DESIGN OF A PROPOSED PRAGMATIC SOFTWARE DEVELOPMENT PROCESS MODEL

OMORODION, FOFO MARTHA

(2006/24468EE)

A Thesis Submitted to the Department of Electrical and Computer Engineering, Federal University of Technology, Minna

NOVEMBER, 2011.

DEDICATION

This project report is dedicated to Almighty God who lovingly gave us life, protects us daily and ensured that this project was successfully completed with inspirations from Him.

It is also dedicated to my parents (Oiseoruemi and Oluwafunminike) and siblings (Ijamenero, Omoniyi, Eki, Idieli, Iyabo and Aimanoshi).

DECLARATION

I, Omorodion, Fofo Martha, declare that this work was done by me and has never been presented elsewhere for the award of a degree. I also hereby relinquish the copyright to the Federal University of Technology, Minna.

OMORODION, FOFO MARTHA
(Name of Student)

[illegible] 10/11/2011
(Signature and date)

CERTIFICATION

I hereby certify that this project, "Design of a Proposed Pragmatic Software Development Process Model, was carried out by Omorodion, Fofo Martha with matriculation number 2006/24468EE and meet the standard deemed acceptable by the department of Electrical and Electronics Engineering, School of Engineering and Engineering Technology, Federal University of Technology, Minna.

Prof. Sergey Misra
(Name of Supervisor)

[illegible] 10/11/11
(Signature and Date)

ENGR A. G. RAJI
(Head of Department)

[illegible] (March 15, 2012)
(Signature and Date)

Egr Dr. G. I. [illegible]
(External Examiner)

[illegible] 6/3/2012
(Signature and Date)

ACKNOWLEDGEMENT

Firstly, all gratitude goes to God Almighty for His provision, protection, support and His inspiration while I was conducting this project and writing the report.

My sincere gratitude goes to my parents, Mr. and Mrs. Samuel Omorodion for their undying love, sacrifice and support for me; my siblings – Ijamenero, Omoniyi, Eki, Idieli, Iyabo and Aimanoshi - also are deeply appreciated for their unconditional love and support for me.

My sincere gratitude also goes to my Supervisor, Professor Sanjay Misra for his selfless assistance in seeing to the success of this project. I will not fail to mention Mr. and Mrs. Daniel Ogundipe, Mr. and Mrs. Afolabi Ogunmola and Prof. and Mrs. Onimisi Abdullahi and family for their selfless assistance. My bosom friend, Yemi Ojo, who has always been a shoulder to lean on in the University, Ojeola Olatubosun and my classmates and friends, will also be acknowledged.

ABSTRACT

The rapid growth in technology and the dynamism it presents in our society today poses a lot of problems for the Software Engineering Practitioners. The result is a series of Software Developments Process Methods which can be used to combat or meet up with the problems it creates for them. This is the dynamism inherent in man - to adapt to change and improve on ourselves and our existing systems. On this basis lay the need to develop the model designed in this project to meet up with variations that exist as a result of technological advancement. The designed model was arrived at by identifying the weaknesses and strengths of various existing models, the practices that software developers apply to enable them meet the ever-changing needs of their clients and the results obtained were comparatively analyzed and possible methods that can be applied to overcome some of these problems based on practices carried out by a developer were pieced together. The results show that although efforts were made to overcome some of the existing problems, new problems that need to be overcome were created.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

Introduction

Software can be referred to as programmes and applications required for proper operation of computers and computerised systems [33]; sometimes they are referred to as the Real Time Operating Systems of hardware. These systems may be produced for either individuals (custom made) or for the general public (generic) [14]. Either way, these programmes are developed based on requirements specifications stated by the clients who ordered for the software product. As a result of the processes and difficulties involved in achieving these goals, a lot of studies have been carried out so as to find out the best ways in which requirements can be met. As such, various development models and techniques seem to have mutated from the three fundamental models viz; The Classic Software Lifecycle Model which is sequential in nature, The Evolutionary Process Model which is iterative and the Rapid Application Development Model which is incremental in nature [13].

Software engineering is an engineering discipline that is concerned with all aspects of software production. Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available [14]. [27] stated in web post that if you could settle the problem of change and really could get an accurate and stable set of requirements you're probably still doomed. In today's economy the fundamental business forces are changing the value of software features too rapidly. What might be a good set of requirements now, is not a good set in six months time. Even if the customers can fix their requirements, the business world isn't going to stop for them. And many changes in the business world are completely unpredictable: and if you cannot get stable requirements you cannot get a predictable plan.

Consequently, a research conducted on various development models and techniques revealed that there is an average of about 40 models already available and yet a consensus has not been reached to agree on one perfect model. Although it could be quite impossible to arrive at a one perfect model, but from the research conducted, software developers seem to have come to terms with the Agile Software Development Methods which is a web of several development techniques applying similar methods to software development such as developing a team, lack of bureaucracy and debugging as the code is being developed so as to minimise time wasted in software checks [20].

This project was carried out for the purpose of designing a development model that will put into consideration most of the benefits and limitations of all the models that were found and make their weak points its strong points while also taking advantage of the benefits they present.

1.2Aim and Objectives

1.2.1 Aim

The aim of this work is to design a practical oriented software development process model putting into consideration the pitfalls of other models and taking steps to avoid them and taking advantage of their benefits.

1.1.2 Objectives

The objectives of this project are:

1. To design a software development process model based on practical activities carried out by developers

2. To carry out a comparative analysis between the benefits of the model and those of past models
3. To design and evaluate a functional model that can and possibly will be implemented for the benefit of all humanity.

1.2 Case Study

The generic software development process models, several other models that evolved from them and the Agile methods which are the modern methods employed.

1.3 Methodology

This project was carried out by conducting a research on existing software development process models, studying the processes involved, conducting a comparative analysis based on the generic models (sequential, iterative and incremental) and then obtaining a list of the benefits and limitations of each model; the suggestions made by a developer about some of the methods he practices in the process of developing software for his clients which are obviously practicable [9].

1.4 Scope of the Study

It would have been well appreciated if the process model proposed in this study were put to practical use so as to see the practicability of the model in the environment but based on the fact that it was concocted from ideas suggested by a developer as stated earlier; the obvious implication is that it is practicable. As such, the scope of this project does not go beyond conducting a research on the various software process development models that have been designed in the past, proposed and used by several developers. As many models, techniques and methodologies as possible were collected, their practices, weaknesses and strengths were observed, analyzed and put into consideration.

The complaints, praises and suggestions of several developers were also considered and used during the process of the design of the Pragmatic Development Model. At the end of the day, the strengths of the model are clearly compared in tables against the weaknesses of some of the popular and most utilized models in the field.

1.5 Sources of Materials

Most of the materials consulted to conduct the research were obtained from the internet and a few from text books in the Department's Library and some others from my supervisor. I also consulted some lecturers and fellow students to see what their suggestions will be and if they can be put to use to boost the study.

1.6 Constraints

The major factor that militated against the success of this project was difficulty in access to the internet. Using a modem turned out to be almost next to nothing because of poor signals provided by the GSM (Global System for Mobile Communications) networks within the premises. The wireless system provided by the Institution also turned out to be almost of no help as their signals don't extend to the hostel facility provided and the part of the environment where signals can be found are far from the hostels and signals are quite poor and navigation slow during the day. Thus, it is better to browse at midnight (not just at night, emphasis is on midnight) because of the reduced amount of users but it's quite dangerous for a helpless female student to walk down the road at such an outrageous hour. Another constraint is the inability to put the model to practice because of time and lack of developers in the environment to put it to use.

CHAPTER TWO

Theoretical Background

Software Engineering can easily be assumed to deal with only software products such as web design, writing programmes and the like, but on taking a closer look, one finds that it not only deals with software products but it deals mainly with the problems encountered in the process of developing these software. This misconception among students about the field attracts a lot of them into registering the course only to attend classes and meet the stark opposite of what they expect.

A lot of definitions of Software Engineering have been developed by several authors but a definition proposed by Fritz Bauer [31] states that "Software engineering is the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines". [14] in his book, *Software Engineering,* defined it as an engineering discipline that is concerned with all aspects of software production. The IEEE [15], on developing a comprehensive definition states that Software Engineering (1) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1). Software engineering as an engineering approach for software development can alternatively be viewed as a systematic collection of past experience arranged in the form of methodologies and guidelines. A small program can be written not including the use of software engineering principles; on the other hand, if one wants to develop a huge software product, then software engineering principles are crucial to achieve an excellent quality software cost effectively [8].

