Change Theory: Towards a Better Understanding of Software Maintenance

AMAL ABDEL-RAOUF and MAEDA HANAFI
Computer Science Department
Southern Connecticut State University
501 Crescent Street, New Haven
UNITED STATES
abdelraoufa1@southernct.edu, hanafim1@owls.southernct.edu

Abstract: - A successful Software maintenance process depends on three factors: the maintenance goals, the technical properties of the system and the people performing the software maintenance. Most of the current work to investigate software maintenance only considers the first two factors, ignoring the third factor, which limits the scope and accuracy of these approaches. In this paper, we use change theory to introduce a deeper understanding of the software maintenance process. We utilize three change theories: Lewin's, Prochaska and DiClemente's, and Lippit's theories to introduce three different software maintenance models. These models consider the three success factors and incorporate contextual information to help maintainers better understand the software maintenance task to bring about an effective change.

Key-Words: - Software Engineering, Software Maintenance, Software Quality, Software Properties, Change Theory, Change Stages.

1 Introduction

A software program that is used in a real-world environment inevitably must change; otherwise it becomes gradually less useful in that environment.
Traditionally, software maintenance is defined as "the modification of a software product after delivery to correct faults, to improve performance or other attributes or to adapt the product to a modified environment" [1].

Software maintenance is a very important phase in the software development process, as it is a critical factor in determining software cost. Statistics show that it represents almost 60% of the total cost of any software product [2]. Moreover, software maintainability, the ease with which a software system can be modified, is considered one of the attributes to assess software product quality. Other software quality attributes include portability, usability and reliability [3].

It is very challenging to study software maintenance because it deals with many factors ranging from software features to human dynamics. Software features include correctness, modularity, coupling, code size, and complexity. On the other hand, human dynamics include team activity rate, communication structure, familiarity with the system, and skills level. As a result, the variety of factors affecting the maintenance process limits the generalization of research findings.

Yet most of the research studies have paid little attention to how software engineers understand the system and the information needed to perform a maintenance task. Pizka et al [4] showed that maintainability is dependent on more than just the technical properties of the system; it depends on three different dimensions:

1) The maintenance goals and tasks
2) The technical properties of the system under consideration
3) The people performing software maintenance (team members and a project manager)

Most of the work to investigate software maintenance considers the maintenance goals and the technical properties of the system, thus ignoring the third dimension and limiting the scope and accuracy of these approaches. One major problem is that the effect of the third dimension on software maintenance depends on a deeper understanding of the context of the system, human dynamics and social awareness. This dimension should be considered not only in the software maintenance phase but also in the software development life cycle. Recent research has started to include the social and human dynamic behavior in the software engineering field. In [5], the authors consider both personal and social values as dimensions in the value elicitation techniques during the requirement phase. In [6], the main focus of the study is the

social awareness and how it plays an important role of achieving effective digital communication.

In this paper, we propose the use of change theory to model the software maintenance process in order to incorporate contextual information and help maintainers to better understand the software maintenance task.

The remainder of this paper is organized as follows: the literature related to software maintenance is reviewed in Section 2. Section 3 summarizes three change theories and assumptions about the nature of change. Section 4 presents our models to represent software maintenance in terms of change theory principles. Section 5 then provides a generalization of the three models together with a brief comparison. A case study that shows the effectiveness of our models is presented in Section 6. Finally, Section 7 draws our conclusions.

2 Related Work

In the literature, the term "Software Maintenance" started back in 1960s and was widely accepted after the publication of Cannie's work [7] in 1972. The next significant work was Swanson's [8] that was published in 1976 and included taxonomy of the software maintenance that is still in use. It classifies software maintenance as follows: corrective, adaptive, perfective and preventive maintenance.

Other research has presented alternative taxonomies of software maintenance. Chapin et al. in [9] presented taxonomy of 12 categories based on the nature of changes of the software system's activities. Mens et al. in [10] proposed maintenance taxonomy based on characterizing the factors that influence the software changes.

The distinguished work by Lehman et al. [11], [12] in 1997-98 is still considered for software evolution/maintenance and included in all software engineering textbooks. Lehman et al. carried out several empirical studies of very large-scale industrial systems. From these studies they proposed Lehman's Laws concerning system change.

In recent years, with the emerging use of open source, software services, components, and frameworks some research has challenged Lehman's laws [13], [14] resulting in the need for new models to update them.

