NATIONAL OPEN UNIVERSITY OF NIGERIA

SCHOOL OF SCIENCE AND TECHNOLOGY

COURSE CODE: CIT432

COURSE TITLE: SOFTWARE ENGINEERING II

COURSE GUIDE

CIT432
SOFTWARE ENGINEERING II

Course Team Dr. B.C.E Mbam (Developer/Writer) - EBSU
Dr. Juliana Ndunagu (Programme Leader/Coordinator) - NOUN

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Office
No. 5 Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

Published By:
National Open University of Nigeria

First Printed 2012

ISBN: 978-058-335-1

CONTENTS **PAGE**

Introduction

CIT432: Software Engineering II is a 2 -credit course available for students studying towards acquiring a Bachelor of Science in Computer Science and other related disciplines.

The course is divided into four (4) modules and 14 study units. It entails a brief review of the fundamental concepts of software engineering. It further deals with the different stages involved in developing good, functional, reliable and maintainable software. The course also discusses software engineering models. Formal methods of software development are also treated. It covers software management methodologies and techniques.

What You Will Learn in This Course

The overall aim of this course is to teach you the various models of software development. Program testing, software management methodologies, management techniques and formal methods are also treated. In this course, you will be equipped with the basic and advance means of developing good, functional, reliable and maintainable software. You will also learn about program testing. Finally, you will learn how to develop reliable software in teams and by using the best available models.

Course Aim

This course aims to take a step further in teaching you the basic and best approach to software development. It is hoped that the knowledge would enhance both the software development expertise and your ability to manage developed software.

Course Objectives

It is important to note that each unit has specific objectives. Students should study them carefully before proceeding to subsequent units. Therefore, it may be useful to refer to these objectives in the course of your study of the unit to assess your progress. You should always look at the unit objectives after completing a unit. In this way, you can be sure that you have done what is required of you by the end of the unit.

However, below are the overall objectives of this course:

- What a computer software is
- Different types of computer software
- What engineering is all about

- How engineering principles are applied to software development
- When a software is said to be well engineered.
- Why it is necessary for you to study software engineering.
- The software generations
- The basic features of each generations
- Differences between software and hardware
- Basic characteristics of software
- Explanation with clear examples of system software
- The different application software and where they are being used.
- The knowledge of operating system
- Explanation of programming languages
- The relationship between system software and other software
- The differences between the low level languages and the high level languages
- Definition and explanation of software development model
- The understanding of the background/origin of software development model
- Brief discussion of the classes of software model
- The different types of software engineering model.
- The importance of these models in software development
- The waterfall model of software life cycle
- Phases involved in waterfall model
- Advantages and disadvantages of the waterfall model
- The build-and-fix model
- Advantages and disadvantages of the build-and-fix model
- The appropriate use of build-and-fix model
- Description of the rapid prototype model
- Basic features of the rapid prototyping model
- Use of the rapid prototyping model
- The spiral model and all the phases involved in it.
- Appropriate use of spiral model
- Advantages and disadvantages of the spiral model
- Explanation of SDLC
- Identification of all the stages involved in software development: planning, construction and maintenance.
- Basic factors that must be considered in software development
- The importance of having a standard in software development
- The importance of software testing
- Explanation of program code
- Identification of good code
- The qualities of good code
- The different Program Control Structures
- Modularity in programming
- Software Testing

- Software errors
- Software Testing methods
- Meaning of formal methods
- Some examples of formal method
- Reasons for learning formal method
- Classification of formal methods
- Uses of formal methods
- Caution in the use of formal method
- Limitations of formal methods.
- Explanation of project management
- Project management tools
- Project management process
- Project management methodologies
- Software Project management techniques

Working through This Course

To complete this course, you are required to study all the units, the recommended text books, and other relevant materials. Each unit contains some self assessment exercises and tutor - marked assignments, and at some point in this course, you are required to submit the tutor marked assignments. There is also a final examination at the end of this course. Stated below are the components of this course and what you have to do.

Course Materials

The major components of the course are:

1. Course Guide
2. Study Units
3. Text Books
4. Assignment File
5. Presentation Schedule

Study Units

There are 4 modules and 12 study units in this course. They are:

Module 1 Software Engineering Fundamentals

Unit 1 Introduction
Unit 2 Software Evolution and Software Characteristics
Unit 3 Classifications of Computer Software

Module 2 Software Engineering Models

Unit 1 Overview of Software Development Models
Unit 2 The Waterfall Model and The Build-And-Fix Model
Unit 3 The Rapid Prototyping Model and The Spiral Model

Module 3 Software Development Life Cycle (SDLC)

Unit 1 Overview of the Process Involved
Unit 2 Systems Analysis and Software Requirement Specification
Unit 3 Software Coding

Module 4 Formal Methods

Unit 1 Overview of Formal Methods
Unit 2 Overview of some Formal Methods
Unit 3 Software Project Management

Textbooks and References

These texts will be of enormous benefit to you in learning this course:

Adelard, R., M. & Peter Froome. *Mural and Specbox. In VDM'91*

Bjørner, Dines; Cliff, B. Jones (1978). *The Vienna Development Method: The Meta-Language, Lecture Notes in Computer Science 61*. Berlin, Heidelberg, New York: Springer.

Boehm, B. W. (2001). *Software Engineering Economic*. Englewood Cliffs, New Jersey: Prentice-Hall.

Boehm, B. (1987). A Spiral Model of Software Development and Enhancement, *Computer*, 20(9), 61- 72.

Boehm, B. A.; Egyed, J.; Kwan, D.; Port, A.; Shah, & R. Madachy (1998). Using the WinWin Spiral Model: A Case Study, *Computer*, 31(7), 33-44.

Chatters, B.W.; Lehman, M. M.; Ramil, J.F. & Werwick, P. (1987). Modeling a Software Evolution Process: A Long-Term Case Study, Software Process-Improvement and Practice, 5(2-3), 91-102, 2000. 4, 5, 19-25.

Chatters, B.W.; Lehman, M.M.; Ramil, J. F. & Werwick, P. (1987). Modeling a Software Evolution Process: A Long-Term Case

Study, *Software Process-Improvement and Practice*, 5(2-3), 91-102, 2000.4, 5, 19-25.

Chatters, B.W.; Lehman, M.M.; Ramil, J.F. & Werwick, P. (2000). Modeling a Software Evolution Process: A Long-Term Case Study. *Software Process-Improvement and Practice,* 5(2-3), 91-102.

Derek, A. & Darrel Ince (2001). *Practical Formal Methods.* McGraw Hill Book Corporation.

Fitzgerald, J.S & Larsen, P.G. (1998). *Modelling Systems: Practical Tools and Techniques in Software Engineering*. Cambridge: Cambridge University Press.

Kaiser, G. & Feiler, P. (1988). Intelligent Assistance for Software Development and Maintenance. *IEEE Software,* pages 40-49, May.

Hußmann, H. (1997). Formal Foundations for Software Engineering Methods, *Volume 1322 of Lect. Notes Comp. Sci*. Berlin: Springer.

Bradac, M. *et al.* (1993). Prototyping a Process Monitoring Experiment. In Proceedings of the 15th International Conference on Software Engineering, page 155-165, May

Mbam, B.C.E. (2002). *Information Technology and Management Information System.* Enugu: Our Saviours Press limited.

Mili, A.; Desharnais, J. & Gagne, J.R. (1986). Formal Models of Stepwise Refinement of Programs, *ACM Computing Surveys*, 18, 3, 231-276.

Moore, J.W.; DeWeese, P.R. & Rilling, D. (1997). "U. S. Software Life Cycle Process Standards," *Crosstalk: The DoD Journal of Software Engineering*, 10:7, July.

Sajan, M. (2007). *Software Engineering.* Ram Naga, New Delhi: S. Chand & Company Ltd, – 11Q Q55.

Scacchi, W. & Mi; .P. (1997). "Process Life Cycle Engineering: A Knowledge-Based Approach and Environment". *Intelligent Systems in Accounting, Finance, and Management*, 6(1):83-107.

Scacchi, W. "Understanding Software Process Redesign using Modeling, Analysis and Simulation". *Software Process -- Improvement and Practice* 5(2/3):183-195, 2000.

Basili, V. and Weiss, D. A (1984). Methodology for Collecting Valid Software Engineering Data. *IEEE/ACM Transactions on Software Engineering, SE*-10(6): 728-738, November

Wood, M. & Sommerville, I. (1988). A Knowledge-Based Software Components Catalogue, *Software Engineering Environments*, Ellis Horwood, P. Brererton (ed.), Chichester, England, 116-131.

Yu, E.S.K. & Mylopoulos, J. (1994). Understanding "Why" in Software Process Modelling, Analysis, and Design, *Proc. 16th. Intern. Conf. Software Engineering*, 159 -168,

Assignment File

The assignment file will be given to you in due course. In this file, you will find all the details of the work you must submit to your tutor for marking. The marks you obtain for these assignments will count towards the final mark for the course. Altogether, there are 12 tutor marked assignments for this course.

Presentation Schedule

The presentation schedule included in this course guide provides you with important dates for completion of each tutor marked assignment. You should therefore endeavor to meet the deadlines.

Assessment

There are two aspects to the assessment of this course. First, there are tutor marked assignments; and second, the written examination. Therefore, you are expected to take note of the facts, information and problem solving gathered during the course. The tutor marked assignments must be submitted to your tutor for formal assessment, in accordance to the deadline given. The work submitted will count for 40% of your total course mark. At the end of the course, you will need to sit for a final written examination. This examination will account for 60% of your total score.

Tutor -Marked Assignments (TMAs)

There are 12 TMAs in this course. You need to submit all the TMAs. The best 4 will therefore be counted. When you have completed each

assignment, send them to your tutor as soon as possible and make certain that it gets to your tutor on or before the stipulated deadline. If for any reason you cannot complete your assignment on time, contact your tutor before the assignment is due to discuss the possibility of extension. Extension will not be granted after the deadline, unless on extraordinary cases.

Final Examination and Grading

The final examination for CIT423 will be of last for a period of 2 hours and have a value of 60% of the total course grade. The examination will consist of questions which reflect the self assessment exercise and tutor marked assignments that you have previously encountered. Furthermore, all areas of the course will be examined. It would be better to use the time between finishing the last unit and sitting for the examination, to revise the entire course. You might find it useful to review your TMAs and comment on them before the examination. The final examination covers information from all parts of the course.

Course Marking Scheme

The following table includes the course marking scheme

Table 1: Course Marking Scheme

Assessment	**Marks**
Assignments 1-12	12 assignments, 40% for the best 4 Total = 10% X 4 = 40%
Final Examination	60% of overall course marks
Total	100% of Course Marks

Course Overview

This table indicates the units, the number of weeks required to complete them and the assignments.

Table 2: Course Organiser

Unit	Title of the work	Weeks Activity	Assessment (End of Unit)
	Course Guide	Week 1	
Module 1	**Software Engineering Fundamentals**		
1	Introduction	Week 1	Assessment 1
2	Software Evolution and Software Characteristics	Week 2	Assessment 2
3	Classifications of Computer Software	Week3	Assessment 3
Module 2	**Software Engineering Models**		
1	Overview of Software Development Models	Week 4	Assessment 4
2	The Waterfall Model and The Build-and-Fix Model	Week 5	Assessment 5
3	The Rapid Prototyping Model and The Spiral Model	Week 6	Assessment 6
Module 3	**Software Development Life Cycle (SDLC)**		
1	Overview of The Process Involved	Week 7	Assessment 7
2	Systems Analysis and Software Requirement Specification	Week 8	Assessment 8
3	Software Coding	Week 9	Assessment 9
Module 4	**Formal Methods**		
1	Overview of Formal Methods	Week 10	Assessment 10
2	Overview of some Formal Methods	Week 11	Assessment 11
3	Software Project Management	Week 12	Assessment 12

How to Get the Most Out of This Course

In distance learning, the study units replace the university lecturer. This is one of the huge advantages of distance learning mode; you can read and work through specially designed study materials at your own pace and at a time and place that is most convenient. Think of it as reading from the teacher, the study guide indicates what you ought to study, how to study it and the relevant texts to consult. You are provided with exercises at appropriate points, just as a lecturer might give you an in-class exercise.

Each of the study units follows a common format. The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next to this is a set of learning objectives. These learning objectives are meant to guide your studies. The moment a unit is finished, you must go back and check whether you have achieved the objectives. If this is made a habit, then you will increase your chances of passing the course. The main body of the units also guides you through the required readings from other sources. This will usually be either from a set book or from other sources. Self assessment exercises are provided throughout the unit, to aid personal studies and answers are provided at the end of the unit. Working through these self tests will help you to achieve the objectives of the unit and also prepare you for tutor marked assignments and examinations. You should attempt each self test as you encounter them in the units.

The following are practical strategies for working through this course:

1. Read the course guide thoroughly
2. Organise a study schedule. Refer to the course overview for more details. Note the time you are expected to spend on each unit and how the assignment relates to the units. Important details, e.g. details of your tutorials and the date of the first day of the semester are available. You need to gather together all these information in one place such as a diary, a wall chart calendar or an organiser. Whatever method you choose, you should decide on and write in your own dates for working on each unit.
3. Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they get behind with their course works. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.
4. Turn to Unit 1 and read the introduction and the objectives for the unit.
5. Assemble the study materials. Information about what you need for a unit is given in the table of content at the beginning of each unit. You will almost always need both the study unit you are working on and one of the materials recommended for further readings, on your desk at the same time.
6. Work through the unit, the content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit, you will be encouraged to read from your set books.
7. Keep in mind that you will learn a lot by doing all your assignments carefully. They have been designed to help you meet

the objectives of the course and will help you pass the examination.

8. Review the objectives of each study unit to confirm that you have achieved them. If you are not certain about any of the objectives, review the study material and consult your tutor.
9. When you are confident that you have achieved a unit's objectives, you can start on the next unit. Proceed unit by unit through the course and try to pace your study so that you can keep yourself on schedule.
10. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor marked assignment form and also written on the assignment. Consult you tutor as soon as possible if you have any questions or problems.
11. After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this course guide).

Facilitators/Tutors and Tutorials

There are 8 hours of tutorial provided in support of this course. You will be notified of the dates, time and location together with the name and phone number of your tutor as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course. You must mail your tutor marked assignment to your tutor well before the due date. At least two working days are required for this purpose. They will be marked by your tutor and returned to you as soon as possible. Do not hesitate to contact your tutor by telephone, e-mail or discussion board if you need help. The following might be circumstances in which you would find help necessary, contact your tutor if:

- You do not understand any part of the study units or the assigned readings.
- You have difficulty with the self test or exercise.
- You have questions or problems with an assignment, with your tutor's comments on an assignment or with the grading of an assignment.

You should endeavour to attend the tutorials. This is the only opportunity to have face-to-face contact with your tutor and ask questions which are answered instantly. You can raise any problem

encountered in the course of your study. To gain the maximum benefit from the course tutorials, have some questions handy before attending them. You will learn a lot from participating actively in discussions.

Course Code CIT432
Course Title Software Engineering II

Course Team Dr. B.C.E Mbam (Developer/Writer) - EBSU
Dr. Juliana Ndunagu (Programme Leader/Coordinator) - NOUN

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Office
No. 5 Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

Published By:
National Open University of Nigeria

First Printed 2012

ISBN: 978-058-335-1

CONTENTS **PAGE**

MODULE 1 SOFTWARE ENGINEERING FUNDAMENTALS

UNIT 1 INTRODUCTION

CONTENTS

1.0 INTRODUCTION

The need for computer software is fast growing in our society today. Software controls generation and distribution of electricity, water purification and distribution, robotic systems in production plants, traffic flows, household equipment, aircraft, air traffic, etc. Software also plays an ever-increasing role in business management: it controls equipment maintenance management, logistics, resources allocations etc. In view of this, it is imperative that all and sundry seek to understand what computer software is. In this unit, we shall explain clearly what computer software is? We shall also discuss the need to employ engineering principles in the development of good, functional, reliable and maintainable software. Finally, this unit highlights some reasons you should study software engineering.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- define computer software
- differentiate between the types of computer software
- define software engineering
- apply engineering principles in software development
- determine when a software is said to be well engineered
- describe the essence of studying software engineering.

3.0 MAIN CONTENT

3.1 What is Computer Software?

Software can be defined as aggregates of computer programs together with their appropriate documentation and set of data they need for real life computation/processing. Computer software drives the hardware (physical components) of the computer system. Computer software also enable computer users solve their day-to-day problem.

According to Sajan (2003), software is not just a fancy name for programming. There are some clear distinctions between the two which could be given as follows:

PROGRAM SOFTWARE

Program is usually small in size	Large in size
Single developer	Team of developers
Lacks proper user interface	Well- designed interface
Lacks proper documentation	Is well documented
Ad hoc development	Systematic Development

3.2 Growing Importance of Software

The importance of software is fast growing. For many engineering and other projects, software has become the pivotal tool. For example:

- Software controls generation and distribution of electricity, water purification and distribution, robotic systems in production plants, vehicles and their engines, traffic flows, household equipment, aircraft, air traffic, and passenger bookings etc
- Software also plays an ever-increasing role in business management: it controls equipment maintenance management,

logistics, resource allocations, business processes, financial transactions, accounting, communication, human resources, etc.

SELF-ASSESSMENT EXERCISE 1

1. What is computer program?
2. List some areas you feel software are being used.
3. What do you understand by ad hoc development?

3.3 Engineering

The American Engineers' Council for Professional Development, defines engineering as the creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilising them singly or in combination; or to construct or operate the same with full cognisance of their design; or to forecast their behaviour under specific operating conditions; all as respects an intended function, economics of operation and safety to life and property.

Oxford Advance Learner's Dictionary defines "engineering" as the activity of applying scientific knowledge (knowledge obtained by testing of facts, observation and inferences) to the design, building and control of machines, roads, bridges, software etc. Engineering normally employs well defined procedures/principles to achieve its objective.

3.4 Software Engineering

Software engineering can be defined as the act of employing established engineering principles in the development of good, functional, reliable and maintainable computer software. Software engineering is deals with software developed by teams rather than individual programmers.

However, the definition varies from one author to another, some examples include:

- Software engineering is a discipline that integrates methods, tools, and procedures for the development of computer software (Pressman, 2000).
- Software engineering is the establishment and the use of sound engineering principles to obtain economically software that is reliable and works effectively on computers (Mbam B.C , 2002).
- Software engineering is an emerging discipline that focuses on the creation, development, operation and maintenance of cost effective, reliable, correct, and high quality solutions to software problems (Berry).

- The application of systematic, disciplined, quantifiable approach to development, operation and maintenance of software (IEEE Standard Computer Dictionary).
- Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software (adapted from SWEBOK, the Software Engineering Body of Knowledge).

3.5 What is Well -Engineered Software?

Well-engineered computer software can be described as software that does what the user wants and can be made to continue to do what the user wants. However, Somerville suggests that well-engineered software should:

- Be easy to use
- Be easy to maintain
- Be reliable
- Be efficient
- Provide an appropriate user interface.

3.6 Why Study Software Engineering?

There are many reasons we study software engineering. Some of them include:

- software development needs the structured application of scientific and engineering principles in order to analyse, design, construct, document and maintain it
- like any engineering development, large-scale software development also requires the disciplined application of project management principles
- because of software's growing importance, its development must be managed more carefully than other areas of large projects
- individual approach is no longer appropriate; and the departure point for proper software development should be the realisation that software development has grown from an art to a craft, and to a proper engineering discipline
- to acquire skills to be a better programmer: this makes for higher productivity and better quality programs
- to acquire skills to develop large programs
- it helps you to gather ability to solve complex programming problems

- to learn techniques of specification, design, interface development, testing and integration, project management etc.

SELF-ASSESSMENT EXERCISE 2

1. Define engineering in your own terms.
2. How does software engineering help a programmer to be a better programmer?

4.0 CONCLUSION

Software should be considered as more than a computer program. It includes documentation associated with development, and the user documentation. Software engineering is the establishment and the use of sound engineering principles to obtain economically software that is functional, reliable and works effectively on computers. Well-engineered software should be easy to use, easy to maintain, reliable, be efficient and should also provide an appropriate user interface. Software engineering should be embraced by software developers since it helps in developing in commercial scale.

5.0 SUMMARY

This unit has explained what computer software is, what engineering is all about and how to employ engineering principles in the development of functional computer software. You also saw that some authors simply see software engineering as an emerging discipline that focuses on the creation, development, operation and maintenance of cost effective, reliable, correct, and high quality solutions to software problems. Whatever the definition or explanation may be, the border line is that software engineering helps in the development of functional and reliable software.

6.0 TUTOR-MARKED ASSIGNMENT

1. Define software engineering in your own words.
2. Discuss the importance of software in any sector of the economy of your choice.
3. List and explain any three reasons it is necessary to study software engineering.
4. What is the relationship between program and software?

7.0 REFERENCES/FURTHER READING

Bauer, F. L. Programming as an Evolutionary Process, *Proc. 2nd. Intern. Conf. Software Engineering*, IEEE Computer Society, 223-234, January, 1976. Retrieved from http://www.sciencedaily.com/ articles/e/engineering.htm

Mbam, B.C.E. (2002). *Information Technology and Management Information System*. Enugu: Our Saviours Press Limited.

Oxford Advanced Learner's Dictionary (6th edition).

Sajan, M. (2007). *Software Engineering* (Rev. ed.). New Delhi: S. Chand & Company Ltd, pp. 1-5, 27-36, 138-141, 152-158, 2881-187.

Wood, M. & Somerville, I. (1988). *A Knowledge-Based Software Components Catalogue, Software Engineering Environments*. Ellis Horwood, P. Brererton (Ed.). Chichester, England. pp.116-131.

UNIT 2 SOFTWARE EVOLUTION AND SOFTWARE CHARACTERISTICS

CONTENTS

1.0 INTRODUCTION

Computer software has a very long history. It came into existence starting from when an English mathematician and inventor designed the world's first programmable machine. This machine, called the Analytical Engine, used punch cards similar to those used in the Jacquard loom to select the specific arithmetic operation to apply at each step. Inserting a different set of cards changed the computations the machine performed. From that time till now, computer software had undergone several metamorphoses. These different levels of changes are represented as software generations. The latest generation uses sound, moving images and agents. An automatic self changing piece of software that creates new agent based on the behaviour of the end user. Software also has some unique characteristics when compared with the hardware.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- define software generations
- discuss the basic features of each generations
- differentiate between software and hardware
- list the basic characteristics of software.

3.0 MAIN CONTENT

3.1 Software Evolution

The modern concept of an internally stored computer program was first proposed by Hungarian-American mathematician John von Neumann in 1945. His idea was to use the computer's memory to store the program as well as the data. In this way, programs can be viewed as data and can be processed like data by other programs. This idea greatly simplifies the role of program storage and execution in computers. However, the generation of computer software can classified as follows:

- First generation
- Second generation (2GL)
- Third generation (3GL)
- Fourth generation (4GL)
- Fifth generation (5GL).

3.1.1 First Generation

This generation came up during early 1950s. In this generation, computers were programmed by changing the wires and tens of dials and switches. Sometimes, these setting could be stored on paper tape that looked like a ticker paper from telegraph - a punch tape or punched card. With tape and or card, the computers were commanded what, how and when to do something. Programming then was done using machine language; so to have a flawless, program, a programmer needed to have very detailed knowledge of the computer.

3.1.2 Second Generation (2GL)

This generation came into existence in the mid of 1950s. This generation made use of symbols and are called an assembler (an assembler is a program that translates symbolic instructions to processor instruction.) The programmer no longer work with one's and zero's when using an assembly language but symbols. These symbols are called mnemonics. Each mnemonics stands for one single machine instruction. However, for each processor, a different assembler was written.

3.1.3 Third Generation (3GL)

This generation came into existence at the end of 1950s. This generation witnessed "natural language" but interpreters and compliers were made. (An Interpreter is a translator that translates high level languages on a statement-by-statement basis so that as each language statement is encountered, it is converted to machine executable codes and executed while a complier is a translator that transforms high-level languages into computer executable language). In 3GL, there was no longer need to work in symbols instead, a programmer could use a programming language that resembled more to "natural language" e.g. FORTRAN, COBOL PASCAL etc.

3.1.4 Fourth Generation (4GL)

In the fourth generation, the primary feature was that you do not indicate HOW a computer must perform a task but WHAT it must do. A few instructions in a 4GL will do the same as hundreds of instructions in a lower generation languages. In most of these cases, one deals with database management system. A trained user in this kind of software can create an application in a much shorter time for development and debugging than would be possible with other generations.

3.1.5 Fifth Generation (5GL)

The basis of this generation was laid in the 1990s by using sound, moving images and agents. Software for the end user may be based on principles of knowbot-agents. An automatic self changing piece of software that creates new agent based on the behaviour of the end user. Human-like quality DNA/RNA (intelligent) algorithms could also play a big role.

SELF-ASSESSMENT EXERCISE 1

What is the basic feature(s) of first and second generation software?

3.2 Software Characteristics

Software is logical rather than physical and as such, it possesses characteristics that are different from that of hardware. According to Mbam, B. C. (2003), software characteristics include the following:

3.2.1 Software is Developed or Engineered

Software is not manufactured in a classical sense. Software is different from physical products in a manufacturing plant. Although some similarities exist between software development and hardware manufacturing, the two activities are fundamentally different.

3.2.2 Software is usually Custom-Built

Even though we hear of software components, not many components are there off-the-shelf. Most of the software are custom-built rather than being assembled from existing components. It is however expected that in the coming years, software component and reusability will catch up.

3.2.3 Software does not Wear Out

Unlike any other physical product, software does not wear out. Software is not subjected to the environmental factors like heat, dust, vibration etc.

3.2.4 Cost for Support and Modification of Delivered Software is High

Research has shown that cost for support and modification of delivered software over the life of a system are typically twice the cost of the original acquisition. Even more significant is the fact that the customer's perception of what the software should be able to do changes as experience is gained with its use.

SELF-ASSESSMENT EXERCISE 2

Why is software said to be custom- built?

