COMPARATIVE ANALYSIS BETWEEN CHANGE EFFORT ESTIMATION MODELS AND CHARACTERISTICS OF SOFTWARE DEVELOPMENT PROJECT

SITI AISHAH BINTI ZAKARIA

A Research Proposal

Advanced Informatics School (AIS)
Universiti Teknologi Malaysia
Jalan Sultan Yahya Petra
54100 Kuala Lumpur
Malaysia

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Overview

This chapter describes the background of the research, problem statement, research questions, objectives, scope of research, significance of the study, operational definition, and conceptual framework. Figure 1.1 outlines the overall content structure of this chapter.

Introduction	Overview
	Background of the Research
	Statement of Problem
	Research Hypothesis
	Research Questions
	Research Objectives
	Scope of Research
	Significance of Research
	Operational Definition
	Conceptual Framework

Figure 1.1 Introduction Content Structure

1.2 Background of the Research

In software development phase, Software Project Manager (SPM) plays an important role in order to ensure the success of software project. However, the potential point of failure in the software development phase is still high due to change occur on-going software development phase. In view of the fact that SPM fails to manage the change will be the cause towards the software project's failure. Unfortunately, requirements change is an unavoidable software development activity (Ali & Lai, 2016) or certain to happen in software maintenance. However, through a rigorous requirements change management, timely management of these changes which is vital to successful software development can be achieved (Ali & Lai, 2016).

There are two types of information where helps SPM to make an effective decision on change acceptance which are software change impact analysis (Khurana, Tripathi, & Kushwaha, 2013) and software change effort estimation (Kama, Basri, Asl, & Ibrahim, 2014). A change to a software project, even it is a small change, can lead to several accidental effects where usually not obvious and easy to identify (Khurana et al., 2013). (Khurana et al., 2013) mentioned that the main purpose of impact analysis is not only to find the impact set in terms of coding elements but also in terms of effort and resources required for implementing the change, therefore the impact of the requested change in terms of budget could be analysed by analysts. (Kama et al., 2014) highlighted the information of effort estimation which able to help the SPM to make decision is the estimation of the change effort produced by the changes. Therefore, by having reliable estimation on the change effort, it assists the SPM to decide either to accept or reject the changes (Kama et al., 2014).

Software change impact analysis (SCIA) is an activity of identifying the potential consequences of change in which shows how the cost and effort impacted due to the change (Bennett & Rajlich, 2000). (C.-Y. Chen, Liao, & Lin, 2015) mentioned that a change impact analysis (CIA) becomes essential in order to capture both the potential and contextual impacts of a change proposal, and thereby ensure the consistency of the product's integrated content throughout the process of carrying out any changes. This is because as variant product design involves numerous and concurrent attempts at requirement and product modification (C.-Y. Chen et al., 2015). (W. Chen, Wassyng, & Maibaum, 2014) highlighted that an organization is capable to make test plan or to run tests in advanced, saving the period

of time between deployment and release if impacts can be obtained. Besides, instead of having to run all existing tests, organization can know what to test by using the identify impacts. Last but not least, the test suite can be augmented to cover software entities which are affected but not covered in the existing test suite (W. Chen et al., 2014).

In a complex and large-scale enterprise system, software changes and their impact are critical. This is because it is impossible for tester adequately comprehend the impact of the changes (W. Chen et al., 2014). Furthermore, the results are usually in high costs, over-estimation or under-estimation (W. Chen et al., 2014). Therefore, (W. Chen et al., 2014) mentioned that the testing results are very risky to rely on as it depends on the tester's domain knowledge. So, an organization needs SCIA tool in order to identify the impact of a change after or even before a change has been made (W. Chen et al., 2014). In earlier studies, static analysis is a traditional impact analysis approach has been used as the SCIA tool (Wen Chen et al., 2012). Static analysis examines program code and reasons over all possible behaviours that might arise at run-time, and usually it is conservation and sound (W. Chen et al., 2014). (W. Chen et al., 2014) says that the analysis results guaranteed that an accurate description of the program's behaviour by the soundness, no matter what inputs or in what environment the program is run. Meanwhile, conservatism is reporting weaker properties which are guaranteed to be true, preserving soundness, but may not be strong enough to be useful (Ernst, 2003).

Another impact analysis approach is dynamic analysis which operates by executing a program and observing the executions (W. Chen et al., 2014). It needs run-time executions of the program to collect information. Then software developers can measure dynamic impacts of a change by identifying affected entities in the project. Therefore, with ability to running time and precision in finding impacts make dynamic analysis is more efficient than static analysis. This is a great potential to combine static analysis and dynamic analysis in analysing change impact (W. Chen et al., 2014). This is because static analysis may have imprecision result but provides a conservative way to assess the impacts that lead to soundness and safety, meanwhile dynamic analysis used to make analysis more precise. Hence, by combining static analysis and dynamic analysis, the preservation of both safety and precision, and the capability of dealing with extremely large software project can be achieved (W. Chen et al., 2014).

Software change effort estimation (SCEE) is another vital activity of change management. It is used to determine the amount of effort needed to complete a software project (Anandhi & Chezian, 2014). In a study, (Layman, Nagappan, Guckenheimer, Beehler, & Begel, 2008) proposed software development process, scheduling and predictability are essential components to deliver a project task on time and within budget. Scheduling accuracy and understanding the development process can be improved by the effort estimation artifacts. An accurate estimates of software development effort lead to more reliable schedule prediction. Inaccurate or varying effort estimates can serve as indicators which the feature may require further risk assessment and mitigation as work progresses on a software feature and the amount of remaining effort changes. Among the available prediction models for software effort estimation, classical COCOMO (B. W. Boehm, 1981), COCOMO II (B. Boehm et al., 2000) and function point (Albrecht & Gaffney Jr, 1983) are frequently used. These models focus on an algorithmic approach to build software effort estimation models (Layman et al., 2008). Software effort estimation is a key antecedent to software cost estimation (Anandhi & Chezian, 2014). An algorithmic cost modelling technique such as COCOMO for software cost estimation is mostly used because it is simple to estimate the effort in person-months for a project at different stages (Anandhi & Chezian, 2014).

1.3 Statement of the Problem

Change request is an inevitable activity which occurs at any stage of software development life cycle (SDLC) (Ali & Lai, 2016). A change in software development may lead to several effects which often not obvious and easy to identify (Khurana et al., 2013). Therefore, SCIA (Khurana et al., 2013) and SCEE (Kama et al., 2014) are the information that assists SPM to make an effective decision on change acceptance. Without these two information, it will lead towards the success or failure of the software project. However, at early stages of SDLC, effort estimation is a very difficult task as not much information available at that time (Attarzadeh, Ow, & Ieee, 2009; Bardsiri, Jawawi, Hashim, & Khatibi, 2014). The sequence of the problem give rise to the project delay and cost overrun. Hence, the project lead to failure which cause client dissatisfaction.

To recap, less information at early software development will cause the failure of the project. Thus, information of organization or software project should be taken before implement the project. However, while getting the information, SPM realizes that there is so many different type of characteristics of software project can be found (Nagar & Dixit, 2012). A question that arises, whether the characteristics of software project are suit to all type of the existing effort estimation models? Unfortunately, to the best of our knowledge are not all effort estimation models meet the requirement of characteristics of software development project. This has been mentioned in previous studies (Shepperd, Schofield, & Kitchenham, 1996; Zheng, Wang, Zheng, & Shi, 2009) that predicting using Lines of Code (LOC) fails to reach the accurate result for software effort estimation when the size of software project grows. Although, LOC uses the different languages, it could be a trouble which it could create different values of LOC. However, the Function Point is used to solve the addressed problems which improve the accuracy of effort estimation.