In 2011, Kumar did a comprehensive review on the major studies regarding maintainability models for object-oriented software systems [15. Kumar compiled literature in this particular field along with the variables, methods, and datasets used. For instance, Oman and Hagemeister's [16] research was on constructing and testing software maintainability assessment models. The variables employed in this research are as follows: aveLOC (Average Line of Code), ES (Executable Statement), CM (Line of Comment), and NES (Number of executable Statement). Kumar has compiled other works, which also employed similar variables. Such variables clearly describe software's factors and features.

Moreover, Kumar identified that the methods used in Oman and Hagemeister's research are as follows: MAT (Maintainability Analysis Tool), Regression, Halsted metrics, Cyclomatic Complexity, Assessment Model, and Entropy. The other publications Kumar compiled mostly employ regression models as a method to evaluate software maintenance of object oriented system. Based on Kumar's comprehensive literature review, most of the literature considers software maintenance in terms of the software's factors and features. There is a need for a context model to describe the software maintenance process. In this work, we model software maintenance using change theories principles introduced in the next section.

3 Change Theories

Change theories attempt to describe and organize the process in which human behavior changes. In this work, we have extrapolated the use of change theory to the field of software maintenance.

The most basic of change theories is Kurt Lewin's theory of change [17], used to describe human behaviors, and defined in three stages as follows:

1) Unfreezing: Stage in which change is needed
2) Moving: The change is initiated
3) Refreezing: Equilibrium is reached

Lewin's theory emphasizes the analysis of driving forces and restraining forces before a change is implemented. A change will occur when forces that promote change combined are greater than the forces that resist change combined. In the unfreezing stage, the aim is to unfreeze the current situation, also known as the equilibrium state. During the moving phase, the equilibrium state is in alteration in order to move to a different level of equilibrium. In this phase the driving forces establish a newer equilibrium state that is better than the current one. The last step is refreezing, in which the goal is to make the new equilibrium state continuous and to prevent reverting to the previous equilibrium state. Policies, procedures, and community actions can help uphold a new equilibrium [17].

A more extended and detailed change theory is defined by Ronald Lippit, Watson, and Westley [17]. Lippit's theory includes seven steps to implement a change, as follows:

1) Diagnosing the problem
2) Assess motivation and capacity for change
3) Assess change agent's motivation and resource. A change agent is the person who implements the change (examples of change agents: cheerleader, facilitator, expert)
4) Select progressive change objective; Develop action plans and establish strategies
5) Choose appropriate role of the change agent
6) Maintain change through strong group dynamics
7) Terminate the helping relationship.

Lippit's theory of change consists first of scrutiny of the problem, and thereupon moving on to evaluating the ability to change and the change agent's ability to change. It is important that the change agents clearly understand their roles in order for them to carry out their duty in facilitating change. Lippit, Watson, and Westley pointed out that a change is more likely to be adopted if the neighboring system adapts to it. Once a change is widespread, the behavior would be observed as the norm. An example of applying Lippit's theory of change is given from the research of Gary Mitchell [18] that analyzes planned change in nursing management. However, much of Lippit's theory focuses on the change agents rather than on the change itself.

The third theory of change is Prochaska and DiClemente's cyclical model of change [17], which defines a more general process of change. It was originally developed for the purposes of explaining the patterns of staged behavior change. The defined stages are as follows:

1) Pre-contemplation
2) Contemplation
3) Preparation
4) Action
5) Maintenance

In the pre-contemplation stage, the problem is denied by the individual, and there is absolutely no will to change. The awareness of the problem will occur in the contemplation stage, where change is being thought about. In the preparation stage, the individual is prepared to change their behavior, and consequently the action stage occurs. The action stage is where the individual actually changes their behavior. Finally, the maintenance stage attempts to sustain the change. This process is a spiral since individuals can exit any stage and return to the previous stages. However, it is also possible for the individual to return to the contemplation stage at any time in order to prepare for future action. This model pushes the idea that an individual will not always circle the spiral forever.

The three theories presented so far relate to social and management fields. However, there has been similar research on change in software-related fields. For instance, Lassila proposed the adoption and utilization of new software through the punctuated equilibrium model [19]. The punctuated equilibrium model utilizes incremental adaptations with organizational changes in long periods of stable infrastructures. During these adaptations the equilibrium is disrupted but returns back to stability.