2.1 Background of the Study

2.1.1 Historical Background of Software Development Process Modelling

Discussing software development process modelling requires that one introduce the idea of software processes. A software process can simply be referred to as the path taken to develop functional software. [33], on defining software process as a road map stated that "When you build a product or system, it's important to go through a series of predictable steps—a road map that helps you create a timely, high-quality result [8].

This set of activities includes Specification, Design, Validation and Evolution and they gave rise to the software process modelling which is an abstract representation of a process and it presents a description of a process from some particular perspective [14]. A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes or for building empirically grounded prescriptive models [2]. [48], further stated that a prescriptive model prescribes how a new software system should be developed. Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed, and in what order.

Given a software project, a software process model tells us about the activities that need to be taken up for completing the project. The concept of software process modelling is thus a central idea in Software Engineering, which is a discipline, involved in the study and dissemination of sound software development and management practices [22]. Owing to these necessities and practices and the desire to meet up with dynamism in demand and human nature, several development models have been arrived at and [36] asserted that software development process dates back to as far as the 1950s and that the software industry has evolved through 4 eras; 50‘s –60‘s, mid 60‘s –late 70‘s, mid 70‘s-mid 80‘s, and mid 80‘s-present. Each period has its personal distinctive characteristics, but with time, software has increased in dimension and complexity. He further stated that all the eras share common problems which include hardware advances outpacing the capability to build software for this hardware; the ability to build in pace with the demands; increasing dependency on software’s struggle to build reliable and high quality software, poor design and insufficient resources.

There are two types of models: those that go through each of the activities sequentially only once and those that go through each of the phases many times. Those that execute the stages in a life cycle iteratively produce different artifacts (such as prototypes as well as documents) at the end of each iterate. The major benefit is that even if the whole project is cancelled there are pieces left that could be used in further projects or developments [49]. Various models have therefore evolved over the years and software engineers have concluded that process models can be classified under three generic models [14] which are the Waterfall Model [49] - which is sequential in nature and has separate and distinct phases of specification and development; Evolutionary (iterative) Process Model – which is iterative and its specification, development and validation phases are interleaved; and the Rapid Application Development Model – which exhibits incremental characteristics and systems are assembled from existing components. All other models developed thereafter exhibit the characteristics of one or a combination of two or the three models; thus it’s either it falls under a model explicitly or overlap between two or three of them simultaneously [14].

[13], also noted in their technical report, *Component-based Development Process and Component Lifecycle,* that “different models have been proposed and exploited in

software engineering, and different models have exhibited their (in)abilities to efficiently govern all activities required for a successful development and use of products. Well known examples of sequential models are waterfall model, or V-model, and of evolutionary models, iterative and incremental development and the spiral model.

The software process models history begins with the introduction of a model called "Build and Fix Model". The model has only two steps:

1. Write the Code
2. Fix problems in the code.

Thus, the main theme of the model was to write some code first and then think about different phases of development [4] [12].

A model is an excellent example that deserves to be imitated while a methodology is a convention that a group agrees to and a method is a systematic procedure, similar to technique [1]. As such, we should try to separate the two schools of thought as the three terminologies are used often.

2.2 Sequential Models

2.2.1 The Waterfall Model

This is one of the earliest and the most common approach employed in the process of developing software. It exhibits sequential characteristics and discourages iteration; it was developed in the 1970 [49] on noticing that the process of developing software was an activity that was different from the development of hardware products which required well defined requirements that do not change till after the product has been released into the market [48]. Plate 2.1 shows a view of the waterfall chart showing the various stages involved.

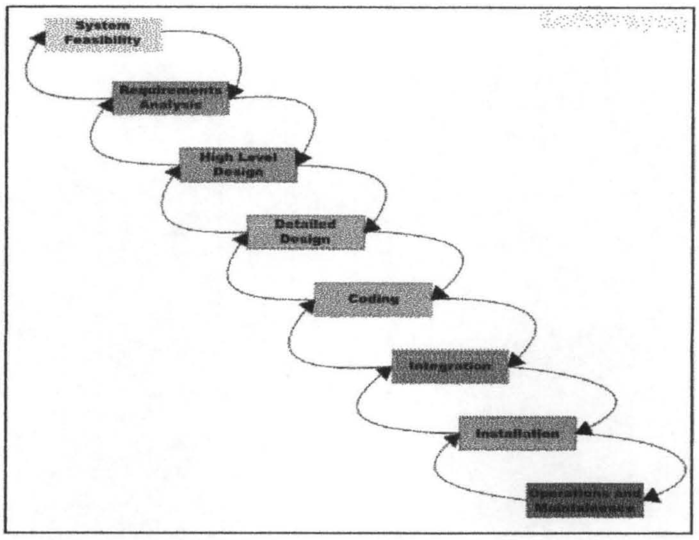

Plate 2.1 A Waterfall Model [49] [22]

This model, sometimes called the Classic Software Lifecycle Model has its stages broken down into as many stages as twelve in some reports [38] but briefly, the stages shown in Plate 2.2.1 apply [44] [48] [34] [14]. The Waterfall Model offers the advantage of simplicity and ease of use, simplified manageability, useful for small projects; while the greatest disadvantage of the waterfall model is that until the final stage of the development cycle is complete, a working model of the software does not lie in the hands of the client. Thus, he is hardly in a position to mention if what has been designed is exactly what he had asked for [46].

2.2.2 The V-Model

Plate 2.2 illustrates the V-model and it is quite similar to the Waterfall Model. In describing the V-model [12] stated that it is assumed to be the extension of Waterfall Model but exhibits its difference from the waterfall such that it doesn't move in a linear way instead its process steps bend upwards after the coding phase to form V-shape showing that each phase has an associated testing phase. Like the activities of the waterfall model, each activity has to be completed before moving on to the next activity. [49].

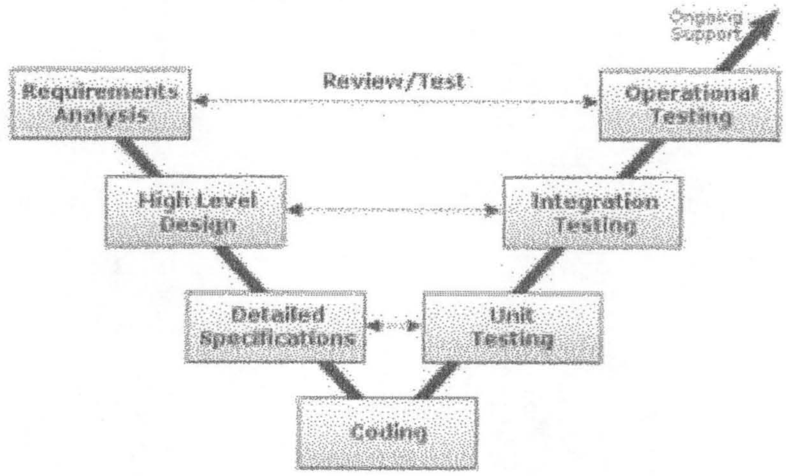

Plate 2.2 A V-Model [40]

[40], a software development expert, on describing the V-models indicated that "another idea evolved which was the traceability down the left side of the V. This means that the requirements have to be traced into the design of the system, thus verifying that they are implemented completely and correctly. The earlier versions of V-models used the first option. For later versions a series of subsequent V-cycles was defined, as shown in Plate 2.3 below".

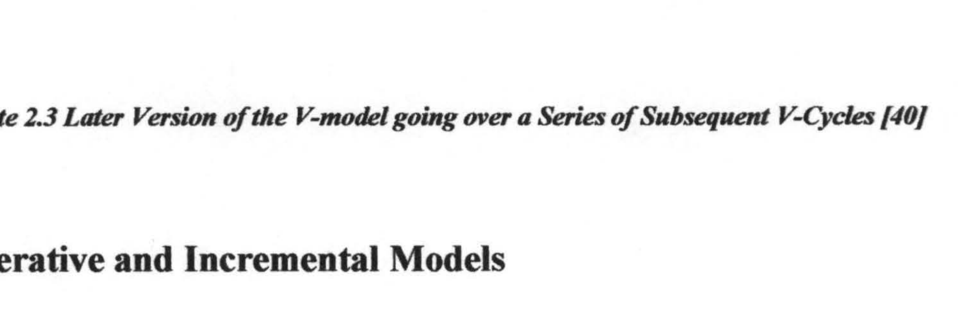

Plate 2.3 Later Version of the V-model going over a Series of Subsequent V-Cycles [40]

2.3 Iterative and Incremental Models

These are the evolutionary models and are characterised in such a way that more complete versions of software are developed. When discussing iterative and incremental development, the terms *iteration* and *increment* are often used freely and interchangeably. They are not, however, synonyms. *Iteration* refers to the cyclic nature of a process in which activities are repeated in a structured manner. And *increment* refers to the quantifiable outcome of each iterate [21].

