

CIT 731

SOFTWARE ENGINEERING METHODOLOGIES



NATIONAL OPEN UNIVERSITY OF NIGERIA



CIT 731

SOFTWARE ENGINEERING METHODOLOGIES

Course Developer/Writer Vivian Nwaocha
National Open University of Nigeria

Programme Leader Prof. Afolabi Adebajo
National Open University of Nigeria

Course Coordinator Vivian Nwaocha
National Open University of Nigeria



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
Nigeria

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

Published by
National Open University of Nigeria

Printed 2008

ISBN: 978-058-977-5

All Rights Reserved

CONTENTS	PAGE
Introduction.....	1
What this You will Learn in this Course.....	1
Course Aims.....	1
Course Objectives.....	2
Working through this Course.....	2
Course Materials.....	2
Online Materials.....	2
Equipment.....	2
Study Units	4
Assessment.....	5
Course Overview.....	5
How to Get the Most from this Course.....	5
Facilitators/Tutors and Tutorials.....	6
Summary.....	7

Introduction

The course, Software Engineering Methodologies, is a core course for students studying towards attaining the Master of Science in Information Technology. In this course we will study the basic notions of software engineering methodology. Various requirements engineering processes as well as system models are discussed in this course. This course also considers the software life-cycle models, software design, testing and prototyping.

The overall aims of this course are to introduce you to various software life-cycle models. Software design, testing, reuse and prototyping are equally discussed.

In structuring this course, we commence with the basic notions of software engineering methodology and move to the requirements engineering and software life-cycle models, software design, testing and prototyping.

There are six modules in this course, each module consists of 5 units of topics that you are expected to complete in 3 hours. The six modules and their units are listed below.

What You will Learn in this Course

The overall aims and objectives of this course is to provide guidance on what you should be achieving in the course of your studies. Each unit also has its own unit objectives which state specifically what you should be achieving in the corresponding unit. To evaluate your progress continuously, you are expected to refer to the overall course aims and objectives as well as the corresponding unit objectives upon the completion of each.

Course Aims

The overall aims and objectives of this course will help you to:

1. Develop your knowledge and understanding of the underlying principles of software engineering methodology
2. Build up your capacity to evaluate software process models
3. Develop your competence in designing software
4. Build up your capacity to test and reuse software

Course Objectives

Upon completion of the course, you should be able to:

1. Describe the basic concepts of software engineering methodology
2. Identify the software process models
3. Explain the software requirements
4. Describe the requirements validation
5. Discuss the system modelling
6. Explain the architectural design
7. Describe the software life-cycle models
8. Know the system models
9. Describe software design
10. Discuss the notion of software prototyping.

Working through this Course

We designed this course in a systematic way, so you need to work through it from Module one, Unit 1 through to Module three, Unit 10. This will enable you appreciate the course better.

Course Materials

Basically, we made use of textbooks and online materials. You are expected to search for more literature and web references for further understanding. Each unit has references and web references that were used to develop them.

Online Materials

Feel free to refer to the web sites provided for all the online reference materials required in this course.

The website is designed to integrate with the print-based course materials. The structure follows the structure of the units and all the reading and activity numbers are the same in both media.

Equipment

In order to get the most from this course, it is essential that you make use of a computer system which has internet access.

Recommended System Specifications:**Processor**

2.0 GHZ Intel compatible processor
1GB RAM
80 GB hard drive with 5 GB free disk
CD-RW drive
3.5" Floppy Disk Drive
TCP/IP (installed)

Operating System

Windows XP Professional (Service Pack)
Microsoft Office 2007
Norton Antivirus

Monitor*

19-inch
1024 X 768 resolution
16-bit high color
*Non Standard resolutions (for example, some laptops) are not supported.

Hardware

Open Serial Port (for scanner)
120W Speakers
Mouse + pad
Windows keyboard
Laser printer

Hardware is constantly changing and improving, causing older technology to become obsolete. An investment in newer, more efficient technology will more than pay for itself in improved performance results.

If your system does not meet the recommended specifications, you may experience considerably slower processing when working in the application.

Systems that exceed the recommended specifications will provide better handling of database files and faster processing time, thereby significantly increasing your productivity.