In the following section, we present how these change theories can be applied to model software maintenance.

4 Software Maintenance Model Based on Change Theories

Our research surrounds the question, "How is software maintenance can be modeled in terms of change theories instead of its features and factors?" We gathered and scrutinize the techniques of other researchers in the field of change theory and compared it to the software maintenance process. In the following subsection, we introduce software maintenance based on the three change theories described in section 3.

4.1 Software Maintenance Model using Kurt Lewin's Theory

In Table 1, we show the stages of software maintenance process based on Lewin's theory.

Table 1. Software Maintenance Model using Lewin's Theory of Change

Stage	***Lewin***	***Description***	***Maintenance Stage***
1.	Unfreezing	Stage in which change is needed.	Prepare and plan for software maintenance. Perform force-field analysis.
2.	Moving	The change is initiated.	Maintenance occurs.
3.	Refreezing	Equilibrium is reached.	Software must pass all tests, be stable, and serve users.

In Lewin's theory, there are only three stages, which place much focus on the idea of equilibrium and moving from one state of equilibrium to another. In terms of software maintenance, such equilibrium can be translated to any running software that is servicing the user and running on a stable system. The system's stability can be quantified with the stability condition from queuing theories, which state

that the servicing rate must be greater than the customer's incoming rate [20].

In the unfreezing stage, much of the aim is to plan and prepare for the change. Maintainers at this stage prepare for a maintenance project. Lewin suggested force-field analysis, where positive and negative factors are compared. Mitchell's research in change management for the nursing field [18] also employs force-field analysis to address resistance and induce stress. The same can be applied to software maintenance. According to Dehaghani et al [21], the factors to be considered are the tools used in maintenance, current practice of proper software development techniques, team stability and skillset, contractual responsibilities, software age and architecture. Once force-field analysis is performed and a plan is finalized, the preparation for change is complete. The benefit of force-field analysis is that it ensures that a change in the software maintenance successfully occurs, while overcoming obstacles.

In the moving stage, the maintenance project begins and the system is analyzed and undergoes change. The change made to the system may be as simple as correcting code or more extensive change to correct design errors or accommodate new system requirements.

Finally, the refreezing stage aims to re-stabilize the system. In this stage, all tests must be passed and the system must be able to serve its users while incorporating the new changes.

4.2 Software Maintenance Model using Lippit's Theory

Lippit's theory consists of more stages, but targets the change agent's ability. In software maintenance, the change agent is the maintainer, since change mostly occurs within the software and only a maintainer is able to do it. Lippit's theory emphasizes the maintainer's skillset and knowledge in order to accomplish the change.

The first three steps of Lippit's theory revolve around analyzing the current status. Diagnosing the problem requires the maintainer's being able to give a brief description of the problem, and perhaps a speculation of its source. Then, the assessment of motivation and capacity of change evaluates the priority, ability, and profit of the proposed maintenance project. If the need for the proposed change is high, then the priority should be high as well. However, assessment of the change agent mainly deals with the team's ability to continue on with the proposed project. For instance, if the proposed maintenance project depends on another project, then the team wouldn't be able to continue the first project without the other being completed.

Once assessment is completed, an action plan must be developed. If there are dependent projects, then those projects must be completed. This stage stresses the importance of an action plan. This action plan can be greatly guided with a maintenance methodology. A maintenance methodology, whether standard or individually defined, can be any process that is defined within a set of tasks and timeline that is feasible and agreed upon by the stakeholders [22]. The maintainers must reassess if the methodology used is consistent with their aims. Factors to consider when choosing a maintenance methodology include project factors (frequency of change or immediate change), developer factors (skill set, enjoyment for collaboration, or desire to be "in fashion"), and organizational factors (organization structure and cultural issues).

Afterwards, the roles of the team members must be determined. Then, the change is put in action. Lippit's theory emphasizes the group dynamics. This is especially important for teams that have scattered members, where each member must be constantly communicating with the rest of the team.