Iterations can offer a development process two key things: iterative refinement, where the process improves on what already exists and is being done, and incremental development, where the process results in progress against project objectives. Additionally, the term *increment* has the obvious implication that there should be more of something at the end of each iterate than there was at the start. Without a clear notion of an increment, iterations are likely to just go in circles [21]. The generic form of the evolutionary model is shown in Plate 2.4 below.

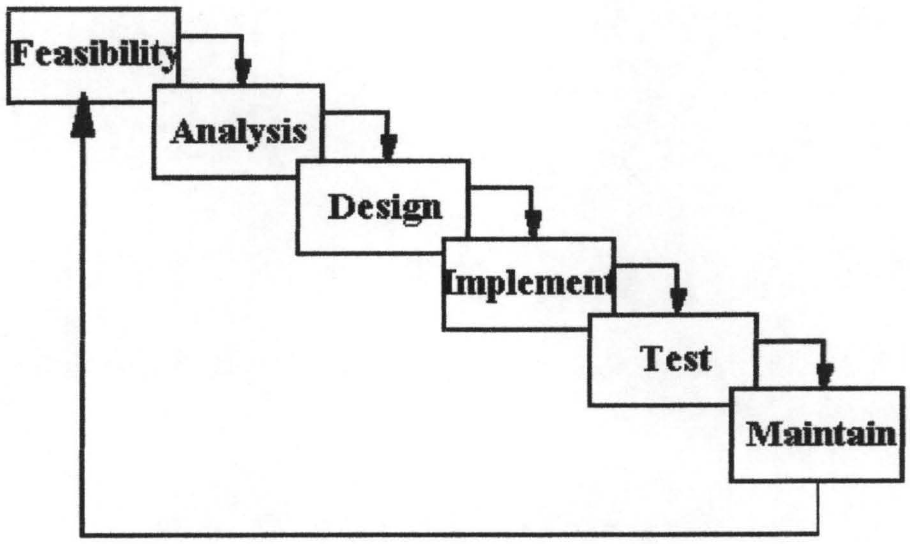

Plate 2.4 A Generic Evolutionary Process Model [14]

2.3.1 The Iterative Waterfall Model

The numerous problems associated with the waterfall model called for some necessary variations in the model and as such this model was designed to accommodate dynamism and changes in the development process. The iterative enhancement life cycle model counters some of the limitations of the waterfall model and tries to combine the benefits of both prototyping and the waterfall model. The basic idea is that the software should be developed in increments, where each increment adds some functional capability to the system until the full system is implemented. At each step extensions and design modifications can be made. An advantage of this approach is that it can result in better testing, since testing each increment is likely to be easier than testing the entire system like in the waterfall model. [32]. Plate 2.5 illustrates the Iterative Waterfall Model.

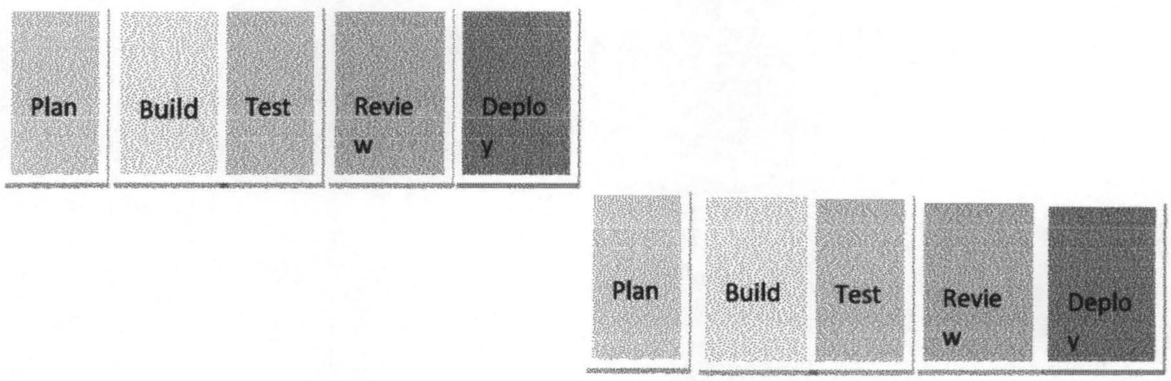

Plate 2.5 An Iterative Waterfall Development Model [42]

2.3.2 The Spiral Model

This is an evolutionary software process model that was proposed by [3] in 1987. It is divided into about six task regions and it couples the iterative methods with some methods in the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced [33]. Plate 2.6 is a view of a Spiral Model showing its different framework activities.

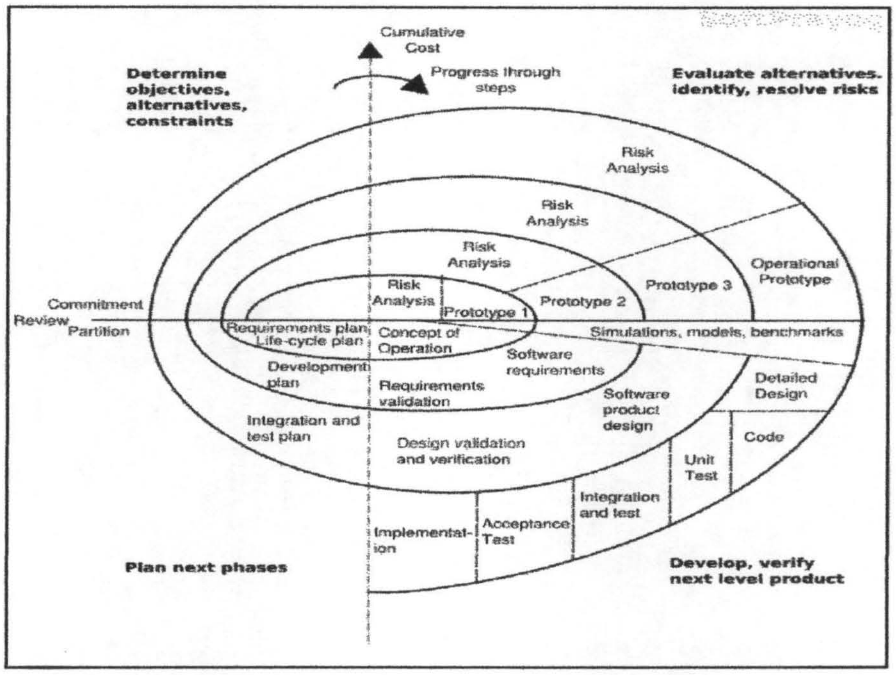

Plate 2.6 A Spiral Model [3]

2.4 The Rapid Application Development (RAD) Model

The last generic model to be described is the RAD model. It is a model that combines the benefits of iterative and incremental models in the process of developing software. It is useful for development when the requirements are not well understood so that prototypes are developed to help the clients have a physical view of what they want. It is a merger of various structured techniques, especially the data driven Information Engineering with prototyping techniques to accelerate software systems development [10]. Sometimes the RAD model is referred to as Component Based Software Engineering (CBSE) [14] used to assemble systems out of existing, independently developed components. Component Based Software Engineering entails more than mere reuse of components, though. It also aims to increase the flexibility of systems through improved modularization. It achieved its name from the "80/20" rule, which states that in development projects, 80% of the work is finished before 20% of the completion time. The remaining 20% of the work takes up the remaining 80% of the time. For example, suppose you're on a development project that takes five months. In the first month, 80% of the work will be done to move the project forward. You'll then spend the remaining four months getting the other 20% of the work done [45]. Plate 2.7 illustrates the RAD model as a system reusability model.