Effort estimation activity deals with predicting the effort required to complete the task of software project (Sharma & Kushwaha, 2012). An accurate estimation is required in software development because inappropriate estimates lead to failure during the implementation of software project (Sharma & Kushwaha, 2012). Thus, effort estimation can be evaluated by executes an efficiency analysis to show the effectiveness of effort estimation models. The software project characteristics are identified based on the SDEE efficiency factors which are development type, organization type, development platform and overall evaluation (Bardsiri et al., 2014). Besides, (Bardsiri et al., 2014) highlighted that the SDEE efficiency factors are defined to increase the applicability and flexibility of the effort estimation models.

There are four characteristics identified which is a part of important requirement of effort estimation models. The characteristics are size of project, complexity of project, programmer capability and language tools experience (Nagar & Dixit, 2012; Rosa, Packard, Krupanand, Bilbro, & Hodal, 2013). The characteristics of size of project, complexity of project, programmer capability and language tools experience are identified based on the SDEE efficiency factors are organization type, overall evaluation, development type and development platform respectively. Based on (Nagar & Dixit, 2012; Sehra, Kaur, Brar, & Kaur, 2014), there are three models of complexity factor which are organic, semi-detached, and embedded. It says that organic is a small-scale project and simple complexity, semi-

detached is medium-scale project and average complexity, meanwhile embedded is a large-scale project and complex (Nagar & Dixit, 2012; Sehra et al., 2014).

In software development phase, SPM plays a vital role in order to make correct decision which lead the success of the project. Therefore, to resolve the problems as mentioned earlier, a comparative analysis of effort estimation models need to be performed. Hence, it is even more useful because the characteristics of software project are obtained at very early stage of SDLC which helps SPM to choose the effort estimation model that suit the best with the project characteristics. Last but not least, the characteristics of software project is identified based on the efficiency factors of effort estimation which helps SPM to identify the most flexible and applicable effort estimation models. This will avoid the project delay, cost overrun and the failure of effort estimation. Next, it leads toward the success of project which make the client satisfied.

1.4 Research Hypothesis

Not all effort estimation models are suit to the characteristics of software development project.

1.5 Research Questions

i. What are the current change effort estimation models in software development project?
ii. What are the characteristics of software development project?
iii. How to perform the comparative analysis between the characteristics of software development project and the current change effort estimation models?
iv. What the effort estimation models that suit the characteristics of software development project?

1.6 Research Objectives

i. To analyse current change effort estimation models for software development phase.
ii. To identify list of characteristics of software development project.
iii. To perform comparative analysis between the characteristics of software development project and the current change effort estimation models.
iv. To identify the best effort estimation models that suit to the characteristics of software development project?

1.7 Scopes of Research

The main reason of defining a research scope is to focus the research area and emphasize the boundaries and constraints of the study. Limitation of the research scopes are as follow:

1.7.1 Research Context

The research aims to produce a comparative analysis between the selected characteristics of software development project and the selected of effort estimation models. The selected characteristics of software development project are: (1) size of project; (2) complexity of project; (3) programmer experience; and (4) language and tools experience. The selected characteristics of software development project are based on the efficiency analysis which are: (1) organization type; (2) overall evaluation; (3) development type; and (4) development platform. The selected effort estimation models are: (1) LOC; (2) FPA; (3) UCP; and (4) CEPM.

1.7.2 Research Challenges

Since this research will focus on software development phase, there will be challenges in order to apprehend the actual information in real software projects during software development process. The following challenges might affect the research's timeline and milestones. Hence, the researcher outlines following challenges:

- Real software project – Real software project require participation from business organization or industry. Since this study might not be able to benefit them directly, it is hard to capture and collect relevant data for this research. This is because real software project in the industry are constraint with other factors; for instance are confidentiality, commercial obligations, politics, and complex organization structures.

- Model / Tool – This research executes a comparative analysis between four existing effort estimation models which developed by other researchers. As this study may give less benefit to them, it is difficult to get the real models of effort estimation.

1.8 Significance of the Research

A change may occurs at any phase of software development life cycle which can be impacted the cost, time and effort of software development. Without an effective decision made by SPM, the project may lead towards cost overrun and project delay. Hence, effort estimation models are an effective technique to predict the amount of effort needed in a software development project which used by SPM in order to make a good decision. However, there are many effort estimation that exists nowadays. A question arises in which change effort estimation models should be used by SPM in order to make an effective decision in change acceptance.

At early of software development, SPM notices that there are many characteristics of software development project exist. Thus, the selection of reliable and effective effort estimation model is important in order to meet the requirement of the characteristics of software development project. Therefore, a comparative analysis between effort estimation

models and the characteristics of software development project is performed. The researcher believes this analysis will helps SPM to make an effective decision by gaining the information of which effort estimation models that suit to the characteristics of software development project.

1.9 Operational Definition

The operational definitions of terminologies used in this research are presented below:

Software development	:	The stages of the software process in completing a software development project.
Change	:	The amendment that occurs during software development phase to meet the current requirement of the project.
Change impact analysis	:	A process of identifying the potential consequences of cost and effort due to the change.
Effort estimation	:	An activity of determining the amount of effort required to develop a software development project.
Effort estimation model	:	A technique used to determine the amount of effort required to develop a project.
Efficiency	:	A process that measured the effectiveness of effort estimation model.
Suit	:	A reliable and effective of an effort estimation model for the characteristics of software development project.

1.10 Conceptual Framework

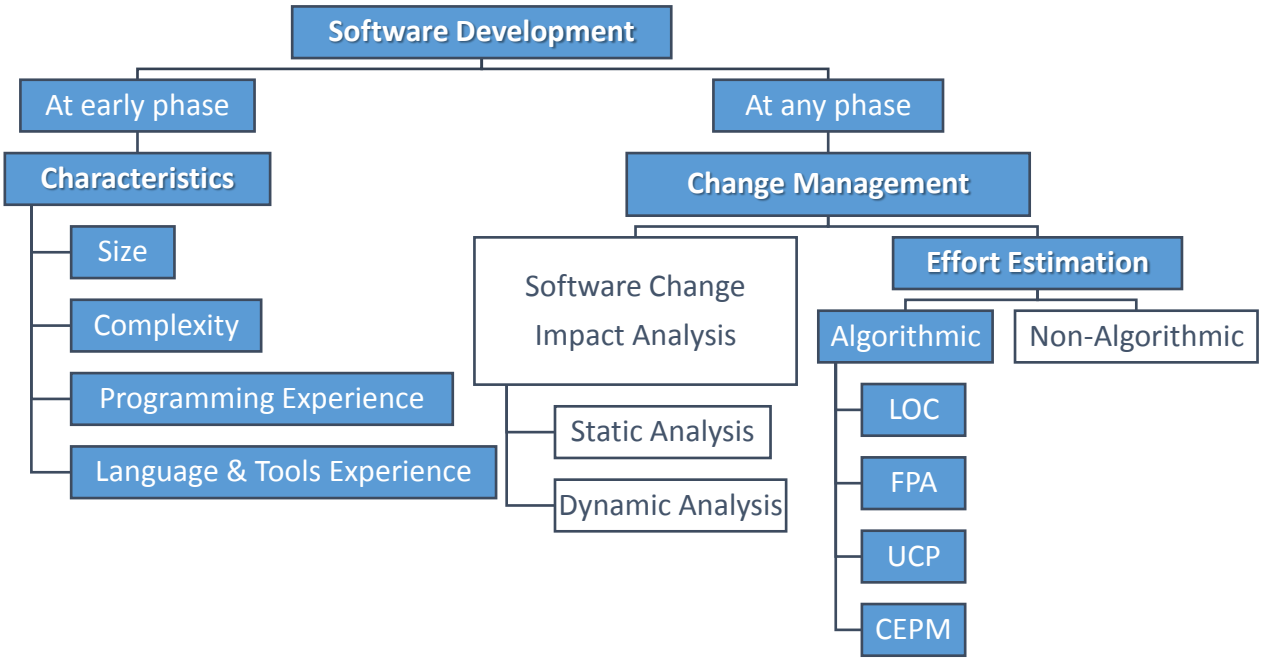

Figure 1.2 Conceptual Framework

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter describes the introduction and software development life cycle (SDLC) as shown in Figure 2.1 which outlines the overall content structure for this chapter.