Table 2. Softwae Maintenance Model using Lippit's Theory of Change

Stage	***Lippit***	***Software Maintenance Tasks***
1.	Diagnosing the problem	Give a brief description of the problem. Recommended to speculate on its source.
2.	Assess motivation and capacity for change	Evaluate ability, priority, and profit.
3.	Assess change agent's motivation and resource	Evaluate team's ability.
4.	Select progressive change objective; Develop action plans and establish strategies	Reassess maintenance methodology and develop action plans.
5.	Choose appropriate role of the change agent i.e. cheerleader, facilitator, expert	Pick appropriate maintainers, and set roles for each one.
6.	Maintain change through strong group dynamics	Make necessary changes while upholding team communication.
7.	Terminate the helping relationship.	End the project. Return the system to running and stable.

During this stage, a maintainer constantly faces unfamiliar code. In this case, the maintainer would search and pick a relevant node to begin comprehending the structure and program flow. The maintainer would decide if the current node is relevant to the change. This process of searching for relevant nodes would continue until there is enough information to implement a solution and create changes in code. Searching and comprehending is

easier on the maintainer when the environment provides ease for judging a node's relevancy. Another factor that provides further straightforwardness is the environment's reliability to provide relevant information for a given node [23]. If these two factors are not considered, this stage will take longer time.

When a solution is implemented and the tests are passed, then the maintenance project ends. If needed, tests can be recreated to cover new cases, and the system can be retested. However, if a need for change i.e. bug, fault, new feature, or adaption change, appears due to the new change, then the team must aim to continue to maintain the change. Once those needs are addressed, the project may be terminated. The system must be running and stable as well. The tabular form of software maintenance tasks based on Lippit's theory is shown in Table 2.

4.3 Software Maintenance Model using Prochaska and DiClemente's Theory

In Prochaska and DiClemente's theory, there is a pre-contemplation stage that doesn't exist in the other theories we have described. This stage considers the situation where a need for change is required but is not yet noticed by the maintenance team. The users may perceive it, but it requires the maintenance team to notice it before a maintenance project is initiated, moving to the contemplation stage. The software is similar to a patient at this stage; others perceive his/her abnormal behavior, but the patient has no intention of changing.

But even in this stage, it is advised that the possibilities of future requirements are analyzed. The aim of this is to ensure that all architectural changes are explored and thoroughly defined [24]. In fact, the pre-contemplation stage defines a stage for the team to have special attention to the architecture's changeability. A coherent architecture allows flexibility to meet the constant altering of user's requirements. In order to have extensive architecture coherence, the team must gain enough knowledge about it so that the software can evolve. If the architectural knowledge is lost, then the software goes through coherent architectural loss known as code decay [24]. In other words, the team may make changes that do not take advantage of the software's architecture, thus losing its coherency.

Moreover, Prochaska and DiClemente's spiral model allows for flexibility that is not offered by Lippit's and Lewin's. At any stage, the team can suspend a maintenance project and exit a stage. Anytime the team decides to return to the maintenance project, the project's stage winds back to the contemplation stage. This is a convenience for the team especially when an emergency maintenance happens or a need for another change appears.

The contemplation stage occurs when the team considers the change, but is not yet ready to analyze and implement it. At this point, the team is conscious of the need for change and a maintenance project must be formulated. The priority, profit, and ability of the proposed change are decided before it is moved to the preparation stage.

The preparation stage is when the team decides that it is ready to undertake the change. This is when a definite action plan is set and the re-analysis of current methodology is done and revised as needed. Consequently, the team re-analyzes the code and changes as needed, which is the action stage. In Prochaska and DiClemente's patient model, the patient is supported with counseling, social support, and assistance. In software maintenance, the tools and environment assist the maintainers in searching for relevant nodes to form a solution [25]. The pressure from users provides motivation, and the maintenance team members provide further assistance in the process.

Once the changes are done and the tests are passed, the changes are integrated into the running system. A change in software can be maintained if all the tests are successful, the users are updated about it, and the system is stable. If the system is not stable, then another maintenance project is initiated and repeats throughout all the defined steps. Moreover, it is recommended that a change be evaluated by surveying the users. A tabular form of software maintenance tasks based on Prochaska and DiClemente's theory is shown in Table 3.

Table 3. Softwae Maintenance Model using Prochaska and DiClemente's Theory of Change

Stage	***Prochaska and DiClemente's***	***Maintenance Stage and Tasks***
1.	Pre-contemplation	The need for change is not yet perceived by maintainers. Speculate about future changes especially architectural changes.
2.	Contemplation	The problem is detected by the team, and priority, ability, and profit are decided.
3.	Preparation	Reassess maintenance methodology and develop action plans.
4.	Action	Analyze, search, and change code.
5.	Maintenance	Rerun tests to ensure the new change is passed, and the users are updated about it. Evaluation of the changes is recommended.