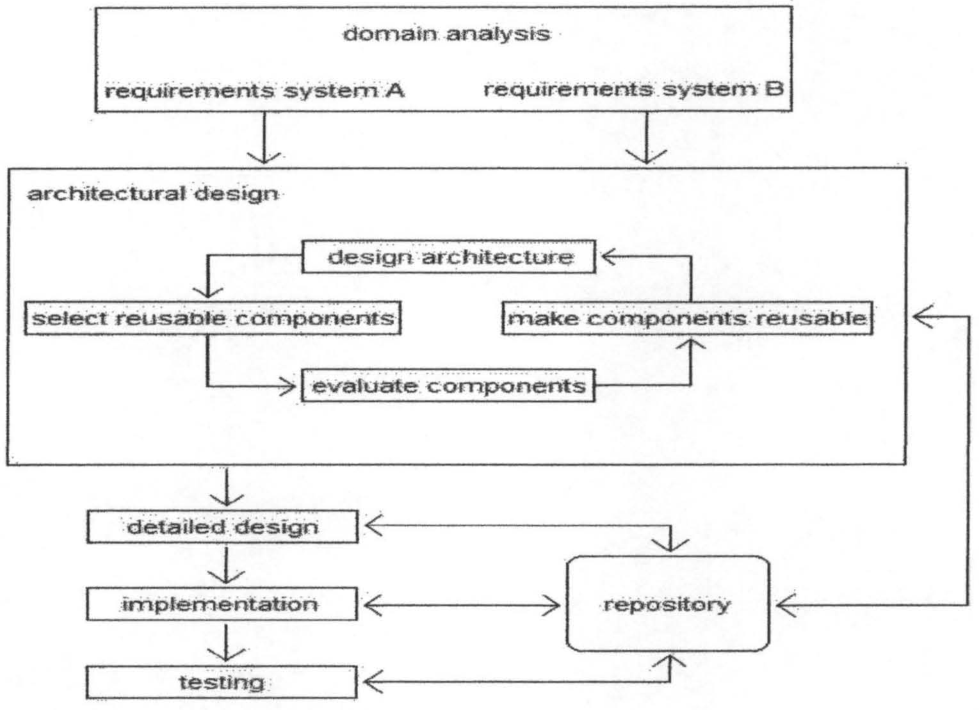

Plate 2.7 A System Reuse Model [37]

2.5 Other Models in Brief

- **The Rapid Prototyping Model (RPM)** [47] – It falls under the evolutionary process. This model is useful when users are not sure about their requirements; a prototype model is then developed to verify the specification. It has the advantage of meeting user requirements and it useful when requirements change rapidly. It has the disadvantages of being too iterative, not being cost effective and is a vague idea.

- **V-Model XT Process Framework** ⁻ The VM XT, (XT - eXtreme Tailoring) is an improvement on the V-Model (97). It is a recent model announced as the standard for public-sector projects in Germany and its framework has a flexible customizing ability focused on meeting the requirements specifications [38]. It cannot be used for all software application and is family oriented rather than individual.
- **The WinWin Spiral Model (WSM)** – It deals mainly with how to handle negotiations between customer and developer so that at the end of the day, the customer wins by getting a product that satisfies most of his needs while the developer wins by being able to work out achievable budgets and time limits [5]. It is cost effective, customer's needs and time limits are met; but the two parties may not reach and agreement and it involves compromise.
- **The Concurrent Process Model (CPM)** – It involves series of activities that take place existing simultaneously but still remaining in different developmental stages [33]. It is applicable to most software development and it shows the current state of a project accurately. It also faces the problem of being too iterative and events in an activity triggers transitions in other activities.
- **The Formal Systems Development Model (FSDM)** – Its development process is based on formal mathematical transformation of system models to executable programs. Though it is similar to the waterfall model, but it has clearly defined phase boundaries; sometimes it is called the Formal Methods Model (FMM) [33].

- **Component-Based Development Model (CBDM)** – This model appears to be similar to the CBSE but they differ in the sense that while CBSE lays more emphasis on reuse of components, CBDM focuses on development of systems using components [13].

The following are development methodologies named as methodologies, techniques, processes or approaches.

- **Family-Oriented Software Development Process (FSDP) -** This process focuses on clarifying the properties of family related systems and not the individual systems. It is useful mainly for aero-engine software. It follows the traditional lifecycle process [18].
- **Incremental Delivery (IDR)** [14] –It falls under the evolutionary process. Also called the Incremental Development and Release. It combines the advantages of the iterative and waterfall approach.
- **Joint Application Development** – This is a technique that involves the use of a group of software developers, testers, and possible end-users to interact to generate requirements and prototypes of the software being produced [48].
- **Object-Oriented Design (OOD) -** This approach to software development was proposed in late 1960s. It requires that object-oriented techniques be used during the analysis, and implementation of the system. It requires the analyst to determine the systems' objects, their behaviour with time, response to events and relationships with other objects and how the system manipulates the objects.
- **The Fourth Generation Techniques (4GT)** [33] –It is a modern day technique that involves the use of automated code generation. This technique deals with the

ability to use specialized language forms to identify software that explains the problems that need to be solved in ways that customers will understand.

2.6 Agile Development Methodologies

The term Agile stands for 'moving quickly'. These methodologies are quite recent and a lot of methodologies are involved in this group. They were developed as a result of the bureaucracy involved in the earlier models. They are called "Agile" as a result of the flexibility involved when using them to develop a process model. These software development methodologies are based on iterative and incremental models of software development. The requirements and the solutions are a product of collaboration between self organizing and cross functional teams. It is a lightweight software development model, which was developed in the 1990s [27].

[24] asserted that in February 2001, several software engineering consultants joined forces and began to classify a number of similar change-sensitive methodologies as *agile* (a term with a decade of use in flexible manufacturing practices [25] which began to be used for software development in the late 1990's [26]). The term promoted the professed ability for rapid and flexible response to change of the methodologies. The consultants formed the Agile Alliance and wrote The Manifesto for Agile Software Development and the Principles behind the Agile Manifesto [20, 28]. The methodologies originally embraced by the Agile Alliance were Adaptive Software Development (ASD) [16], Crystal [1], Dynamic Systems Development Method (DSDM) [17], Extreme Programming (XP) [19], Feature Driven Development (FDD) [30, 39] and Scrum [23].

Most of the ideas were not new; indeed many people believed that much successful software had been built that way for a long time. There was, however, a view that these ideas had been stifled and not been treated seriously enough, particularly by people interested in software process [27]. Plates 2.8 and 2.9 show the pictorial views of the Scrum and Lean Development Methodologies.

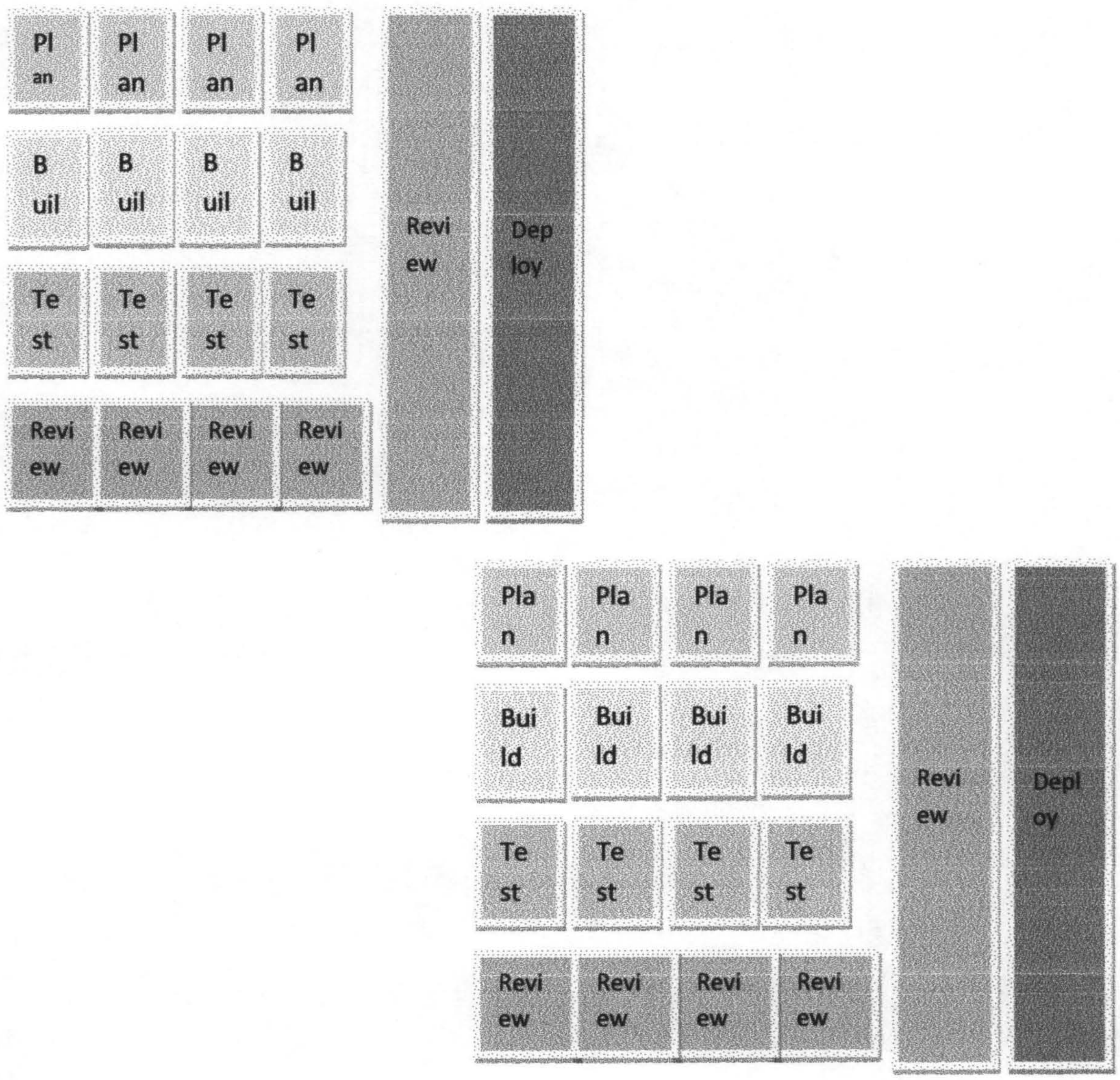

Plate 2.8 Scrum Development Methodology [42]