Study Unit

Module 1 Introduction to Software Engineering Methodology

Unit 1	Software Engineering Methodology – Basic Notions
Unit 2	Software Process, Professional and Ethical Issues
Unit 3	Software Process- Model
Unit 4	Evolutionary and Incremental Development
Unit 5	Spiral Development and Process Activities

Module 2 Introduction to Software Engineering Methodology

Unit 1	Computer-Aided Software Engineering (CASE)
Unit 2	Software Requirements
Unit 3	Functional and Non-Functional Requirements
Unit 4	Requirements
Unit 5	Domain Requirements

Module 3 Requirements Engineering Processes

Unit 1	Concepts of Requirements Engineering
Unit 2	Viewpoints
Unit 3	Interviewing
Unit 4	Requirements Validation
Unit 5	System Modelling

Module 4 Requirements Engineering Processes

Unit 1	Formal Methods
Unit 2	Specification
Unit 3	Introduction to Architectural Design
Unit 4	Models
Unit 5	Sub-systems and Modules

Module 5 Software Engineering

Unit 1	Software Life-cycle Models
Unit 2	Requirements Engineering
Unit 3	Formal Specification
Unit 4	System Models
Unit 5	Software Design

Module 6 Software Engineering

Unit 1	Software Testing
Unit 2	Software Inspection
Unit 3	Software Reliability

Unit 4	Software Reuse
Unit 5	Software Prototyping

From the preceding, the content of the course can be divided into three major blocks:

1. Introduction to Software Engineering Methodology
2. Requirements Engineering Processes
3. Software Engineering

Module 1 and 2 describes the basic notions of software engineering methodology, Module 3 and 4 explains the processes involved in requirements engineering and Module 5 and 6 discusses the software life-cycle models, software design, reliability, reuse and prototyping.

Assessment

The course, Software Engineering Methodologies entails attending a three-hour final examination which contributes 50% to your final grading. The final examination covers materials from all parts of the course with a style similar to the Tutor-marked assignments.

The examination aims at testing your ability to apply the knowledge you have learned throughout the course, rather than your ability to memorise the materials. In preparing for the examination, it is essential that you receive the activities and Tutor-marked assignments you have completed in each unit. The other 50% will account for all the TMAs at the end of each unit.

Course Overview

This section proposes the number of weeks that you are expected to spend on the three modules comprising of 30 units and the assignments that follow each of the units.

We recommend that each unit with its associated TMA is completed in one week, bringing your study period to a maximum of 30 weeks.

How to Get the Most from this Course

In order for you to learn various concepts in this course, it is essential to practice. Independent activities and case activities which are based on a particular scenario are presented in the units. The activities include open questions to promote discussion on the relevant topics, questions with standard answers and program demonstrations on the concepts. You may try to delve into each unit adopting the following steps:

1. read the study unit
2. read the textbook, printed or online references
3. perform the activities
4. participate in group discussions
5. complete the tutor-marked assignments
6. participate in online discussions

This course makes intensive use of materials on the world-wide web. Specific web addresses will be given for your reference. There are also optional readings in the units. You may wish to read these to extend your knowledge beyond the required materials. They will not be assessed.

Facilitators/Tutors and Tutorials

About 20 hours of tutorials will be provided in support of this course. You will be notified of the dates, time and location for these tutorials, together with the name and phone number of your tutor as soon as you are allotted 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 TMAs to your tutor well before the due date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by phone, e-mail if you need help. The following might be circumstances in which you would find help necessary. You can also contact your tutor if:

- i. you do not understand any part of the study units or the assigned readings
- ii. you have difficulty with the TMAs
- iii. you have a question or problem with your tutor's comments on an assignment or with the grading of an assignment.

You should try your best to attend tutorials, since it is the only opportunity to have an interaction with your tutor and to ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain maximum benefit from the course tutorials, you are advised to prepare a list of questions before attending the tutorial. You will learn a lot from participating in discussions actively.

Summary

The course, Software Engineering Methodologies is intended to develop your understanding of the basic concepts of software engineering methods, thus enabling you understand the requirements of engineering processes. This course also provides you with hands-on experience in designing, testing and reusing software.

We hope that you will find the course enlightening and that you will find it both interesting and useful. In the longer term, we hope you will get acquainted with the National Open University of Nigeria and we wish you every success in your future.