5 Generalization and Comparison

Table 4. General Maintenance Tasks in Terms of Change Theories

Lewin	*Prochaska and DiClemente's*	*Lippit*	*Maintenance Stage and Tasks*
	Pre-contemplation		The need for change is not yet perceived by maintainers. Speculate about future changes especially architectural changes.
Unfreezing	Contemplation	Diagnosing the problem	The problem is detected by the team, and priority, ability, and profit are decided. Give a brief description of the problem. Recommended to speculate on its source.
		Assess motivation and capacity for change	Perform force-field analysis. Evaluate ability, priority, and profit.
		Assess change agent's motivation and resource	Perform force-field analysis. Evaluate team's ability.
Moving	Preparation	Select progressive change objective; Develop action plans and establish strategies	Reassess maintenance methodology and develop action plans.
		Choose appropriate role of the change agent i.e. cheerleader, facilitator, expert	Pick appropriate maintainers, and set roles for each one.
	Action	Maintain change through strong group dynamics	Maintenance occurs. Analyze, search, and change code. Make necessary changes while upholding team communication.
Refreezing	Maintenance	Terminate the helping relationship.	End the project. Ensure passing of tests. Return the system to running and stable. The users must be updated about it. Evaluation of the changes is recommended.

Lewin's theory is the most basic with only three stages, with a focus on analyzing negative and positive factors through force-field analysis. Lippit's theory is more detailed, helping to define more specific tasks in the software maintenance process. Lippit's theory also emphasizes the importance of assessing the specific team's abilities, since a stress is placed on the change agent. Furthermore, the team's dynamics, including communication, are emphasized. On the other hand, Prochaska and DiClemente's theory is a more evolved theory, allowing flexibility for the maintenance team to spiral around the stages. Moreover, their theory has a pre-contemplation stage, which propels the team to constantly think of the future changing requirements and needs.

In Table 4, we provide a comparison of the three change theories together with the corresponding software maintenance tasks in each stage.

5.1 Assessing the Current Situation

Prochaska and DiClemente's pre-contemplation stage does not correspond to any stages from the other theories. However, the next stage of contemplation corresponds to Lewin's unfreezing stage and encompasses Lippit's first three stages. These stages all share the characteristics of reviewing and reassessing the current situation. The highlighted task is force-field analysis of driving forces and negating forces. Lippit's theory splits the tasks into 1) the analysis of the problem itself, where a maintenance request is sent in and a maintenance project is initiated, 2) analyzing ability to change, and 3) analyzing the change agent's ability, i.e. team's ability.

5.2 Developing an Action Plan

Lewin's moving stage corresponds to Prochaska and DiClemente's preparation and action stages. Lewin's moving stage also corresponds to the next three stages of Lippit's theory. These stages are specific to developing an action plan and acting upon it. Lippit's theory divides Prochaska and DiClemente's preparation stage into two stages: 1) developing action plans and 2) choosing team member roles.

Finally, change takes place in Lippit's last stage in this category, which corresponds to Prochaska and DiClemente's action stage.

5.3 Maintenance

The final stage in all theories depicts a maintenance stage. Lewin's refreezing stage aims to incorporate the new changes into the operational system successfully. Lippit's theory explains that the team terminates the maintenance project and return the system to the running state. Prochaska and DiClemente's last stage includes passing all tests, addressing all other needed changes, and ensuring the stability of the system.

6 Case Study: Website Logo

The application chosen to conduct this case study is a web-based application for a company that has the logo on the top right side of a page. After using the website, the manger realized that most advertisement appear on top of the logo and most customers can't see it. The solution is to move the logo to the left side of the page. The change theory models for this case study are shown in Table 6.

6.1 Assessing the current situation for the Website Logo Case Study

Once the problem is detected, this phase starts with analyzing the problem. A change is needed, is it fixing a bug or adding a new feature to the software. From the customer point of view, this is fixing a bug while from the maintainer's point of view this could be a completely new feature to add. Unfortunately contracts have different consideration and cost for the different types of change. A negotiation should start till both customer and developers agree on the change definition and the cost (force-field analysis).