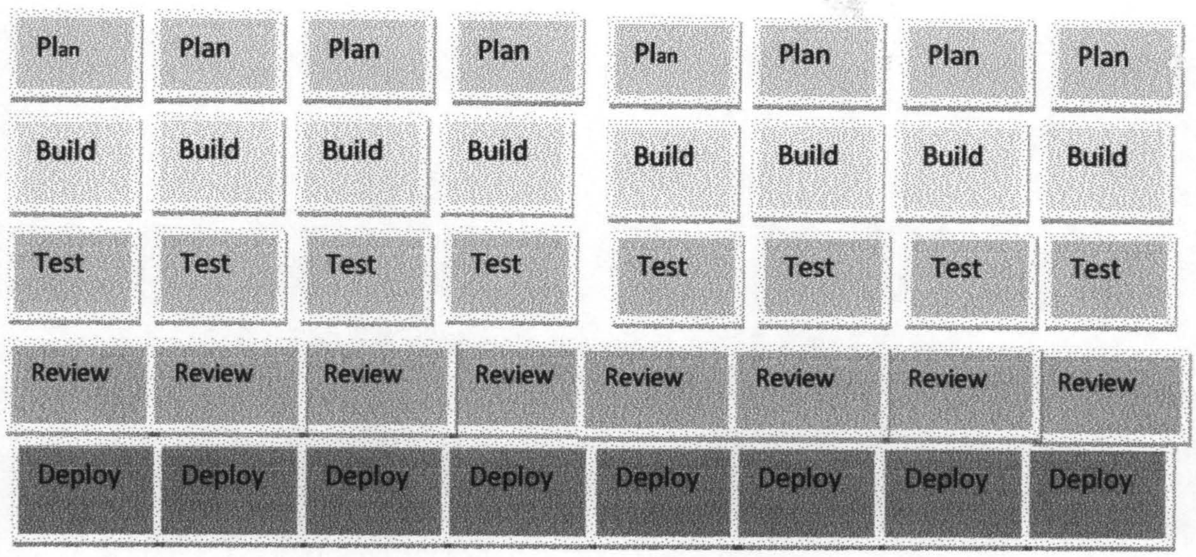

Plate 2.9 Lean Development Methodology [42]

2.6.1 Manifesto for Agile Software Development [20]

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more. The pioneers of the Agile manifesto and principles include Kent Bent, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Scwaber, Jeff Sutherland and Dave Thomas.

2.6.2 Other Agile Methodologies in Brief

- **Adaptive Project Framework (APF)** [35] – this method allows for constant adjustment of the project so as to ensure the delivery of maximum business value by adjusting scope at each iterate.
- **The Rational Unified Process Model (RUP)** – sometimes it is referred to as the unified process. It emphasizes on the development and maintenance of existing models. It also centres on the use of the Unified Modelling Language (UML).
- **Scrum** [40]–This method involves an agile approach to software development. It is a framework and not a full process or methodology. Thus it does not provide a complete detail of descriptions of how the project is done. It has a team called the Scrum team which makes much of the decisions used during the project. Team work is emphasized. It uses short iteration and frequent reviews.
- **Extreme Programming (XP)** [19] - It is actually a deliberate and disciplined approach to Agile software development. One of the first approaches to gain main-stream success, Extreme was found to be most successful at smaller companies especially in the dot-com boom. It involves team work [6].
- **Lean Software Development (LSD)** [29]–The National Institute of standards and Technology Manufacturing Extensions Partnership's Lean Network, describes lean as "A systematic approach to identifying and eliminating waste through continuous improvement, flowing the product at the pull of the customer in pursuit of perfection." It is not a project management technology but a set of principles applicable to any method for improving project planning and execution.
- **Crystal Methodologies (CM)** [1] – It employs an evolutionary process and involves the use of team work; it is also an agile process that consists of about five

different methods viz; crystal clear, crystal orange, crystal red, crystal yellow and crystal maroon. The method is created with different requirements or problems posed by the customer. It is useful for customized products.

- **The Dynamic Systems Development Method (DSDM)** [17] – It is an agile software technique that is base on the RAD techniques. Here time is fixed and functionality variable as opposed to traditional development methodologies where functionality is fixed, and time and resources variable. It is useful when the time required for delivery is short, thus it uses incremental prototyping in a controlled environment with each increment delivering enough functionality to move to the next increment.
- **Feature Driven Development (FDD)** [30] - It is very iterative and collaborative and it is an agile development method. It is composed of five processes viz; develop an Overall Model, Build a Features List, Plan by Feature, Design by Feature, Build by Feature. FDD has eight practices: domain object modelling; developing by feature; individual class ownership; feature teams; inspections; regular builds; configuration management; reporting/visibility of results.
- **Adaptive Software Development (ASD)** [16] – It is an agile method which involves teamwork and involves the use of JAD approach for requirements gathering. It is mission driven, component-based, iterative, risk driven and change tolerant. It evolved from RAD.
- **Test Driven Development (TDD)** – It is an agile method. It produces tests that specify and validate what the code does before the final the production code is designed [19].

2.7 Other Development Methods Not Described

There are numerous other models that cannot be discussed as a result of space and time; they include Structured programming [36] developed in 1969, Structured Systems Analysis and Design Methodology (SSADM) [36] in 1980, Object-oriented programming (OOP) [36] developed since the early 1960s, and it was the dominant programming methodology during the mid-1990s. The Team Software Process developed by Watts Humphrey at the Software Engineering Institute (SEI), The Personal Software Process was also developed by [46] in an attempt to apply the underlying principles of the Software Engineering Institute's (SEI) Capability Maturity Model [46] to the software development practices of a developer. Integrated Methodology (QAIassist-IM) since 2007 [46], Systems Development Life Cycle (SDLC) [43], Clean Your Room (CYR) [7] comes as a response to the Cleanroom Software Engineering process, its focus is upon implementing cool, new features and not on tests, documentation, or bug-fixes. Cliff, developed as a response to the Waterfall model, leaps suddenly from the starting point (known as the *Hairbrained Idea*) to the end (known as the *Product*, but informally Cliff teams refer to it as the *Corpse*) [7]. Testosterone-Driven Development, [7] like Test-driven Development, focuses on testing first. But extremely aggressive, requiring that entire test suites be produced; Conference Driven Development (CDD), [7] Fragile Programming [7], an important element of the *Delicate* software pattern related to Agile programming; Conference Drive Development (CDD) [14], Prince2 [45] and Cleanroom Software Engineering [7].

Chapter Three

Materials and Methods

3.1 Materials

This model was designed based on some suggestions made by a software developer on techniques he applies during the development of a system for his clients where he indicated that he uses Agile methods while meeting their needs for predictability; the following are methods he claims he practices [9]:

- Make sure it is a project which can use Agile methodology
- Propose a fixed burn-rate
- Propose a team that is right-sized to the skills and scale of the project
- Propose a fixed sprint size (2 or 3 weeks is typical) for scrum and 6-8weeks releases composed of 3-4 2 week iterations
- Propose a contract where the client can change anything they want for the next sprint or iteration and where they can accept/reject each story as it completes. In exchange the client can be sure there is a single voice (not a committee) who is available to (or embedded with) the team
- Propose that the client can cancel at the end of any sprint/release with no penalty and will be left with shippable software that has been accepted

He claims that he has uses this approach in some of his engagements as well stating that it is a good compromise between time-and-materials and fixed-price models. The client can quickly receive value and see the cost vs. value model emerge quickly from the results of each sprint/release.

From initiative, I have also tried to apply common sense and knowledge to its development. As is generally known, the human nervous system, as a whole, is sometimes considered as a computer system. Necessary features for development are made available at birth even when they are not really useful to the individual at that time and may never be useful to it at maturity. But considering the perfection and intelligence involved in the development of the nervous system, it can be seen that at the end of the day, from hindsight, nothing seems to be lacking for the individual provided he develops himself positively and the human software was designed in such a way that whatever paths an individual chooses to take, he can be totally different from (and similar to) the next individual. The problem of a clash of wills never comes to play or the possibility that the software cannot be adjusted to go in any direction. After all software is supposed to be *soft;* so not just are requirements changeable, they ought to be changeable. It takes a lot of energy to get customers of software to fix requirements. It's even worse if they've ever dabbled in software development themselves, because then they "know" that software is easy to change [27].