Course Code	CIT 731
Course Title	Software Engineering Methodologies
Course Developer/Writer	Vivian Nwaocha National Open University of Nigeria
Programme Leader	Prof. Afolabi Adebajo National Open University of Nigeria
Course Coordinator	Vivian Nwaocha National Open University of Nigeria



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
Nigeria

e-mail: centralinfo@nou.edu.ng

URL: www.nou.edu.ng

Published by
National Open University of Nigeria

Printed 2008

ISBN: 978-058-977-5

All Rights Reserved

CONTENTS	PAGE
Module 1	1
Unit 1 Software Engineering Methodology – Basic Notions.....	1
Unit 2 Software Process, Professional and Ethical Issues....	6
Unit 3 Software Process – Model.....	11
Unit 4 Evolutionary Incremental Development.....	15
Unit 5 Spiral Development and Process Activities.....	19
 Module 2	 24
Unit 1 Computer-Aided Software Engineering (CASE).....	24
Unit 2 Software Requirements.....	30
Unit 3 Functional and Non-Functional Requirements.....	35
Unit 4 Requirements.....	40
Unit 5 Domain Requirements.....	45
 Module 3	 48
Unit 1 Concepts of Requirements Engineering.....	48
Unit 2 Viewpoints.....	52
Unit 3 Interviewing.....	56
Unit 4 Requirements Validation.....	60
Unit 5 System Modelling.....	64
 Module 4	 69
Unit 1 Formal Methods.....	69
Unit 2 Specification.....	73
Unit 3 Introduction to Architectural Design.....	75
Unit 4 Models.....	81
Unit 5 Sub-Systems and Modules.....	85
 Module 5	 88
Unit 1 Software Life-Cycle Models.....	88
Unit 2 Requirements Engineering.....	97
Unit 3 Formal Specification.....	101
Unit 4 System Models.....	105
Unit 5 Software Design.....	109
 Module 6	 113

Unit 1	Software Testing.....	113
Unit 2	Software Inspection.....	118
Unit 3	Software Reliability.....	122
Unit 4	Software Re-Use	126
Unit 5	Software Prototyping.....	130

MODULE 1 INTRODUCTION TO SOFTWARE ENGINEERING METHODOLOGY

Unit 1	Software Engineering Methodology – Basic Notions
Unit 2	Software Process, Professional and Ethical Issues
Unit 3	Software Process – Model
Unit 4	Evolutionary Incremental Development
Unit 5	Spiral Development and Process Activities

UNIT 1 SOFTWARE ENGINEERING METHODOLOGY-BASIC NOTIONS

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	Software Engineering Methodology
3.2	Software Costs
3.3	Software
3.4	Software Engineering
3.5	Software Engineering vs. Computer Science
3.6	Attributes of a Good Software
4.0	Conclusion
5.0	Summary
6.0	Tutor-Marked Assignment
7.0	References/Further Readings

1.0 INTRODUCTION

This unit introduces some basic concepts that the student needs to be familiar with before attempting to develop any software. It describes software and software engineering methodology, explaining the attributes of good software. The unit introduces you to the fundamental notions of software engineering methodology, thus guiding you through and facilitating your understanding of the subsequent units.

2.0 OBJECTIVES

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

- explain software engineering methodology
- describe software engineering
- explain what a software is
- list the attributes of a good software.

3.0 MAIN CONTENT

3.1 Software Engineering Methodology

The body of methods, rules, postulates, procedures, and processes that are used to manage a software engineering project are collectively referred to as a software engineering methodology.

3.2 Software Costs

Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost. Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs. Software engineering is concerned with cost-effective software development.

3.3 Software

Software simply refers to computer programs and associated documentation such as requirements, design models and user manuals. Software products may be developed for a particular customer or may be developed for a general market. Software products may be

1. Generic - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
2. Bespoke (custom) - developed for a single customer according to their specification.

New software can be created by developing new programs, configuring generic software systems or reusing existing software.

SELF ASSESSMENT EXERCISE 1

Define the term 'software'.

3.4 Software Engineering

Software engineering is an engineering discipline that is concerned with all aspects of software production. Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

3.5 Software Engineering vs. Computer Science

Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software. Computer science theories are still insufficient to act as a complete underpinning for software engineering (unlike e.g. physics and electrical engineering).

3.6 Attributes of a Good Software

The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

- a. **Maintainability:** Software must evolve to meet changing needs
- b. **Dependability:** Software must be trustworthy
- c. **Efficiency:** Software should not make wasteful use of system resources
- d. **Usability:** Software must be usable by the users for which it was designed
- e. **Robustness:** Software should fail only under extreme conditions
- f. **Portability:** Should be possible to move from one environment to another

SELF ASSESSMENT EXERCISE 2

What are the attributes of good software?