4.0 CONCLUSION

The changes in software world are classified into generations. Each of these generations showcases some basic feature that makes it unique from others. The latest generation seeks to imitate human beings. We have also explained that software is developed or engineered, software does not wear out, software is custom-built and that cost for support and modification of delivered software is high. These characteristics differentiate software from the hardware.

5.0 SUMMARY

In this unit, you were thought evolution of the computer software and the basic characteristics of software. Computer software was classified into first, second, third, fourth and fifth generations. You were also made to known that software is logical rather than physical and as such, it possesses characteristics different from that of hardware.

6.0 TUTOR-MARKED ASSIGNMENT

1. Compare and contrast between software and hardware
2. Write short notes on the following software generations
   (a) second generation (b) fourth generation

3. Trace the origin of computer software.

7.0 REFERENCES/FURTHER READING

Chatters, B.W.; Lehman, M.M.; Ramil, J. F. & Werwick, P. (2000). Modeling a Software Evolution Process: *A Long-Term Case Study, Software Process-Improvement and Practice*, 5(2-3), 91-102.

Mbam, B.C.E. (2002). *Information Technology and Management Information System.* Enugu: Our Saviours Press Limited.

Royce, W. W. Managing the Development of Large Software Systems, *Proc. 9th. Intern. Conf. of Software Engineering*, IEEE Computer Society, 1987, 328-338. Originally published in Proc.WESCON, 1970. Retrieved online on 26/09/2010 at http://www.sciencedaily.com/articles/e/engineering.htm

Sajan, M. (2007). *Software Engineering* (Rev. ed.). New Delhi: S. Chand & Company Ltd, pp. 1-5, 27-36, 138-141, 152-158, 2881-187

Wood, M. & Somerville, I. (1988). A Knowledge-Based Software Components Catalogue, *Software Engineering Environments*, Ellis Horwood, P. Brererton (Ed.). Chichester, England, 116-131.

UNIT 3 CLASSIFICATIONS OF COMPUTER SOFTWARE

CONTENTS

1.0 INTRODUCTION

The computer software is categorised basically into three; the system software, the application software and the programming languages. Similar to natural languages, such as English, programming languages have a vocabulary, grammar, and syntax. However, natural languages are not suited for programming computers. These three basic classifications of software are the main focus in this unit.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- describe Instances and Schemes. Explain clearly with examples of system software
- differentiate between application software and their uses
- discuss operating system
- describe what programming languages are
- explain the relationship between system software and other software
- differentiate between the low level languages and the high level languages.

3.0 MAIN CONTENT

3.1 System Software

System software is computer software designed to operate the computer hardware and to provide and maintain a platform for running application software. System software helps use the operating system and computer system. The purpose of system software is to insulate the applications programmer as much as possible from the details of the particular computer complex being used, especially memory and other hardware features, and such accessory devices as communications, printers, readers, displays, keyboards etc. System software includes diagnostic tools, compilers, servers, window systems, utilities, language translator, data communication programs, database systems and more.

The most basic types of system software are: Operating system, the computer BIOS and device firmware, and the utility software.

3.1.1 Operating System (OS)

Operating System (OS) is the basic software that controls a computer. The operating system has many functions: It coordinates and manipulates computer hardware, such as computer memory, printers, disks, keyboard, mouse, and monitor; it organises files on a variety of storage media, such as floppy disk, hard drive, compact disc, digital video disc, and tape. It also manages hardware errors and the loss of data among others. It provides a platform to run high-level system and application software. Prominent examples are Microsoft Windows, Mac OS X and Linux.

3.1.2 The Computer BIOS and Device Firmware

The computer BIOS and device firmware (software routines stored in read-only memory (ROM). Unlike random access memory (RAM), read-only memory stays intact even in the absence of electrical power. Startup routines and low-level input/output instructions are stored in firmware which provides basic functionality to operate and control the hardware connected to or built into the computer.

3.1.3 The Utility Software

Utility software helps to analyse, configure, optimise and maintain the computer system.

Utility software such as an editor or a debugger, are designed to perform a particular function. The term *utility* usually refers to software that

solves narrowly focused problems or those related to computer system management.They could be other system software.

SELF-ASSESSMENT EXERCISE 1

Do you think that system software is indispensible in the computer? Give reasons for your answer.

3.2 Application Software

Application software, also known as an application, is computer software designed to help the user to perform singular or multiple related specific tasks. Examples include word processor, presentation software, spreadsheet, desktop publishing, enterprise software, accounting software, office suites, graphics software, and media players etc. Let us now highlight some common application software.

3.2.1 Word Processor

Word processor (more formally known as document preparation system) is a computer application used for the production (including composition, editing, formatting, and possibly printing) of any sort of printable material. Word processors are descended from early text formatting tools (sometimes called text justification tools, from their only real capability). Word processing was one of the earliest applications for the personal computer in office productivity. Although early word processors used tag-based markup for document formatting, most modern word processors take advantage of a graphical user interface providing some form of What You See Is What You Get editing. Most are powerful systems consisting of one or more programs that can produce any arbitrary combination of images, graphics and text, the latter handled with type-setting capability.

Microsoft Word is the most widely used word processing software. Microsoft estimates that over 500,000,000 people use the Microsoft Office suite, which includes Word. Many other word processing applications exist, including WordPerfect and others. Word processing typically implies the presence of text manipulation functions that extend beyond a basic ability to enter and change text, such as automatic generation of:

- batch mailings using a form letter template and an address database (also called mail merging)
- indices of keywords and their page numbers
- tables of contents with section titles and their page numbers
- tables of figures with caption titles and their page numbers

- cross-referencing with section or page numbers
- footnote numbering
- new versions of a document using variables (e.g. model numbers, product names, etc.).

Other word processing functions include "spell checking" (actually checks against wordlists), "grammar checking" (checks for what seem to be simple grammar errors), and a "thesaurus" function (finds words with similar or opposite meanings). Other common features include collaborative editing, comments and annotations, support for images and diagrams and internal cross-referencing.

3.2.2 Electronic Spreadsheet Applications

The word “spreadsheet” came from “spread” in its sense of a newspaper or magazine item (text and/or graphics) that covers two facing pages, extending across the center fold and treating the two pages as one large one. The compound word “spread-sheet” came to mean the format used to present book-keeping ledgers—with columns for categories of expenditures across the top, invoices listed down the left margin, and the amount of each payment in the cell where its row and column intersect—which were, traditionally, a “spread” across facing pages of a bound ledger (book for keeping accounting records) or on oversized sheets of paper ruled into rows and columns in that format and approximately twice as wide as ordinary paper.

A spreadsheet is a computer application that simulates a paper, accounting worksheet. It displays multiple cells that together make up a grid consisting of rows and columns; each cell contains alphanumeric text, numeric values or formulas. A formula defines how the content of that cell is to be calculated from the contents of any other cell (or combination of cells) each time any cell is updated. Spreadsheets are frequently used for financial information because of their ability to re-calculate the entire sheet automatically after a change to a single cell is made.

Lotus 1-2-3 was the leading spreadsheet when DOS was the dominant operating system. Excel now has the largest market share on the Windows and Macintosh platforms.

3.2.3 Desktop Publishing Software

The term “desktop publishing” is commonly used to describe page layout skills. However, the skills and software are not limited to paper and book publishing. The same skills and software are often used to create graphics for point of sale displays, promotional items, trade show

exhibits, retail package designs and outdoor signs. Desktop publishing began in 1985 with the introduction of MacPublisher, the first of What You See Is What You Get (WYSIWYG) layout program. This was run on the original 128K Macintosh computer. Desktop publishing programs were specifically designed to allow elaborate layout for publication, but often offered only limited support for editing. Typically, desktop publishing programs allowed users to import text that was written using a text editor or word processor. Before the advent of desktop publishing, the only option available to most persons for producing typed (as opposed to handwritten) documents was a typewriter, which offered only a handful of typefaces (usually fixed-width) and one or two font sizes.

Desktop publishing (also known as DTP) combines a personal computer and WYSIWYG page layout software to create publication documents on a computer for either large scale publishing or small scale local multifunction peripheral output and distribution.

3.2.4 Presentation Software

A presentation program is a computer software package used to display information, normally in the form of a slide show. It typically includes three major functions: an editor that allows text to be inserted and formatted, a method for inserting and manipulating graphic images and a slide-show system to display the content. In the mid-1980s, developments in the world of computers changed the way presentations were created. Inexpensive specialised applications now made it possible for anyone with a PC or Macintosh to create professional-looking presentation graphics.

A presentation program is supposed to help both the speaker with an easier access to his ideas and the participants with visual information which complements the talk. There are many different types of presentations including professional (work-related), education, entertainment, and for general communication. Presentation programs can either supplement or replace the use of older visual aid technology, such as pamphlets, handouts, chalkboards, flip charts, posters, slides and overhead transparencies. Text, graphics, movies, and other objects are positioned on individual pages or “slides” or “foils”. The “slide” analogy is a reference to the slide projector, a device that has become somewhat obsolete due to the use of presentation software. Slides can be printed, or (more usually) displayed on-screen and navigated through at the command of the presenter. Transitions between slides can be animated in a variety of ways, as the emergence of elements on a slide itself.

Typically, a presentation has many constraints and the most important is the limited time to present consistent information. The first commercial computer software specifically intended for creating WYSIWYG presentations was developed at Hewlett Packard in 1979 and called BRUNO and later HP-Draw. The first software displaying a presentation on a personal computer screen was VCN ExecuVision, developed in 1982. This program allowed users to choose from a library of images to accompany the text of their presentation. A typical example of presentation software in use today is Microsoft PowerPoint.

Apart from these few application software discussed, there are a lot of other application software in use today.

SELF-ASSESSMENT EXERCISE 2

List out other application software in use today.

3.3 Programming Languages

Programming languages are the various methods of writing computer instructions. The instructions adhere to a particular set of rules for each language. The programming languages are used to create application software discussed in section 3.2 and other software similar to them. There are basically two forms of programming languages; the low level and the high level languages.

3.3.1 Low Level Languages

The low level languages are divided into two; the machine language and the assembly language.

- **Machine Language:** It deals directly with the computer hardware. It uses 0's and 1's to form commands that cause the computer to perform series of operations as specified by the programmer. Machine language is difficult to use and more time consuming.
- **Assembly Language:** This is also a low level language. The assembly language uses symbols instead of 0's and 1's. This language reduced the complexity of program authoring. However, each computer or family of computers has its own assembly language which prevented the software of one computer model from being used on a different computer model.

3.3.2 High Level Languages

High level languages are more like natural languages of the computer users. These types of languages do not bother about the knowledge of the computer hardware. They were developed for two reasons; for the programmer to work on different computers without having to learn a new assembly language each time, and secondly, for software written on one computer could be used on another. Translators (a complier, or interpreter) were used to help solve these problems by translating program into machine language and checking the program syntax errors.

SELF-ASSESSMENT EXERCISE 3

Explain why it is easier to program in high level languages than in low level languages.

4.0 CONCLUSION

The system software is dedicated to operate the computer hardware and to provide and maintain a platform for running application software. The purpose of system software is to insulate the applications programmer as much as possible from the details of the particular computer complex being used, especially memory and other hardware features. The application software helps computer users to solve their day-to-day problem. The programming languages enable programmers to instruct the computer on what to do and how to do them.

5.0 SUMMARY

In this unit, you have learnt that:

- computer software can be classified into three. Namely: the system, software, the application software and the programming languages
- system software operates the hardware
- application software helps users solve their problems
- programming languages enable programmers create instructions for the computer
- the machine language uses 0’s and 1’s to form commands
- the assembly language uses symbols
- the high level languages use natural language like English
- utility usually refers to software that solves narrowly focused problems or those related to computer system management.
- Operating System (OS) is the basic software that controls a computer.

6.0 TUTOR-MARKED ASSIGNMENT

1. Explain the relationship between the system software and the application software.

(a) Explain the following:

i. The low level languages
ii. The high level languages

(b) Which of the languages in (1) above needs a translator and why?

2. Itemise the specific functions of the operating system.
3. What are the basic features of a word processor?
4. Machine language is the language the computer understands. Explain.

7.0 REFERENCES/FURTHER READING

Bauer, F. L., Programming as an Evolutionary Process, *Proc. 2nd. Intern. Conf. Software Engineering*, IEEE Computer Society, 223-234, January, 1976. http://alistair.cockburn.us/The+end+of+software+engineering+and+the+start+of+economic-cooperative+gaming.retrieved 24/09/2010.

Royce, W. W. Managing the Development of Large Software Systems, *Proc. 9th. Intern. Conf. of Software Engineering*, IEEE Computer Society, 1987, 328-338 Originally published in Proc.WESCON, 1970. Retrived online on 26/09/2010 at http://www.sciencedaily.com/articles/e/engineering.htm

Sajan, M. (2007). *Software Engineering* (Rev. ed.). New Delhi: S. Chand & Company Ltd., pp. 1-5, 27-36, 138-141, 152-158, 2881-187.

Somerville, I. (1999). *Software Engineering* (7th ed.). Menlo Park, CA: Addison-Wesley.

Wood, M. & I. Sommerville, A. (1988). Knowledge-Based Software Components Catalogue, *Software Engineering Environments*, Ellis Horwood & P. Brererton (Eds.). Chichester, England, 116-131.

MODULE 2 SOFTWARE ENGINEERING MODELS

UNIT 1 OVERVIEW OF SOFTWARE DEVELOPMENT MODELS

CONTENTS

1.0 INTRODUCTION

Standard software product goes through series of processes to evolve. To enable software developers produce good, functional, maintainable software, there have been some software models which enhance the developmental effort. This models account for their inception, initial development, productive operation, upkeep, and retirement from one generation to another. This unit takes an overview of most of the common software development models. We shall begin with background and definitions of traditional software life cycle models that dominate most textbook discussions and current software development practices. This will be followed by a more comprehensive review of few models of software evolution that are of current use as the basis for organising software engineering projects and technologies.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain software development model
- describe the origin of software development model
- discuss briefly the classes of software model
- differentiate between the types of software engineering model
- discuss the importance of these models in software development.

3.0 MAIN CONTENT

3.1 Definition and Meaning of Life Cycle Model

Different tutors/authors may define software life cycle model in different ways but let us simply define “software life cycle model” as the series of steps through which the software product progresses. Software life cycle model often represent a networked sequence of activities, objects, transformations, and events that embody strategies for accomplishing software evolution. Such models can be used to develop more precise and formalised descriptions of software life cycle activities. Their power emerges from their utilisation of a sufficiently rich notation, syntax, or semantics, often suitable for computational processing.

3.1.1 Background/Origin of Software Models

Explicit software life cycle models evolution date back to the earliest projects developing large software systems in the 1950s and 1960s (Hosier 1961, Royce 1970). At that time, the apparent purpose of these early software life cycle models was to provide a conceptual scheme for rationally managing the development of software systems. Such a scheme could therefore serve as a basis for planning, organising, staffing, coordinating, budgeting, and directing software development activities. Since the 1960s, many descriptions of the classic software life cycle have appeared (Hosier, 1961; Royce, 1970; Boehm, 1976; Distaso, 1980; Scacchi, 1984; Somerville, 1999). Royce (1970), originated the formulation of the software life cycle using the now familiar "waterfall” model and other recent models.

3.1.2 Classes of Software Model

A software life cycle model is either a descriptive or prescriptive characterisation of how software is or should be developed (Marciniak, 2001). 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 (Curtis, Krasner; Iscoe, 1988). Descriptive life cycle models describe how particular software systems are actually developed in specific settings. As such, they are less common and more difficult to articulate for an obvious reason: one must observe or collect data throughout the life cycle of a software system, a period of elapsed time often measured in years. Also, descriptive models are specific to the systems observed and only realisable through systematic comparative analysis.

On the other hand, a prescriptive model prescribes how a new software system should be developed. Prescriptive models are used as guidelines or frameworks to organise and structure how software development activities should be performed, and the order it should be performed. Typically, it is easier and more common to articulate a prescriptive life cycle model for how software systems should be developed. This is possible since most of such models are intuitive or well reasoned when developing different kinds of application systems, in different kinds of development settings, using different programming languages, with differentially skilled staff, etc.

However, prescriptive models are also used to package the development tasks and techniques for using a given set of software engineering tools or environment during a development project. Therefore, this suggests the prescriptive software life cycle models will dominate attention until a sufficient base of observational data is available to articulate empirically grounded descriptive life cycle models.

SELF-ASSESSMENT EXERCISE 1

Explain the descriptive model.

3.2 Reasons for Articulating Software Life Cycle Model

From the above explanation of descriptive and prescriptive software life cycle model, we can see that there are a variety of reasons for articulating software life cycle models. Some of them are:

- Software life cycle model serves as a guideline to organise, plan, staff, budget, schedule and manage software project work over organisational time, space, and computing environments
- Software life cycle model serves as prescriptive outline for what documents to produce for delivery to client
- Software life cycle model serves as a basis for determining what software engineering tools and methodologies will be most appropriate to support different life cycle activities
- Software life cycle model serves as a framework for analysing or estimating patterns of resource allocation and consumption during the software life cycle (Boehm 1981).

It also serves as a basis for conducting empirical studies to determine what affects software productivity, cost, and overall quality.

SELF-ASSESSMENT EXERCISE 2

How does a software life cycle model serve as a prescriptive outline for the documents to produce for delivery to client?

3.3 Different Software Development Models

In this section, we shall take a little time to introduce the different software life cycle models in use today. We will only highlight them here and in subsequent units dwell comprehensively with some of them we feel are most popular.

- **Build-and-Fix Model:** Build-and-fix model is a software development model where the entire software product is built and delivered to the client. The client points out what has to be changed and changes are made until the client is satisfied (Stephen, 2003). The product then goes into operation mode.
- **Spiral Model:** Spiral model is a risk-based model (Stephen, 2003). Risk-based implies that the major objectives here is to determine the risks involved in developing that software product and then resolve each risk in turn; that is, attempt to remove or at least minimise that risk.
- **Rapid Prototyping Model:** Rapid prototyping model is a type of model where the requirement team gathers information and presents their findings in form of a document to the client. They proceed to build a rapid prototype (build code that reflects much of the functionality that the client sees; such as input screens and reports but omits "hidden" aspects, such as file updating).
- **Waterfall Model:** Waterfall model is a type of model whereby the software developer follows a well-defined engineering procedure in the development of a software product. The phase to

go through include; requirement analysis, specification, design, coding, testing and validation, deployment and maintenance.

- **Incremental Model:** The incremental model performs the waterfall in overlapping sections attempting to compensate for the length of waterfall model projects by producing usable functionality earlier. This may involve a complete upfront set of requirements that are implemented in a series of small projects. As an alternative, a project using the incremental model may start with general objectives. Then some portion of these objectives is defined as requirements and is implemented, followed by the next portion of the objectives until all objectives are implemented.
- **Clean Room:** The clean room technique attempts to keep contaminants (software bugs) out of the product. The idea is to control cost by detecting bugs as early as possible, when they are less costly to remove. Rather than using natural languages like English, more formal notations are used to produce specifications on which all software design and requirements validation is based. Off-line review techniques are used to develop understanding of the software before it is executed. Software is intended to execute properly the first time. Programmers are not allowed to perform trial and error executions, though automation checks syntax, data flow, and variable types. Testing uses statistical examination to focus on the detection of the errors most likely to cause operational failures.
- **Extreme Programming**: Extreme Programming does not use specifications. The test cases initially defined are used as a description of the requirements. These are then used after the implementation to help check the (sub-) product. The idea in this excerpt from extreme programming can also be found in the W-model: the left part of the "W" can simply be omitted. This then leaves just the testing activities as tasks up to the point of implementation. The requirements for the system to be developed are then extracted from the specified test cases.
- **V- Model**: In V-model, the "V" describes the graphical arrangement of the individual phases. The "V" is also a synonym for verification and validation. This model is very simple and easy to understand. By the ordering of activities in time sequence and with abstraction levels, the connection between development and test activities becomes clear. Oppositely laying activities complement one another i.e. serve as a base for test activities. So, the system test is carried out on the basis of the results specification phase. Many of the process models currently used can be more generally connected by the V-model where the "V" describes the graphical arrangement of the individual phases. The "V" is also a synonym for verification and validation. The coarse view of the model gives the impression that the test activities first

start after the implementation. However, in the description of the individual activities, the preparatory work is usually listed. So, for example, the test plan and test strategy should be worked out immediately after the definition of the requirements.

- **Synchronise and Stabilise Model**: In this model, during the requirements analysis, interview of potential customers are conducted and requirement document is developed. Once these requirements have been captured, specifications are drawn up. The project is then divided into 3 or 4 builds. Each build is carried out by small team working in parallel. At the end of each day, the code is synchronised (test and debug). Also, at the end of the build, it is stabilised by freezing the build and remaining defects because of the synchronisation components always work together.
- **Object-Oriented Model**: The object-oriented approach to software development focuses on real-world objects. It is based on the premise that there exists a fundamental human limitation to manage more than seven different objects or concepts at one time. Grady Booch suggests that the principles of software engineering can help us decompose systems so that we never simultaneously deal with more than seven entities. Object-oriented popularity is increasing in concert with the increasing complexity of software systems. Object-oriented includes object-oriented analysis (OOA), object oriented design (OOD), and object oriented programming (OOP).

In the subsequent units of this module, we shall take much time to look at four of these models in details. These four engineering models will be treated as prescriptive model even though they may serve as descriptive models.

3.4 Emerging Trends and New Directions in Software Development

In addition to the ongoing interest, debate, and assessment of process-centered or process-driven software engineering environments that rely on process models to configure or control their operation (Ambriola, 1999; Garg and Jazayeri, 1996), there are a number of promising avenues for further research and development with software process models. These opportunities areas and sample direction for further exploration include:

- Software process simulation (Raffo *et al.*, 1999; Raffo and Scacchi, 2000) efforts which seek to determine or experimentally evaluate the performance of classic or operational process models using a sample of alternative parameter configurations or

empirically derived process data. Simulations of empirically derived models of software evolution or evolutionary processes also offer new avenues for exploration (Chatters, Lehman *et al.*, 2000; Mockus, 2000).

- Web-based software process models and process engineering environments (Bolcer, 1998; Grundy, 1998; Penedo, 2000; Scacchi and Noll, 1997) that seek to provide software development workspaces and project support capabilities that are tied to adaptive process models (e.g. Engineering Web Applications with Java).
- Software and business processes reengineering which focuses attention to opportunities that emerge when the tools, techniques, and concepts for each disciplined are combined to their relative advantage (Scacchi and Mi, 1997; Scacchi and Noll, 1997; Scacchi, 2000). This in turn is giving rise to new techniques for redesigning, situating, and optimising software process models for specific organisational and system development settings (Scacchi and Noll, 1997; Scacchi, 2000) (e.g. Business Reengineering in the Age of the Internet).
- Understanding, capturing, and operationalising process models that characterise the practices and patterns of globally distributed software development associated with open source software (DiBona, 1999; Fogel, 1999; Mockus, 2000), as well as other emerging software development processes, such as extreme programming and Web-based virtual software development enterprises or workspaces (Noll and Scacchi, 1999, 2001; Penedo, 2000).

SELF-ASSESSMENT EXERCISE 3

1. List any seven (7) software engineering models you know.
2. Which of the software techniques (models) attempt to keep contaminants (software bugs) out of the product**?**
3. What is a prototype?

4.0 CONCLUSION

In this unit, you have learnt that in order to develop good functional and maintainable software, there are some software models that serve as a guide for software developers. We went further to define and explain what a software development model is. We also categorised the software models into descriptive and prescriptive models. Moreover, we x-rayed some specific reasons for articulating software development models. We again surveyed the emerging trends and new directions in software development. We ended the unit by taking a view at some common software development models.

5.0 SUMMARY

You can now explain what a software development model is. You can as well comfortably state and explain reasons why software models are needed. Descriptive and prescriptive software were also examined. Furthermore, you learnt the various software development models that can be employed in the development of good, functional and maintainable software.

6.0 TUTOR-MARKED ASSIGNMENT

1. Explain clearly, the prescriptive model and the descriptive model.
2. Explain in few sentences, the basic features of the following models:

a. Synchronise and stabilise model
b. Object-oriented model
c. Extreme programming
d. V-model

3. List and explain any five reasons for articulating the software development model
4. Examine briefly, any two emerging trends and new directions in software development efforts.

7.0 REFERENCES/FURTHER READING

Garg, P.K.; Mi, P.; Pham, T.; Scacchi, W. & Thunquest, G. (1994). The SMART Approach for Software Process Engineering, *Proc. 16th. Intern. Conf. Software Engineering*, 341 – 345.

Sajan, M. (2007). *Software Engineering*. (Rev. ed.) New Delhi: S. Chand & Company Ltd., , pp. 1-5, 27-36, 138-141, 152-158, 2881-187.

Scacchi, W. & Mi, P. (1997). Process Life Cycle Engineering: A Knowledge-Based Approach and Environment. *Intelligent Systems in Accounting, Finance, and Management*, 6(1):83-107.

Scacchi, W. & Noll, J. (1997). Process-Driven Intranets: Life Cycle Support for Process Reengineering. *IEEE Internet Computing*, 1(5):42-49.

Scacchi, W. (1984). Managing Software Engineering Projects: A Social Analysis, *IEEE Trans. Software Engineering*, SE-10,1, 49-59, January.

Scacchi, W. (2000). Understanding Software Process Redesign using Modeling, Analysis and Simulation. *Software Process -- Improvement and Practice* 5(2/3):183-195.

Selby, R.W.; Basili, V.R. & Baker, T. (1985). *CLEANROOM Software Development*. New York: Empirical Press.

Wood, J. & Silver, D. (1995). *Joint Application Development*. New York: Wiley and Sons, Inc.

Wood, M. & Sommerville, I. (1988). A Knowledge-Based Software Components Catalogue, *Software Engineering Environments*, Ellis Horwood, P. Brererton (Ed.). Chichester, England, pp. 116-131.

Yu, E.S.K. & Mylopoulos, J. (1994). Understanding "Why" in Software Process Modelling, Analysis, and Design, *Proc. 16th. Intern. Conf. Software Engineering*, 159 -168.