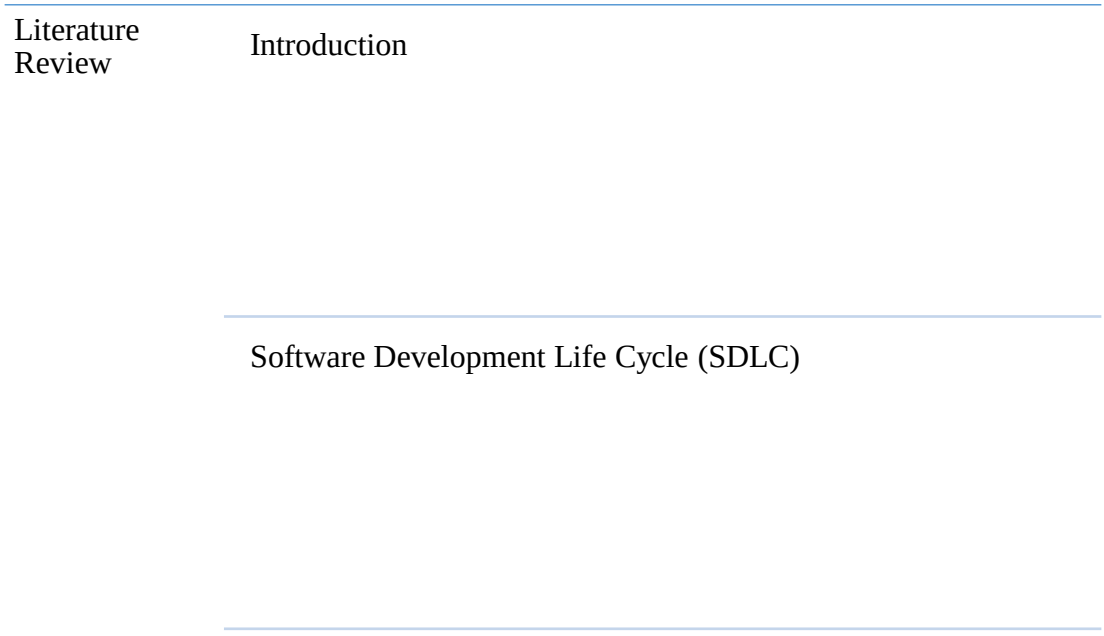

Figure 2.1 Literature Review Content Structure

2.2 Software Development Life Cycle (SDLC)

Software engineering (SE) is "a systematic approach to the analysis, design, assessment, implementation, test, maintenance and reengineering of software, that is, the application of engineering to software. In the software engineering approach, several models for the software life cycle are defined or known as SDLC, and many methodologies for the definition and assessment of the different phases of a life-cycle model" (Laplante, 2007). (Maheshwari & Jain, 2012) describes that SE is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of a software, and the study these approaches; that is, the application of engineering to software because it integrates significant mathematics, computer science and practises whose origins are in Engineering.

A software development life cycle (SDLC) is a structure imposed on the development of a software product which often considered as a subset of SDLC (Maheshwari & Jain, 2012; Taya & Gupta, 2011). There are several models for such processes, each describing approaches to variety of tasks or activities that take place during the process. It aims to be the standard that defines all the tasks required for developing and maintaining software (Maheshwari & Jain, 2012). Basically, SE processes are composed of many activities which involved requirements analysis, specification, software architecture, implementation, testing, documentation, training and support, and maintenance (Maheshwari & Jain, 2012).

2.2.1 SDLC Models

There are several types of SDLC models are Waterfall Model, V-Shape Model, CMM Model, RUP Model, Prototype Model, Incremental Model, Spiral Model, RAD Model, JAD Model, and Agile Model. The following sub-sections describes the SDLC models in details.

2.2.1.1 Waterfall Model

The waterfall model is a conventional, linear, sequential or traditional waterfall software life cycle model (Maheshwari & Jain, 2012). It was originally proposed by Benington in 1959 (Benington, 1987) and later popularized by Royce in 1970 (Royce, 1987). The model is also known as cascade model and has been considered as the oldest and widely used model. Waterfall model identifies design flaws before it develops as early as at planning stage. It is a sequential development approach which the process flows steadily downward like a waterfall through the phases of (1) Software Planning, (2) Software Requirements, (3) Requirement Analysis, (4) Program Design, (5) Implementation, (6) Testing & Validation, and (7) Operation & Maintenance. There are two types of waterfall models (Maheshwari & Jain, 2012) which are original waterfall model which introduced by Benington (Benington, 1987) and iterative waterfall model that popularized by Royce (Royce, 1987).

2.2.1.1.1 Original Waterfall Model

The original waterfall model has three basic principles which are project, emphasis and tight control (Maheshwari & Jain, 2012). The project is divide into sequential stages, with some overlap and splash back acceptable between stages. The emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time. Last but not least, tight control is maintained over the life of the project via extensive written documentation, formal reviews, and approval or signoff by the user and information technology management occurring at the end of the most phases before beginning the next stage.

The advantage of original waterfall model is each phase has well defined deliverable or milestone. It is simple to use and understandable. Besides, it acts as template into which methods for analysis, design, code test and maintenance can be placed.

The disadvantage of this model is each of the development phase is not able to go to the previous step as shown in Figure 2.2. Therefore, if the design phase has gone wrong, the development process can get very complicated in the implementation phase. Typically, the

clients are not very clear of what they need from the software which cause changes that lead to a lot of confusion. The small changes or errors which arise in the completed software will cause a lot of problems.

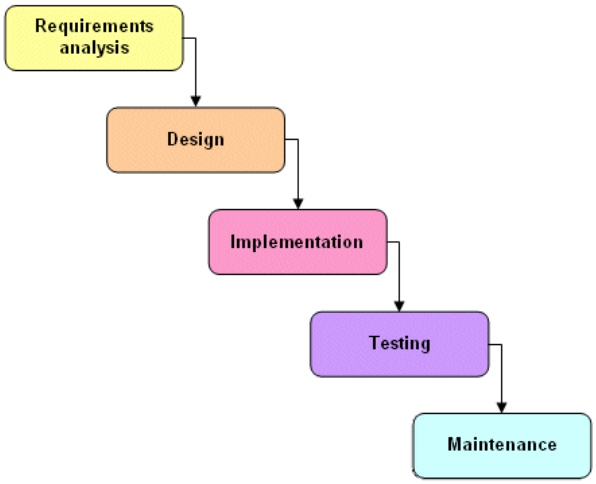

Figure 2.2 Original Waterfall Model (Benington, 1987)

2.2.1.1.2 Iterative Waterfall Model

The iterative waterfall model is introduced to resolved the problems of the original waterfall model (Maheshwari & Jain, 2012). This is because the problems created a demand for a new method of developing systems which could provide faster results, require less up-front information and offer greater flexibility. This model, the project is divided into small parts. Hence, it allows the development team to demonstrate preliminary results in the process and obtain the valuable feedback from system users. Each of iteration is a Mini-Waterfall process with the feedback from one stage which provides an important information for the design of the next stage.

