to understand for end users. A new criterion proposed above is the insertion of a line to let users know that a large amount of effort rework arises because unexpected errors correlate to the lack of complete, correct, consistent attributes of the lines of source code in developing a new module, or in an existing interface, and the lack of users' and development involved to understand change effort and unavailability of user documentation to refer to the change.

An example of new criteria in Change Request Form

New Criteria

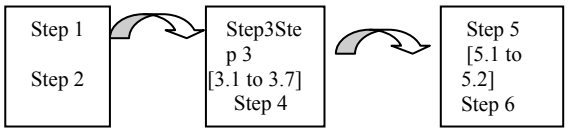
A large amount of rework can be expected if attributes declared in the source code are incomplete, inconsistent, incorrect or missing, or if there is a lack of user communication or no user documentation. Developers are to ensure the input of correct, complete, consistent attributes into the program and must check that these criteria are met.

For the attention of developers

Please tick if you have already informed users of what you have updated, inserted, or deleted from a new module or an existing module. What would you change?

7. FRAMEWORK REFINEMENT

Though our framework has validated a case study, the evidence from the exploratory study supports the need to refine some steps in our proposed framework. Step 3, in particular, is a review of change on the attributes types on the lines of source code for change types of an existing application, and step 4 identifies new criteria in change request forms to establish the relationship between the vertical and horizontal dimension. Steps 5 and 6 identify the relationship between effort and various change types, and estimate the amount of person effort required. This can be calculated based on a correct change type estimate on the amount of effort required. The revise framework is as follows:



Step 1: Categorize requirements change into first order change and second order change. (Remains unchanged)
Step 2: Note the reasons for the RC. (Remains unchanged)
Step 3: Understand the factors relating to, and impacting on the RC.

- 3.1 Know what the change type is.
- 3.2 Find out what to do with the change type. Focus on the mode of execution.
- 3.3 Check the sequence of change: updates, delete or insert
- 3.4 For updates, unexpected errors that arise will add more rework. Check variables such as attributes in data declaration and data statement must be complete, correct and consistent.
- 3.5 Check user documentation is available.
- 3.6 Check the change has been communicated to users.
- 3.7 Review criteria in change request forms of any missing information.

Step 4: Distinguish vertical and horizontal dimensions of requirements change relationships**.** In this context**,** an example of a vertical dimension would be data coding or data declaration and on the horizontal dimension would be referred to inconsistent, incorrect, incomplete, missing or ambiguity.
Steps 5: Identify the relationship between effort and various change types.

- 5.1 Review the qualifications, experience and skill of software developers. Have they been in projects for many years and are they aware of a change?
- 5.2 Determine the number of software developers involved in a project.

Step 6 Estimate the amount of person effort required. (To be incorporated with COCOMO 2.0 or other parametric models.)

8. CONCLUSION AND FUTURE WORK

Many researchers in the literature focus on rework problems and blame them on project managers and software maintainers due to their lack of experience or knowledge of estimation [22, 24]. They discuss the types of defects found that impact on project risk, the impact of ripple effect on the analysis, improper costing and time planning, and the inability of staff to make good decisions on change requests [2, 7, 27].
The evidence from our exploratory study confirms that the other trigger to cause rework arises from unexpected errors which correlate to 1) weak characteristics and weak attributes in data declaration or statement; 2) lack of user and developers' involvement, and 3) lack of user documentation in any of the change types.

The contribution made by this paper is to provide software maintainers and change management committees with an insight into various types of software maintenance issues which are drivers for software rework effort, using cases where our framework has been applied. Having more detailed information can help change management committees to narrow their focus when they are considering change request forms and the mandatory criteria they require on these forms. This will enable a better understanding of the rework effort required for necessary changes and will significantly reduce the risk involved in effort estimation for the rework; better control of the changes will be possible and they will be organized more effectively and efficiently. In addition, the refinement of the framework will then contribute to the successful validation in other case studies. These studies include open source development.

REFERENCES

[1] D.V. Edelstein, Report on the IEEE STD 1219-1993-Standard For Software Maintenance ACM SIGSOFT Software Engineering Notes.1993, Vol 18, No. 4, Pp 94-95.

[2] B. P. Lientz, and E. B. Swanson, Software Maintenance Management.Addison, Wesley Publishing Company, Reading, Massachusetts. 1980.

[3] ANSI/IEEE.. IEEE Standard Glossary of Software Engineering Terminology. Technical Report 729. 1983.