UNIT 2 THE WATERFALL MODEL AND THE BUILD-AND-FIX MODEL

CONTENTS

1.0 INTRODUCTION

Building good functional, maintainable software is a challenging adventure. In spite of this, good and functional software is of paramount importance to almost every human endeavour today. Engineering approach to software development has brought a new dawn to the software world. In the earliest days of software development, code was written and then debugged. There was no formal design or analysis. This code and debug approach rapidly became less than optimal as complex software systems were required. Since the approach to developing complex hardware systems was well understood, it provided a model for developing software. This brought about most of the software engineering models. The waterfall model was derived from engineering models to put order in the development of large software products (Sajan Mathew, 2001). The waterfall model consists of several stages/phases which are processed in a linear fashion. Code-and-fix model is very similar to the waterfall model but in this case, the entire product is built and then delivered to the client. The client points out what has to be changed and the developer affects the changes to the satisfaction of the client. In this unit, we shall consider basically the waterfall model and the code-and-fix model.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain the waterfall model of software life cycle
- list the phases involved in waterfall model
- discuss the advantages and disadvantages of the waterfall model
- explain the build-and-fix model
- discuss the advantages and disadvantages of the build-and-fix model
- explain when the build-and-fix model is appropriate to use.

3.0 MAIN CONTENT

3.1 Description of Waterfall Model

The waterfall model is an approach to software development that emphasizes completing a phase of the development before proceeding to the next phase. The waterfall model was derived from engineering models to put some order in the development of large software product. It consists of different stages which are processed in a linear fashion. In conjunction with certain phase completions, a baseline is established that "freezes" the products of the development at that point. If a need is identified to change these products, a formal change process is followed to make the change. The graphic representation of these phases in software development resembles the downward flow of a waterfall. In the waterfall model, no phase is started until the result of the previous phase has been carefully verified.

3.1.1 Phases Involved in Waterfall Model and how it Works

The different stages/phases involved are:

- Requirements stage/ phase
- Specification stage/phase
- Planning stage/phase
- Design stage/phase
- Implementation stage/phase
- Integration stage/phase
- Operations stage/phase
- Retirement.

The life cycle phases of the waterfall model are shown in the figure below:

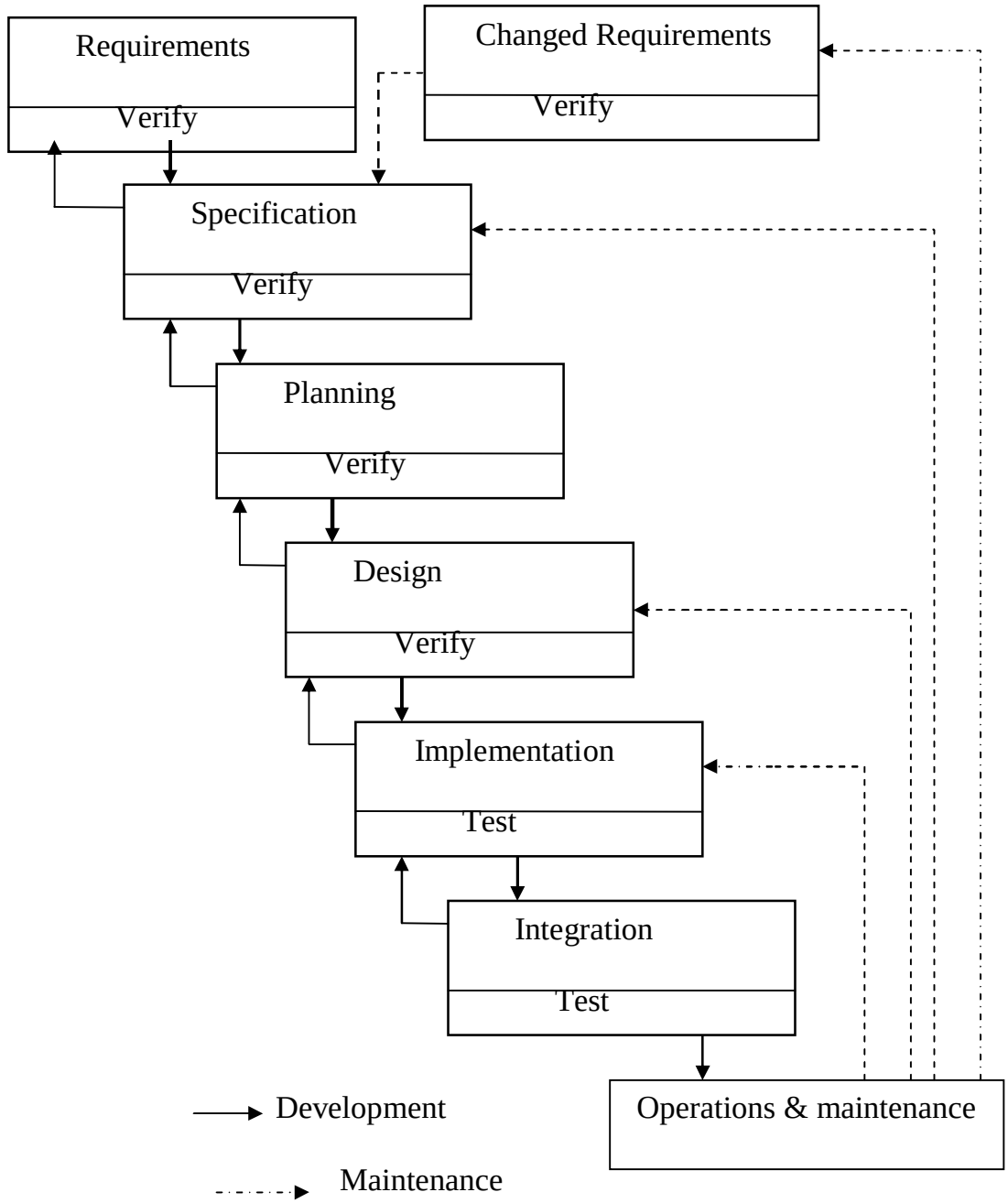

Fig. 2.1: Stephen, (2001). Life Cycle Phases of the Waterfall Model.

Let us discuss in details, each of the stages/phases shown in the figure above.

- **Requirement Stage/Phase**: In this phase (first phase), members of the development team meet with the client (customer) and members of the client organisation. Here, the development team aims at determining exactly what the client's needs are. If the client currently has a manual system in place that is to be replaced by the proposed automated products, the developers will obtain a detailed understanding of the manual system and why it is considered inadequate. To help developers gather the needed understanding, developers will have to interview appropriate members of the client's organisation. Developers should also study relevant documents (organisational charts, procedure manuals, operational manuals etc). The findings of the requirement team are presented in the form of a document. The document will be thoroughly checked (refined) before proceeding to the next stage.
- **Specification Stage/Phase**: Specification is the second stage. At this stage, the development team having ascertained the client's real needs in form of a requirement document draws the specification. A specification document is a written document that states exactly what the proposed system (product) is to do. While drawing the specification document, the developers may obtain more understanding/insights into the client's requirement. If this happens, requirement document will be revisited and amended to reflect the latest discovery. The carefully drawn specification document is presented to the client for checking. The client goes through the specification document and certifies it ok if all is well to him. On the other hand, if the client's notices any error, the correction must be affected before moving on to the next stage.
- **Planning Stage/Phase**: This is the third phase of the waterfall model. Once the specification document has been approved by the client, the developers draw up the software project management plan (SPMP). The SPMP usually contains the description of what is to be done, how long it will take, how much it will cost, the human and computer resources that will be needed and a detailed timetable showing who will do what and when. The overall cost of the project is also made known to the client. If the client accepts the above specifications, the design begins.
- **The Design Stage/Phase**: This phase consists of two steps: architectural and detailed designs. During the architectural design step, the product is broken down into modules. During the detailed design, each module in turn is designed. The function

each module is to carry out what is determined, and what algorithm and data structures are to be used. If any fault in the plan or specification document is detected, the feedback loops takes care of the corrections. This takes us to the implementation stage.

- **Implementation Stage/Phase:** At this stage, each module in turn is coded in the particular programming language specified in the contract. The coded modules are then tested. While this is being done, if any design or specification error is detected, the feedback loops are followed and the fault corrected.
- **Integration Stage/Phase:** At this stage, the different modules are brought together (linked) to form a complete system (product). The source code together with all documentations is now tested. When the developers are convinced that the software satisfies its specification document and that all the documentation is correct and complete, the software product is handed over to the client for acceptance testing. Once the client tests and certifies that the product does what it is supposed to do, he/she signs off the product.
- **Operation and Maintenance Stage/Phase:** Once the client sign off the product, the product is then installed on the client's computers and it goes into operation mode. From this time on, any changes made to the product whether to the source code or to the documentation are maintenance.
- **Maintenance** could be a corrective maintenance (where a fault in the code or documentation is detected and corrected), or an enhancement maintenance (maintenance that involves a change in the requirement). The rightmost dashed line in the figure shows what happens when the requirements are change and these changes in turn trigger changes in the specification document, design document, and implementation of the product.

3.2.1 Advantages and Disadvantages of the Waterfall Model

Advantages of the waterfall model include, among others, the following:

a. It enforces a disciplined engineering approach
b. Verification is done after each phase
c. Documentation produced can reduce maintenance cost
d. Feedback loops help to correct faults immediately.

Some disadvantages of the waterfall model include:

a. Documentation slows down the process
b. It lacks flexibility
c. Generate few visible sign of progress until the very end.

3.1.3 Where to Use the Waterfall Model

Because of the strengths and the weaknesses shown above, the application of the waterfall model should be in situations where the requirements and the implementation of those requirements are very well understood and also when the software to be produced is large.

For example, if a company has experience in building accounting systems, I/O controllers, or compilers, then building another such product based on the existing designs is best managed with the waterfall model.

SELF-ASSESSMENT EXERCISE 1

1. Explain what happens in the specification phase of the waterfall model
2. Outline any three strengths of the waterfall model.
3. What is corrective maintenance?

3.2 Description of the Build-and-Fix Model

Build-and-fix model is a software development model where the entire software products are built and delivered to the client. The client points out what has to be changed and changes are made until the client is satisfied (Stephen. 2003). The product then goes into operation mode.

3.2.1 Phases Involved in Build-and-Fix Model and How it Works

The figure below illustrates the build-and-fix model:

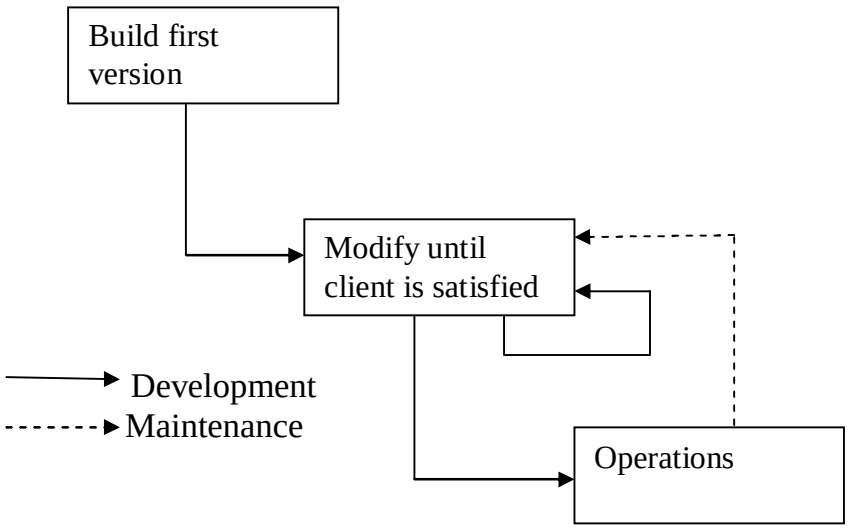

Fig. 2.2: Stephen, (2001). Life Cycle Phases of Build-and-Fix Model.

The build-and-fix model does not follow series of stages/phases religiously like the waterfall model. Here, the specification phase, the planning phase, the design phases are all omitted. The development team immediately writes the code and delivers the product to the client who now tests it and points out things to be changed. This means that there is no coherent and cohesive overall structure, and maintenance becomes a big problem.

3.2.2 Advantages and Disadvantages of the Build-and-Fix Model

Advantages include, among others, the following:

a. It provides immediate feedback to developers: This shows immediate signs of progress
b. Removes planning/design/documentation overhead.

Disadvantages

a. Much time is spent debugging
b. Does not promote documentation and therefore produced software is costlier to maintain
c. Design changes cost much more after coding has started

After the sequences of changes, the codes structure becomes so messy that subsequent corrections become harder to apply and results become less reliable.

3.2.3 When to Use the Build-and-Fix Model

The build-and-fix model should be used when the product is small and there is no possibility of the product ever having to be maintained in the future. For example, if a student is to write a 25-50 line home work problem to solve a particular computational need (e.g. program to keep track of student's record in the class), then it would be a waste of time to specify, plan and design the development effort.

SELF-ASSESSMENT EXERCISE 2

1. When is it most appropriate to apply the build-and-fix model?
2. The build-and-fix model does not follow the series of stage like the waterfall model. TRUE or FALSE
3. Why is maintenance a big problem with the build-and-fix model approach?

4.0 CONCLUSION

Two life cycle models were discussed in this unit, namely; waterfall and the build-and-fix models. The waterfall model was shown to be a very good model that brings about success in most software development effort. This success was attributed to the fact that waterfall is document based and follows engineering process rigidly. The build-and-fix model should almost never to be used, because changing a product in operation mode costs is much. However, both waterfall and build-and fix models has a number of advantages as well as some disadvantages.

7.0 SUMMARY

There were several issues discussed in this unit which can be summarised as follows:

- the waterfall model was derived from engineering model and as such series of stages starting from the requirement phase down to operation and maintenance phase
- the waterfall model creates room for feedback at every phase. this gives room for error corrections
- the application of the waterfall model should be in situations where the requirements and the implementation are very well understood and also when the software to be produced is large

There are several advantages of the waterfall model

- The build-and-fix model does not follow series of phases; rather it is an approach where the entire software product is built and delivers to the clients who points out changes to be made
- The build-and-fix model should be used when the product is small and there is no possibility of the product ever having to be maintained in the future.

8.0 TUTOR-MARKED ASSIGNMENT

1. Explain the need for the feedback loops in the waterfall model.
2. The term “operation mode” and “maintenance phase” both refer to the time period after software have been delivered to the client. What is the essential difference between the two terms?
3. Describe the sort of product that you feel that will be an ideal application for the waterfall model.
4. Why is the build-and-fix model not appropriate for building large software product?
5. List and explain any three disadvantages of the waterfall model
6. What is the major risk in using the waterfall model? How can this risk be resolved?

7.0 REFERENCES/FURTHER READING

Boehm, B. (1987). A Spiral Model of Software Development and Enhancement, *Computer*, 20(9), 61- 72.

Chatters, B.W.; Lehman, M.M.; Ramil, J.F. & Werwick, P. (1987). Modeling a Software Evolution Process: *A Long-Term Case Study, Software Process-Improvement and Practice*, 5(2-3), 91-102, 2000. 4, 5, 19-25.

Garg, P.K.; Mi, P.; Pham, T.; Scacchi, W. & Thunquest, G. (1994). The SMART Approach for Software Process Engineering, *Proc. 16th. Intern. Conf. Software Engineering*, 341 - 345.

Mili, A.; Desharnais, J. & Gagne, J.R. (1986). Formal Models of Stepwise Refinement of Programs, *ACM Computing Surveys*, 18, 3, 231-276.

Mills, H.D.; Dyer, M. & Linger, R.C. (1970). Cleanroom Software Engineering, *IEEE Software*, Royce, W. W. (Ed.). Managing the Development of Large Software Systems, *Proc. 9th. Intern. Conf. of Software Engineering*, ,IEEE Computer Society, 1987 ,328-338 Originally published in Proc.WESCON.

Moore, J.W.; DeWeese, P.R. & Rilling, D. (1997). "U. S. Software Life Cycle Process Standards," *Crosstalk: The DoD Journal of Software Engineering*, 10:7, July.

Neighbors, J. (1984). The Draco Approach to Constructing Software from Reusable Components, *IEEE Trans. Software Engineering*, 10, 5, 564-574.

Sajan, M. (2007). *Software Engineering* (Rev. ed.). New Delhi: S. Chand & Company Ltd., pp. 1-5, 27-36, 138-141, 152-158, 2881-187.

Scacchi, W. (1984). Managing Software Engineering Projects: A Social Analysis, *IEEE Trans. Software Engineering*, SE-10, 1, 49-59, January.

Stephen, S. (2001). *Software Engineering*. ISBN-0-256-LL454-4.

UNIT 3 THE RAPID PROTOTYPING MODEL AND THE SPIRAL MODEL

CONTENTS

1.0 INTRODUCTION

In unit one, we discussed two important software engineering development models; the build-and-fix and the waterfall models. You learnt that it was imperative to have software engineering model which serves as an outline for developing good, functional and maintainable software. In this unit, we shall examine another two important software engineering models: the rapid prototyping model and the spiral model. In this unit, we shall consider basically the waterfall model and the code-and-fix model. The rapid prototyping model is a model where software developers after gathering information in requirement process, builds a reduced version of the product in question to enable them gather the real and exact need of the customer. The spiral model on the other hand, is a model that emphasizes risk analysis at every stage of the development process. Basic features of these two software engineering models shall be discussed in details in this unit. All the issues discussed were presented in a simple and clear manner.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- describe the rapid prototype model
- discuss the basic features of the rapid prototyping model
- explain when to use the rapid prototyping model
- describe the spiral model and all the phases involved in it
- explain when it is best to use the spiral model
- discuss the advantages and disadvantages of the spiral model.

3.0 MAIN CONTENT

3.1 Description of Rapid Prototyping Model

Rapid prototyping model is a type of model where the development team gathers information as in waterfall model, and presents their findings in form of a document to the client. After this, they proceed to build a rapid prototype. A prototype is a reduced functionality or a limited performance version of a software system. The key points are to build code rapidly that will show the client the inputs and the outputs in order for the client to say either, "yes that's exactly what I want" or "No what I really want is something else."

3.1.1 Phases Involved in Rapid Prototyping and how it Works

The phases involved in the rapid prototyping model are similar to that of waterfall model but they are not the same. In rapid prototyping model, the phases start with "rapid prototype". What happens is that the requirement team gathers information as before and presents their findings in form of a document to the client. After presenting their finding to the client, they proceed to build a rapid prototype. The prototype is built as quickly as possible to speed up the software development process. The sole purpose of the prototype is to capture the client's need; once this has been determined, the rapid prototype is effectively discarded. The development team uses information captured from the client to draw up the specification document. From here, the process continues as it is in the waterfall model. Another difference in the rapid prototyping model is that the feedback loop is missing. This is because the rapid prototype built is normally used to capture all the errors/faults to be corrected.

The rapid prototyping model can therefore be represented as shown in Figure 3.1below:

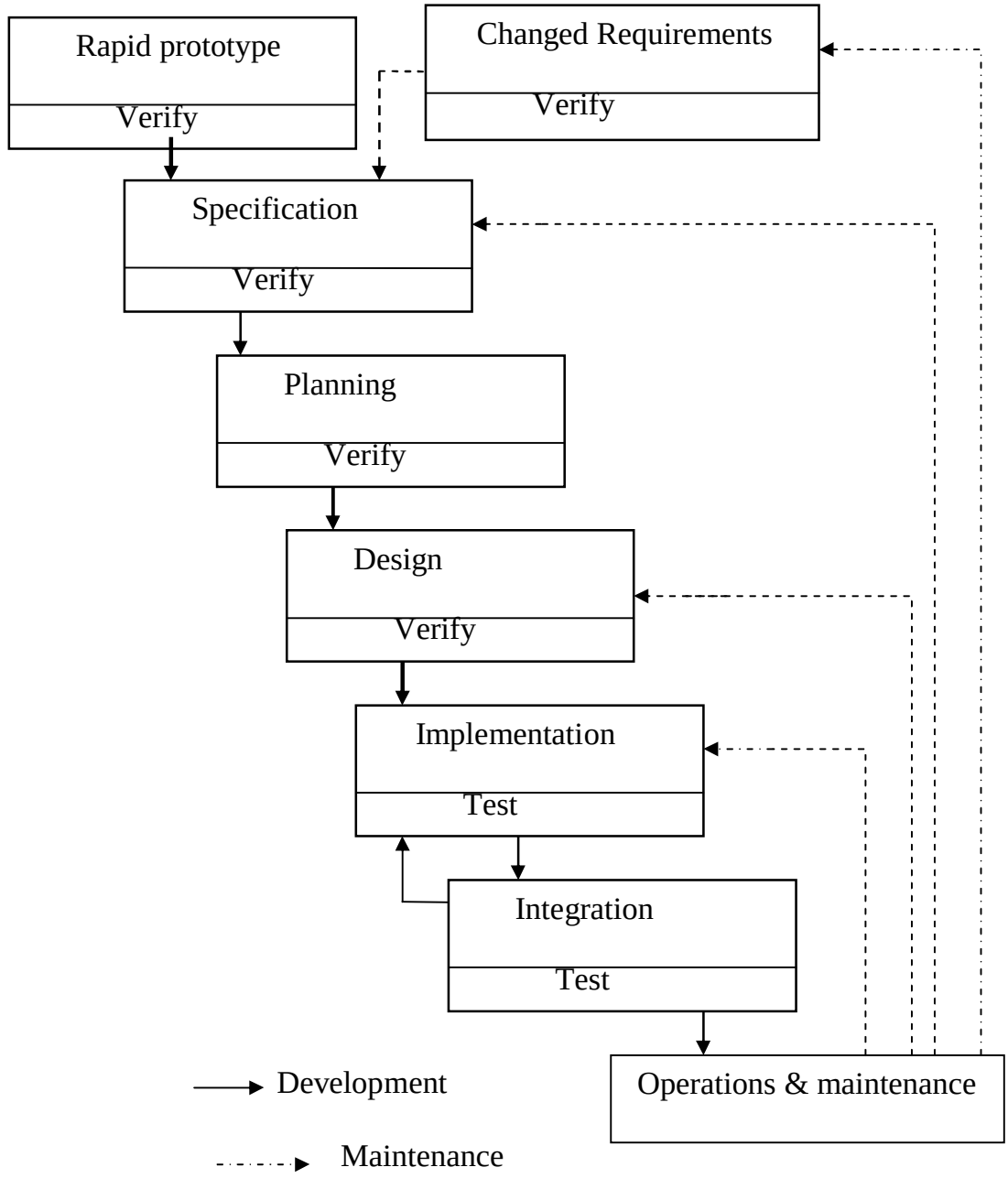

Fig. 3.1: Stephen, (2001). Life Cycle Phases of Rapid Prototyping Model

Rapid prototype model is used to overcome issues related to understanding and capturing of user's requirements. An essential aspect of rapid prototype is embedded in the word "rapid". The developer should endeavour to construct the prototype as quickly as possible to speed up the software development process.

3.1.2 Advantages and Disadvantages of the Rapid Prototyping Model

Advantages include

a. developers can benefit from the experience gained from building the prototype and apply this experience towards building a better product
b. gathers client's feedback early in the process to avoid costly re-design later
c. early functionality
d. provides a process to perfect the requirements definition.
e. provides risk control
f. documentation focuses on the end product not the evolution of the product.

Disadvantages include

a. adding the rapid prototype will lengthen the requirements phase
b. depending on the type of software system, it may not be possible to build a meaningful prototype without considerable effort
c. less applicable to existing systems than to new, original development.

3.1.3 When to Use the Rapid Prototyping Model

Rapid Prototyping model is very useful to demonstrate technical feasibility when the technical risk is high. It can also be used to better understand and extract user requirements. In either case, the goal is to limit cost by understanding the problem before committing more resources. Prototyping can always be used with the analysis and design portions of objected-oriented model.

3.2 Description of the Spiral Model

The spiral model of software development and evolution represents a risk-driven approach to software process analysis and structuring (Boehm, 1987; Boehm *et al.*, 1998). The main objective here is to determine the risks involved in developing a particular software and then to resolve each risk in turn. For example, a typical risk is that the delivered software may not satisfy the client's real needs. This type of risk can be taken care of by building a rapid prototype and have the client experiment on it. A project is terminated if at any time it becomes clear that the product to be built will not be cost effective. The spiral model uses the best waterfall model and the rapid prototyping model.

3.2.1 Phases involved in Spiral Model and How it Works

The spiral model's development is divided into four quadrants; planning, risk analysis, engineering, and client evaluation quadrants. Starting at the planning quadrant (innermost quadrant), you move clockwise direction through the quadrants spiraling outwards until you have completed a final product. At each interaction cycle, a progressively more complete version of the prototype is built. At each client evaluation, the engineering work is evaluated and suggestions for modifications are made. The project is terminated if at risk analysis, it is determined that the risks are too much. The figure below illustrates the spiral model.