The advantage of this model is much better of the software process compared to original waterfall model. It allows feedback to proceeding phases as shown in Figure 2.3. This model also can be used for project at which point the requirements are difficult to understand.

The disadvantage of this model is it is hard to manage. This is because it is no clear milestones in the development process. Besides, no phase is really completed.

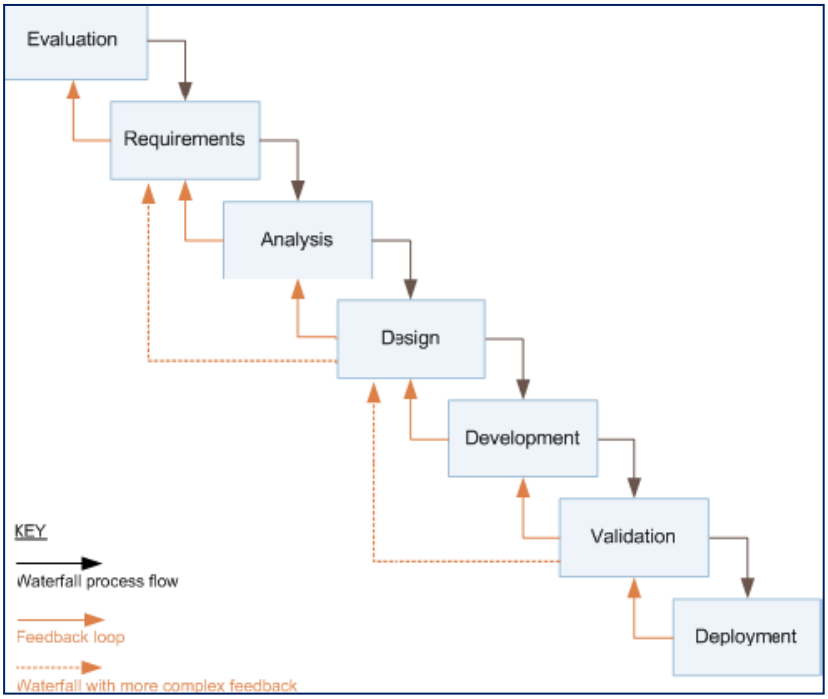

Figure 2.3 Iteration Waterfall Model (Royce, 1987)

2.2.1.2 V-Shape Model

The V-shape model was initially popularized by Osborne (Osborne, Brummond, Hart, Zarean, & Conger, 2005) as shown in Figure 2.4. This model can be considered as an extension of the waterfall model (Forsberg & Mooz, 1991). It is linear software development model, but the process steps are bent upwards after the implementation stage, to the form the typical V shape instead of flows downward as waterfall model. This model demonstrates the relationships between each stage of the development life cycle and its related stage of testing. The vertical and horizontal axes of V-shape model represent level of abstraction and time or project completeness, respectively.

The advantage of V-shape model is it is simple and easy to use. Besides, it saves a lot time as the testing activities are happened very well before coding. Therefore, it has higher chance of success compared to the waterfall model. It also a proactive defect tracking which is defects are found at early phase. Last but not least, this model works well for small projects as the requirements are easily understood.

The disadvantage of this model is it is very rigid and least flexible. Software is developed in the implementation stage; thus no early prototypes of the software are produced. If any changes happen in the middle stage of the development process, then the test documents and requirement documents have to be updated.

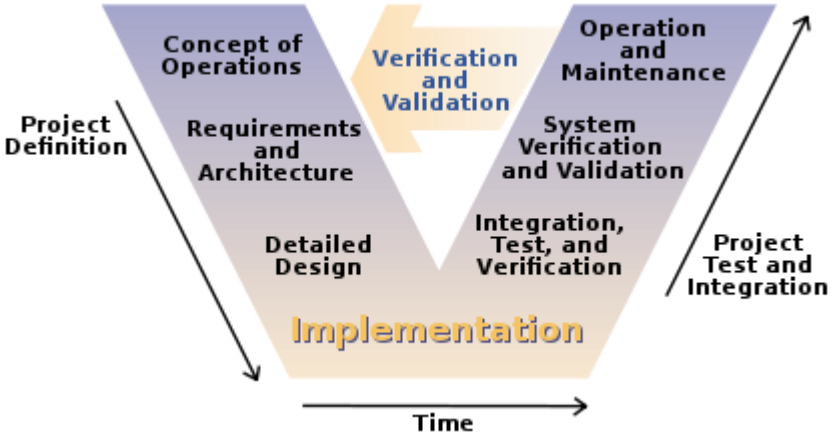

Figure 2.4 V-Shape Model (Osborne et al., 2005)

2.2.1.3 Capability Maturity Model (CMM) Model

The CMM model was first described by Watts Humphrey in 1989 and it was later published in 1993 (Paulk, Curtis, Chrissis, & Weber, 1993). It is originally developed as a tool for objectively assessing the ability of government contractor's processes to implement a contracted software project. In the field of software development, this model is used as a general model to aid in business process normally, and has been used widely in government, industry, commerce and software-development organizations.

The advantage of CMM model is easily understood as shown in Figure 2.5 and high guarantee of success. It is also highly flexible software development model.

The disadvantage of this model is it is not able to go to the previous phase where each phase looks as a target. Hence, it can be dangerous when fixated on reaching to the next phase while losing the perspective and the real goal which is to improve the development processes.

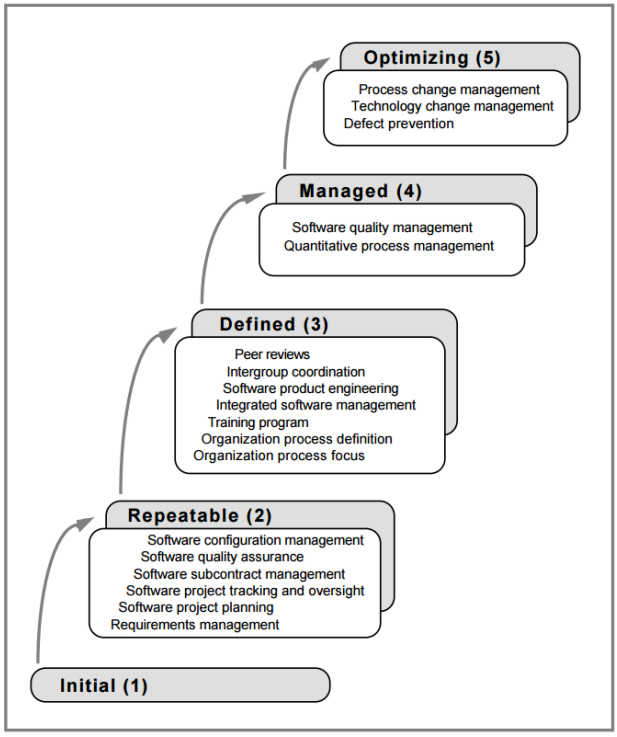

Figure 2.5 CMM Model (Paulk et al., 1993)

2.2.1.4 Rational Unified Process (RUP) Model

The RUP model is originally developed in 1990s and later marketed by Rational Software Corporation, a division of IBM (Kruchten, 2004). It is a use case driven model (Jacobson, Booch, Rumbaugh, Rumbaugh, & Booch, 1999). This model was designed with regards to the development requirements of object-oriented software. It consists nine core disciplines of fundamental workflows (Khan & Beg, 2013) which are:

- Business Modelling
- Requirements
- Analysis and Design
- Implementation
- Test
- Deployment
- Configuration and Change Management
- Project Management
- Environment

These nine core disciplines are not separated in time. Conversely, they are implemented at the same time throughout the four phases; (1) inception, (2) elaboration, (3) construction, and (4) transition. As Figure 2.6 indicates that the code is slightly get written at early of the project lifecycle, but the amount is not zero. Tardy in the project, most of the requisites are kenned, but some incipient ones are still identified.