That is to say, if I choose to build cars today, am free to do so. Nothing binds me to being a farmer or something else I have been pre-programmed to do. In short, the system is very flexible; and funny enough, it is so flexible that it has the ability to change at any time even at old age without getting corrupted. Thus, this model was designed to exhibit such flexibility and dynamism that can be observed in humans. From the specification stage, it creates room for variation, improvement and increments. It has the ability to encourage development of the software in whatever direction – based on specifications made – that users want it to. It also involves team work both within the system and without. Though, it is not as perfect as the human nervous system, but it is a model that seems to want to put as much as is possible into consideration just as the nervous system does.

A point to note though is that one model or methodology cannot cut across different technologies. Some are useful for large systems, others for small systems and still others for systems for long term use. The different technologies, cultures, and project priorities call for different ways of working [1].

Developers must be able to make *all* technical decisions. XP gets to the heart of this where in its planning process it states that only developers may make estimates on how much time it will take to do some work. But the technical people cannot do the whole process themselves. They need guidance on the business needs. This leads to another important aspect of adaptive processes: they need very close contact with business expertise. This goes beyond most projects involvement of the business role. Agile teams cannot exist with occasional communication. They need continuous access to business expertise [27].

3.1.1 Comparative Analyses of the Generic Models

A comparative analysis of the three generic models is shown in Table 3.1.1. Only these three models have been analyzed because as stated earlier, all the other models and methodologies are variations of these three.

Table 3.1 Comparative Analysis of the Generic Models

Waterfall Model	Evolutionary Process Model	RAD
It lacks flexibility	It is very flexible	It is relatively flexible
It is useful when long life complex systems are being built	Useful mainly when small systems are being developed	Useful mainly for commercial systems
Delivery is relatively fast	Delivery is slow	It is characterized by fast delivery
Very low risks	High risks	Low risks
Higher chances that user needs will not be met.	Needs are definitely met else the product will not be released	May not meet needs because of requirement compromise and time limit.
Badly structured systems	Poorly structured systems	Relatively well structured systems
It is simple and stable	Fairly clear but needs confirmation	Fairly clear, stable and large
It is static	Its is dynamic	It is fairly dynamic
It is not cost effective because value for money is not achieved	It is not cost effective	It is relative cost effective

3.2 Methods

Plate 3.2 shows an illustration of the Pragmatic Process model and the following sections give a description of each stage shown in the model. Take note that following the arrows, one can move from any one stage to which ever one he needs to go to depending on the problems that arise.

3.2.1 Requirements and Specification

Specifications are gathered and negotiations are made on what they hope to achieve. The team (a group of experts useful for the development of the software) meets with the clients or end users to discuss whatever features cannot be considered and also they can make suggestions too on certain features the clients did not consider. Executing an adaptive process is not easy. In particular it requires a very effective team of developers. The team needs to be effective both in the quality of the individuals and in the way they blend together. There's also an interesting synergy: not just does adaptivity require a strong team, most good developers prefer an adaptive process [27]. [11] in his book also stated that the quality of the people on a project, and their organization and management, are more important factors in success than are the tools they use or the technical approaches they take

They should also be informed that at any time they can come up with new specifications that will help to improve the features of the software. The clients should also fix a time frame for which the software should be ready to be delivered to the end users. To the clients' and team's convenience, they can fix meeting days and times within a week, preferably twice a week. They should meet the first time to collect requirements and the second time to not only collect more requirements but also show the clients the designs and if possible developments they have been able to come up with. This way in making more requirements, their suggestions will be more in depth with understanding than the first time. All decisions made should be documented for reference purposes. A fruitful way to think about software development is to consider it as a cooperative game of communication and invention [1].

One primary goal of this model is to give room for flexibility not only to the development process but also to the developers themselves. In applying the methods suggested by this model, if somewhere along the line the developers discover that a path they are taking will probably lead to a dead end, they can come up with ideas of their own and modify the model to their own benefit and to avoid wastage. This is so that it can be useful for large systems as well as moderate and small systems. Don't hesitate to document all the necessary processes no matter how many they are or how small and irrelevant they may seem to be; documentation is very vital in any process as all the ideas cannot be stuffed in the brain and remembered [14]. This is an area where the Waterfall Model has an edge over the Agile Methods. Don't worry about them littering the place; put them in soft copies and you can easily delete the ones that are not needed.

3.2.2 Planning

The team, after collecting requirements and specifications, should hold a meeting and decide on which methodology they want to use for the development process and they should pay attention to the size of the system they are about to develop (small or large systems). Time should not be wasted here; the meeting can last for at most an hour, but they should make sure that the opinion of every individual is considered before arriving at a final conclusion. If they meet problems somewhere along the line then they should come back to this stage and reconsider their position once again.

3.2.3 Design and development

As much as possible, the development of the software should commence as soon as a set of requirements and specifications have been made. Since they will be allowed to change specifications at any time, little or no time should be wasted on waiting for them to arrive at a consensus on what they might not ask for at the end of the day. This idea is coined from the Concurrent Process Model where rather than confining software engineering activities to a sequence of events; it defines a network of activities. Each activity on the network exists simultaneously with other activities. Events generated within a given activity or at some other place in the activity network trigger transitions among the states of an activity [33]. Research has shown that shorter iterations have lower complexity and risk, better feedback, and higher productivity and success rates [24]. At this stage, the source code should be developed inculcating features that will give room for development, changes, total removal from the system or increments. After this point, whatever changes they come up with will only be fixed in as additional features to the already designed features for such development. A maximum of four iterations can be used, if possible, less. After each design, the client should be consulted for possible changes, improvements, rejections and so on. Often the most valuable features aren't at all obvious until customers have had a chance to play with the software. Agile methods seek to take advantage of this, encouraging business people to learn about their needs as the system gets built, and to build the system in such a way that changes can be incorporated quickly [27].

3.2.4 Verification and Validation

This stage can be divided into three sub-stages viz; unit testing, integration and testing and review. After each development, tests should be carried out on each unit developed to ensure that flaws, errors and omissions are corrected. This process is quite a difficult one as errors cannot easily be noticed especially by the programmer. Suggestions are either two programmers develop their programmes independently and exchange their work to check for errors or interns, being eager to learn, excited about their new job and probably, being youths, are agile and can easily notice any errors, should be allowed to go through their work as they go along so that such errors can be corrected.

As soon as an adequate number of units that can be integrated have been built, they should be integrated and tested to verify that they can function together; making sure also that the emergent properties suit the needs of the clients. After three iterations, the clients should be informed that there is the need for a final validation of the software so that it can finally be released into the market as there is still room for one more iterate. Informing them about moving on to the verification and validation stage will help to ensure that no more changes are made after the fourth iteration. At this stage final tests should be carried out with the clients present agreeing that all needs have been met and are satisfied, else, if there is need for more variations or additions then they should be carried out and then reviewed again. From practice data indicates that many times clients stop development at around 85-90% of the original budget because the product is good enough to be shipped thereby saving costs for the company [9].

3.2.5 Operations and Maintenance

This stage is also divided into three parts – deployment, feedback and evolution. At this stage, a complete functional and acceptable system should be delivered to the client for shipping. The complaints of customers and possible suggestions on improvements they may have should be considered and implemented for another release.