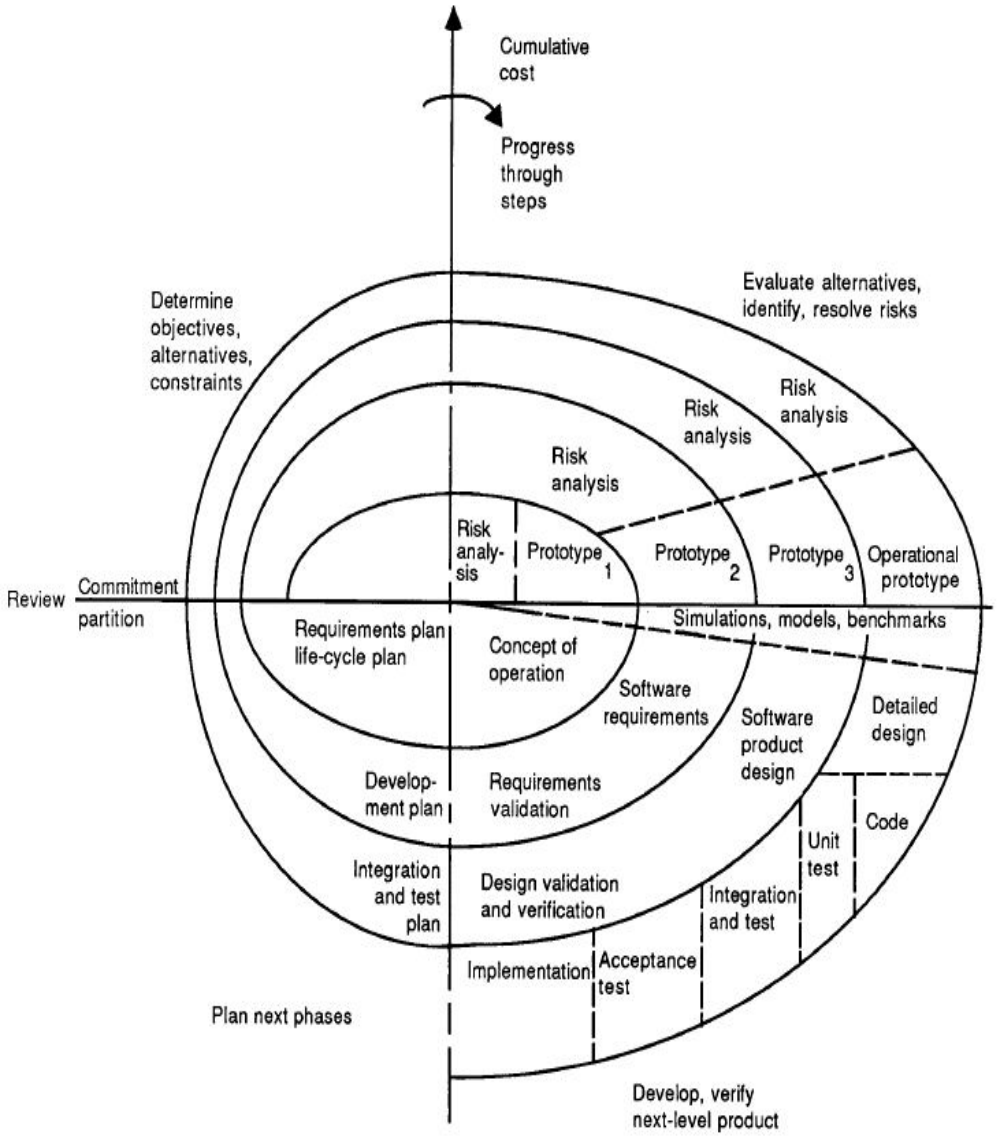

Fig. 3.2: Boehm, (1987). The Spiral Model Diagram

This approach, developed by Barry Boehm, incorporates elements of specification-driven, prototype-driven process methods, together with the classic software life cycle.

System development in this model therefore spirals out only so far as needed according to the risk that must be managed. Risk analysis, which seeks to identify situations that might cause a development effort to fail or go over budget/schedule, occurs during each spiral cycle. In each cycle, it represents roughly the same amount of angular displacement, while the displaced sweep volume denotes increasing levels of effort required for risk analysis.

3.2.3 Advantages and Disadvantages of Spiral Model

Advantages include:

a. spiral model combines the best features of the waterfall and prototyping models
b. it address risks associated with software development
c. it enables the developer to apply prototyping at any stage in the evolution of the software product
d. it control costs and risk through prototyping
e. allows for work force specialisation
f. facilitates allocation of resources
g. does not require a complete set of requirements at the onset.

Disadvantages include:

a. the overall cost is comparatively high
b. it is complicated
c. it is unstable for small projects were risks are modest
d. requires considerable risk assessment expertise

3.2.3 When to Use the Spiral Model

Spiral model is particularly useful in ADE (Aerospace, Defense and Engineering) projects, because they are risky in nature. They tend to use mature technology and to work well-known problems. Spiral model is also applicable to many business applications, especially those for which success is not guaranteed or the applications require much computation, such as in decision support systems.

SELF-ASSESSMENT EXERCISE

1. List the four quadrants of the spiral model.
2. What does the word “spiral” connotes in this model.
3. If after the risk analysis of any stage, it is determined that the risks are too much, what happens?

4.0 CONCLUSION

This unit highlights another two software engineering development models, namely; the rapid prototyping and the spiral models. The rapid prototyping model ensures that the delivered product satisfies the client’s real needs. This is because the model gives client room to interact with an implementation of a portion of the functionality of the intended product. The spiral model was also discussed. The spiral model is a risk-driven model. A proper risk analysis is carried out at every stage of the development process and the project terminated at any time the risk discovered seems to be too much. The spiral model is made up of four quadrants and a new prototype is built at each quadrant to access the clients view. When to use the rapid prototype as well as when to use the spiral models in software developments were also discussed.

6.0 SUMMARY

In this unit, you have learnt that:

- a prototype is a reduced functionality version of the product in question
- the sole purpose of the prototype is to capture the client’s need
- the rapid prototyping model makes up for some shortcomings of the waterfall model
- rapid prototyping model is very useful to demonstrate technical feasibility when the technical risk is high
- rapid prototyping has many strengths as well as some weaknesses
- the spiral model makes room for proper risk analysis at every stage of the process development
- a prototype is built at every four quadrants of the spiral model
- spiral model is particularly useful in ADE (Aerospace, Defense and Engineering) projects, because they are risky in nature.

7.0 TUTOR-MARKED ASSIGNMENT

1. Explain why you think a new prototype is necessary at every quadrant of the spiral model.
2. What is meant by “risk analysis?”
3. Why is the rapid prototyping model probably not being employed?
4. When should developers use the spiral model
5. Explain the four quadrants of the spiral model.

7.0 REFERENCES/FURTHER READING

Boehm, B. (1987). A Spiral Model of Software Development and Enhancement, *Computer*, 20(9), 61- 72.

Boehm, B. W. (1981). *Software Engineering Economics*. New Jersey, Englewood Cliffs: Prentice-Hall, pp. 20.

Boehm, B.; Egyed, A.; Kwan, J.; Port, D.; Shah, A. & Madachy, R. (1998). Using the WinWin Spiral Model: A Case Study. *Computer*, 31(7), 33-44.

Chatters, B.W.; Lehman, M.M.; Ramil, J.F. & Werwick, P. (1987). Modeling a Software Evolution Process: *A Long-Term Case Study, Software Process-Improvement and Practice*, 5(2-3), 91-102, 2000. 4, 5, 19-25.

Garg, P.K.; Mi, P.; Pham, T.; Scacchi, W. & Thunquest, G. (1994). The SMART Approach for Software Process Engineering, *Proc. 16th. Intern. Conf. Software Engineering*, 341 - 345.

Moore, J.W.; DeWeese, P.R. & Rilling, D. (1997). "U. S. Software Life Cycle Process Standards". *Crosstalk: The DoD Journal of Software Engineering*, 10:7.

Mili, A.; Desharnais, J. & Gagne, J.R. (1986). Formal Models of Stepwise Refinement of Programs, *ACM Computing Surveys*, 18, 3, 231-276.

Mills, H.D.; Dyer, M. & Linger, R.C. (1987). Cleanroom Software Engineering, *IEEE Software*, Royce, W. W., Managing the Development of Large Software Systems, *Proc. 9th. Intern. Conf. of Software Engineering, IEEE Computer Society*. pp.328-338 Originally published in Proc.WESCON, 1970.

Neighbors, J. (1984). The Draco Approach to Constructing Software from Reusable Components, *IEEE Trans. Software Engineering*, 10, 5, 564-574.

Sajan, M. *Software Engineering* (Rev. ed.). New Delhi: S. Chand & Company Ltd., pp. 1-5, 27-36, 138-141, 152-158, 2881-187.

MODULE 3 SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

UNIT 1 OVERVIEW OF THE PROCESS INVOLVED

CONTENTS

1.0 INTRODUCTION

Software refers to the aggregates of computer programs and the appropriate data needed for their operations with proper documentation. Development of good, functional and maintainable software does not evolve from harp-hazard activities. Software developers usually employ a well defined algorithm in order to produce standard software. In this

unit, we shall take an overview of the systematic procedure involved in software development. This step-by-step affair that cuts across many activity stages of software development is what is known as Software Development Life Cycle (SDLC). Some calls it "system life cycle" while others simply refers to it as phases in software development. Though the exact terminology varies from reference to reference, the essential concepts remain the same.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain what is SDLC
- identify all the stages involved in software development: planning, construction and maintenance.
- list the basic factors that must be considered in software development
- explain the importance of having a standard in software development
- explain the importance of software testing.

3.0 MAIN CONTENT

3.1 Software Project Management

Software project management is the set of activities involved in planning, controlling and directing the software project. Before any software development team or organisation will be able to develop workable software, the first thing to do is to make a comprehensive plan of the software project before the actual construction. In most situations, new feasible systems replace or supplement existing information processing mechanisms whether they were previously automated, manual, or informal. This section looks at some of the basic issues in software project planning.

3.1.1 Initial Problem Statement

"Necessity is the mother of invention". There should be a clear computational need that will necessitate the development of software. The initial problem statement specifies in definite terms the computational need at hand. It normally gives a clear picture of what the physical system is. A proper survey of the existing system is usually done to enable an organisation identify clearly the coverage of the new system and hence ensure a well articulated organisational need.

3.1.2 Feasibility Study

The next step in planning is feasibility study. This is a detailed analysis of the whole problem area specified in the initial problem statement. The feasibility study seeks to determine whether the computational effort envisaged is feasible (possible). The feasibility study ensures that the project is possible in terms of budget, expertise, and technology.

- **Budget:** The budget aspect seeks to find out if there are financial constraints that will make the realisation of the software project impossible. It extends to examining an estimate of the overall cost of the project
- **Expertise:** The expertise aspect of the feasibility study deals with the technical know-how needed to make the project feasible. It seeks to answer the questions such as; are there some experts who specialise in that area?
- **Technology:** The technology aspect of the feasibility study examines the enabling technology that will make the realisation of the software project feasible (possible). You need to check for example, if there are programming languages available that can be used to develop such software project etc.

Other issues like policy statements of the firm or of the government that may make such computerisation effort difficult or impossible are also taken into considerations during the feasibility study.

3.1.3 Requirement Analysis

Requirement analysis is the stage when the things that are required (needed) to make the software project a success are identified. Requirement analysis identifies the problems the new software is expected to solve. It also explores the new system's capabilities, its desired performance characteristics and the resource infrastructure needed to support system operation and maintenance. According to Inyiama H. & Alo U (2009), requirement analysis entails for example, knowing whether personal computers should be used or if the situation requires minicomputers. If personal computers are to be used, will there exist as standalone computers? Or is networking required and what network topology will be preferred? The requirement analysis helps to clarify the overall requirement of the project. The requirement analysis report provides the basis for the next stage in the development.

3.1.4 Software Maintenance

Another phase in software project management is software maintenance. Any addition, correction, or subtraction made to software after delivery is tagged maintenance. Maintenance in real sense, deals with adjusting the installed packages to cope with on-going changes in content and environment. Maintenance helps to sustain the useful operation of a system in its host/target environment by providing requested functional enhancement repairs, performance, improvement and conversions.

SELF-ASSESSMENT EXERCISE 1

1. Why is feasibility study very important in any project development?
2. What does the **initial problem statement** specify?
3. Explain software maintenance in your own understanding.

3.2 Software Project Construction

The construction activities are activities that are directly related to the development of software. Such activities include, among others, systems analysis, specification, design, coding, testing, integration etc.

3.2.1 System Analysis and Specification

Whereas, the system analysis describes an environment which will place constraints on the system to be developed, the system analysis seeks to really understand the technical details involved in the process within the scope of the computerisation. System analysis involves so many practical steps which include, among others, data gathering, input form design, database file, data flow diagram design etc. In subsequent unit, we shall discus in details some of these practical steps. In line with these practical steps, system specification is drawn. System specification gives detailed description of what the system should look like, in terms of function, required effect and technology. This detailed description serves as a basis for design of the new system according to the data produced from the above analysis.

3.2.2 System Design

Based on the specification drawn above, system design describes an overview of the new system in terms of interface, structure and technology Sajan, M. (2003). System design depicts the program logic. In most cases, false codes popularly known as pseudo codes are used to specify and explain the program logic contained in the specification. It is at this stage that projects are normally broken down into subsystems

and modules. The mechanism to transfer control from one subsystem to another is also spelt out. Various techniques employed in system design were discussed in software engineering 1.

3.2.3 Software Coding

Software coding implies translating the design made in 3.2.2 above to a computer understandable format using a particular programming language. It should be noted that whereas design is done without making use of any particular programming language, coding is done with a particular high level programming language specified in the contract. The actual programming must follow a structured programming constructs to arrive at functional maintainable software. Some of these programming constructs shall be discussed in subsequent units.

SELF-ASSESSMENT EXERCISE 2

What question(s) does systems analysis seeks to answer?

3.3 Software Installation and Management

Once system test has been carried out and the system certified ok, preparation begins to take place the new system into operation using the design specifications to the new system. This section presents different ways of changing over from the old system to the newly developed system.

3.3.1 Software Testing and Integration

Before deployment and commissioning of new developed system, a test run of the system is carried out to remove errors (bugs). Each of the modules is tested. The subsystems in turn are also tested. After the module and subsystem testing, the individual components are integrated (linked) to form one complete system and a system test carried on it. Software testing and integration affirms and sustains the overall integrity of the software, architectural configuration through verifying the consistency and completeness of implemented modules. It also verifies the resource interface and interconnections against their specifications and validates the performance of the system against its requirements.

The software codes produced in software coding are tested in a planned way. Typically, a test plan is drawn and test data used to carry out the test following the test plan.

NOTE: A test plan must be comprehensive and must be drawn from the beginning of the project. It should also be noted that unit test is first

carried out. This is when the component units/modules are individually tested with a prepared set of test data and all bugs corrected. After this unit test, system test is carried out. System test is a test on the integrated (complete) system using actual data (main data the software is built to handle).

3.3.2 Software Conversion

Conversion is simply a change over from the manual or less automated existing system to the newly developed system. It is however expedient to employ a systematic change over from the old process to the newly automated process. Conversion strategies commonly used in installation include among others the following.

3.3.2.1 Abrupt Conversion (Cut-Over)

This is a conversion strategy where on a specific date chosen, the old system is terminated and the new system is placed into operation. Jeffery L. et al (2001). This approach of software conversion may be a high risk approach since they may still be major problems that will not be discovered until the new system has been in operation for some time.

3.3.2.2 Parallel Conversion

Here, both the old and the new system are operated simultaneously for some time period. Jeffery L. et al (2001). This approach ensures that all major problems in the new system are discovered and solved before the old system is discarded. The final cut-over may be either abrupt or gradual as portions of the new system are deemed adequate.

3.3.2.3 Staged Conversion

This is the type of conversion based on the version concept. Jeffery L. et al (2001). Here, each successive version of the new system is converted as it is developed. The conversion of each version may be abrupt, parallel or location.

3.3.2.4 Location Conversion

This type of conversion applies when the same system will be used in many different geographical areas. Here, location takes place at one location first using either abrupt or parallel conversion. When the new system is certified by the site, it will be adopted by other sites. The cut-over now may be abrupt since other sites have certified the newly built product ok.

3.3.3 Software Documentations

Documentation is done at each stage of the software development life cycle. However, at this stage, one thinks in terms of committing the entire documentation into an automated database for future reference.

Documentation details development descriptions and user guides, all in form suitable for dissemination and system support. These documentations characterise what the developed system is supposed to do, how it does it, how it was developed, how it was put together, validated and how to install use and maintain it. Software documentations are often a primary medium for communication between developers, users, maintainers; thus each of these group can benefit from automated mechanism that allow them to browse, query, retrieve and selectively print out documents (Garg and Scacchi 1989;1990).

3.4 The Need for a Standard

Each of the stages discussed above requires a professional standard document in which the correct way of doing it is clearly specified. Having established a professional stand, a quality control group is always set up to ensure that the standards are been maintained and followed. Establishment of a professional standard document helps to ensure that every work done in the computing world/environment meets up with the quality standard. Again, since any member of the software development team may be sacked, retire, cease to function for any reason or even die. It will be difficult for anyone else to understand what he/she has done so far or continue where he stopped if there is no standard. Hence, it is expedient to discuss the issue of standards before a software contract is signed. One should cause the contractor to adopt your own standard if possible.

SELF-ASSESSMENT EXERCISE 3

1. Differentiate between unit testing and system testing.
2. What should be the coverage areas of software documentation?

4.0 CONCLUSION

Software development life cycle (SDLC) means the different phases involved in developing good, functional, maintainable software. You learnt that the three supportive activities involved in the software development life cycle are: software management, software construction, and software quality assurance. The management aspect of software development includes set of activities involved in planning,

controlling and directing software project. Furthermore, you learned that software construction involves all activities that analyse requirements, develops design, writes code and structures databases. On the other hand, you were made to understand that software quality assurance involves activities such as software testing and integration, software conversion, documentation etc. that make sure the management and construction efforts put forth result in a product that meets all of its requirements. We also have shown why it is expedient to have a standard guiding each of these phases of software development.

5.0 SUMMARY

In this unit, you have learnt that:

- software development life cycle can be divided into three: management, construction and quality assurance
- feasibility study seeks to determine whether computerisation effort is possible in terms of budget, expertise and technology
- the requirement analysis is a stage when all that needed to make the project a success are analysed
- system analysis seeks to understand the technical details involved in the process with the scope of computerisation
- specification document gives a detailed description of what the system should look like in terms of function, effect and technology
- software design describes an overview of the new system in terms of interface, structure and technology
- coding is the act of translating the design into a computer understandable format using a particular programming language
- testing implies test-running the newly developed software product to check if it meets the targeted functions while integration deals with linking the different modules and subsystems to form one entity
- software conversion means changing over from the already existing manual or less automated system to the newly developed software
- maintenance is an addition or subtraction made to the software product after delivery
- there is a need to have a common stand (standard) in software development process.

6.0 TUTOR-MARKED ASSIGNMENT

1(a) Explain software conversion.
(b) Explain the different types of software conversion giving vivid example(s) on each of the conversion types.

2. Why is there need for systematic approach in the development of software project?
3. Where there are no standard, what happens?
4. Explain the three basic aspect of the feasibility study. Find out some other aspects that are also part of the feasibility study.
5. Why is it necessary to plan a software project before its construction.
6. In your own words, explain any three phase in software development life cycle.

7.0 REFERENCES/FURTHER READING

Boehm, B. (1987). A Spiral Model of Software Development and Enhancement, *Computer*, 20(9), 61- 72.

Boehm, B. W. (1981). *Software Engineering Economics.* Englewood Cliffs, New Jersey: 20.

Boehm, B.; Egyed, A.; Kwan, J. & Port, D.; *et al. (*1998). Using the WinWin Spiral Model: A Case Study. *Computer*, 31(7), 33-44.

Chatters, B.W.; Lehman, M.M.; Ramil, J.F. & Werwick, P. Modeling a Software Evolution Process: A Long-Term Case Study. *Software Process-Improvement and Practice*, 5(2-3), 91-102, 2000. 4, 5, 19-25, 1987.

Garg, P.K.; P. Mi; Pham, T.; Scacchi, W. & Thunquest, G. The SMART Approach for Software Process Engineering, *Proc. 16th. Intern. Conf. Software Engineering*, 341 - 345, 1994.

Jeffery, L.; *et al.* (2001). *System Analysis and Design Methods* (5th ed.). McGraw-Hill Higher Education, pp. 163-210.

Mili, A.; Desharnais, J. & Gagne, J.R. Formal Models of Stepwise Refinement of Programs, *ACM Computing Surveys*, 18, 3, 231-276, 1986.

Mills, H.D.; Dyer M. & Linger, R.C. Cleanroom Software Engineering, *IEEE Software*, Royce, W. W., Managing the Development of Large Software Systems, *Proc. 9th. Intern. Conf.*

of Software Engineering, IEEE Computer Society, 1987 ,328-338 Originally published in Proc.WESCON, 1970.

Moore, J.W.; De-Weese, P.R. & Rilling, D. "U. S. Software Life Cycle Process Standards," *Crosstalk: The DoD Journal of Software Engineering*, 10:7, July 1997.

Sajan, M. (2007). *Software Engineering* (Rev. ed.). Ram Naga, New Delhi: S. Chand & Company Ltd., pp. 1-5, 27-36, 138-141, 152-158, 2881-187.

Stephen, S. (2001). *Software Engineering.* ISBN-0-256-LL454-4.

UNIT 2 SYSTEMS ANALYSES AND SOFTWARE REQUIREMENT SPECIFICATION

CONTENTS

1.0 INTRODUCTION

In unit 1, you learnt an overview of software development life cycle (SDLC). In this unit, you will learn how to carry out a practical system analyses. System analysis deals with the study of a problem domain aiming at recommending improvements or solutions. The implication is that system analysis deals mainly with the studies of the existing system to ascertain it workability with the aim to design a new system which will overcome the shortcomings of the existing system. At this stage, the software development team seeks to understand in-depth the technical details in the processes within the scope of the computerisation effort. They also seek to establish the required materials for processes outside the scope of computerisation. System analysis is strategic because in our society today, a very big gap exists between those who need computer software in the various areas of endeavours and those who understand the processes involved and the enabling technologies.

System analysis seeks to bridge this big gap. During the analysis period, the development team must ensure that:

- they maintain absolute confidentiality of client data
- they maintain high integrity
- they offer the best advice to the client to help the client take appropriate decision
- they documented all agreement reached with the client
- they do not emotionally caught up in the internal affairs of the client.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain the practical system analysis
- list the stages in carrying out system analysis on the existing system
- explain the various steps involved in system analysis
- discuss the importance of data gathering
- explain the advantages and disadvantages of various method of data gathering
- define system requirement specification
- list the major goals of software requirement analyses.

3.0 MAIN CONTENT

3.1 Data Collection

Data collection also called data gathering is the process of acquiring relevant data relating to a particular field of study. The development team needs organised methods of data collection in order to be able to dig out relevant facts about the system in question. Data collected will help the development team in taking decision about the system to be built. Data collection tools include among others the following:

3.1.1 Questionnaire Methodology

Questionnaire is one of the important tools of data collection. Questionnaires are specially arranged document that enable the development team and indeed all researchers to collect information and opinions from respondent. A questionnaire maintains a uniform response format like: **strongly agree, agree, neutral, disagree** etc. Information on the questionnaire must be expressed in a few, clear

unambiguous manner to ensure quick understanding and easy analysis of collected data.

There are basically two formats of questionnaires: free-format and fixed-format.

3.1.1.1 Free-Format Questionnaire

This is the type of questionnaire that offers the respondent greater latitude (freedom) in the answer. A question is asked and the respondent records the answer in the space provided. Examples of free-format questions are:

What is your feeling about the subject matter?
Are there some reasons why you think the subject should be made compulsory?

It is recommended that the development team/system analyst phrase the questions in simple sentences and not use words that may be confusing. The analyst should also ask questions with three or fewer sentences so that the questions will not take more time than the respondent is willing to sacrifice. It should also be noted that sometimes the respondent's answer will not match the question asked.

3.1.1.2 Fixed-Format Questionnaire

This type of questionnaire contains questions that require selection of predefined responses. In this type or questionnaire, when given any question, the respondent must choose from the available answer. This makes the results easier to tabulate. There are three types of fixed format:

Multiple-choice questions: Here, the respondent is given several answers. It is also specified if more than one answer can be selected:

Example: Do you feel that the operation should be computerised?

Yes ☐
No ☐

Rating questions: Here, the respondent is given a statement and asked to use supplied responses to state an opinion. There should be an equal number of positive and negative ratings.

An example: Automation of the entire process will bring about better output

Strongly agree ☐
Agree ☐
No option ☐
Disagree ☐
Strongly disagree ☐

Ranking questions: Here, the respondent is given several possible answers which are to be ranked in order of preference of experience. Example:
Rank the following according to the time you spent processing:

---------% result processing
---------% new student Registration
---------% continuing student Registration
---------- % staff salary computation

Advantages:

Use of questionnaire in data collection has a number of advantages:

a. most questionnaires can be answered quickly
b. questionnaire allows individuals to maintain anonymity
c. responses van be tabulated and analysed quickly
d. questionnaire provides relatively inexpensive means of data gathering etc.

Disadvantages include:

a. number of respondent is often low
b. good questionnaires are difficult to prepare
c. there is no immediate opportunity to clarify a vague or incomplete answer to any question.

3.1.2 Existing Documentation Method

Collection of data from existing documentation helps the development team understand how the manual system operates. Existing documents may include organisational charts, interoffice memoranda, studies, minutes of meeting, suggestion box notes and reports that document the problem area. Other documents to look at according to Jeffery, L. *et al.* (2003) include, among others, the following:

- company's mission statement and strategic plan
- accounting records, performance reviews work measurement reviews and other scheduled operating reports
- sample of manual and computerised database
- various types of flowcharts and diagrams
- program documentations
- standard operating procedures, job outlines or task instruction for specific day-to-day operations
- policy manuals that may place constraints on the proposed system
- sample of manual and computerised screens and reports
- completed forms that represent actual transaction at various point in the processing cycle etc.

3.1.3 On-Site Observation Method

Many times, some technical details of how operational processes are carried out are not always documented. On-site observation helps the analyst to understand better how operational processes are physically carried out. In this technique, the system analyst/development either participates or watches a person perform activities to learn more about the system. Sometimes, this approach of data gathering is used when the validity of data collected through other methods is in question. In other for the system analyst to be able to obtain data through this approach, the analyst does not arrive at the site and begins to record everything anyhow. The systems analyst must first determine how data will actually be captured. He/she must also identify the ideal time to observe a particular aspect of the system. Observation should be first made when the workload is normal, afterward, observation will be made during peak periods to gather information for measuring the effect caused by the increased volume.

Advantages

a. data gathered by observation can be highly reliable
b. the development team will be able to see exactly what is being done
c. observation is relatively inexpensive compared with other fact-finding methodology etc.

Disadvantages

a. because people usually feel uncomfortable when being observed, they may unknowingly perform differently when being observed.
b. the work being observed may not involve the level of difficulty or volume experienced during that time period.
c. the tasks being observed are subject to various types of interruptions etc.

3.1.4 Interview Method

Interview is a type of data collection methodology in which the development team or system analyst collects information from individuals through face-to-face interactions. Interview can be used to achieve any or all of the following goals: verify facts, find facts, clarify facts, generate enthusiasm and get the end user involved.