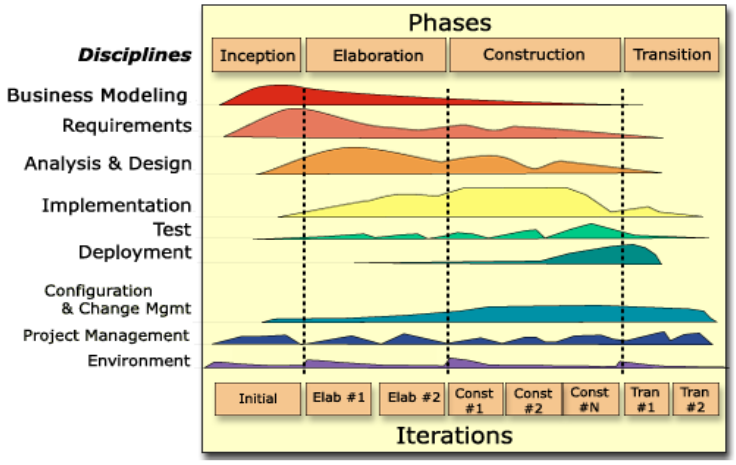

Figure 2.6 RUP Model (Jacobson et al., 1999; Khan & Beg, 2013)

2.2.1.5 Prototype Model

The prototype model is the development approach of activities in software development, the creation of prototypes like incomplete versions of the software program being developed (Maheshwari & Jain, 2012). This model is not a standalone and complete development methodology. It attempts to minimize inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process. User is involved throughout the development process, which increases the likelihood of user acceptance of the final implementation. Small-size mock-ups of the system are developed following an iterative modification process until the Prototype evolves to meet the users. While most prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system. A basic understanding of the fundamental business problem is essential to avoid solving the wrong problem.

The advantage of this model is users get an idea of what the final system looks like by early visibility of the prototype which is encourages active involvement between producer and users. It enables a higher output for user. Besides, this model improves the system development speed and cost effective. In addition, it helps to identify any problems with the efficacy of earlier design, requirements analysis and coding activities.

The disadvantage of prototype model is there is possibility of system leads to be left uncompleted and executing system before they are prepared. The producer might produce a system insufficient for overall organization requires. This model usually lacks of flexibility. It is not suitable for large applications and difficult to manage a project.

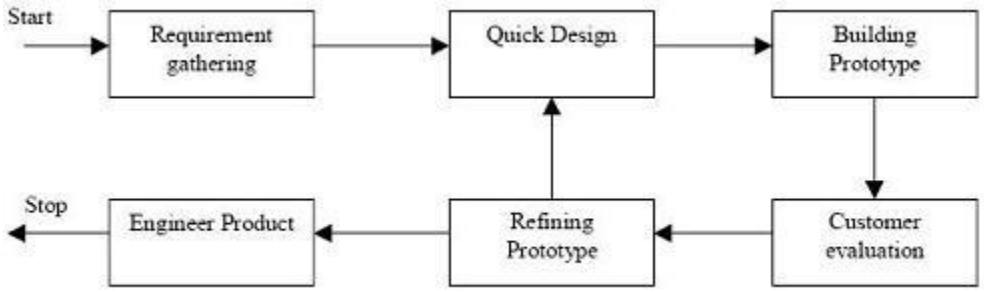

Figure 2.7 Prototype Model (Smith, 1991)

2.2.1.6 Incremental Model

The incremental model is a technique of software development that the product is designed, implemented and tested incrementally till the product is completed as shown in Figure 2.8. It includes both software development and software maintenance. The product is determines as completed when it meets all the requirements. This model applies the waterfall model in stages (Bindal & Mehta, 2015; Pressman, 2010). It is multiple linear development stages which making a cycle a "multi-level" cycle (Bindal & Mehta, 2015). Cycles are divided up into smaller, more easily managed modules.

The advantage of this model is it generates working software rapidly during software life cycle. Besides, it is more flexible, less costly to change requirements and scope. It is easier to test and debug during a smaller changing process. In addition, it is easy to manage risk and the risky piece is quickly identified.

The disadvantage of incremental model is it requires a good planning, structure and design. Besides, it requires a clear and complete processing of the whole system before broken down and built incrementally. It iteration processing is very rigid and they do not overlap.

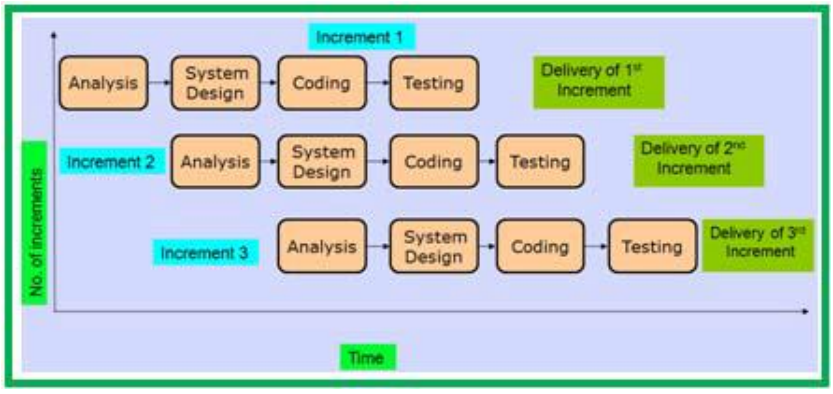

Figure 2.8 Incremental Model (Khan & Beg, 2013)

2.2.1.7 Spiral Model

The spiral model is introduced by Boehm. It is similar to incremental model (Bindal & Mehta, 2015) which combining elements of both design and prototyping in software development phases (Maheshwari & Jain, 2012). This model is focus on risk assessment and on decreasing project risk by breaking a project into smaller parts and providing more ease-of-change during the development process, and also providing the opportunity to evaluate risks and weigh consideration of project continuation throughout the life cycle. Each cycle involves a progression through the same sequence of steps, for each part of the product and for each of its levels of elaboration, from an overall concept-of-operation document down to the coding of each individual program.

The advantage of every time a new prototype is obtained, it is revaluated again and again by the client, and so more client involvement is there. Better productivity through reuse capabilities. Proper control over cost, time and manpower requirement for a project work. Errors are eliminated in early phases of project development only.