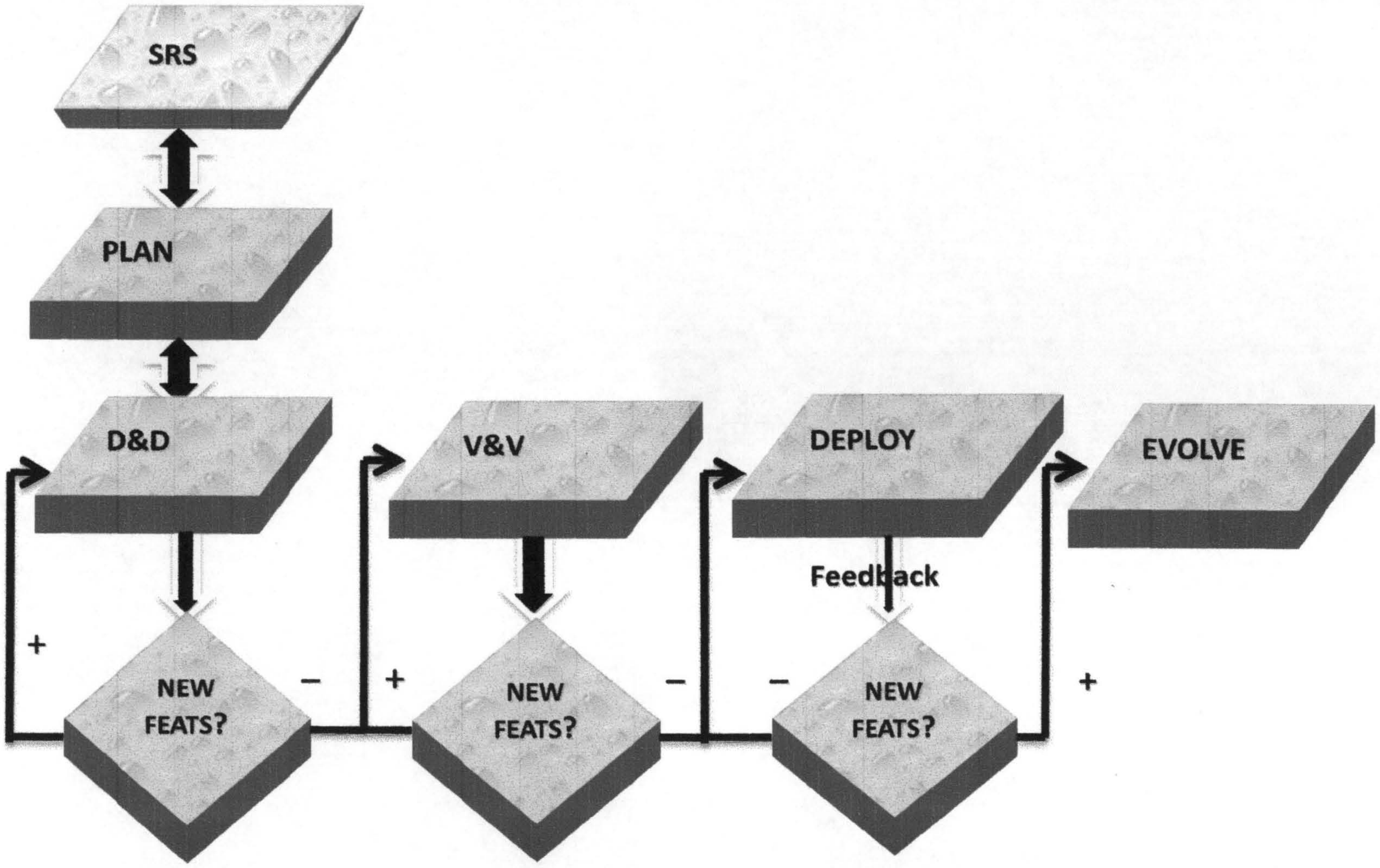

Figure 3.0 The Pragmatic Development Model

CHAPTER FOUR

Results and Discussion

4.1 Comparative Analyses between the Pragmatic Model and Others

This section analyzes the model with respect to the way it improves on some of the shortcomings of some of the mostly used models. Only the three Generic Models are used for comparison since all other models described earlier in Chapter Two are offshoots from them, but the Agile Methodology being a more recent, far removed methodology from the others and the Spiral Model because of its uniqueness in terms of risk assessment will be added. The weaknesses of the models mentioned were gotten from reports, textbooks, and web posts and sometimes by observation.

Tables 4.0, 4.1, 4.2, 4.3, 4.4 shows the uniqueness of the model over the waterfall, RAD, Evolutionary Process and the Spiral Models respectively; the care and effort put into the development of the Agile Methodologies makes it a bit difficult to come up with its shortcomings and as such only a few will be mentioned with respect to the Agile Methodologies.

Table 4.1: Differences between the Waterfall Model and the Pragmatic Model

Waterfall Model	**The Pragmatic Model**
Very bureaucratic	Very flexible
It was built as a flawed and non-working model	It is practical based
The client has no idea of what is being built and cannot make variations until the project is completed	The client is worked with closely and can make changes as the work progresses
When a stage is completed, it discourages backtracking in spite of errors that may be present	Clients are encouraged to make changes whenever they come up with them

Table 4.2: Differences between the RAD Model and the Pragmatic Model

RAD Model	**The Pragmatic Model**
Due to time constraints the product may not be at top quality	Aimed at producing high quality, shippable product
It burns out good technicians at an alarming rate.	They are free to make modifications on the model to suit their needs

Table 4.3: Differences between the Evolutionary Process Model and the Pragmatic Model

Evolutionary Model	**The Pragmatic Model**
High risk	Risks are controlled and monitored (fixed burn rate)
This method is used as an excuse for hacking to avoid documenting the requirements even if they are well understood [10]	It involves proper documentation and is no excuse for crime
Users/acquirers do not understand the nature of the approach and can be disappointed when results are unsatisfactory	The nature of the software may not be understood at first but all these would be eliminated before final delivery

Table 4.4: Differences between the Spiral Model and the Pragmatic Model

Spiral Model	**The Pragmatic Model**
The risk assessment is rigidly anchored in the process and in some cases the risk assessment may not be necessary in this detail [40].	Risks are avoided by trying to ensure that needs are met to minimize backtracking
Assuring customers that the evolutionary approach is controllable is difficult.	Assuring customers that an Agile approach is to be used is not difficult

Table 4.5: Differences between the Agile Model and the Pragmatic Model

Agile Model	**The Pragmatic Model**
The ability to cope with corrections or deficiencies introduced into the product is difficult [24]	Appropriate measures are taken while designing the code for possible modifications and improvements
There is the possibility that the project will go off track because of lack of emphasis laid on documentation and designing	A lot of emphasis is laid on documentation to ease feedback processes

4.2 Discussion

4.2.1 The Waterfall Model

Figure 2.1 shows the pictorial representation of the waterfall model reflecting its linear features. The waterfall model is the about the oldest model used by software engineers to develop software packages. It was presented as a non-practicable model with many flaws but because of the advantages it presented, it soon became widely used in the field. [32]. As indicated in Table 4.1, the new model overrides the Waterfall Model on the grounds that it is based on practice; not to mention that it also reduces bureaucracy to the barest minimum.

4.2.2 The RAD Model

In table 4.2 are highlighted the features that the Pragmatic Model has over the RAD Model. It showed that although the RAD Model method creates a situation where the products are delivered rapidly, it also posed the problem of not being able to meet the needs of consumers in the long run. These shortcomings have been noted and avoided in the new model.

4.2.3 The Spiral Model

As shown in Table 4.4 the problem of risk assessment was taken advantage of during the design of the new model because it created the realization that the whole process is very risky and a lot of care has to be taken so as not to incur losses in the long run. The spiral model was a big improvement as it stressed on evolutionary incremental development, risk management, prototyping and project overview and planning [22]. Generally speaking, the Spiral Model is not much esteemed and not much used, although it has many advantages and could have even more if the risk assessment phases would be tailored down to the necessary amount [40].

4.2.4 The Agile Methodologies

Agile methodologies came into being when developers felt the need for change from the older bureaucratic methods which don't necessarily provide the customers with what they really want. In Agile methods, all software development processes are considered to be empirical rather than defined (also known as prescriptive) [24] and it involves adaptivity and feedback throughout the process of developing a system.

Features of the methodologies practiced include simplicity, adaptivity, incremental and Maximize stakeholder value [6]. Agile methodologies are also people oriented rather than process oriented; meaning that a process cannot replace the skill of a development team. It is also adaptive rather than predictive indicating that the methodology responds well to change and doesn't try to assume what the process should be like thereby resisting change.

The ideas coined by the Agile Methods really served as a booster for the reasoning behind the Pragmatic Model. They introduced the idea of flexibility and less bureaucracy and the need for the process to be adaptive, not predictive which renders the work rigid and liable to failure. Although it has it shortcomings as indicated in Table 4.5, which were taken advantage of in this project, it has also played a vital role in the development processes practiced in the field.

4.2.5 The Pragmatic Model

A major shortcoming present in this project is the fact that the model could not be tested before its presentation because of lack of developers in the environment. For a cogent conclusion to be arrived at and an improvement on the beauty of the project, there is the need for the model to be put to test so that real life results can be obtained. For instance, the assumptions made about the edge it has over the older models are theoretical and for a fact, it is mentioned regularly that assumptions made theoretically are not necessarily practicable in real life situations not to mention that these practices are not necessarily carried out within the environment here in Nigeria.

CHAPTER FIVE

Conclusions and Recommendation

5.1 Conclusions

This Project showed a brief discussion of a few software development process models, highlighted some of their weaknesses and strengths and went further to design a model based on practical events. The edge the new model has over the existing ones were also highlighted in tables and a brief discussion was carried out eventually.