[4] B. J. Cornelius, M. Munro; and D.J. Robson, An approach to software maintenance education. Software Engineering Journal. 1988, Vol. 4, No. 4, Pp233-240.

[5] G. Kotonya and I. Sommerville. Requirements Engineering Processes and Techniques. John Wiley and Sons Ltd. 1985.

[6] S. D. Harker, K. D.Eason and D. E. Dobson. The change and evolution of requirements as a challenge to the practice of software. 1993.

[7] G. E. Stark, P. Oman, A.Skillicorn, A. Ameele. An examination of the effects of requirements change on software maintenance releases. Journal of Software Maintenance: Research and Practice. 1999, VOL. 11. NO. 5, PP 293-309

[8] W. Lam. Achieving requirements reuse: A domain-specific approach from avionics. The Journal of Systems and Software. 1997,Vol 38, No.3, Pp197–209.

[9] T. E.Bell, and T. A. Thayer. Software Requirements: Are They Really a Problem? In conference proceedings of 2nd IEEE international confernce on software engineering, San Francisco, California,United States. 1976.

[10] B. B. Chua, J. M. Verner. IT Practitioner's Perspective on Australian Risk Management Practices and Tools for Software Development Projects: A Pilot Study, SoMeT:2005, Pp111-125.

[11] B. B. Chua, J M. Verner, D. Dalcher. A Framework for Predicting Person-Effort on Requirements change. SoMeT: 2006, Pp 439-451

[12] B. B. Chua, D. V. Bernardo, J M. Verner. Criteria for Estimating Effort for Requirements change. EuroSPI 2008, Pp36-46

[13] V. R. Basili, and D. M. Weiss. Evaluation of a Software Requirements Document By Analysis of Change Data. In Proceedings of the 5th international conference on Software Engineering. San Diego, Califonia, United States. 1980.

[14] T. E. Bell, and T. A. Thayer. Software Requirements: Are They Really a Problem? In conference proceedings of 2nd IEEE international confernce on software engineering, San Francisco, California,United States. 1976.

[15] G. Walia, and J. Carver. A Systematic Literature Review to identify a classify Software Requirement Errors." Information and Software Technology, 2009. Vol. 51, No.7. Pp. 1087-1109.

[16] P. Grubb and A. A. Takang. Software Maintenance Concepts and Practice. Second Edition. World Scentific Publishing Company Ltd. G. Alkatib. The maintenance problem of application software: An empirical analysis. Journal of Software Maintenance Resarch and Practice. 2003, Vol. 1 No.2. Pp83-104.

[17] L. J. Arthur, Software Evolution.. The Software Maintenance Challenge. John Wiley and Sons, New York. 1988.

[18] W. M. Osborne. Building and sustaining software maintainability. In proceedings of Conference on Software Maintenance. 1987, Pp13-23

[19] H. L. Putman. A General Empirical Solution to the Macro Software Sizing and Estimating Problem". IEEE Transactions on software engineering, 1978, Vol. 4, No. 4, Pp 345-361.

[20] B. W. Boehm, B. Clark, E. Horowitz, J. Christopher Westland, R.J. Madachy, R W. Selby. Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. Ann. Software Engineering. 1995,Vol.1: Pp57-94

[21] M. Jorgensen, D. Sjoberg, and G. Kirkeboen.The Prediction Ability of Experienced Software Maintenance. In the 4th Proceeding of European Conference on Software Maintenance and Reengineering. Zurich, New Zealand. 2000

[22] F. Bergeron, J.F.St-Arna. Estimation of Informaiton Systems Development Efforts: A Pilot Study Journal of Information Management. 1992, Vol 22, No. 4, Pp239-254

[23] K, Molokken, M. Jorgensen. A review of software surveys on software effort estimation. In the proceedings of IEEE Transaction Sympoisum on Empirical Software Engineering. ISESE. 2003. Pp 223-230

[24] S.J. Hoch, and D.A. Schkade. A psychological approach to decision support systems. Journal of Management Science. 1996. Vol.42, No 1**,** Pp 51-64

[25] B. W. Boehm.. Software Engineering Economics. IEEE Transactions Software Engineeirng, 1984,Vol. 10. Pp 61-71

[26] B. B. Chua, J. M. Verner. esigning Criteria for change request forms. In proceedings of SMEF, Rome. Italy. 2008.

[27] J. D. Glazier and R. Powell. Qualitative Research in Information Management. 1992.