The disadvantage is the model requires risk identification, its projection, risk assessment and risk management which is not an easy task. The cost and time estimations are also not very easy. This model is not suitable for smaller project as then the cost of risk analysis is greater than cost of the entire project.

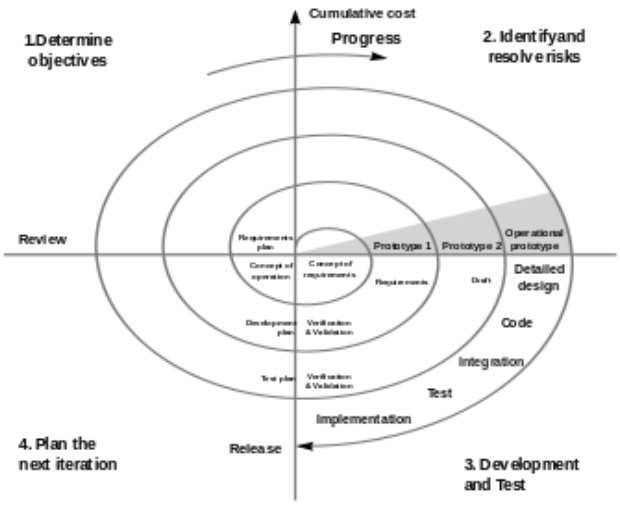

Figure 2.9 Spriral Model

2.2.1.8 Rapid Application Development (RAD) Model

Rapid Application Development (RAD) is a methodology that uses prototyping as a mechanism principally developed by Martin (1991). Martin (1991) approaches to RAD divides the process into four distinct phases as in **Error! Reference source not found.** which re: (1) requirement planning phase; (2) user design phase; (3) construction phase; and (4) cutover phase.

In requirement planning phase, the process starts by the combining the elements of the system planning and systems analysis phases of the SDLC. Team of users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements. It ends when the team agrees on the key issues and obtains management authorization to continue.

In user design phase, users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. User Design is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.

Next, the construction phase focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.

Finally, the cutover phase resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner.

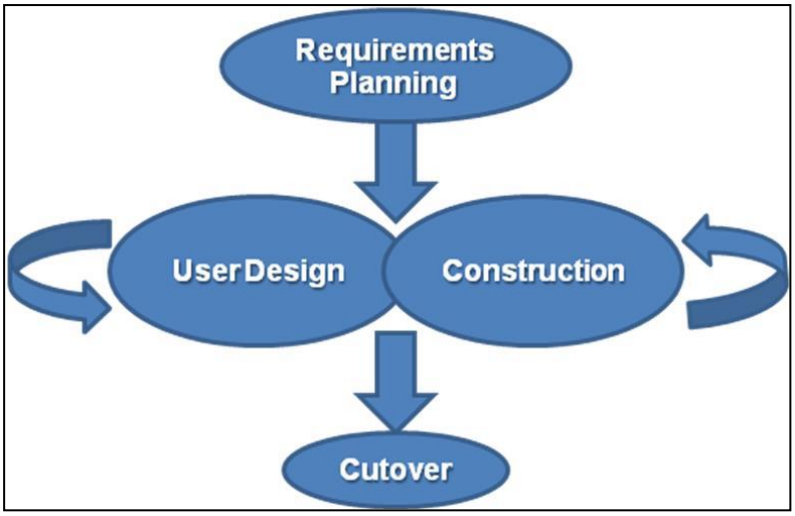

Figure 2.10 Prototyping approach used by RAD (Martin, 1991)

2.2.1.9 Joint Application Development (JAD) Model

The JAD model is a methodology that involves the client or end user in the design and development of an application, through a succession of collaborative workshops called JAD sessions. Before the advent of JAD, requirements were identified by interviewing stakeholders individually. The ineffectiveness of this interviewing technique, which focused on individual input rather than group consensus, led to the development of the JAD approach.

The advantage of this model is it allows key users to participate effectively. While properly used, JAD can result in a more accurate statement of system requirements, a better understanding of common goals, and a stronger commitment to the success of the new system. However, the disadvantage of JAD model is expensive and can be cumbersome if the group is too large relative to the size of the project.

2.2.1.10 Agile Model

The agile model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. The agile method breaks the product into small incremental builds. These build are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is displayed to client and important stakeholder.

The advantage of agile model is easy to manage and very realistic approach to software development. It promotes teamwork and cross training. Its functionality can be developed rapidly and demonstrated. It is suitable for fixed or changing requirements. Besides, it enables concurrent development and delivery within an overall planned context.

The disadvantage of this model is not suitable for handling complex dependencies. Besides, it is more risk of sustainability, maintainability and extensibility. There is very high individual dependency, since there is minimum documentation generated. An overall plan, an agile leader and agile PM practice is a must without which it will not work.