The system analyst/development team is the researcher and responsible for organising and conducting the interview. The client or system user is the interviewee who is asked to respond to series of questions. To use the interviewing techniques, you must possess good human relation skills for dealing effectively with different types of people.

Advantages

The interview methodology of gathering information has some advantages which include among others, the following:

a. interview gives the system analyst an opportunity to motivate the interviewee to respond freely and openly to questions
b. interview allows the system analyst to probe for more feedback from the interviewee
c. interview permits the system analyst to adapt questions for each individual
d. interview permits the development team/systems analyst to observe the interviewee's nonverbal communication.

Disadvantages

a. interviewing is very time-consuming and costly
b. success of interview is highly dependent on the system analyst/development's human relation skills.

3.1.5 Work Sampling Method

Because it may not be possible that the development team/system analyst studies every occurrence of every form or record in a file or database, the system analyst normally employs the sampling method to get a large enough cross section to determine what can happen in the system. Sampling is the process of collecting a representative sample of document, forms and records etc. This fact finding technique involves a large number of observations taken at random intervals. When using the sampling methodology, he/she needs to predefine the operations of the job to be observed, make many random observations, being careful to observe activities at the different time of the day. This is done by

counting the different number of occurrences of each operation during the observation. The following guideline will help him/her develop observation skill:

- determine the who, what, where, when, why and how of the observation
- obtain permission from appropriate supervisors or managers
- keep a low profile
- review observation notes with appropriate individuals
- do not interrupt the individuals at work
- do not make assumptions etc.

3.1.6 Survey Method

Survey is a fact-finding technique which employs telephone, mails/e-mails or internet for data gathering. In telephone survey, the interviewer follows a prepared script that is exactly the same as a written questionnaire. This type allows opportunity for some opinion probing. The mail/e-mail survey involves use of electronic mail and non-electronic mail to achieve data collection. The internet method involves use of the international network and the World Wide Web in data collection. The internet is the highest data repository in our age.

SELF-ASSESSMENT EXERCISE 1

1. List any four advantages of interview method of data collection.
2. Questionnaire method and observation method, which one do you prefer.
3. Explain the existing document method of data collection.

3.2 System Design

This form of system design is as a result of the system analysis done in 3.1 above. The design involves such processes like form design (input and output), data flow diagram, system diagram etc. Here, the analyst will only highlight form design, data flow diagram. Other form of design has been treated in software engineering 1

3.2.1 Form Design

Forms are used to capture data from system users (input) and also to display outputs (output form). The development team/system analyst should therefore design an input form for example, to be used in order to ensure that the required information is supplied in a consistent manner. On an input form, the data to be entered into files are clearly set out, typically one character per cell on the form. A well designed input form

would contain various blocks of data in order in which they will be prompted. The output form should also be designed in the format that the information it will display will be clear and easily understood.

SELF-ASSESSMENT EXERCISE 2

Find out the importance of good form design in software development.

3.3 Software Requirement Specification

A thorough systems analysis leads to appropriate software requirement specification (SRS). SRS states in precise and explicit language those functions and capabilities the software must provide and state any required constraints by which the system must abide. According to Sajan, M. (2001), Software requirement specification is often referred to as the "parent" document because all subsequent project management documents such as design specification, software architectural specifications, testing and validation etc. are all related to it. A well defined, well written SRS accomplishes the following four goals Sajan, M. (2001).

3.3.1 The Goals of Software Requirement Specification

The four major goals of SRS according to Sajan, M. (2001) include:

a. SRS provides feedbacks to customer. An SRS is the customers' assurance that the development team understands the issues or problems to be solved and the software behaviour necessary to address this problem. In line with this, SRS must be written in natural language in an unambiguous manner that may also include charts, tables, data flow diagrams, and decision tables among others.
b. SRS decomposes problem into component parts. It must be in a well defined format which organises information, places borders around the problems and solidifies ideas.
c. SRS serves as an input to the design specification and statement of work. SRS must contain sufficient details in the functional system requirement so that a design solution can be devised.
d. SRS serves as a product validation check. SRS serves as a parent document for testing and validation strategies that will be applied to the requirements for validation.

3.3.2 What you should know about writing SRS

While writing SRS, there are few points you should bear in mind. Some of these points are:

- each requirement should have a specific purpose and represent characteristics of problem domain
- when writing SRS, always imagine that the requirement will be given to others to interpret
- it is important to remain objective while writing SRS
- never assume that everyone will understand the requirement the way you understand it
- all terms that could have multiple meanings should be defined in a glossary where its meaning is made more specific.

SELF-ASSESSMENT EXERCISE 3

1. System analysis leads to drawing the system requirement specification. Explain how?
2. How can SRS serve as product validation check?
3. Why must you as a systems analyst remain objective while writing the SRS.

4.0 CONCLUSION

In this unit, you have learnt how you can carry out a successful and detailed system analysis on existing manual or less automated system with a view to designing a new software project. It was noted that during the analysis period, the development team must ensure good working ethics in order to maintain confidence reposed on them by their clients. We also discussed the different methodologies the development team/system analyst can employ in his/her data collection. Furthermore, software requirement specification was discussed. We explored the four major goals of software requirement specification. We also explained some of the factors the development team must take into consideration while trying to prepare the software requirement specification. Finally, we highlighted the need to make a good design after the systems analysis in order to help programmers put up proper code that will achieve the expected functions of the software project

9.0 SUMMARY

Proper and detailed system analysis is strategic to the development of a good software product. There are several tools which enables the development team to gather needed data for any type of software project. These tools include among others: questionnaire, interview, on-site observation, survey, working sample, existing document etc. Having done a proper system analysis, it forms the basis for developing the software requirement specification (SRS) document. The SRS document states in precise and explicit language those functions and capabilities the software must provide as well as states any required constraints by which the system must abide. This SRS is a parent document that guides the development team throughout the development process.

10.0 TUTOR-MARKED ASSIGNMENT

1. Discuss the roles of software requirement specification (SRS) document.
2. Explain the two formats of questionnaire treated in this unit. Give example of each format.
3. What is data collection? And why is data collection necessary in software development.
4. Mention and explain any five ways of analysing an existing system.
5. What do you understand by software requirement specification (SRS)
6. Discuss the advantages and disadvantages of on-site observation method of data collection.

7.0 REFERENCES/FURTHER READING

Boehm, B.; Egyed, A.; Kwan, J.; Port, D.; Shah, A. & Madachy, R. (1998). Using the WinWin Spiral Model: A Case Study.*Computer*, 31(7), 33-44.

Garg, P.K.; Mi, P.; Pham, T.; Scacchi, W. & Thunquest, G. (1994). The SMART Approach for Software Process Engineering, *Proc. 16th. Intern. Conf. Software Engineering,* 341 - 345.

Jeffery, L.; *et al.* ((2001). *System Analysis and Design Methods*. (5th ed.). McGraw-Hill Higher Education. A Division of the McGraw Hill Companies, pp. 163-210.

Mills, H.D.; Dyer, M. & Linger, R.C. (1987). Cleanroom Software Engineering, *IEEE Software*, Royce, W. W., Managing the

Development of Large Software Systems, *Proc. 9th. Intern. Conf. of Software Engineering,IEEE* Computer Society, 328-338 Originally published in Proc.WESCON, 1970.

Moore, J.W.; P.R. DeWeese, & Rilling, D. (1997). "U. S. Software Life Cycle Process Standards," *Crosstalk: The DoD Journal of Software Engineering*, 10:7.

Sajan, M. (2007). *Software Engineering(*Rev. ed.). Ram Naga, New Delhi: S. Chand & Company Ltd., pp. 1-5, 27-36, 138-141, 152-158, 2881-187.

Stephen S. (2001). *Software Engineering.* ISBN-0-256-LL454-4.

UNIT 3 SOFTWARE CODING AND TESTING

CONTENTS

1.0 INTRODUCTION

In the previous unit, you learnt how to analyse the existing system, draw a correct specification document for the new project, and also put up a good design for the new system. In this unit, you will learn how to translate design into actual program codes that will perform the functions specified in the specification document. We may not go into writing real computer programs, so if you are not a good programmer do not be afraid, you will understand the issues we are about to discuss.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- explain program code
- describe when program code is said to be good
- list the qualities of good code
- describe the different program control structures
- discuss modularity in programming

- define software testing
- explain reasons for software errors
- describe software testing methods.

3.0 MAIN CONTENT

3.1 Program Code

Program code simply means the actual computer syntax or computer program used to perform real life computations (Sajan, M., 2001). Coding breaks analysis down into two basic parts: coding the data to put it into useable form and computing with the data. Coding is normally achieved with specific computer languages. At this level, computer languages that are employed are mostly the high level programming languages (high level languages are computer programming languages that uses close to natural languages e.g. (English) but with special syntax and semantics.)

3.1.1 What is "Good Code?"

"Good code" can be defined to be a code that works, free of errors (bugs), readable and maintainable (Sajan, M., 2001). Most software development team have coding "standard" that all programmers are supposed to adhere to. So, though every programmer may have his own style of coding, there are some basic facts that every programmer must consider while writing codes. Some of the points to consider are:

- seek to minimise or eliminate use of global variables. this will help you avoid some errors
- descriptive variable names should be used and try to avoid abbreviation
- use whitespace generously vertically and horizontally
- organise code in easy to read form
- use comment statements always to avoid confusion
- coding style should be consistent throughout the program (e.g. naming convections, identifications)
- make use of descriptive functions and method name
- an application should include proper documentations of overall program function and flow.

3.1.2 Qualities of "Good Code"

There are some important qualities of "good code". Let us now briefly discuss some of these qualities:

- **Consistency:** consistent code is written in such a way that it is easy for people to understand how the program works. When reading consistent code, one subconsciously forms a number of assumptions and expectations about how the code works.
- **Cleanliness**: Clean code is written in a form that is easy to read. It is in a form that people can read it with minimum effort so that they can understand it easily.
- **Correctness**: A code is designed to be correct to a point that people spend less time worrying about bugs and more time enhancing the features of a program. Correct code does not crash.
- **Extensibility**: Extensible code is written in a form that programmers can easily add new features to the program. These are codes that are easier to reuse and modify without much assumptions.

SELF-ASSESSMENT EXERCISE 1

1. What is "bad code"?
2. What are some basic facts that every programmer must consider while writing codes.

3(a) program code
(b) Programming.

3.2 Modularity in Programming

A module is a program unit usually designed to perform a particular task in the main program. Modular programming is defined as a programming style where complex software project is broken down into smaller manageable units. The complex system is usually broken into subsystems and in-turn into program modules. In the same vein, during programming in a particular programming high level language, it is important to continue with the modular form of design to facilitate their easy representations following a standard control logic structures. Some of these standard control structures will be discussed in the next section.

3.3 Program Control Structures

If you want to create a powerful, intelligent, versatile and useful program, you need some methods or techniques for these forms of program flow. These standard programming techniques used in programming is what is known as *program control structures*. Program control structures help programmers to avoid arbitrary large and complex flowcharts and replace them with modular forms. This modular form facilitates representations by any of the three basic program control structures, namely; sequence structure, selection structure and iteration structure. Let us now discuss each of these program structures one after the other.

Please note that in our discussion, we shall use pseudo codes which are an English-like method of expressing the logic of a flowchart. However, in some of our illustrations, we shall simply show the syntax (format) of each structure as it is supported by most common high level languages e.g. Qbasic, visual basic etc.

3.3.1 Sequence Structure

In this program structure, data flow arrow moves from one process to the next process in a sequential order. This implies that two processes are executed one after the other in a chronological manner. Sequential structure can be represented using pseudo codes as:

- Process A
- Process B

Diagrammatically, it can be represented as follows:

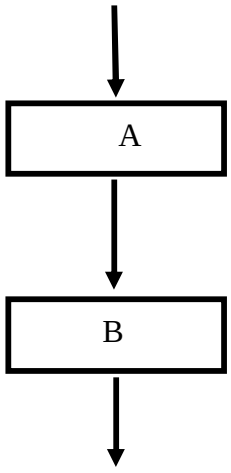

Fig. 3.1: Flowchart of a Sequence Program Structure

3.3.2 Selection Structure

This is the type of program construct that uses a test condition between alternative action (Okey, A.2003). Three main selection mechanisms exist. These are IF-THEN –ELSE, Block IF-THEN-ELSE, SELECT CASE CONSTRUCTS.

Let us now discuss each of them using their syntax as it is supported by most of the high level programming languages.

3.3.2.1 IF-THEN-ELSE Structure

This form of selection exists in a situation where the first statement is evaluated if the specified condition is TRUE; else the second statement is evaluated. The syntax for this construct is thus:

- IF condition THEN statement [ELSE statement]

Where condition is any expression that can be evaluated as true or false. The ELSE part is optional

Let us again use a pseudo code to illustrate that:

```
IF (X) THEN
        Process A
ELSE
        Process B
END IF
```

3.3.2.2 Block IF-THEN-ELSE Structure

In this construct, each of the process contains complex logical structure requiring nested IF-THEN-ELSE. It is nested in the sense that IF-THEN-ELSE constructs is found in another IF-THEN- ELSE. The syntax is thus:

```
IF condition -1 THEN
        [statementblock-1]
ELSEIF condition-2 THEN
        [statementblock-2]
ELSE
        [statementblock-n]
END IF
```

Where

Condition 1& 2: An expression that can be evaluated as True or false
Statementblock-1, statementblock-2, statementblock-n: is one or more statements on one or more lines

3.3.2.3 Select Case Constructs

This is a program structure where a solution may need to take one of many different actions for execution based on the value of a correct variable. Only one option is allowed to be taken (Inyiama, H. & Alo, R., 2009). The syntax for the SELECT CASE CONSTRUCT is:

```
SELECT CASE testexpression
CASE expressionlist1
        [statementblock-1]
CASE expressionlist2
        [statementblock-2]
CASE ELSE
        [statementblock-n]
END SELECT
```

Where
Testexpression: Is any numeric or string expression
Expressionlist 1 & 2: one or more expression to match tstexpression
Statementblock-1-n : one or more statement on one or more lines

For example, in order to compute the performance of a student in a particular course using the SELECT CASE construct, the score becomes the control variable. Since no score can have two grades, the computation can be represented using pseudo code as follows:

SELECT CASE SCORE

```
CASE 1          PROCESS GRADE A
CASE 2          PROCESS GRADE B
CASE 3          PROCESS GRADE C
CASE 4          PROCESS GRADE D
CASE 5          PROCESS GRADE E
CASE 6          PROCESS GRADE F
END SELECT
```

3.3.3 Iteration Structure

This is the type of program construct that allows a controlled repetition of portion of a program code. The basic iteration constructs are WHILE…. WEND, DO…….LOOP, DO…..UNTIL, FOR…..TO…NEXT. Let us now give the syntax of each of the repetitive constructs.

FOR….NEXT Construct

The syntax is:

FOR index = initialvalue TO finalvalue [STEP stepsize]
Statement(s)
NEXT [index]

Where

- Index represents a numeric variable used as a loop counter
- Initialvalue represents the initial(first) value of the counter
- Finalvalue represents the final (last) value of the counter
- Stepsize represents the amount the loop counter variable is changed each time through the loop.

Note: There is also the nested FOR……TO…….NEXT

DO….. LOOP Construct

The syntaxes are as follows:

DO [{WHILEIUNTIL} condition}]
[statementblock]
LOOP

DO
[statementblock]
LOOP [{WHILEIUNTIL} condition}]

NOTE: Syntax of form (a) tests its loop exit condition at the top while the syntax (b) tests its exit condition at the bottom of the loop.

Also items in a pair of square brackets [ ] are optional. Items in a pair braces- { } separated by "I" character mean that one of the items must be used.

WHILE…….WEND Construct

This construct also tests its exit condition at the top. The syntax is as follows

WHILE condition
Statementblock
WEND

Where

- Condition: represents a numeric expression that the program evaluates as true or false
- Statementblock: represents one or more statements that must be repeatedly executed while the condition remains true.

SELF-ASSESSMENT EXERCISE 2

1. Enumerate the three main selection structure.
2. Differentiate between a module and a subsystem.
3. Give the syntax of the WHILE ----WEND structure.

3.4 Software Testing

Software testing is the process of executing a program or system with the intent of finding errors (bugs); that is to say that testing involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results (Sajan, M., 2001).

3.4.1 Testing Fundamentals

There are some basic things you must be conversant with to be able to successfully carry out software testing. Some of these issues include knowing some reasons for software (bugs) and few principles in testing, test plan etc.

Few **principles** in testing: Listed below are few principles to take note of:

- Testing should be based on user requirements
- Test planning should be done early
- Testing should start at the module
- Test time and resources are limited
- You must use effective resources to test.

3.4.1.1 Reasons for Software Errors (Bugs)

The following among other, are the causes of error in programs:

- Poorly documented codes
- Software complexity
- Programming error
- Software development tools
- Changing requirements etc.

3.4.1.2 Test Plan

A software project plan is a document that describes the objectives, scope, approach and focus of a software testing (Sajan, 2001). Though the content of a test plan may vary from project to project, every test plan must contain all that was specified in the specification document. Examples of some items found in a test plan include among others, the following:

- Title of the project
- Software product overview
- Relevant standard or legal requirement.
- Project risk analysis
- Software entrance and exit criteria etc.

3.4.2 Purpose of Testing

According to Sajan, M. (2001), testing is usually performed for the following purposes:

- **Verification and Validation**

These two words come to play here because they are the program codes that actually perform the functions spelt out in the specification document. These two processes help to ensure that the product satisfies requirements of the software product. **Verification** is the testing that takes place at the end of each phase. It is a test in each step in the process of building the software to ensure the software yield the right product. **Validation** is the testing that takes place after the complete product has been developed before acceptance testing. Validation can also be defined as the process of determining whether the product as a whole satisfies its specification document. These two processes therefore involve activities such as reviews, inspections, walkthroughs and testing.

- **To Improve Quality**

With regards to testing, quality means the conformance to the specified design requirement that is performing as required under specified circumstances. Typical software quality factors include correctness, efficiency, flexibility, usability, documentation, maintainability, reusability etc. Quality is very necessary because error could cause serious losses, disasters.

- **For Reliability Estimation**

Software reliability has important relations with many aspects of software, including the structure, and the amount of testing it has been subjected to. Based on an operational profile (an estimate of the relative frequency of use of various inputs to the program), testing can be used as a statistical sampling method to gain failure data for reliability estimation.

3.4.3 Testing Methods

According to Sajan, M. (2001), testing methods can be broadly classified into two:

- White box or structural testing
- Black box or functional testing

3.4.3.1 White Box or Structural Testing

White box testing is a form of testing that concentrates on the procedural details. Using this testing method, the internal structure is being disclosed with the main goal to detect faults. White box testing relies on the intimate knowledge of the code and procedural design to drive the test case. It is most widely utilised in unit testing to determine all possible paths within a module to execute all loops and to test all logical expression. White box testing helps the software engineer to:

- Guarantee that all independent paths within a module have been exercised at least once.
- Examine all logical decisions on their true and false sides.
- Executes all loops and test their operations at their limits.
- Exercise data structures to ensure their validity.

3.4.3.2 Black Box Testing or Functional Testing

Black box testing focuses on the overall functionality of the software. This testing method allows the functional testing to uncover faults like incorrect or missing functions, errors in any of the interfaces, errors in data structures or databases and errors related to performance and program initialisation or termination. To perform a successful black box test, the relationships between the many different modules in the system model need to be understood. Next, all necessary ways of testing all object relationships need to be defined. Black box testing aims to test a given program's behaviour against its specification without making any reference to the internal structures of the program or the algorithms used.

3.4.4 Types of Testing

There many types of testing. Let us highlight few of them here:

- **Unit testing**: This is a test of a particular function or code modules. Unit testing requires detailed knowledge of the internal program design and code and is typically done by the programmer himself.
- **Incremental integration testing**: It is a continuous testing of an application as new functionality is added; it requires that various aspects of an application's functionality be independent enough to work separately before all parts of the program are completed.
- **Integration testing**: This is the testing of combined parts of an application to determine if they function together correctly. The "part" include code modules, individual applications etc. Integration testing procedure can be performed in three ways: Top-down, bottom-up, or using "Big-Bang" approach (Humphery).
- **Top-down integration strategy**: This is a strategy where modules are developed and tested starting from the top level of the programming hierarchy and continuing with the lower levels. Top-down strategy is an incremental approach since the testing is done one level at a time. The benefits of top-down integration are that having a skeleton, we can test major functions early in the development process.
- **Bottom-up integration**: This integration strategy starts with building and testing the low level modules first, working its way up the hierarchy. The advantage of bottom-up integration is that there is no need for program stubs as we start developing and testing with actual modules.
- **Big-Bang approach:** In this integration approach, all the modules or builds are constructed and tested independently of

each other and when these are finished, they are all put together at the same time. The advantage of this approach is that it is very quick as no driver or stubs are needed, thus cutting down the development time.

- **System testing:** System testing is a form of testing based on overall requirements specifications. It embraces all combined parts of a system. This type of test involves examination of the whole computer system. All the software components, all hardware components and the interfaces are tested together.
- **Function testing:** Is a testing process that is black-box in nature. It is aimed at examine the overall functionality of the product. Function testing usually includes testing of all the interfaces and should therefore; involve the client in the process. The specification in function testing should be very detailed describing who, where, when and how the test should be conducted and what exactly will be tested.

SELF-ASSESSMENT EXERCISE 3

1. Is verification and validation the same?
2. Why is high level languages mostly used in the development of software products today?
3. What is white box testing?

4.0 CONCLUSION

In this unit, you have learnt that in order to facilitate maintenance, a clear programming style is needed and a good programming practice must be followed. It was noted that organisations have programming standard which all programmer working for them must adopt. We went further to discuss some program control structures which when followed in programming, helps one to come up with "good" programs. Moreover, software testing was discussed. You also learnt that testing enables one to ensure that the software product if free of bugs and that it satisfies the conditions specified in the specification document. Two methods of carrying out software testing; black box/functional testing and white box/structural testing were highlighted. A comprehensive test plan must also be followed in order to arrive at the expected result.

8.0 SUMMARY

The software product designed can only be realised if translated to program codes using any high level programming language of choice or as specified in the contract. Programmers must put up codes that are clean, consistent, correct and extensible following the standard control programming structures. After the coding, software testing must be

carried out to ensure that the software product is working according to specifications. There are also standards followed while carrying out software testing. These standards help developers to achieve the very purpose of software testing.

9.0 TUTOR-MARKED ASSIGNMENT

1a. Define good code
b. Explain any three qualities of good code.

2. Explain vividly, any two program control structures of your choice.
3. What is modularity in programming?
4. Give and explain the syntax of the block IF-THEN-ELSE structure.
5. Explain any two purposes of software testing you know.
6. Give any four reasons for software bugs.

7.0 REFERENCES/FURTHER READING

Ani, C.O. (n.d). *Programming with Microsoft Basic* (New ed.). Enugu: Immaculate Publications Limited, pp. 76-87.

Boehm, B.; Egyed, A.; Kwan, J.; Port, D.; Shah, A. & Madachy, R. (1998). Using the WinWin Spiral Model: A Case Study, *Computer*, 31(7), 33-44.

Garg, P.K.; Mi, P.; Pham, T.; Scacchi, W. & Thunquest, G. (1994). The SMART Approach for Software Process Engineering, *Proc. 16th. Intern. Conf. Software Engineering,* 341 - 345.

Jeffery, L.; *et al.* (2001). *System Analysis and Design Methods* (5th ed.). McGraw-Hill Higher Education, pp. 163-210.

Mills, H.D.; Dyer, M.; & Linger, R.C. (1987). Cleanroom Software Engineering, *IEEE Software*, Royce, W. W., Managing the Development of Large Software Systems, *Proc. 9th. Intern. Conf. of Software Engineering,* IEEE Computer Society,328-338 Originally Published in Proc.WESCON, 1970.

Moore, J.W.; DeWeese, P.R. & Rilling, D. (1997). "U. S. Software Life Cycle Process Standards," *Crosstalk: The DoD Journal of Software Engineering*, 10:7.

Sajan, M. (2007). *Software Engineering (*Rev. ed.). New Delhi: S. Chand & Company Ltd., pp. 1-5, 27-36, 138-141, 152-158, 2881-187.

Stephen, S. (2001). *Software Engineering*. ISBN-0-256-LL454-4.

MODULE 4 FORMAL METHODS

UNIT 1 OVERVIEW OF FORMAL METHODS

CONTENTS

1.0 INTRODUCTION

Every software engineering methodology is based on a recommended development processes which are of several phases: analysis, specification, design, coding, unit testing, integration and system testing and maintenance. Formal methods can be a foundation for describing complex systems. It can also be a foundation for reasoning about systems or provides support for program development. The use of formal methods for software and hardware designs is motivated by the expectation that, as in other engineering disciplines, it will perform appropriate mathematical analysis that can contribute to the reliability and robustness of a design. However, the high cost of using formal

methods implies that they are only used in the development of high-integrity systems, where safety or security is of utmost importance. In this unit, we shall simply take an overview of what formal methods are, their classifications, uses, limitations and advantages. In subsequent units, we may pick on a particular formal method to enable you appreciate more what they look like.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- describe the formal methods
- list some examples of formal methods
- explain reasons for learning formal methods
- discuss the classification of formal methods
- describe the uses of formal methods
- explain precautions in the use of formal method
- describe the limitations of formal methods.

3.0 MAIN CONTENT

3.1 What are Formal Methods?

In computer science and software engineering, formal methods are a particular kind of mathematically-based techniques for the specification, development and verification of software and hardware systems. Formal methods are best described as the application of a fairly broad variety of theoretical computer science fundamentals, in particular logic calculi, formal languages, automata theory, and program semantics, but also type systems and algebraic data types to problems in software and hardware specification and verification**.**

Formal methods allow a software engineer to create a specification that is more complete, consistent, and unambiguous than those produced using conventional or object-oriented methods. Set theory and logic notation are used to create a clear statement of facts (requirements). This mathematical specification can then be analysed to prove correctness and consistency. Because the specification is created using mathematical notation, it is inherently less ambiguous than informal modes of representation. The following topic categories are presented.