Technically, there is a big difference between adding a new behavior to the software and changing old behavior. Users like it when we add new behavior but stop trusting us if we change or remove old behavior they depend on. In this case study, we are doing both, adding a new behavior (logo on the left side) and removing old behavior (logo on the right side)

6.2 Developping Action Plan for the Website Logo

In this phase, we start to decide about the parts that will change in the system. During software maintenance, three parts could change in the system: structure, functionality and resource usage [26]. Special care should be given to changing functionality, if we add a new function we don't need to change the existing functionality. On the other hand fixing bugs may result in both adding a new function and changing the existing functionality. Table 5 shows the changing software in each case.

Table 5. Changing Software

	Adding a Feature	***Fixing a Bug***
Structure	Changes	Changes
New Functionality	Changes	N/A
Functionality	N/A	Changes
Resource Usage	N/A	N/A

Once the parts that need to be changed are recognized, appropriate team members are picked to perform the change. An action plan is defined with a set of tasks and timeline that is feasible and agreed upon by both customer and maintainers. Then change takes place. At this stage, the maintainer tries to locate the code related to adding the logo on the right side and delete it. Then the new function is added by including code to add the logo on the left side of the web page. All maintainers uphold team communication to implement the change.

6.3 Maintenance of the Website Logo

First, run tests to make sure that changes have taken place. Then, ensure that the website page has been updated, and that the logo no longer appears on the right side. Instead, the logo appears on the left side. Use different web-browsers to make sure that the change is stable and users using any web browser can see the change.

Finally, it is recommended to evaluate the change in software by surveying the website users to know if they better recognize the page logo after the change.

7 Conclusion

Most of the current research views software maintenance in terms of technical factors and ignores human dynamics. Our research proposes the use of change theories to model software maintenance. We utilize three change theories: Lewin's, Prochaska and DiClemente's, and Lippit's theories to introduce three different software maintenance models.

Lewin's model is the most basic with only three stages, with a focus on analyzing negative and positive factors through force-field analysis. Lippit's model is more detailed, emphasizing the maintainer's skillset and knowledge in order to

accomplish the change. Lastly, Prochaska and DiClemente's model is a more evolved model, allowing flexibility for the maintenance team to spiral around the stages.

Our findings suggest that considering software maintenance from a different perspective will further benefit the software maintenance process, help developers to better understand their role and increase the possibility of success.

Table 6. Change Theory Models for Website logo Case study

Lewin	***Prochaska and DiClemente's***	***Lippit***	***Website Logo Maintenance Tasks***
	Pre-contemplation		Users don't see Logo as advertisements hide it. Logo Problem is not yet detected.
Unfreezing	Contemplation	Diagnosing the problem	The problem is detected by the customer and conveyed to the maintenance team. Maintainers give a brief description of the problem.
		Assess motivation and capacity for change	Fixing bug or adding a new feature. Cost negotiation.
		Assess change agent's motivation and resource	Perform force-field analysis. Evaluate team's ability.
Moving	Preparation	Select progressive change objective; Develop action plans and establish strategies	Changes are recognized to be at: the structure of the Software, adding new functionality and minimum change of existing functionality
		Choose appropriate role of the change agent i.e. cheerleader, facilitator, expert	Pick appropriate maintainers, and set roles for each one.
	Action	Maintain change through strong group dynamics	The code for adding the logo on the right side is deleted and a new code is added to include the logo on the left side. Website page has been updated, logo no longer appears on the right side. Instead the logo appears on the left side.
Refreezing	Maintenance	Terminate the helping relationship.	End the project. Ensure passing of tests using different web browsers.. Evaluate the change in software. Survey on the website users to know if they better recognize the page logo after the change.

References:

[1] K. Denis, J. Koskinen, and M. Sakkinen, "Fault-Proneness Of Open Source Software: Exploring Its Relations To Internal Software Quality And Maintenance Process," *Open Software Engineering Journal* , vol. 7, pp. 1-23, 2013.

[2] P. Bhatt, K. Williams, G. Shroff, and K. Misra, "Influencing Factors on Outsources Software Maintenance," *ACM SIGSOFT Software Engineering Notes*, 31, 3, pp. 44-49, May 2006.

[3] Kirti Tyagi and Arun Sharma, "Reliability of Component Based Systems – A Critical Survey," *WSEAS Transaction on Computers*, Issue 2, Volume 11, February 2012.