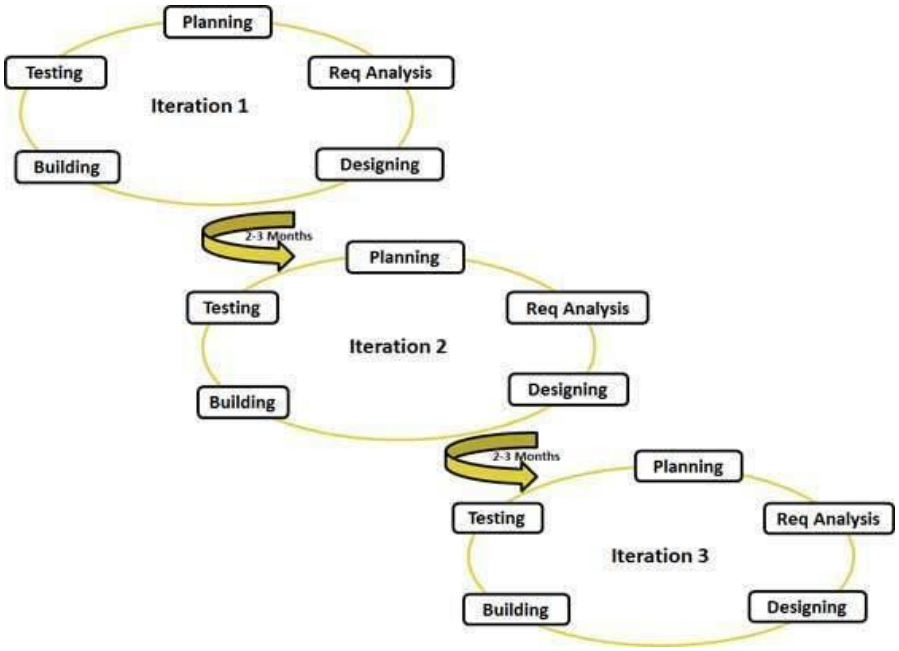

Figure 2.11 Agile Model

2.2.2 Comparative analysis of SDLC Models

Table 2.1 Comparative analysis of SDLC

X**Model/ Features**	**Waterfall**	**V-Shape**	**CMM**	**RUP**	**Prototype**	**Incremental**	**Spiral**	**RAD**	**JAD**	**Agile**
Requirement Specifications	Beginning	Beginning	At second level	Beginning	Frequently changed	Beginning	Beginning	Time-box released	Prototyped	Frequently changed
Understanding Requirements	Well understood	Easily understood	Easily understood	Difficult to understood	Not well understood	Well understood	Well understood	Easily understood	Easily understood	Well understood
Cost	Low	Expensive	High	Expensive	High	Low	Expensive	Low	Expensive	Very High
Guarantee of Success	Low	High	High	Not guaranteed	Good	High	High	Good	Low but for long period	Very high
Resources Control	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No
Cost Control	Yes	Yes	Varies	Yes	No	No	Yes	Yes	Yes	Yes
Simplicity	Simple	Intermediate	Intermediate	Simple and clear	Simple	Intermediate	Intermediate	Very simple	Simple	Complex
Risk Involvement	High	Low	Varies acc to level	Critical risks in the early stages	Low	Easily manage	Low	Very low	Varies	Reduced
Expertise Required	High	Medium	Varies acc to level	Yes	Medium	High	High	Medium	High	Very high
Changes Incorporated	Difficult	Difficult	Medium	Easy	Easy	Easy	Easy	Easy	Medium	Difficult
Risk Analysis	Only at beginning	Yes	Yes	Yes	No risk analysis	No risk analysis	Yes	Low	Yes	Yes
User Involvement	Only at beginning	At the beginning	Only at beginning	At the beginning and at the last phase	High	Intermediate	High	Only at beginning	In the design and development	High
Overlapping Phases	No such phase	No	No	Yes	Yes	No	Yes	No	No	Yes
Flexibility	Rigid	Little flexible	Highly flexible	Considerable	Highly flexible	Less flexible	Flexible	High	Flexible	Highly flexible
Maintenance	Least glamorous	Least	Typical	Promote maintainability	Routine maintainability	Promotes maintainability	Typical	Easily maintained	Rigorously at all times	Promotes maintainability
Integrity & Security	Vital	Limited	Limited	Very important	Weak	Robust	High	Vital	Limited	Demonstrable
Reusability	Limited	To some extent	Yes	Supports reusability of the existing classes	Weak	Yes	Yes	Some extent	Limited	Use Case *Reuse*
Interface	Minimal	Minimal	Crucial	User interface	Crucial	Crucial	Crucial	Minimal	Crucial	Model-driven
Documentation & Training required	Vital	Yes	Yes	Yes	Weak	Yes	Yes	Limited	Limited	Yes
Time Frame	Long	Acc to project size	Quite long	Short *time frame*	Short	Very long	Long	Short	Medium	Least possible

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter discusses the research methodology which includes discussions in following topics: (1) introduction, (2) SDLC process, and (3) research schedule. Figure 3.1 outlines the overall content structure for this chapter.

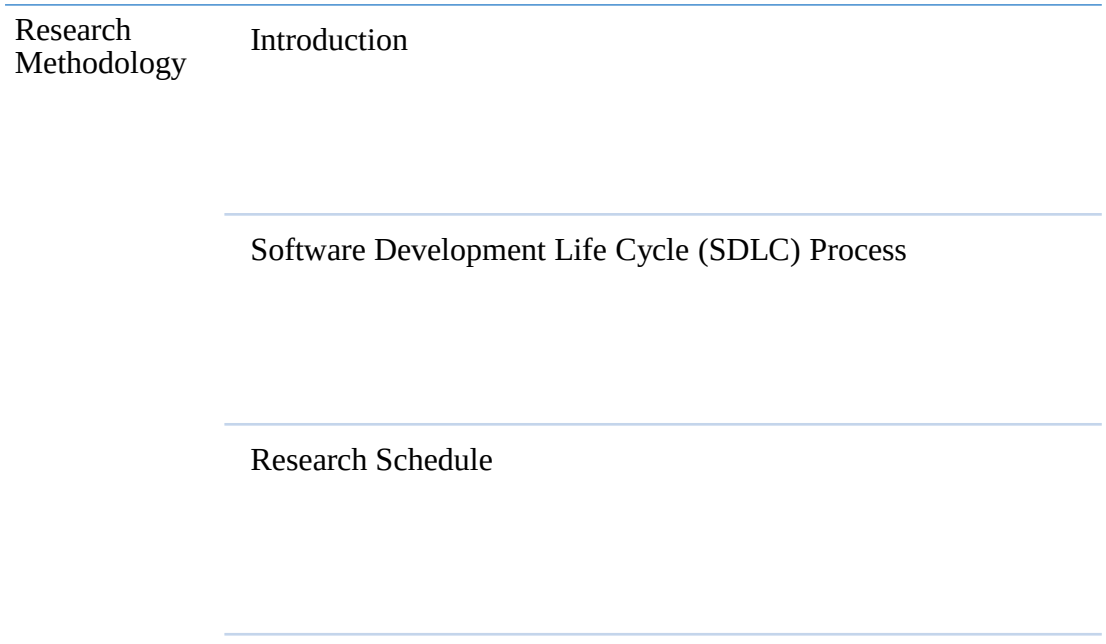

Figure 3.1 Research Methodology Content Structure

3.2 Software Development Life Cycle (SDLC) Process

Figure 3.2 Software Development Life Cycle (SDLC) Process

3.2.1 Statement of problem

The research starts with identifying the problems of the current system. The critical problem is identified and defines as the highest priority requirement. In this respect, it helps to understand the output or goal of the study. Besides, this stage is used to be a guide line to determine the scope of the study. Therefore, implementation phase will depend generally to this stage. These are not the only advantages of using problems statement in developing life cycle. Nevertheless, it increases the understanding the purpose of the study.

3.2.2 Literature Review

Next, literature reviews which related to problem statement are collected and make comparison between current models and the characteristics of software development project. The main idea is to find new or adequate approach or method to compare the current models to the new system successfully and accurately. Besides, at this stage, it is important to group authors who draw similar conclusion to prove a statement is correct and true. However, authors are in disagreement should be noted to highlight the scopes or limitations of the study. Last but not least, this stage will show how our study relates to the previous studies and relates to the literature in general. What method used in searching of the related literature view will be explains in the next stage which is data collection.

3.2.3 Data Collection

Then, the following stage is data collection which is one of important aspects in capturing system requirements. This stage generally shows the quality of the project. It is means by how the data collection is well organized, thus the quality of the project and user satisfaction is increased. There are several methods are used in this study. First, data can be collected by surfing via Internet to the web page such as www.library.utm.my. From there, we can visit subscribed online database like IEEEXplore Digital Library and Web of Science

including e-journal, e-book, e-standard, e-thesis, e-proceeding, etc. It is only accessible to UTM Library registered members and remotely via EZprozy.

Next, most of the system requirements were obtained through frequent discussions with my supervisor, Dr. Mohd Nazri Kama whom is consider as a potential client in the software development project. This process is very important in order to get better understanding about the system. The purpose of data collection techniques is:

- To identify the client's need and constraints.
- To identify the characteristics in the software development project.
- To know the current models used in effort estimation.
- To be able to understand the current effort estimation models.

3.2.4 Data Analysis

Next, data analysis is a vital process which is it has been executed after the data is collected. Data analysis is a stage of inspecting or modelling data with the aim of discovering useful information, suggesting conclusions, and supporting decision-making. In this stage, comparison between the selected current effort estimation models is done. The results of the comparison will be used to analyse the most reliable and effective models that can be used by software project manager which convenient to the characteristics of software development project. Hence, at this stage, the analysed data answers the research questions and hypothesis or disprove the research hypothesis.

3.2.5 Evaluation

The last stage is evaluation which allows software project manager get the information in which the most suitable effort estimation model that should be used regarding the characteristics of software development project. In short, this stage aims to assist software project manager to make an effective decision at early of software development phase.

Hence, the project runs smoothly as software project manager manage to make a good decision with help of this research.