The results show that some of the shortcomings of most of the models discussed were put into consideration and improved upon where necessary and changes were made when possible. This model may not necessarily bring about the solution to the problems in the field and the wish of any designer is that someone else out there will learn from his mistakes and improve on them someday. Obviously, the model still possibly has a few weaknesses that passed unnoticed during the process of its design.

The aim of the paper so far was achieved as indicated in the report and there is hope that it will be useful to not just the Software Engineering Society but to all of humanity.

5.2 Recommendation

As a recommendation, if it is possible, a student should put the model to practical use to see how practicable the Pragmatic Model is in real life situations so that he can suggest some possible modifications that can be carried out on it to further improve the model.

References

1. A. Cockburn, Agile Software Development. Reading, Massachusetts: Addison Wesley Longman, 2001, pp. 164-173.
2. B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems", *Communications ACM*, 1988, 31, 11, 1268-1287.
3. B. W. Boehm, "A Spiral Model of Software Development and Enhancement" *Computer*, 20(9), 1987, pp. 61-72.
4. B. W. Boehm, "Model of Software Development and Enhancement", *IEEE Computer*, May 1988.
5. B. W. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, "Using the WinWin Spiral Model": *A Case Study, Computer*, 31(7), 1998, pp .33-44.
6. C. Bodea, "Agile Software Project Management Methodologies. The Agile Approach". *A Technical Paper. Economic Informatics Department, Academy of Economic Studies, Bucharest,* 2005, pp. 27-31.
7. C. Haase, "Crystal Methodology". 2008, *(Retrieved on the 14th of December, 2011).* http://weblogs.java.net/blog/chet/archive/2008/01/crystal_methodo.html
8. D. Chase, and B. Denneen, "Dynamic Adaptive Framework". *Embracing the Power of Change through L.E.A.N Cycle Planning*, 2001, pp. 1-17.
9. D. Stein, In: T. Hamilton-Whitaker, "The Difference between Waterfall, Iterative Waterfall, Scrum and Lean Software Development (In Pictures!) " *(Retrieved from the Web on the 11th Of December, 2010),* 2009, http://agile101.net/2009/09/08/the-difference-between-waterfall-iterative-waterfall-scrum-and-lean-in-pictures/
10. "Development Models". *Retrieved from the Web on the 20th of March, 2011.* http://myprojects.kostigoff.net/methodology/development_models/development_models.htm
11. F. P. Brooks, "The Mythical Man-Month", *Anniversary Edition: Addison-Wesley Publishing Company, Harlow England,* 1995.
12. H. Gull, F. Azam, W. B. Haider and S. Z. Iqbal, "A New Divide & Conquer Software Process Model", *World Academy of Science, Engineering and Technology 60,* 2009, pp. 255-260.
13. I. Crnkovic, S. Larsson, M. Chaudron, "Component-based Development Process and Component Lifecycle". *A Technical Paper.*
14. I. Sommerville, "Software Engineering". *Eighth Edition. Addison Wesly. Harlow England,* 2007, pp 63-85.
15. IEEE Standards Collection: Software Engineering, IEEE Standard 610.12—1990, IEEE, 1993.
16. J. Highsmith, "Adaptive Software Development". *New York, NY: Dorset House*, 1999.
17. J. Stapleton, "DSDM, The Method in Practice", *Second ed: Addison Wesley Longman,* 2003.
18. K. Allenby, et al. A Family-Oriented Software Development Process for Engine Controllers. *A Technical Paper. Rolls-Royce University Technology Centre in Systems and Software Engineering,* York, UK, 2001, pp. I-XV.
19. K. Beck, "Test Driven Development -- by Example". *Boston: Addison Wesley,* 2003.
20. K. Beck et al. "The Agile Manifesto," http://www.agileAlliance.org, 2001.
21. K. Henney, "Iterative & Incremental Development Explained". 2007, *A web page retrieved from the web on the 11th of April, 2011.* www.searchsoftwareQuality.com
22. K. Johri, "The Agile Concurrent Software Process". *Retrieved from the Web on the 2nd of February, 2011.* http://www.softprayog.in/papers/software_process.shtml&usg
23. K. Schwaber, and M. Beedle, "Agile Software Development with SCRUM". *Upper Saddle River, NJ: Prentice-Hall,* 2002.
24. L. Williams, "A Survey of Agile Development Methodologies". 2007, pp 209-227.
25. Lehigh University, "Agile Competition is Spreading to the World". http://www.ie.lehigh.edu/ , 1991.
26. M. Aoyama, "Agile Software Process and its Experience," *International Conference on Software Engineering, Kyoto, Japan,* 1998, pp. 3-12.
27. M. Fowler, "The New Methodology". 2005, An Article Posted to the Web. (Retrieved on the 11th of December, 2010). http://martinfowler.com/articles/newMethodology.html.
28. M. Fowler, & J. Highsmith, "The Agile Manifesto," in *Software Development,* August 2001, pp. 28-32.

29. M. Poppendieck, and T. Poppendieck, "Lean Software Development: An Agile Toolkit". *Addison-Wesley Professional*, 2003.
30. P. Coad, E. LeFebvre, and J. DeLuca, "Java Modelling in Color with UML" *Prentice Hall*, 1999.
31. P. Naur, and B. Randall (eds.), "Software Engineering", *A Report on a Conference Sponsored by the NATO Science Committee*, NATO, 1969.
32. "Prototyping Software Lifecycle Model". *Retrieved from the Web on the 18th of April, 2011.* http://www.freetutes.com.
33. R. S. Pressman, "Software Engineering", *A Practitioner's Approach", 5th Edition. New York: McGraw-Hill*, 2001, pp. 20-47.
34. R. S. Pressman, "Software Engineering", *A Practitioner's Approach, Sixth Edition, New York: McGraw Hill*, 2005.
35. R. Wysocki, "Introduction to the Adaptive Project Framework". *A White Paper*, 2010.
36. "Software Development Methodology". *A White Paper*, http://www.cs.umd.edu/class/spring2003/cmsc838p/process/software_development_methodology *(retrieved on the 22nd of August, 2010).*
37. "Software Reusability". *A Web Page Posted to the Web on the 4th of January, 2010. (Retrieved on the 11th of April, 2011).* http://centurion2.com/SEHomework/SoftwareReusability/SoftwareReusability.html
38. S. Biffl, D. Winkler, R. Höhn, and H. Wetzel, "Software Process Improvement in Europe: Potential of the New V-Modell XT and Research Issues Practice Section". *Published online in Wiley InterScience (www.interscience.wiley.com) DOI: 10.1002/spip.* 2006, pp. 266, 229-238.
39. S. R. Palmer, and J. M. Felsing, "A Practical Guide to Feature-Driven Development". *Upper Saddle River, NJ: Prentice Hall PTR*, 2002.
40. Software Experts. "Software Process Models". *An Article Retrieved from the Web on the 5th of February, 2011.* http://www.the-software-experts.de/e_dta-sw-process.htm
41. T. Gilb, "Estimation or Control?" – *Thesenpapier, abgerufenim* Juni 2007, *http://www.dasma.org/*
42. T. Hamilton-Whitaker, "The Difference between Waterfall, Iterative Waterfall, Scrum and Lean Software Development (In Pictures!)". 2009, (*Retrieved from the web on the 11th of December, 2010*). http://agile101.net/2009/09/08/the-difference-between-waterfall-iterative-waterfall-scrum-and-lean-in-pictures/
43. "Ten-Step Project Management Process". http://www.mariosalexandrou.com/methodologies/systems-development-life-cycle.asp *(retrieved from the Web in December, 2010*).
44. "The Waterfall Model Advantages and Disadvantages". *A web page retrieved on the 5th of March, 2011.* http://www.buzzle.com/editorials/1-5-2005-63768.asp
45. W. Moore, "About RAD". *An Article Retrieved from the Web on the 14th of April, 2011*, 1997, http://wmoore.ca/demo/opinion/rad.htm.
46. W. S. Humphrey, "Using a Defined and Measured Personal Software Process". Published *In IEEE Software*, 1996, pp. 77-88.
47. W. S. Jawadekar, "Software Engineering. Principles and Practices". *Tata Mcgraw-Hill Publishing Company Limited*, 2004, pp. 19-33.
48. W. Scacchi, "Process Models in Software Engineering". *John Wiley and Sons, Inc, New York*, 2001, 1-24.
49. W. W. Royce, "Managing the Development of Large Software Systems", *Proc. 9th. Intern. Conf. Software Engineering, IEEE Computer Society*, 1987, 328-338. *Originally published in Proc. WESCON, 1970.*