3.2 Why Considering Formal Method?

There are some reasons we consider formal methods. They include among others, the following:

- Complexity of systems with embedded software has increased rapidly
- Maintaining reliability in software-intensive systems is very difficult
- Systems are increasingly dependent on software components.

3.3 Classification of Formal Method

3.3.1 Basic Classifications

Formal methods can be used at a number of levels as shown below:

- **Level 0:** Formal specification may be undertaken and then a program developed from this informally. This may be the most cost-effective option in many cases.
- **Level 1:** Formal development and formal verification may be used to produce a program in a more formal manner. For example, proofs of properties or refinement from the specification to a program may be undertaken. This is most appropriate in high-integrity systems involving safety or security.
- **Level 2:** Theorem provers may be used to undertake fully formal machine-checked proofs. This can be very expensive and is only practically worthwhile if the cost of mistakes is extremely high (e.g., in critical parts of microprocessor design).

3.3.2 Classification as with Programming Language Semantics

As with programming language semantics, styles of formal methods may be roughly classified as follows:

- **Denotational semantics:** Here**,** the meaning of a system is expressed in the mathematical theory of domains. Proponents of such methods rely on the well-understood nature of domains to give meaning to the system; critics point out that not every system may be intuitively or naturally viewed as a function.
- **Operational semantics**: Here, the meaning of a system is expressed as a sequence of actions of a (presumably) simpler computational model. Proponents of such methods point to the simplicity of their models as a means to expressive clarity; critics oppose that the problem of semantics has just been delayed (who defines the semantics of the simpler model?).
- **Axiomatic semantics**: Here, the meaning of the system is expressed in terms of preconditions and postconditions which are true before and after the system performs a task, respectively. Proponents note the connection to classical logic; critics note that

such semantics never really describe what a system does (merely what is true before and afterwards).

SELF-ASSESSMENT EXERCISE 1

Do you think that application of formal methods in software development is necessary? Give reason(s) for your answer.

3.4 Lightweight Formal Methods

Some practitioners believe that the formal methods community has overemphasized full formalisation of a specification or design. They contend that the expressiveness of the languages involved, as well as the complexity of the systems being modelled, makes full formalisation a difficult and expensive task. As an alternative, various *lightweight* formal methods, which emphasize partial specification and focused application, have been proposed. Examples of this lightweight approach to formal methods include the Alloy Object Modelling Notation, Denney's synthesis of some aspects of the Z notation with use case driven development, and the Vienna Development Method (VDM) Tools.

3.5 Uses of Formal Method

Formal methods can be applied at various points through the development process.

3.5.1 Specification

Formal methods may be used to give a description of the system to be developed, at whatever level(s) of detail desired. This formal description can be used to guide further development activities (this will be discussed in subsequent sections). Additionally, formal methods can be used to verify that the requirements for the system being developed have been completely and accurately specified. The need for formal specification systems has been noted for years. In the ALGOL 60 Report, John Backus presented a formal notation for describing programming language syntax (later named Backus normal form or Backus-Naur form (BNF)); Backus also described the need for a notation for describing programming language semantics.

3.5.2 Development

Once a formal specification has been developed, the specification may be used as a guide while the concrete system is developed (i.e. realised in software and/or hardware). Examples:

- If the formal specification is in an operational semantics, the observed behaviour of the concrete system can be compared with the behaviour of the specification (which itself should be executable or simulated). Additionally, the operational commands of the specification may be amenable to direct translation into executable code.
- If the formal specification is in an axiomatic semantics, the preconditions and postconditions of the specification may become assertions in the executable code.

3.5.3 Verification

Once a formal specification has been developed, the specification may be used as the basis for proving properties of the specification (and hopefully by inference the developed system).

3.5.3.1 Human-Directed Proof

Sometimes, the motivation for proving the correctness of a system is not the obvious need for re-assurance of the correctness of the system, but a desire to understand the system better. Consequently, some proofs of correctness are produced in the style of mathematical proof: handwritten (or typeset) using natural language, using a level of informality common to such proofs. A "good" proof is one which is readable and comprehensible by other human readers. Critics of such approaches point out that the ambiguity inherent in natural language allows errors to be undetected in such proofs; often, subtle errors can be present in the low-level details typically overlooked by such proofs. Additionally, the work involved in producing such a good proof requires a high level of mathematical sophistication and expertise.

3.5.3.2 Automated Proof

In contrast, there is increasing interest in producing proofs of correctness of such systems by automated means. Automated techniques fall into two general categories:

- **Automated theorem proving**: Here, a system attempts to produce a formal proof from scratch, given a description of the system, a set of logical axioms, and a set of inference rules.
- **Model checking:** Here, a system verifies certain properties by means of an exhaustive search of all possible states that a system could enter during its execution.

Neither of these techniques works without human assistance. Automated theorem provers usually require guidance as to which properties are

"interesting" enough to pursue; model checkers can quickly get bogged down in checking millions of uninteresting states if not given a sufficiently abstract model. Proponents of such systems argue that the results have greater mathematical certainty than human-produced proofs, since all the tedious details have been algorithmically verified. The training required to use such systems is also less than that required in producing good mathematical proofs by hand. This makes the techniques accessible to a wider variety of practitioners.

Critics note that some of those systems are like oracles: they make a pronouncement of truth, yet give no explanation of that truth. There is also the problem of "verifying the verifier"; if the program which aids in the verification is itself unproven, there may be reason to doubt the soundness of the produced results. Some modern model checking tools produce a "proof log" detailing each step in their proof, making it possible to perform, given suitable tools, independent verification.

SELF-ASSESSMENT EXERCISE 2

Explain any two development processes where formal method is applied and how?

3.6 Cautions in the Use of Formal Methods

Judicious application to suitable project environments is critical if benefits are to exceed costs

Formal method and problem domain expertise must be fully integrated to achieve positive results.

3.7 Limitations of Formal Methods

Limitations of formal method include, among others, the following:

- Formal method is used as an adjunct to, not a replacement for, standard quality assurance methods
- Formal methods are not a panacea, but can increase confidence in a product’s reliability if applied with care and skill
- Formal methods, though very useful for consistency checks, cannot assure completeness of a specification.

4.0 CONCLUSION

Formal method is the application of a fairly broad variety of theoretical computer science fundamentals, in particular, logic calculi, formal languages, automata theory, and program semantics. You learnt that

increase in software complexity, and the problem of maintaining software reliability among others, are reasons why formal methods are needed. We also classified formal methods on basic classification and also based on programming language semantics. The reasons for the use of formal method were also highlighted. Finally, we discussed some limitations of formal methods.

5.0 SUMMARY

In this unit, you have learnt that formal method:

- could be a foundation for describing complex systems
- could be a foundation for reasoning about systems
- provides support for program development
- are no panacea
- can detect defects earlier in life cycle
- can be applied at various levels of resource investment
- can be integrated within existing project process models
- can improve quality assurance when applied judiciously to appropriate projects.

6.0 TUTOR-MARKED ASSIGNMENT

1. Explain clearly the various stages where formal method is applied in software development.
2. Dwell concisely on the two basic classifications of formal methods as discussed in this unit.
3. Explain any three limitations of formal method.
4. What is Human-directed proof?

7.0 REFERENCES/FURTHER READING

Adelard, R. M.; Peter, F.; Mural & Specbox. (1991). In *VDM'91 Formal Software Development Methods.* Springer Berlin / Heidelberg.

Bjørner, D. & Cliff B. J. (1978). *The Vienna Development Method: The Meta-Language, Lecture Notes in Computer Science 61*. Berlin, Heidelberg, New York: Springer.

Derek, A. & Darrel, I. (1991). *Practical Formal Methods*. McGraw Hill. *15th International Symposium on Formal Methods*, 2008. http://www.fm2008.abo._/.

Fitzgerald, J. S. & Larsen, P.G. (1998). *Modelling Systems: Practical Tools and Techniques in Software Engineering*. Cambridge: University Press.

Hußmann, H. (1997). Formal Foundations for Software Engineering Methods, *Vol. 1322 of Lect. Notes Comp. Sci.* Berlin: Springer.

Dawes, J. (1991). *The VDM-SL Reference Guide*. Pitman.

Kemmerer, R. A. (1990). Integrating Formal Methods into the Development Process. *IEEE Software*, 7(5):37–50.

UNIT 2 OVERVIEW OF SOME FORMAL METHODS

CONTENTS

1.0 INTRODUCTION

In this unit, you will learn about some specific formal methods. The two formal methods treated here, are VDM and Z notation. There are many different approaches to formalisation, but one of the longest established formal methods for development of computer-based systems is VDM, the Vienna Development Method. The VDM is a collection of techniques for the modelling, specification and design of computer-based systems. VDM has its roots in the IBM laboratories in Vienna in the mid-1970s. The corresponding standardised definition language is called VDM-SL. There is also an object-oriented extension of VDM, called VDM++. There are also several industrial applications of VDM (especially VDMTools), for example Boeing used VDMTools for reverse engineering from Java back to VDM++. This unit will also take a brief overview of another formal method known as Z notation. Z notation is a formal method based on the standard mathematical notation used in axiomatic set theory, lambda calculus, and first-order predicate logic. All expressions in Z notation are typed, thereby avoiding some of the paradoxes of naive set theory. Z contains a standardised catalog (called the mathematical toolkit) of commonly used mathematical functions and predicates.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- describe the Vienna Development Method (VDM)
- discuss the history of VDM
- list the basic features of VDM
- explain the language of VDM
- describe structuring in VDM
- explain VDM tools support
- discuss the Z notation
- describe the history of Z notation
- list the various usages and notation of the Z notation.

3.0 MAIN CONTENT

3.1 What is VDM?

The Vienna VDM is a formal language developed at the IBM laboratories in Vienna. VDM stands for "Vienna Development Method": a collection of techniques for the formal specification and development of computing systems. It consists of a specification language called VDM-SL; rules for data and operation refinement which allow one to establish links between abstract requirements specifications and detailed design specifications down to the level of code; and a proof theory in which rigorous arguments can be conducted about the properties of specified systems and the correctness of design decisions. Computing systems may be modeled in VDM-SL at a higher level of abstraction than is achievable using programming languages, allowing the analysis of designs and identification of key features, including defects, at an early stage of system development. Models that have been validated can be transformed into detailed system designs through a refinement process. The language has a formal semantics, enabling proof of the properties of models to a high level of assurance. It also has an executable subset, so that models may be analysed by testing and can be executed through graphical user interfaces, so that models can be evaluated by experts who are not necessarily familiar with the modeling language itself. The term "VDM" is sometimes used a little carelessly to mean the specification language only.

3.2 History of VDM

VDM's origins lie in the research on formal semantics of programming languages at IBM's Vienna Laboratory in the 1960s and 70s, including the VDL and Meta-IV notations. VDM is their modern descendent, now

used well beyond the bounds of language semantics in industrial systems development as well as academic research. The first version of the language was called the **V**ienna **D**efinition **L**anguage (VDL). The VDL was essentially used for giving operational semantics descriptions in contrast to the VDM - Meta-IV which provided denotational semantics. Towards the end of 1972, the Vienna group again turned their attention to the problem of systematically developing a compiler from a language definition. The overall approach adopted has been termed the "Vienna Development Method". The meta-language actually adopted ("Meta-IV") is used to define major portions of PL/1 (as given in ECMA 74 - interestingly a "formal standards document written as an abstract interpreter") in BEKIČ 74.

The Language

As a mature and accepted language which has been used in a wide variety of applications, VDM supports lots of features for creation of formal models and proofs.

3.3 Functions

Functions can be defined in two ways; implicit or explicit. Both methods have advantages and disadvantages and are used in different kinds of situations.

3.3.1 Explicit Functions

An explicitly defined function consists of already known functions, operators, constants and parameters. The first line of an explicit function definition is the signature specifying the functions name, the input parameters and the output. The second line starts again with the name of the function, followed by a pair of brackets containing names for the input parameters so that they can be used later on. The Greek delta (≙) is used as definition symbol. The equality sign (=) is not used to avoid confusion with predicates involving equalities (e.g. square $(2^2) = 4$).

3.3.2 Implicit Functions

An implicitly defined function does not specify how to calculate the solution, but what has to be calculated. The explicit definition can be referred to as the implementation of the implicit specification. The most significant reason to give an implicit specification is simplicity. Implicit definitions are mostly shorter than explicit ones. For example it is easy to define a square root function implicitly, but it is much harder to implement an algorithm to approximately calculate it. However it is not always easier to give an implicit definition. The implicit specification of

the algorithm for the UK's income tax is not really shorter than the actual implementation. Another reason to give an implicit definition is that they are easier to understand. What the square root function does is easy to understand. How it is done is much more complicated and in many cases also unimportant. A big advantage of implicit definitions is that they can not only yield a result for single values, but that it is also possible to evaluate the range of plausible results. The danger of implicit with implicit specifications is, that they have to be very exact. The square root of a function can be either negative or positive. An implicit definition has to deal with this. It is only "correct" if it defines all properties the user want's to rely on. The implicit definition of a function starts similar to an explicit one. The first line of the specification is the signature: names are given to argument and result and the names are followed by the type. Names are given as link to pre- and post-conditions. The second contains the precondition and the third one the postcondition. Pre- and post-conditions are arbitrary complex, boolean-valued functions, specifying the valid input values respectively the possible output. Since the possible input values for an implicit defined function are limited to values fulfilling the precondition, such functions can be seen as partial functions.

3.4 Features of VDM

The toolkit has lots of useful features from syntax checking to code generation:

- **Syntax checking**: The syntax-checker verifies whether the syntax of the selected files matches the VDM++ language specifications. If the check passes, it gives access to the other features of VDMTools.
- **Type checking**: The type-checker tests mis-uses of values and operators and can also show places, where runtime errors may occur.
- **Code generation**: VDMTools is able to generate a fully executable code for about 95% of all VDM++ constructs. Code generation is available for Java and C++.
- **Specification manager**: A manager-window displays all classes and files in the specification. It also shows the status for each file.
- **Interpreter and Debugger**: VDMTools allows the execution of all executable VDM++ constructs. Debugging is also supported.
- **Integrity examiner**: It extends the static checking capabilities of VDM++. The tool scans through all sources to find possible inconsistencies or integrity violations. The examiner creates expressions which should evaluate to be true. If they evaluate to be false, there may be a problem.

- **Rose-VDM++ Link**: The Rational-Rose-Link provides a bi-directional link between Rational Rose (UML) and the Toolbox (VDM++).
- **Java to VDM++ Translator**: It is possible to generate a VDM++ specification from a Java application. The generated model can be examined at VDM++ level.
- **Several input types**: Models for VDMTools can be written either with Microsoft Word (RTF) or in Latex. Plain text is also possible but is not recommended.

3.5 Structuring

The main difference between the VDM-SL and VDM++ notations are the way in which structuring is dealt with. In VDM-SL there is a conventional modular extension whereas VDM++ has a traditional object-oriented structuring mechanism with classes and inheritance.

3.5.1 Structuring in VDM-SL

In the ISO standard for VDM-SL, there is an informative annex that contains different structuring principles. These all follow traditional information hiding principles with modules and they can be explained as:

- **Module naming**: Each module is syntactically started with the keyword module followed by the name of the module. At the end of a module the keyword end is written followed again by the name of the module.
- **Importing**: It is possible to import definitions that have been exported from other modules. This is done in an imports section that is started off with the keyword imports and followed by a sequence of imports from different modules. Each of these module imports is started with the keyword from followed by the name of the module and a module signature. The module signature can either simply be the keyword all indicating the import of all definitions exported from that module, or it can be a sequence of import signatures. The import signatures are specific for types, values, functions and operations and each of these are started with the corresponding keyword. In addition these import signatures name the constructs that there is a desire to get access to. In addition optional type information can be present and finally it is possible to rename each of the constructs upon import. Type one needs also to use the keyword structure if one wishes to get access to the internal structure of a particular type.
- **Exporting**: The definitions from a module that one wish other modules to have access to are exported using the keyword

exports followed by an exports module signature. The exports module signature can either simply consist of the keyword all or as a sequence of export signatures. Such export signatures are specific for types, values, functions and operations and each of these are started with the corresponding keyword. In case one wishes to export the internal structure of a type, the keyword structure must be used.

- **More exotic features**: In earlier versions of the VDM-SL tools, there were also support for parameterised modules and instantiations of such modules. However, these features were taken out of VDMTools around 2000 because they were hardly ever used in industrial applications and there were substantial number of tool challenges with these features.

3.5.2 Structuring in VDM++

In VDM++ structuring are done using classes and multiple inheritance. The key concepts are:

- **Class**: Each class is syntactically started with the keyword class followed by the name of the class. At the end of a class, the keyword end is written followed again by the name of the class.
- **Inheritance**: In case a class inherits constructs from other classes, the class name in the class heading can be followed by the keywords is subclass, followed by a comma-separated list of names of superclasses.
- **Access modifiers**: Information hiding in VDM++ is done in the same way as in most object-oriented languages using access modifiers. In VDM++ definitions are per default private but in form of all definitions, it is possible to use one of the access modifier keywords: private, public and protected.

3.6 Tool Support

A number of different tools support VDM:

- VDMTools are the leading commercial tools for VDM and VDM++, owned, marketed, maintained and developed by CKS Systems, building on earlier versions developed by the Danish Company, IFAD. The full versions include automatic code generation for Java and C++, dynamic link library and CORBA support.
- Overture is a community-based open source initiative aimed at providing freely available tool support for VDM++ on top of the eclipse platform. Its aim is to develop a framework for

interoperable tools that will be useful for industrial application, research and education.

- SpaceBox: from Adelard, it provides syntax checking, some simple semantic checking, and generation of a LaTeX file enabling specifications to be printed in mathematical notation. This tool is freely available but it is not being further maintained.
- LaTexand LaTeX2e macros are available to support the presentation of VDM models in the ISO Standard Language's mathematical syntax. They have been developed and are maintained by the National Physical Laboratory in the UK.

SELF-ASSESSMENT EXERCISE 1

Outline the industrial applications of VDM.

3.7 Z Notation

The Z notation is another formal method. The **Z notation** (formally pronounced /ˈzɛd/ notation), named after Zermelo-Fraenkel set theory, is a formal specification language used for describing and modelling computing systems. It is targeted at the clear specification of computer programs and computer-based systems in general.

3.8 History of Z Notation

Z notation was originally proposed by Abrial in 1977 with the help of Steve Schuman and Bertrand Meyer. It was developed further at the Programming Research Group at Oxford University, where Abrial worked in the early 1980s, having arrived at Oxford in September 1979. Abrial answers the question "Why Z?" with "Because it is the ultimate language.

3.9 Usage and Notation

Z is based on the standard mathematical notation used in axiomatic set theory, lambda calculus, and first-order predicate logic. All expressions in Z notation are typed, thereby avoiding some of the paradoxes of naive set theory. Z contains a standardised catalog (called the *mathematical toolkit*) of commonly used mathematical functions and predicates. Although Z notation (just like the APL Language,) uses many non-ASCII symbols, the specification includes suggestions for rendering the Z notation symbols in ASCII and in Latex.

SELF-ASSESSMENT EXERCISE 2

What is Z notation targeted at?

4.0 CONCLUSION

In the history of formal methods, the Vienna Development Method is one of the longest established formal methods. During its long lifetime, many different tools, standards and formalisations arose and disappeared. The most popular tool for VDM today (VDMTools) is a rather useful tool for development of formal models in VDM++ or VDM-SL. The code creation features of VDMTools for Java and C++ are very helpful and work properly. Syntax- and type-checking ensure syntactically correct models and the integrity-examiner provides integrity-conditions which have to be proven or at least observed. The Vienna Development Method has a great community. Z notation is another good formal method in use. Notation is based on the standard mathematical notation used in axiomatic set theory, lambda calculus, and first-order predicate logic. All expressions in Z notation are typed, thereby avoiding some of the paradoxes of naive set theory.

11.0 SUMMARY

In this unit, two formal methods namely; the Vienna Development Method and the Z notation were treated. We discussed that though there are many different approaches of formalisation, one of the longest established formal methods for development of computer-based systems is VDM; the Vienna Development Method. The Vienna Development Method (VDM) is a collection of techniques for the modeling, specification and design of computer-based systems. We also explained that Z notation is another good formal method that is widely used for development of computer-based system.

12.0 TUTOR-MARKED ASSIGNMENT

1. Differentiate between the explicit functions and the implicit functions
2. Trace the history of VDM
3. List and explain the uses of any five features in the toolkit of VDM
4. Differentiate between structuring in VDM-SL and structuring in VDM++.

7.0 REFERENCES/FURTHER READING

Bicarregui, J.C.; Fitzgerald, J.S.; Lindsay, P.A.; Moore, R. & Ritchie, B. *Proof in VDM: a Practitioner's Guide.*

Bjørner, D. & Cliff, B. J. (1978). The Vienna Development Method: The Meta-Language. *Lecture Notes in Computer Science 61*. Berlin, Heidelberg, New York: Springer.

Bjørner, D. & Jones, C.B. (1982). *Formal Specification and Software Development.* Prentice Hall International.

Dawes, J. (1991). *The VDM-SL Reference Guide*. Pitman.

Jones, C. B. (1986). *Systematic Software Development Using VDM* (2nd ed.). Prentice-Hall International.

Jones, C.B. (1990). *Systematic Software Development using VDM*. Prentice Hall.

O'Regan, G. (2006). *Mathematical Approaches to Software Quality*. London: Springer.

UNIT 3 SOFTWARE PROJECT MANAGEMENT

CONTENTS

1.0 INTRODUCTION

This unit will discuss the rules, processes and tools for project planning and management. Every project requires some preparation to achieve a

successful outcome, so every project would probably be done better by using a few project management methods somewhere in the process. Project management methods can help in the planning and managing of all sorts of tasks, especially complex activities. There exist numerous project management methodologies and techniques in the world. This unit however aims to make a selection and present an overview of the commonly used software project management methodologies and techniques. Their applications, advantages and disadvantages are discussed as well as their relation to each other. The methodologies RUP, SSADM, Crystal Clear are discussed, as well as the techniques PMBOK, COCOMO, MTA, EV and Critical path.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- describe project management
- explain the project management tools
- discuss the project management process
- explain project management methodologies
- describe the software project management techniques.

3.0 MAIN CONTENT

3.1 Definition of Software Project Management

3.1.1 Definitions

Project management can be defined as the process of planning, organising, staffing, directing and controlling the production of software. Project management can also be said to mean the art of planning, organising, directing and controlling of resources for a finite period of time to complete specific goals and objectives.

3.2 Project Management Tools

Here, we shall examine four commonly used tools in project planning and management, namely: Brainstorming, Fishbone Diagrams, Critical Path Analysis Flow Diagrams, and Gantt Charts.

3.2.1 Brainstorming

Brainstorming is usually the first crucial creative stage of the project management and project planning process. Unlike most project management skills and methods, the first stage of the brainstorming

process is ideally a free-thinking and random technique. Consequently, it can be overlooked or under-utilised because it is not a natural approach for many people whose main strengths are in systems and processes. Consequently, this stage of the project planning process can benefit from being facilitated by a team member who is capable of managing such session, specifically to help organised people to think randomly and creatively.

3.2.2 Fishbone Diagrams

Fishbone diagrams are chiefly used in quality management fault-detection, and in business process improvement, especially in manufacturing and production, but the model is also very useful in project management planning and task management generally. Within project management, fishbone diagrams are useful for early planning, notably when gathering and organising factors, for example during brainstorming. Fishbone diagrams are very good for identifying hidden factors which can be significant in enabling larger activities, resources areas, or parts of a process. Fishbone diagrams are not good for scheduling or showing interdependent time-critical factors.

Fishbone diagrams are also called 'cause and effect diagrams' and Ishikawa diagrams, after Kaoru Ishikawa (1915-89), a Japanese professor who specialised in industrial quality management and engineering. He devised the fishbone diagram technique in the 1960s.

Ishikawa's diagram became known as a fishbone diagram, obviously, because it looks like a fishbone:

- a fishbone diagram has a central spine running left to right, around which is built a map of factors which contribute to the final result (or problem)
- for each project the main categories of factors are identified and shown as the main 'bones' leading to the spine
- into each category can be drawn 'primary' elements or factors (shown as P in the diagram), and into these can be drawn secondary elements or factors (shown as S). This is done for every category, and can be extended to third or fourth level factors if necessary.

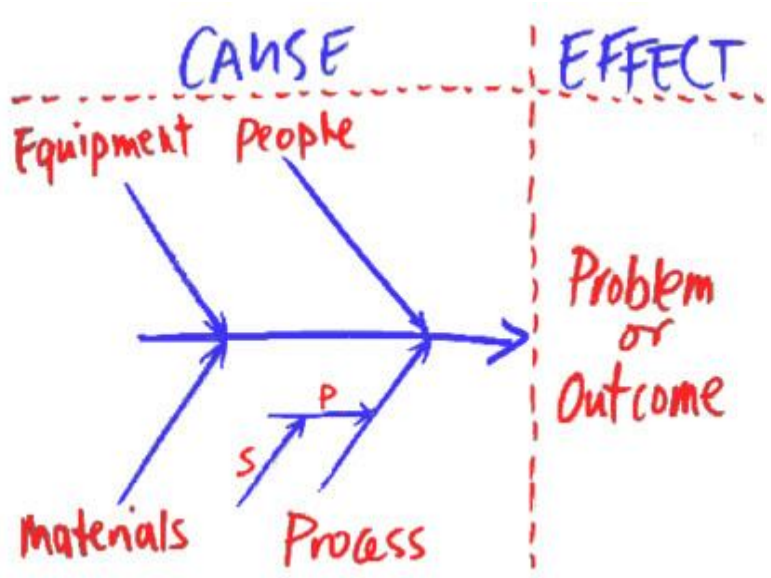

Fig. 3.1: Fishbone Diagram

The diagram above is a very simple one. Typically fishbone diagrams have six or more main bones feeding into the spine. Other main category factors can include: environment, management, systems, training, legal, etc.