Table 3.1 Evaluation Form

Effort estimation models/ Characteristics of software development project	**Size**			**Complexity**			**Programmer Experience**			**Language and Tools Experience**		
	Small	Medium	Large	Simple	Average	Complex	Entry	Intermediate	Expert	Entry	Intermediate	Expert
LOC												
FPA												
UCP												
CEPM												

3.3 Research Schedule

Table 3.2 Research Schedule

No	Activity	2016					2017			
		Feb - Apr	Apr - Jun	Jun - Aug	Aug - Oct	Oct - Dec	Dec - Feb	Feb - Apr	Apr - Jun	Jun - Aug
1	Planning									
2	Problem Statement									
3	Literature Review									
4	Data Collection									
5	Data Analysis									
6	Evaluation									
7	Documentation									

REFERENCES

Albrecht, A. J., & Gaffney Jr, J. E. (1983). Software function, source lines of code, and development effort prediction: a software science validation. *Software Engineering, IEEE Transactions on*(6), 639-648.

Ali, N., & Lai, R. (2016). A method of requirements change management for global software development. *Information and Software Technology, 70*, 49-67. doi:10.1016/j.infsof.2015.09.005

Anandhi, V., & Chezian, R. M. (2014). *Regression Techniques in Software Effort Estimation Using COCOMO Dataset.* Paper presented at the Intelligent Computing Applications (ICICA), 2014 International Conference on.

Attarzadeh, I., Ow, S. H., & Ieee. (2009). *Proposing a New High Performance Model for Software Cost Estimation*.

Bardsiri, V. K., Jawawi, D. N. A., Hashim, S. Z. M., & Khatibi, E. (2014). A flexible method to estimate the software development effort based on the classification of projects and localization of comparisons. *Empirical Software Engineering, 19*(4), 857-884.

Benington, H. D. (1987). *Production of large computer programs*. Paper presented at the Proceedings of the 9th international conference on Software Engineering, Monterey, California, USA.

Bennett, K. H., & Rajlich, V. (2000). *Software maintenance and evolution: a roadmap*. Paper presented at the Proceeding of the Conference on The Future of Software Engineering, Limerick, Ireland.

Bindal, N., & Mehta, A. (2015). SURVEY ON SOFTWARE DEVELOPMENTPROCESSING MODELS. *Analysis, 2*(03).

Boehm, B., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., . . . Steece, B. (2000). Cost estimation with COCOMO II. *ed: Upper Saddle River, NJ: Prentice-Hall*.

Boehm, B. W. (1981). *Software engineering economics* (Vol. 197): Prentice-hall Englewood Cliffs (NJ).

Chen, C.-Y., Liao, G.-Y., & Lin, K.-S. (2015). An attribute-based and object-oriented approach with system implementation for change impact analysis in variant product design. *Computer-Aided Design, 62*, 203-217. doi:10.1016/j.cad.2014.11.006

Chen, W., Iqbal, A., Abdrakhmanov, A., Parlar, J., George, C., Lawford, M., . . . Wassyng, A. (2012). Large-scale enterprise systems: Changes and impacts *Enterprise Information Systems* (pp. 274-290): Springer.

Chen, W., Wassyng, A., & Maibaum, T. (2014). Combining Static and Dynamic Impact Analysis for Large-Scale Enterprise Systems. In A. Jedlitschka, P. Kuvaja, M.

Kuhrmann, T. Mannisto, J. Munch, & M. Raatikainen (Eds.), *Product-Focused Software Process Improvement, Profes 2014* (Vol. 8892, pp. 224-238). Berlin: Springer-Verlag Berlin.

Ernst, M. D. (2003). *Static and dynamic analysis: Synergy and duality.* Paper presented at the WODA 2003: ICSE Workshop on Dynamic Analysis.

Forsberg, K., & Mooz, H. (1991). *The relationship of system engineering to the project cycle.* Paper presented at the INCOSE International Symposium.

Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., & Booch, G. (1999). *The unified software development process* (Vol. 1): Addison-Wesley Reading.

Kama, N., Basri, S., Asl, M. H., & Ibrahim, R. (2014). COCHCOMO: A Change Effort Estimation Tool for Software Development Phase. In H. Fujita, A. Selamat, & H. Haron (Eds.), *New Trends in Software Methodologies, Tools and Techniques* (Vol. 265, pp. 1029-1045).

Khan, P., & Beg, M. S. (2013). *Extended decision support matrix for selection of sdlc-models on traditional and agile software development projects.* Paper presented at the Advanced Computing and Communication Technologies (ACCT), 2013 Third International Conference on.

Khurana, P., Tripathi, A., & Kushwaha, D. S. (2013). Change Impact Analysis and its Regression Test Effort Estimation. In B. M. Kalra, D. Garg, R. Prasad, & S. Kumar (Eds.), *Proceedings of the 2013 3rd Ieee International Advance Computing Conference* (pp. 1420-1424). New York: Ieee.

Kruchten, P. (2004). *The rational unified process: an introduction*: Addison-Wesley Professional.

Laplante, P. A. (2007). *What every engineer should know about software engineering*: CRC Press.

Layman, L., Nagappan, N., Guckenheimer, S., Beehler, J., & Begel, A. (2008). *Mining software effort data: preliminary analysis of visual studio team system data.* Paper presented at the Proceedings of the 2008 international working conference on Mining software repositories.

Maheshwari, S., & Jain, D. C. (2012). A Comparative Analysis of Different types of Models in Software Development Life Cycle. *International Journal of Advanced Research in Computer Science and Software Engineering*, *2*(5), 285-290.

Martin, J. (1991). *Rapid application development*: Macmillan Publishing Co., Inc.

Nagar, C., & Dixit, A. (2012). Efforts Estimation by Combining the Use Case Point and COCOMO. *International Journal of Computer Applications*, *52*(7).

Osborne, L., Brummond, J., Hart, R. D., Zarean, M., & Conger, S. M. (2005). *Clarus: Concept of operations*. Retrieved from

Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). Capability maturity model, version 1.1. *Software, IEEE, 10*(4), 18-27.

Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. Boston: McGraw Hill.

Rosa, W., Packard, T., Krupanand, A., Bilbro, J. W., & Hodal, M. M. (2013). COTS integration and estimation for ERP. *Journal of Systems and Software, 86*(2), 538-550.

Royce, W. W. (1987). *Managing the deelopment of large software systems: concepts and techniques*. Paper presented at the Proceedings of the 9th international conference on Software Engineering, Monterey, California, USA.

Sehra, S. K., Kaur, J., Brar, Y. S., & Kaur, N. (2014). *Analysis of Data Mining Techniques for Software Effort Estimation.* Paper presented at the Information Technology: New Generations (ITNG), 2014 11th International Conference on.

Sharma, A., & Kushwaha, D. S. (2012). Estimation of Software Development Effort from Requirements Based Complexity. *Procedia Technology, 4*, 716-722.

Shepperd, M., Schofield, C., & Kitchenham, B. (1996). *Effort estimation using analogy.* Paper presented at the Proceedings of the 18th international conference on Software engineering.

Smith, M. F. (1991). *Software Prototyping: Adoption, Practice and Management*: McGraw-Hill, Inc.

Taya, S., & Gupta, S. (2011). Comparative analysis of software development life cycle models. *ijcst, 2*(4).

Zheng, Y., Wang, B., Zheng, Y., & Shi, L. (2009). *Estimation of software projects effort based on function point.* Paper presented at the Computer Science & Education, 2009. ICCSE'09. 4th International Conference on.

APPENDIX A

GANTT CHART