[4] M. Pizka, and F. Deissenboeck, "How to effectively define and measure maintainability," *Software Measurement European Forum*, Rome, Italy, 2007.

[5] Ghulam Murtaza, Naveed Ikramand Abdul Basit, "A Framework for Eliciting Value Proposition from Stakeholders," *WSEAS Transaction on Computers*, Issue 6, Volume 9, June 2010

[6] Zainura Idrus, Siti Z. Z. Abidin, R. Hashim, N. Omar, "Social Awareness: The Power of Digital Elements in Collaborative Environment," *WSEAS Transaction on Computers*, Issue 6, Volume 9, June 2010

[7] R. Canning, "That maintenance iceberg," *EDP Analyzer*, 10(10), 1972.

[8] E. B. Swanson, "The dimensions of maintenance," *Proceedings of the 2nd Intl. Conference on Software Engineering*, San Francisco, CA, October 1976.

[9] N. Chapin, J. Hale, K. Khan, J. Ramil, and W. Tan, "Types of software evolution and software maintenance," *Journal of Software Maintenance and Evolution*: *Research and Practice*, 13(1), January/February 2001.

[10] T. Mens, J. Buckley, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change," *Journal of Software Maintenance and Evolution: Research and Practice*,17(5), September 2005.

[11] M. M. Lehman, D. E. Perry, and J. F. Ramil, "Implications of evolution metrics on software maintenance," *Proceedings Of the IEEE Intl. Conference on Software Maintenance*, Bethesda, Maryland, November 1998.

[12] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and laws of software evolution," The nineties view. In *Proceedings of the Fourth Intl. Software Metrics Symposium*, Albuquerque, NM, November 1997.

[13] M. W. Godfrey and Q. Tu., "Evolution in open source software: A case study. In *Proceedings of 2000 IEEE Intl. Conference on Software Maintenance*, October 2000.

[14] G. Robles, J. M. Gonzalez-Barahona, M. Michlmayr, J. J. Amor, and D. M. German, "Macro-level software evolution: A case study of a large software compilation," *Empirical Software Engineering*, vol. 14, pp. 262–285, 2009.

[15] D. Kumar, Sanjay, A. Sharma, and A. Rana, "Analysis Of Maintainability Models For Object Oriented System," *International Journal On Computer Science & Engineering* 3.12 , pp. 3837-3844, 2011.

[16] F. Zhuo, B. Lowther, P. Oman and J. Hagemeister, "Constructing and testing software maintainability assesement models," *IEEE Computer Society* pp. 61-70, 1993.

[17] A. Kritsonis, "Comparison of Change Theories," *International Journal of Scholarly Academic Intellectual Diversity* vol. 8.1 pp. 1-7, 2004- 2005.

[18] G. Mitchell, "Selecting The Best Theory To Implement Planned Change," *Nursing Management* - UK 20.1, pp. 32-37, 2013.

[19] Lassila, Kathy S., and James C. Brancheau, "Adoption And Utilization Of Commercial Software Packages: Exploring Utilization Equilibria, Transitions, Triggers, And Tracks," *Journal Of Management Information Systems*, vol. 16.2 , pp. 63-90, 1999.

[20] J. Virtamo, "Queueing Theory / Priority Queues." Web. 10 Feb., 2014. <http://www.netlab.tkk.fi/opetus/s383143/kalvot/E_priority.pdf>.

[21] Dehaghani, S. M. Hejazi, and N. Hajrahimi, "Which Factors Affect Software Projects Maintenance Cost More?," *Acta Informatica Medica* , vol. 21.1, pp. 63-66, 2013.

[22] D. Edberg, P. Ivanova, and W. Kuechler, "Methodology Mashups: An Exploration of Processes Used to Maintain Software," *Journal of Management Information Systems*, vol. 28.4, pp. 271-303, 2012.

[23] Ko. J. Andrew, B. Myers, M. Coblenz, and H. Aung, "An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks," *IEEE Computer Society* , vol. 32.12 , pp. 971-987, 2006.

[24] K. Bennet, and V. Rajlich, "Software Maintenance and Evolution: A Roadmap." *ACM* , pp.73-87, 2000.

[25] Ian Sommerville, *Software Engineering*, 9th Edition, Addison-Wesley Publishers Ltd, New York, 2011.

[26] Michael Feathers, *Working Effictively with Legacy Code*, Prentice Hall PTR, 2005.