The categories used in a fishbone diagram should make sense for the project. Various standard category sets exist for different industrial applications, however it is important that your chosen structure is right for your own situation, rather than taking a standard set of category headings and hoping that it fits.

At a simple level, the fishbone diagram is a very effective planning model and tool - especially for 'mapping' an entire operation. Where a fishbone diagram is used for project planning thus, the 'effect' is shown as the aim, outcome or result, not a problem.

3.2.3 Project Critical Path Analysis (Flow Diagram or Chart)

Critical Path Analysis sounds very complicated, but it is a very logical and effective method for planning and managing complex projects. A critical path analysis is normally shown as a flow diagram, whose format is linear (organised in a line), and specifically a time-line. Critical path analysis is also called critical path method; it means the same thing and the terms are commonly abbreviated, to CPA and CPM.

A commonly used tool within CPA is PERT (Program/Programme/Project Evaluation and Review Technique) which is a specialised method for identifying related and interdependent activities and events, especially where a big project may contain hundreds or thousands of connected elements. PERT is not normally relevant in simple projects, but any project of considerable size and complexity, particularly when timings and interdependency issues are crucial, can benefit from the detailed analysis enabled by PERT methods. PERT analysis commonly feeds into CPA and to other broader project management systems, such as those mentioned here.

Critical path analysis flow diagrams are very good for showing interdependent factors whose timings overlap or coincide. They also enable a plan to be scheduled according to a timescale. CPA flow diagrams also enable costings and budgeting, although not quite as easily as Gantt charts (below), and they also help planners to identify causal elements, although not quite so easily as fishbone diagrams in figure 3.1 above.

Creating Critical Path Analysis

As an example, the project is a simple one; making a fried breakfast. First note down all the issues (resources and activities in a rough order), again for example:

- Assemble crockery and utensils, assemble ingredients, prepare equipment, make toast, fry sausages and eggs, grill bacon and tomatoes, lay table, warm plates, serve.

Note that some of these activities must happen in parallel - and they are crucially interdependent. This means, if you tried to make a fried breakfast by doing one task at a time, and one after the other, things would go wrong. Certain tasks must be started before others, and certain tasks must be completed in order for others to begin. The plates need to be warming while other activities are going on. The toast needs to be done while the sausages are frying, and at the same time the bacon and sausages are under the grill. The eggs need to be fried last. A critical path analysis is a diagrammatical representation of what needs done and when. Timescales and costs can be applied to each activity and resource. Here is the critical path analysis for making a fried breakfast:

- This critical path analysis example below shows just a few activities over a few minutes. Normal business projects would see the analysis extending several times wider than this example, and the time line would be based on weeks or months. It is possible to use MS Excel or a similar spreadsheet to create a critical path

analysis, which allows financial totals and time totals to be planned and tracked. Various specialised project management software enable the same thing. Beware, however, of spending weeks on the intricacies of computer modelling, when in the early stages especially, a carefully hand drawn diagram - which requires no computer training at all - can put 90% of the thinking and structure in place and available for just a tiny fraction of the price of all the alternatives.).

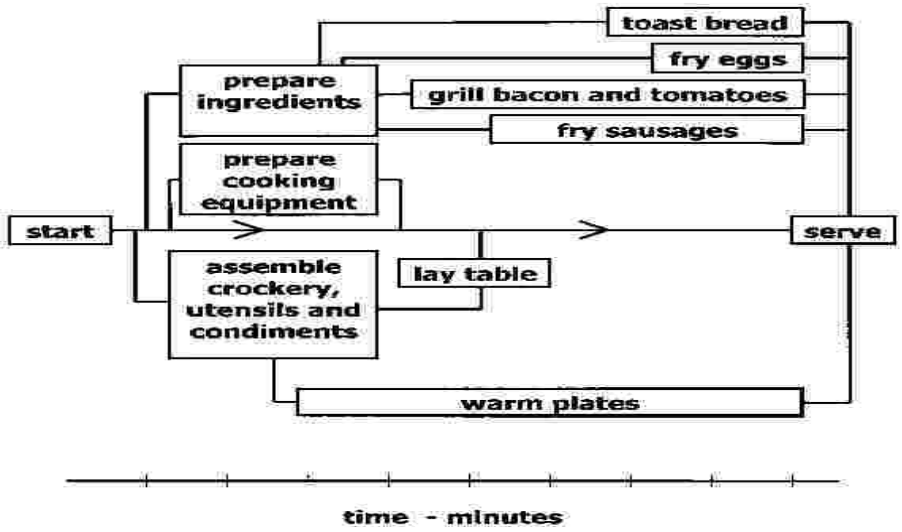

Fig. 3.2: Critical Path Analysis Flow for Making a Fried Breakfast

3.2.4 Gantt Charts

Gantt charts (commonly wrongly called gant charts) are extremely useful project management tools. The Gantt chart is named after US engineer and consultant Henry Gantt (1861-1919), who devised the technique in the 1910s. Gantt charts are excellent models for scheduling and for budgeting, and for reporting and presenting and communicating project plans and progress easily and quickly, but as a rule Gantt charts are not as good as a critical path analysis flow diagram for identifying and showing interdependent factors, or for 'mapping' a plan from and/or into all of its detailed causal or contributing elements.

You can construct a Gantt chart using MS Excel or a similar spreadsheet. Every activity has a separate line. Create a time-line for the duration of the project (the breakfast example shows minutes, but normally you would use weeks, or months for very big long-term projects). You can colour code the time blocks to denote type of activity (for example, intense, watching brief, directly managed, delegated and left-to-run, etc.) You can schedule review and insert break points. At the end of each line you can show as many cost columns for the activities as you need. The breakfast example shows just the capital cost of the

consumable items and a revenue cost for labour and fuel. A Gantt chart like this can be used to keep track of progress for each activity and how the costs are running. You can move the time blocks around to report on actuals versus planned, and to re-schedule, and to create new plan updates. Costs columns can show plan and actuals and variances, and calculate whatever totals, averages, ratios, etc., that you need. Gantt charts are probably the most flexible and useful of all project management tools, but remember they do not very easily or obviously show the importance and inter-dependence of related parallel activities, and they will not obviously show the necessity to complete one task before another can begin, as a critical path analysis will do, so you may need both tools, especially at the planning stage, and almost certainly for large complex projects.

Example of Gantt chart is as follows:

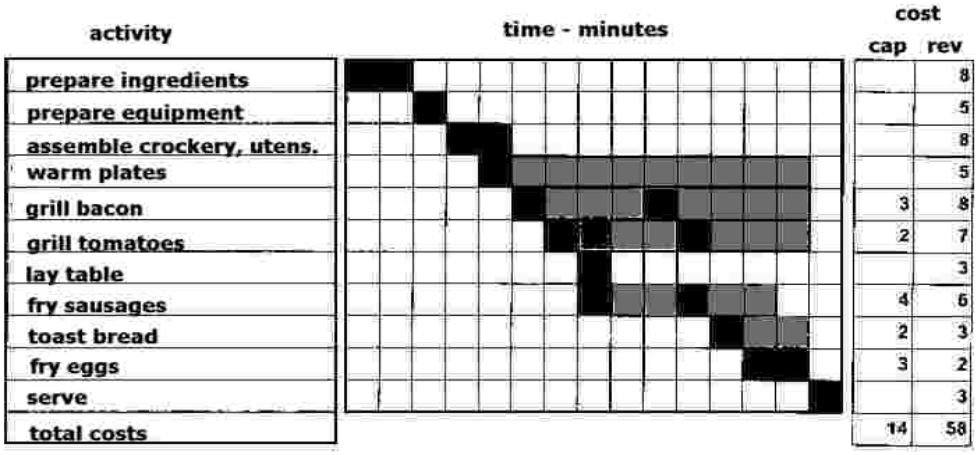

Fig. 3.3: Sample Gantt Chart for Toasting Bread

A wide range of computerised systems/software now exists for project management and planning, and new methods continue to be developed. Project planning tools naturally become used also for subsequent project reporting, presentations, etc., and you will make life easier for everyone if you use formats that people recognise and find familiar.

SELF-ASSESSMENT EXERCISE 1

1. How fishbone is diagram a tool for project planning and project planning?
2. Brainstorming is usually the first crucial creative stage of the project management and project planning process. Why?

3.3 Project Management Process

The following are the processes in software project management:

- Agreed precise specification for the project
- Plan the project
- Communicate the project plan to your project team
- Agree and delegate project actions.
- Manage and motivate
- Check, measure, monitor, review project progress
- Complete project
- Project follow-up.

3.3.1 Agreed Precise Specification for the Project (Terms of Reference)

The project specification should be an accurate description of what the project aims to achieve, and the criteria and flexibilities involved, its parameters, scope, range, outputs, sources, participants, budgets and timescales. It should also be specified with superiors, or with relevant authorities. The specification may involve several drafts before it is agreed. A project specification is essential because it creates a measurable accountability for anyone wishing at any time to assess how the project is going, or its success on completion. Project terms of reference also provide an essential discipline and framework to keep the project on track, and concerned with the original agreed aims and parameters. A properly formulated and agreed project specification also protects the project manager from being held to account for issues that are outside the original scope of the project or beyond the project manager's control.

This is the stage to agree special conditions or exceptions with those in authority. Once you have published the terms of reference you have created a very firm set of expectations by which you will be judged. So if you have any concerns, or want to renegotiate, this is usually the time to do that. For big projects, the terms of reference may require several weeks to produce and agreed on. Most normal business projects however require a few days to think and consult in order to produce a suitable project specification. Establishing and agreeing on a project specification is an important process even if your task is simple one.

A template for a project specification:

- describe purpose, aims and deliverables
- state parameters (timescales, budgets, range, scope, territory, authority)
- state people involved and the way the team will work (frequency of meetings, decision-making process)
- establish 'break-points' at which to review and check progress, and how progress and results will be measured.

3.3.2 Plan the Project

During this period, plan the various stages and activities of the project. Where possible (and certainly where necessary), involve your team in the planning. A useful tip is to work backwards from the end aim, identifying all the things that need to be put in place and done, in reverse order. Additionally, from the bare beginnings of the project, use brainstorming (noting ideas and points at random - typically with a project team), to help gather points and issues and to explore innovations and ideas. Fishbone diagrams (diagrams used for identifying hidden factors which can be significant in enabling larger activities, resources areas, or parts of a process) are also useful for brainstorming and identifying causal factors which might otherwise be forgotten.

For complex projects, or when you lack experience of the issues, involve others in the brainstorming process. Thereafter is a question of putting the issues in the right order, and establishing relationships and links between each issue. Complex projects will have a number of activities running in parallel. Some parts of the project will need other parts of the project to be completed before they can begin or progress. Such 'interdependent' parts of a project need particularly careful consideration and planning. Some projects will require a feasibility stage before the completion of a detailed plan. Gantt charts and critical path analysis flow diagrams are two commonly used tools for detailed project management planning, enabling scheduling, costing and budgeting and other financials, and project management and reporting.

Project contingency planning is also part of the project plan. Planning for and anticipating the unforeseen, or the possibility that things may not go as expected, is called 'contingency planning'. Contingency planning is vital in any task when results and outcomes cannot be absolutely guaranteed. Often a contingency budget needs to be planned as there are usually costs associated. Contingency planning is about preparing fall-back actions, and making sure that leeway for time, activity and resource exists to rectify or replace first-choice plans. A simple contingency plan

for the fried breakfast would be to plan for the possibility of breaking the yolk of an egg, in which case spare resource (eggs) should be budgeted for and available if needed. It may be difficult to anticipate precisely what contingency to plan for in complex long-term projects, in which case simply a contingency budget is provided, to be allocated later when and if required.

3.3.3 Communicate the Project Plan to your Team

This serves two purposes: it informs people what is happening, and it obtains essential support, agreement and commitment. If your project is complex and involves a team, then you should involve the team in the planning process to maximise buy-in, ownership, and thereby accountability. Your project will also benefit from input and consultation from relevant people at an early stage.

Also consider how best to communicate the aims and approach of your project to other people in your organisation and wider network. Your project 'team' can extend more widely than you might first imagine. Consider all the possible 'stakeholders' - those who have an interest in your project and the areas it touches and needs to attract support or tolerance. Involvement and communication are vital for cooperation and support. Failing to communicate to people (who might have no great input, but whose cooperation is crucial) is a common reason for arousing suspicion and objections, defensiveness or resistance.

3.3.4 Agree and Delegate Project Actions

Having identified those responsible for each activity in your plan, the next action is to clearly describe all the activities, including all relevant parameters, timescales, costs, and deliverables. Use of proper delegation methods is vital for successful project management involving teams. Delegated tasks may fail if they have not been explained clearly, agreed with the other person, or supported and checked while in progress. So publish the full plan to all in the team, and consider carefully how to delegate medium-to-long-term tasks in light of team members' forward-planning capabilities. Long-term complex projects need to be planned in more detail and great care must be taken in delegating and supporting them.

3.3.5 Manage, Motivate, Inform, Encourage, Enable the Project Team

As a project manager, you should be able to manage the team and activities in meetings, communicate, support, and help with decisions (but not making them for people who can make them for themselves). 'Praise loudly; blame softly’; a wonderful maxim attributed to Catherine the Great. One of the big challenges for a project manager is deciding how much freedom to give for each delegated activity. Tight parameters and lots of checking are necessary for inexperienced people who like clear instructions, but this approach is the kiss of death to experienced, entrepreneurial and creative people. They need a wider brief, more freedom, and less checking. Manage these people by the results they get - not how they get them. Look out for differences in personality and working styles in your team. Misunderstanding personal styles can get in the way of team cooperation. Your role here is to enable and translate. Face to face meetings, when you can bring team members together, are generally the best way to avoid issues and relationships becoming personalised and emotional. Communicate progress and successes regularly to everyone. Give the people in your team the plaudits, particularly when someone high up expresses satisfaction - never accept plaudits yourself. Conversely, you must take the blame for anything that goes wrong and never 'dump' (your problems or stresses) on anyone in your team. As project manager, any problem is always ultimately down to you anyway. Look out for signs of stress and manage it accordingly. A happy positive team with a basic plan will outperform a miserable team with a brilliant plan, every time.

3.3.6 Check, Measure, and Review Project Performance

Check the progress of activities against the plan. Review performance regularly and at the stipulated review points, and confirm the validity and relevance of the remainder of the plan. Adjust the plan if necessary in light of performance, changing circumstances, and new information, but remain on track and within the original terms of reference. Be sure to use transparent, pre-agreed measurements when judging performance. (This shows how essential it is to have these measures in place and clearly agreed before the task begins.) Identify, agree and delegate new actions as appropriate. Inform team members and those in authority about developments, clearly, concisely and in writing. Also plan team review meetings and stick to the monitoring systems you established. Probe the apparent situations to get at the real facts and figures. Analyse causes and learn from mistakes. Identify reliable advisors and experts in the team and use them. Keep talking to people, and make yourself available to all.

3.3.7 Complete Project; Review and Report on Project

At the end of your successful project, hold a review meeting with the team. Ensure you understand what happened and why. Reflect on any failures and mistakes positively, objectively, and without allocating personal blame. Reflect on successes gratefully and realistically. Write a review report, and make observations and recommendations about follow up issues and priorities.

3.3.8 Follow up - Train, Support, Measure and Report Project Results and Benefits

Traditionally, this stage would be considered part of the project completion, but increasingly an emphasized additional stage of project follow-up is appropriate. This is particularly so in very political environments, and/or where projects benefits have relatively low visibility and meaning to stakeholders (staff, customers, investors, etc.), especially if the project also has very high costs, as ICT projects tend to do.

ICT (information and communications technology) projects often are like this - low visibility of benefits but very high costs, and also very high stress and risk levels too.

Project management always involves change management too, within which it is very important to consider the effects of the project on people who have to adapt to the change. There is often a training or education need. There will almost certainly be an explanation need, in which for example methods like team briefing have proved very useful. Whatever, when you are focused on project management it is easy to forget or ignore that many people are affected in some way by the results of the project. Change is difficult, even when it is good and for right reasons. Remembering this during and at the end of your project will help you achieve a project that is well received, as well as successful purely in project management terms. As project manager, to be at the end of a project and to report that the project plan has been fully met, on time and on budget, is a significant achievement, whatever the project size and complexity. The mix of skills required is such that good project managers can manage anything.

SELF-ASSESSMENT EXERCISE 2

1. Why must a project manager first draw a precise specification for any software project?
2. Do you think it is necessary for the project manager to follow up - train, support, measure and report project results and benefits to

stakeholders (staff, customers, investors). What are your reasons?

3.4 Project Management Methodologies

3.4.1 Definition

a. Methodology: A codified set of recommended practices, sometimes accompanied by training materials, formal educational programs, worksheets and diagramming tools.
b. Thick methodology: A methodology that includes a large amount of formal process paperwork and documentation
c. Thin methodology: A methodology that eschews formal process paperwork and documentation commonly used software development methodologies

Let us at this point discuss some methodologies. Methodologies such as the rational unified process (RUP), Structured Systems Analysis and Design Methodology (SSADM), PRINCE2 extreme programming and others are considered thick methodologies. When discussing each of the methodologies, we shall however focus on the management and business focus of the methodology.

3.4.2 Rational Unified Process (RUP) Methodology

In RUP, an iterative approach is used. A software product is designed and built in a succession of incremental iterations. Each iteration includes some, or most, of the development disciplines (requirements, analysis, design, implementation, testing, and so on). Figure 3.4 shows one iteration of a RUP project in a graphical way. One of the major differences between RUP and other methodologies like SSADM is that RUP doesn't use a waterfall approach for software development. The phases of requirements, analysis, design, implementation, integration and testing are not done in strict sequence. RUP is a thick methodology; the whole software design process is described with high detail. RUP is hence particularly applicable on larger software projects. The RUP methodology is general enough to be used out of the box, but the modular nature of RUP. RUP is designed and documented using unified Modeling Language (UML). This also makes it easy to adapt the methodology to the special needs of a single project or company. The Rational unified Process (RUP) is a software design methodology created by the Rational Software Company. The Rational Software Company was acquired by IBM in 2003.

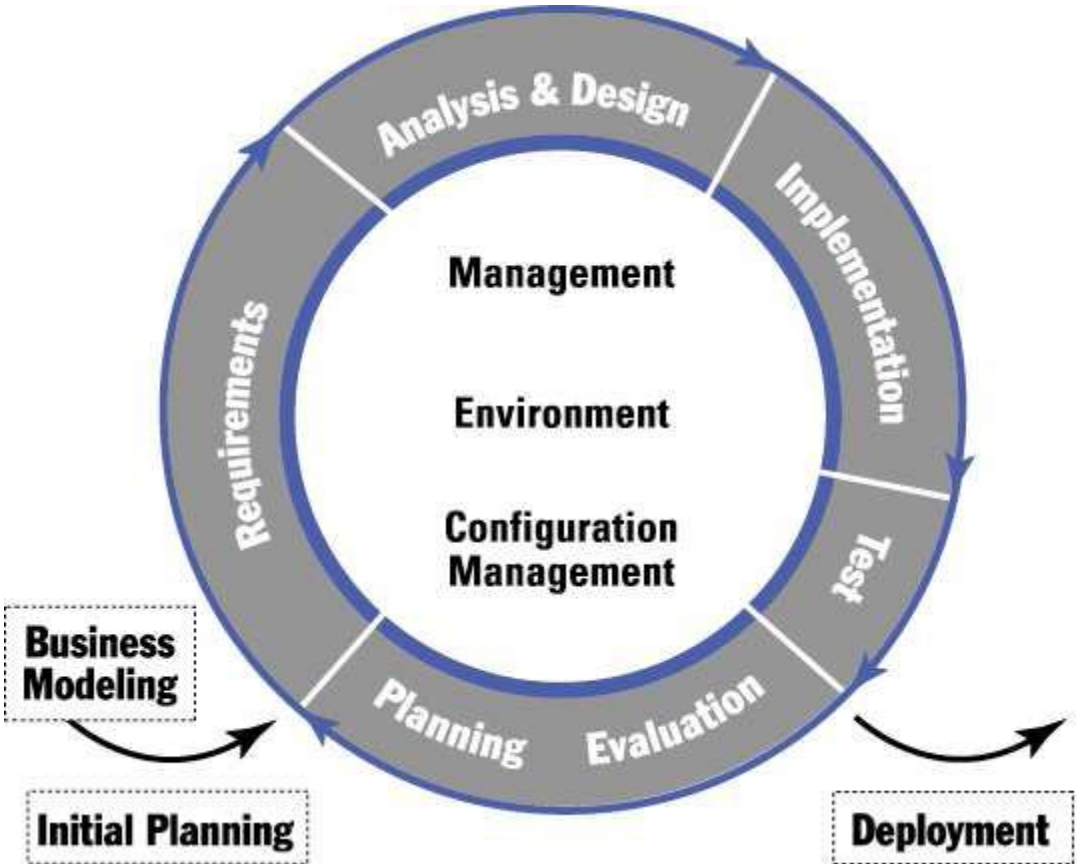

Fig. 3.4: The Iterative Approach to Software Development [RUP-IMG]

Application Area

Due to the modular nature of RUP, it can be used for all sorts of software projects. It is even possible to use RUP for non-software projects. However, because of the complexity of the RUP methodology, it is used mostly for larger software projects.

Advantages

a. The iterative approach leads to higher efficiency. Testing takes place in each iteration, not just at the end of the project life cycle. This way, problems are noticed earlier, and are therefore easier and cheaper to resolve. When using a waterfall approach, it can happen that, for example, software programmers have to wait for the completion of the design phase before starting implementing and integrate the design. Designing and building a software project with an iterative approach solves this problem. Integration and implementation will not only happen at the end of the

project, but at every iteration. This saves time, since more team members can work more.

b. Managing changes in software requirements will be made easier by using RUP. Unless a software project is very small, it is nearly impossible to define all the software requirements at the beginning of a project. It will almost always take more than one step to know what the final software product will look like, for the customer as well as for the project members. Developing with iterations makes this process of changing, requirements, that often leads to missed schedules and dissatisfied customers, less troublesome.

c. RUP itself is software, too, and is distributed in an electronic and online form. Team members don't need to leave their computers for RUP related activities. No more searching in big, dusty books. All information about the software development methodology is available at the project members' fingertips. Also, the newest version of RUP is always present on the computer of each team member. And even more important, it makes sure that every team member is using the same version of RUP. RUP is designed and documented using UML, in an object oriented way. This makes it easy to adapt RUP to the special needs of a single project or organisation.

Disadvantages

a. RUP is a commercial product, no open or free standard. Before RUP can be used, the RUP has to be bought from IBM, as an electronic software and documentation package (a trial version can be downloaded from the IBM website, however). The RUP only exists in an electronic form, which can sometimes limit its use.

b. RUP, as said before, describes the whole software design process with high detail; it is a very complex methodology, difficult to comprehend for both project managers and project members. Therefore, it is not the most appropriate software design methodology for most small projects.

c. Starting to use RUP as software development methodology is difficult everyone participating in the project will have to learn working with RUP.

3.4.3 Structured Systems Analysis and Design Methodology (SSADM)

SSADM sets out a cascade or waterfall view of systems development, in which there are a series of steps, each of which leads to the next step as shown in Fig. 3.5. It was commissioned by the Central Computing and

Telecommunications Agency (CCTA) in a bid to standardise the many and varied IT projects being developed across government departments. Structured Systems Analysis and Design Methodology (SSADM) is a widely used computer application development method in the UK. Its use is often specified as a requirement for government computing projects. Today, it is increasingly being adopted by the public sector in Europe. SSADM is an open standard, which means it is freely available for use in industry and many companies offer support, training and Computer Aided Software Engineering (CASE) tools for it. The series of process in SSADM is shown below: SSADM is a thick methodology of software management.

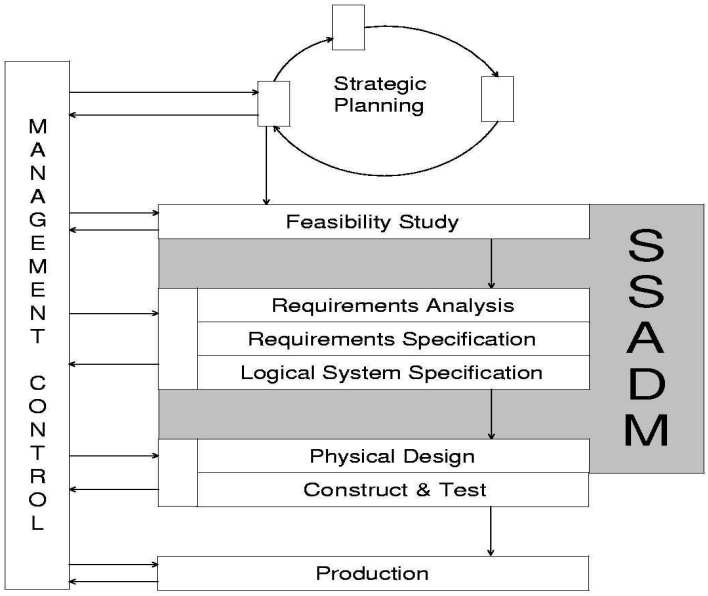

Fig 3.5: SSADM Process Model SSADM

Feasibility Study

The feasibility study consists of one single stage, which involves conducting a high level analysis of a business area to determine whether a system can cost effectively support the business requirements. In the feasibility study, an overview Data Flow Diagram (DFD) is produced together with a high level Logical Data Structure (LDS). At this stage, the DFD will represent the existing system and the LDS may be incomplete and contain unresolved many-to-many relationships.

Requirements Analysis

- **Investigation of the current environment**: During this stage, the systems requirements are identified and the current business

environment is modelled in terms of the processes carried out and the data structures involved. In this, DFDs and LDSs are used to produce detailed logical models of the current system.

- **Business Systems Options (BSO):** During this stage, up to six business systems options are produced and presented and one of these options (chosen option) is adopted and refined. DFDs and LDSs are produced to support each business system option and the final chosen option. The transition from the former stage to this stage is a key part of SSADM: this is where we move from a logical model of the current system to a logical model of the required system. This means the DFDs and LDSs have to be refined to cater for new or changed requirements.

Requirements Specification

The requirements specification consists of a single stage which involves further developing the work carried out in the Requirements Analysis: detailed functional and non-functional requirements are identified and new techniques are introduced to define the required processing and data structures.

Logical Systems Specification

a. **Technical system options**: In this stage, up to six technical options (specifying the development and implementation environments) are produced, one being selected.

b. **Logical design**: In this stage, the logical design of update. enquiry processing and system dialogues (menus etc.) are carried out.

c. **Physical Design**: The physical design consists of a single stage in which the logical system specification and technical system specification are used to create a physical database design and a set of program specification. SSADM revolves around the use of three key techniques:

d. **Logical Data Modelling (LDM):** This is the process of identifying, modelling and documenting the data requirements of a business information system. A LDM consists of a LDS and the associated documentation. LDSs represent Entities (things about which a business needs to record information) and Relationships (necessary associations between entities).

e. **Data Flow Modelling (DFM):** This is the process of identifying, modeling and documenting how data flows around a business information system. A Data Flow Model consists of a set of integrated DFDs supported by appropriate documentation. DFDs represent processes (activities which transform data from one form to another), data stores (holding areas for data), external

entities (things which send data into a system or receive data from a system and finally data flows (routes by which data can flow).

f. **Entity/Event Modelling (EM):** This is the process of identifying, modelling and documenting the business events which affect each entity and life history and appropriate supporting documentation.

Application Area

SSADM was originally developed to standardise the many and varied IT projects being developed across government departments. Today, SSADM *Version 4,* can be used in all kinds of analysis and design stages of system development. SSADM can be used practically for any size of project: small (1-2 persons, less than one man year), medium (4 - 10 persons, 1-20 man years) and large projects. Furthermore, SSADM can be used to develop new projects, but it can also be used to maintain existing systems.

Advantages

i. As mentioned earlier, SSADM is an open standard, which means that it is freely available for use in industry and many companies that offer support, training and CASE tools for it.
ii. SSADM divides an application development project into modules, stages, steps, and tasks, and provides a framework for describing projects in a fashion suited to manage the project.

Objectives of SSADM

SSADM can reduce the chances of initial requirements being misunderstood and of the systems functionality straying from the requirements through the use of inadequate analysis and design techniques.

Disadvantages

i. SSADM is a typical example of a structured methodology, which means that the purpose of it is to:
ii. Formalise the requirements elicitation process to reduce the chances of misunderstanding the requirements. Introduce best practice techniques to the analysis and design process.
iii. As mentioned earlier, SSADM can reduce the chances of initial requirements being misunderstood and of the systems functionality straying from the requirements through the use of inadequate analysis and design techniques. However, SSADM

assumes that the requirements (in the form of an agreed requirements specification) will not change during the development of a project. Following each step of SSADM rigorously can be time consuming and there may be a considerable delay between inception and completion.

3.4.4 Crystal Clear Methodology

The crystal clear methodology is a thin methodology. Crystal clear is a highly optimised way to use a small, collocated team, prioritising for safety in delivering a satisfactory outcome, efficiency in development, and habitability of the working conventions. The crystal clear methodology is part of the crystal family of methodologies, where every methodology is characterised by a colour (Clear, Yellow, Orange, Red, Maroon, Blue, Violet). That colour represents the number of people for which the methodology is suited; crystal clear is the lightest colour and is meant for the smallest project groups, of between two to eight people. Darker colours are for larger groups. Crystal clear has at its core seven properties that should be established for every project that wishes to adhere to the methodology. While all of these are desired, only the first three are mandatory; the other four will get the project further into the safety zone. These seven properties are:

1. **Frequent Delivery**: When delivering is working, tested code to the actual software users once every few months (or more often, if possible), users will be able to deliver feedback on implemented requirements, sponsors will see progress and developers will get a morale boost.
2. **Reflective Improvement**: Taking time to let the team reflect on what works and what does not work for the project, and improving the things that do not work.
3. **Osmotic Communication**: Having the entire team so close together (if possible in the same room, otherwise in adjacent rooms) that people do not have to go to a lot of trouble to raise answer or answer questions, but can do so instantly, will make people work together naturally, inspect each others' work and pick up relevant information as if by osmosis.
4. **Personal Safety:** If people feel safe to speak up without fear of reprisal, they can give constructive criticism on other people's work and admit their own mistakes, leading to honesty and ultimately, trust.
5. **Focus:** If everybody has time to focus on their primary objectives for two hours a day, for two consecutive days every week, without any distractions that can make them lose their train of thought (like meetings or other work), people will be more focused and work will be finished quicker.

6. **Easy Access to Expert Users**: If expert users are available to the team, they can answer questions and deliver feedback on quality and design decisions.
7. **Technical Environment with Automated Tests, Configuration Management and Frequent Integration**: A proper technical environment where testing and configuration management/version control tasks (like making backups and merging changes) do not have to be done by hand will make life easier for developers.

Crystal clear offers several concrete procedures/techniques that can help establish these critical properties, but these are optional: If the team knows of other ways to satisfy the properties, there is nothing that stands in their way. In general, it can be said that crystal clear values properties over techniques. This also makes crystal clear a low threshold methodology: project groups can carry over their established methods and techniques which the group has either grown into or were developed to fit their specific situation to crystal clear, and thus will not have to learn a set of new ones before coming up to speed.

Application Area

As explained above, crystal clear is meant for project groups consisting of two to eight people working at the same physical location, with one or more expert users available. In general, this means any setting where the first three (but ideally, all seven) of the properties can be fulfilled are applicable.

However, the above does not have to be strictly adhered to. All methodologies in the Crystal family support the stretch to fit principle, which states that when a potential project does not fit within the target methodology, the principles and practices to be carried out by the methodology can be stretched to fit the particular case. For example, teams that are significantly larger than eight people have carried out crystal clear successfully by stretching it to fit their needs.

Advantages

a. Because the seven properties are based upon behaviour that has been observed in successful project groups, those practicing crystal clear might well be on the right track to bringing the project to an end successfully. While this is of course no guarantee of success, there are always other factors that contribute to or detract from a project's success. It is likely that these properties contain at least some quality that does indeed

make the difference between a successful and an unsuccessful project.

b. Unlike traditional, thick methodologies like SSADM or PRINCE, crystal clear is flexible as to what project teams are supposed to do and how to do it. This is expressed in the properties over techniques and stretch to fit principles. In fact, crystal clear was explicitly designed to be usable by as many project groups as possible, with the least number of new techniques to learn. It differs in this respect even from a fellow agile methodology.

Disadvantages

a. One of crystal clear's major strengths is also its principal disadvantage: It tries to be a methodology that is applicable in as many cases as possible. This clearly prevents it from ever being a “best" methodology (like XP strives to be)

b. Another disadvantage might be that crystal clear is still relatively new: Most of the principles behind the methodology are all based on real experiences drawn from real projects, so perhaps wider exposure will reveal that crystal clear indeed works “as advertised".

Coupling with other methodologies

Because crystal clear is such a lightweight methodology, it can be coupled with several other methodologies to reap the benefits of both. This can be done in one of two ways: either by adding one or more techniques from the other methodology to crystal clear, or by merging both methodologies to practice them at the same time. As might be expected, the first is easier to attain than the second.

SELF-ASSESSMENT EXERCISE 3

1. Explain any four (4) of the seven properties of the crystal clear methodology
2. Draw the SSADM Process Model.

3.5 Software Management Techniques

Software management techniques are ways of efficiently acquiring information of a software project in a manner that is not immediately obvious or straightforward. These techniques can be used as an aid to estimate, track and evaluate different aspects of the project. Some of the software management techniques to be discussed include discuss PMBOK. COCOMO, MTA, EV and Critical Path.

3.5.1 Constructive Cost Model (COCOMO)

a. COCOMO is an empirical, algorithmic model for estimating the effort, schedule and costs of a software project. It was derived by collecting relevant data from a large number of software projects, then analysing the data to discover the formulae that were the best-fit to the observations (CCM-SWENG, p. 522). The first version of the COCOMO model (now known as COCOMO 81) was a three-level model where the levels reflected the detail of the analysis of the cost estimate. The first level (basic) provided an initial rough estimate; the second level modified this using a number of project and process multipliers and the most detailed level produced estimates for different phases of the project.

b. COCOMO 81 makes various assumptions about the software development process in order to produce its estimates. The latter will only be somewhat accurate when the project uses the waterfall process model and every line of code is produced from scratch. It also fails to take into account that nowadays higher-level programming languages are employed, supported by various automated tools. We will not elaborate on this version since it has been obsoleted by COCOMO 2. COCOMO 2 includes support for various development methodologies such as component-based development and prototyping, fourth generation programming languages and CASE support tools. COCOMO 2 still consists of three levels, but these have been given slightly different interpretations:

The early prototyping level: Size estimates are based on object points. These object points are a simple way of quantifying the perceived complexity of requirements that need to be implemented. The required effort is then computed by applying a simple extrapolation from the object points and programmer productivity. Object points are based on the number of screens, reports and modules in third generation programming languages, and can be weighed by the perceived complexity of the screen, report or module in question.

The early design level: This level corresponds to the completion of the system requirements with (perhaps) some initial design. Estimates are based on function points, which are obtained by working out the object points in detail. More specifically, the total number of points is computed by measuring or estimating the following program features: external inputs and outputs, user interactions, external interfaces and files used by the system. The function points are then converted to number of lines of source code using the tables provided by the COCOMO model.

The post-architecture level: Once the system architecture has been designed a reasonably accurate estimate of the software size can be made. The estimate as this level uses a more extensive set of multipliers reflecting personnel capabilities, product and project characteristics [CCM-SWENG, p. 523{524

Application Area

COCOMO is a well-known empirical algorithmic cost estimation technique. It is well-documented, in the public domain and is supported by public domain and commercial tools. It has been widely used and has a long pedigree from its first instantiation in 1981 .The application of the first instantiation of the model was limited due to the rather large constraints on the development process. This issue has been mitigated by continued improvements on, and extensions of the model, resulting in COCOMO 2.

Advantages

a. Although it's hard to pinpoint the exact cost of any given project, one can still obtain usable data by calculating optimistic and pessimistic estimates. Implementation and execution of the model is very simple and efficient. As a result, it is supported by public as well as commercial tools.
b. COCOMO is a well-known and well-documented technique.

Disadvantages

a. It is quite difficult to come up with satisfactory estimates for the size of a project when the latter still in an early stage of development.
b. The use of the number of lines of source code as a measure of complexity is highly disputable. Even though COCOMO tries to take this into account by providing different tables for all major programming languages, there are still lots of inconsistencies such as: expressivity differences between programmers, usage of subroutines, general code reuse, etc.

Usage in Methodologies

As stated above, the COCOMO model can only be applied when the project in question satisfies a given number of criteria. Additionally, it is advisable to try out other estimation techniques in order to get a feeling of the accuracy of the estimates that have been obtained.

3.5.2 Project Management Body of Knowledge (PMBOK)

The project management body of knowledge (PMBOK) is an inclusive term that describes the sum of knowledge within the profession of project management (PM). As with other professions such as law, medicine, and accounting, the body of knowledge rests with the practitioners and academics that apply and advance it. The full PMBOK includes knowledge of proven traditional practices that are widely applied, as well as knowledge of innovative and advanced practices that have seen more limited use, and includes both published and unpublished material.

The PMBOK framework splits the project processes into five distinct process groups: initiating, planning, executing, controlling and closing. Note that these groups do not imply that the project has to go through each one in this order; they are only provided in order to be able to structure and categorise the different project processes. PMBOK also identifies several project knowledge areas: integration management, scope management, time management, cost management, quality management, human resource management, communications management, risk management and procurement management. By using this twin categorisation in process groups and knowledge areas, we can classify project processes as shown in table 3.1.

Table 3.1: Summary Table for Project Management Body of Knowledge (PMBOK)

Knowledge Areas / Process Groups	Initiating	Planning	Executing	Controlling	Closing
Project Integration management		Project plan development	Project plan execution	Integrated change control	
Project scope management	Initial scope definition	Scope planning	Scope change control	Scope verification	
Project time management		Activity Definition Activity Sequencing Activity Duration Estimating Schedule Development		Schedule Control	
Project cost management		Resource Planning Cost Control Cost Budgeting		Cost control	
Project quality management		Quality planning	Quality assurance	Quality control	
Project human resource management		Organisational planning	Team development		
Project communication		Communications planning	Information distribution	Performance reporting	Administrative

s management					closure
Risk project management		Risk Management Planning Risk Identification and Qualitative Risk Analysis Quantitative Risk Analysis Risk Response Planning		Risk Monitoring Control	
Project procurement management		Procurement Planning Solicitation Planning	Solicitation Source Selection Contract Administration		Contract closeout

Application Area

PMBOK tries to reflect the growth of knowledge and practices in the field of project management by capturing those practices, tools, techniques and other relevant items that have become generally accepted. Being generally accepted does not mean the knowledge and practices described in the PMBOK framework are or should be applied uniformly on all projects.

Advantages

a. PMBOK provides a general project management framework in the form of process groups and knowledge areas.
b. PMBOK gives a concise summary of and reference to generally accepted project management principles.
c. PMBOK proposes a unified project management terminology.

Disadvantages

a. PMBOK is only a framework; the actual needs of the project in question should be determined by a knowledgeable managerial team.
b. PMBOK provides minimal coverage of various project management methodologies and techniques. One definitely needs to consult specialised texts on these subjects in order to learn the ins and outs.
c. PMBOK only covers those aspects of the project management process that are profession independent.

Usage in Methodologies

Since PMBOK is really a collection of generally accepted project management techniques, these techniques can easily be integrated in other methodologies when applicable.

3.5.3 Milestone Trend Analysis (MTA)

MTA is a software engineering technique for evaluating the actual progress of a project in relation to its planning. This relatively simple technique consists of recording the dates of the milestone deadlines at the times they are changed, i.e. when they are postponed or advanced. In this way, one gets a matrix of data: the columns of the matrix delimit the project milestones, the rows, the dates on which the deadlines were re-evaluated, while an actual cell contains the new deadline estimate for the milestone in question. Of course, one can greatly enhance insight in these data by using some simple visualisation techniques. This can be done by plotting the estimated deadlines against the dates on which they were evaluated. The latter are usually placed on the X-axis while the former is on the Y-axis. The evolution of a project milestone deadline is thus visible as a curve on the graph: downward movement of the curve signifies that the deadline in question was advanced, while upward movement means postponement. One can also easily spot milestone completion: this is the case when the curve intersects the line y = x. The general shape of the graph is often roughly triangular: this is the result of the fact that we stop plotting a curve when the milestone in question has reached completion, i.e. when it intersects with the angular bisector of the first quadrant. An example of a typical MTA chart can be seen in the figure below.

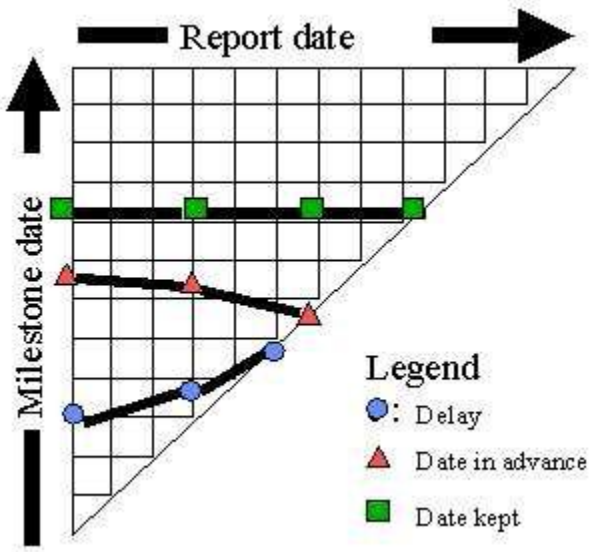

Fig. 3.6: MTA Chart (MTA-SAP)

Application Area

MTA can be applied to every project that uses milestones as the major indicators of progress. It is in essence a very simple and elegant technique that can easily be applied to assess progress. Of course, MTA is an evaluation technique that is to be employed during the execution of a project. Its major uses are preventing and correcting schedule slippage, and post-mortem schedule evaluation.

Advantages

a. MTA is a simple, elegant and effective technique.
b. MTA is widely used and supported.
c. MTA has a large application area.

Disadvantages

a. MTA in itself does not keep track of inter-package dependencies. Therefore, when a certain milestone completion date is altered, one need to make sure its dependencies is altered as well. This does not prove to be much of a problem in practice however, since MTA is available as a plugin for more comprehensive project management tools that can keep track of dependencies.
b. The inputs of the MTA technique are of course estimates of milestone completion deadlines. As such, it is imperative these estimates are made by knowledgeable and experienced engineers. MTA will not be of much use if these estimates are not reasonably accurate.

Usage in Methodologies

As stated above, the only prerequisite is that the project under scrutiny uses milestones. MTA does not impose any further restrictions on the process model and can help to clarify progress assessment in almost any project.

3.5.4 Earned Value (EV) Management

In earned value management, the progress of a project is estimated by comparing what already has been done with the estimates that were made at the beginning of a project. By extrapolating these measurements, a project manager can judge how much resources will be used at the end of a project.

Before we move further, let us look at some common acronyms that are used in the EV management:

Table 3.2: Common Acronyms that are used in the EV Management

Acronym	Meaning
BCWS	Budgeted Cost for Work Scheduled
BCWP	Budgeted Cost for Work Performed
ACWP	Actual Cost of Work Performed
BAC	Budget At Completion
EAC	Estimate At Completion
BCWP	Budgeted Cost for Work Performed

As shown in table 3.2 above, common acronyms that are used in the EV management include:

- Budgeted Cost for Work Performed (BCWP) is also known as the earned value: This value shows what a project has really earned at a certain point in time. The cost of an amount of work can be expressed in different ways; it could be in dollars or hours.

Furthermore, one has to choose when something has been earned. It can be chosen to only set something to be earned when the full task is done. Or say that the part of the task that already had been done has been earned. In the last case, the problem is estimating how far a task has progressed is difficult. In the first case, the problem is that work on a task will skew the figures a little until the task has been done. For example when 95% of the work has been done the earned value of that task is still zero, while there is a significant amount of spent value on it.

EV indicators

- **Cost Variance: CV = BCWP - ACWP**

This shows the difference between the budgeted cost for a certain amount of work (BCWP) and the real cost of an amount of work (ACWP). A negative number indicates that the cost has been underestimated, while a positive number indicates that the cost has been overestimated.

- **Schedule Variance: SV = BCWP - BCWS**

This shows the difference between what has been earned at a certain and what should have been earned.

- **Budget Remaining: BR = BAC – ACWPcumulative**

This shows the amount of budget that is still available to complete the project.

- **Work Remaining: BCWR = BAC – BCWPcumulative**

This shows the amount that still has to be earned in this project, thus the work that remains to be done.

- **Variance at Completion: V AC = BAC - EAC**

This shows the difference between the planned cost at the end and the estimated cost at the end. A negative value indicates that the project is costing more than planned and a positive value indicates that it is costing less.

- **Cost Performance Index (Efficiency): CPIe = BCWP/ACWP**

This shows how efficient the project is being done in terms of cost. For example, a value of 2 shows that the project is currently costing half of the amount planned. Or in other words it is being done twice as efficient as estimated.

- **Schedule Performance Index (Efficiency): SPIe = BCWP/BCWS**

This shows how efficient the project is being done in terms of time. For example a value of 2 shows that the project is going twice as fast as estimated.

- **Estimate At Completion: EAC = BAC/CPIe**

This gives a prediction of the cost at the end of the project. The equivalent but longer version EAC = ACWP + (BAC-BCWP)/CPIe is often used in the literature. It is important to note that CPIe is a moving target and changes during the course of the project. Instead of using the CPIe of the whole project, the CPIe of for example, the last three months can be used. This takes the current performance of the project better into account. Of course choosing shorter timespans for the CPIe will increase the influence of short periods of peak performance.

Advantage

a. During a project, a manager can judge if the project is on schedule/budget. If that is not the case, an estimate can be made on how far the project is over budget.

Disadvantages

a. It is very difficult to estimate the real Earned Value at a certain point in time.
b. Wrong estimates of this value can make a project look like it is doing a lot better than it really is (or the other way around).

Usage in Methodologies

EV management can be used to monitor projects where there is planning beforehand when certain goals should be reached. This encompasses most thick methods. Examples include PRINCE2, SSADM.

3.5.5 Critical Path Analysis

The critical path technique operates on a directed acyclic graph that sequentially orders all tasks that need to be completed in the project. We term this graph the project network. An example of a project network can be seen in Figure 3.7. The tasks connected in a project network are typically the terminal elements of a Work Breakdown Structure. The graph specifies the order in which the different tasks need to be completed, and the dependencies between them. Each task has an associated cost in time. The critical path is the longest path from the start of the project to the finish, and its cost is the shortest period in which the project can be completed. Any delay on tasks on the critical path will delay the entire project. In our example, the critical path is (s, b, d, t), with a cost of 60 days.

A related concept is slack; this is the time that a single activity can be delayed, without delaying the project. By definition, the slack of all activities on the critical path is 0.

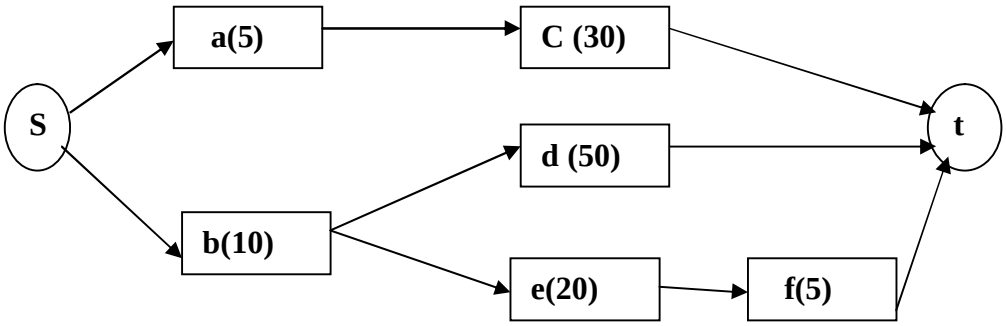

Fig. 3.7: An Example of a Project Network

Application Area

Critical path can be used for task scheduling in just about any project management scheme. However, the grade of dependencies between the tasks must be high enough to make critical path calculation useful. Calculating the critical path for all the deliverables in a (linear) waterfall methodology just would not be all that surprising.

Advantage

a. Critical Path analysis is very clear and unambiguous. It can be used to identify the most important activities, and make sure extra care is given to them. Furthermore, for activities that are not on the critical path, the slack can be calculated and taken into account.

Disadvantage

a. Critical path was designed for routine activities, which can be estimated easily and correctly. Uncertainty about the duration of a task cannot be expressed in the critical path model, and reality can therefore sometimes deviate from the model's predictions.

Usage in Methodologies

For critical path scheduling to be effective, tasks must be known early in advance, and for analysis to be useful, the tasks must have visible dependencies. This makes it unsuitable for methodologies like XP, where activities are small and scheduled only shortly in advance, and tasks have few to no dependencies upon each other.

SELF-ASSESSMENT EXERCISE 4

Why do you think that software management techniques are necessary?

4.0 CONCLUSION

In this unit, you have learnt that project management is the process of planning, organising, staffing, directing and controlling the production of software. You also learnt some project management tools to include brainstorming, fishbone, project critical paths, Gantt chart among others. We went further to look at the project management process. Finally, we concisely reviewed a selection of well-known project management methodologies and techniques.

10.0 SUMMARY

In this unit, you have learnt about:

- the rules, processes and tools for project planning and project management
- project management tools
- project management process
- project management methodologies
- project management techniques.

11.0 TUTOR-MARKED ASSIGNMENT

1. In your own words, explain what you understand by software project management.
2. Give a vivid explanation of the following two project management tools (i) critical path analysis (ii) Gantt charts

3(a) Dwell extensively on any two project management methodologies of your choice
(b) Explain thick and thin methodologies

4(a) Define software management technique
(b) Discuss the following software project management techniques

(i) Milestone trend Analysis (MTA)
(ii) Constructive cost model (COCOMO)
(iii) Earned value (EV) management
(iv) What are the advantages of the critical path technique of project management?

7.0 REFERENCES/FURTHER READING

CC-BOOK Alistair Cockburn | Crystal Clear (due out September 2004, June 2004 Draft Available from http://alistair.cockburn.us/crystal/ books/alistairsbooks.html). Retrieved online 15/10/1010.

[CC-PPP] The Portland Pattern Repository Wiki | Crystal Clear Methodology http://c2.com/cgi-bin/wiki?. Retrieved online 15/10/1010.

CCM-BOEHM B. Boehm, B. Clark, et al. | Cost Models for Future Life Cycle Processes: COCOMO. *Annals of Software engineering*, 1995.

CrystalClearMethodology [CC-WIKI1] Crystal Wiki | Crystal versus XP

http://alistair.cockburn.us/crystal/wiki/FaqCrystalVsXp Retrieved online 15/10/1010.

CC-WIKI2 Crystal Wiki/Crystal Versus RUP http://alistair.cockburn.us/crystal/wiki/FaqCrystalVsRup. Retrieved online 15/10/1010

SSADM-MS Model Systems: SSADM http://www.modelsys.com/msssadm.htm.

Kirchof, N. J. & Adams, J. R. (1986). Conflict Management for Project Managers, Project Management Inst.

Basili, V. & Weiss, D. (1984). A Methodology for Collecting Valid Software Engineering Data. IEEE/ACM Transactions on Software Engineering, SE-10(6): 728-738.

Bradac, M.; *et al.* (1993). Prototyping a Process Monitoring Experiment. In Proceedings of the 15th International Conference on Software Engineering, page 155-165, May 1993.

Kaiser, G. & Feiler, P. (1988). Intelligent Assistance for Software Development and Maintenance. *IEEE Software*, pp. 40-49, May 1988.

CP-MSPROJ Microsoft Project | Critical Path Analysis http://office.microsoft.com/en-us/assistance/HP010404341033.aspx. Retrieved online 15/10/1010

http://evm.nasa.gov/ Earned Value Management

Earned Value Management Web site http://www.acq.osd.mil/pm/