4.0 CONCLUSION

In this unit you have learned about software engineering methodology. You have also been able to understand the difference between software engineering and computer science. Finally, you have been able to understand what software is and the attributes of a good software.

5.0 SUMMARY

What you have learned borders on the basic notions of software engineering. The subsequent units shall build upon these fundamentals.

6.0 TUTOR-MARKED ASSIGNMENT

What do you understand by methodology in software engineering?

7.0 REFERENCES/FURTHER READINGS

- Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).
- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers: San Francisco, CA.
- Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, no. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.
- Constantine L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetchbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>

UNIT 2 SOFTWARE PROCESS, PROFESSIONAL AND ETHICAL ISSUES

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Software Process
 - 3.2 What is a Software Process Model?
 - 3.3 Software Engineering Methods
 - 3.4 Professional and Ethical Responsibility
 - 3.5 Issues of Professional Responsibility
 - 3.6 What are CASE Tools
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

In this unit, you will learn about software process and software process model. You will also learn about the professional and ethical responsibilities as well as CASE tools.

2.0 OBJECTIVES

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

- describe a software process
- explain the term ‘software process model
- outline the professional and ethical responsibilities of a software engineer
- describe CASE tools.

3.0 MAIN CONTENT

3.1 Software Process

A software process is a set of activities and associated results whose goal is the development or evolution of software product Generic activities in all software processes are:

Specification - what the system should do and its development constraints

Development - production of the software system

Validation - checking that the software is what the customer wants

Evolution - changing the software in response to changing demands

3.2 What is a Software Process Model?

A software process model is a *simplified* representation of a software process, presented from a specific perspective. Examples of process perspectives are:

1. Workflow perspective - sequence of activities
2. Data-flow perspective - information flow
3. Role/action perspective - who does what

Generic process models include:

Waterfall

Evolutionary development

Formal transformation

Integration from reusable components

3.3 Software Engineering Methods

Software engineering methods refer to structured approaches to software development which include system models, notations, rules, design advice and process guidance

Model descriptions: Descriptions of graphical models which should be produced

Rules: Constraints applied to system models

Recommendations: Advice on good design practice

Process guidance: What activities to follow

SELF ASSESSMENT EXERCISE 1

Explain the concept of rules within the context of software engineering methods.

3.4 Professional and Ethical Responsibilities

Software engineering involves wider responsibilities than simply the application of technical skills.

Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.

Ethical behaviour is more than simply upholding the law.

3.5 Issues of Professional Responsibility

Confidentiality: Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.

Competence: Engineers should not misrepresent their level of competence. They should not knowingly accept work which is not within their competence.

Intellectual property rights: Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.

Computer misuse: Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

3.6 What are CASE Tools?

CASE tools are software systems which are designed to support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run.

SELF ASSESSMENT EXERCISE 2

What are CASE tools?

4.0 CONCLUSION

In this unit you have learned about the software process and software process model. You have also been able to understand the concept of software engineering methods. Finally, you have become aware of the professional and ethical responsibilities of a Software Engineer.

5.0 SUMMARY

What you have learned in this unit is focused on process and process models as well as professional and ethical responsibilities of a software engineer.

6.0 TUTOR-MARKED ASSIGNMENT

Describe a software process model.

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience*, (John Wiley & Sons, Inc.), vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (November 2006). (Eds.), *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Oriented Development: The Fusion Method*, Prentice-Hall: Inc.

Englewood Cliffs, N J, Constantine L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press, Reading, MA.

Grogono, P. (1999). *Software Engineering*, (2nd Edition). New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>

UNIT 3 SOFTWARE PROCESS MODEL – WATERFALL MODEL

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Software Process vs. Software Process Model
 - 3.2 The Waterfall
 - 3.3 The Waterfall Model Phases
 - 3.4 The Waterfall Model Problems
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

What you will learn in this unit borders on software process model, basically the waterfall model.

2.0 OBJECTIVES

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

- distinguish between software process and software process model
- describe the waterfall model
- outline the phases of a waterfall model
- point out the problems of a waterfall model.

3.0 MAIN CONTENT

3.1 Software Process vs. Software Process Model

A software process refers to a structured set of activities required to develop a software system

Specification
Design
Validation
Evolution.

While a software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

3.2 The Waterfall Model

The waterfall model consists of separate and distinct phases of specification and development.

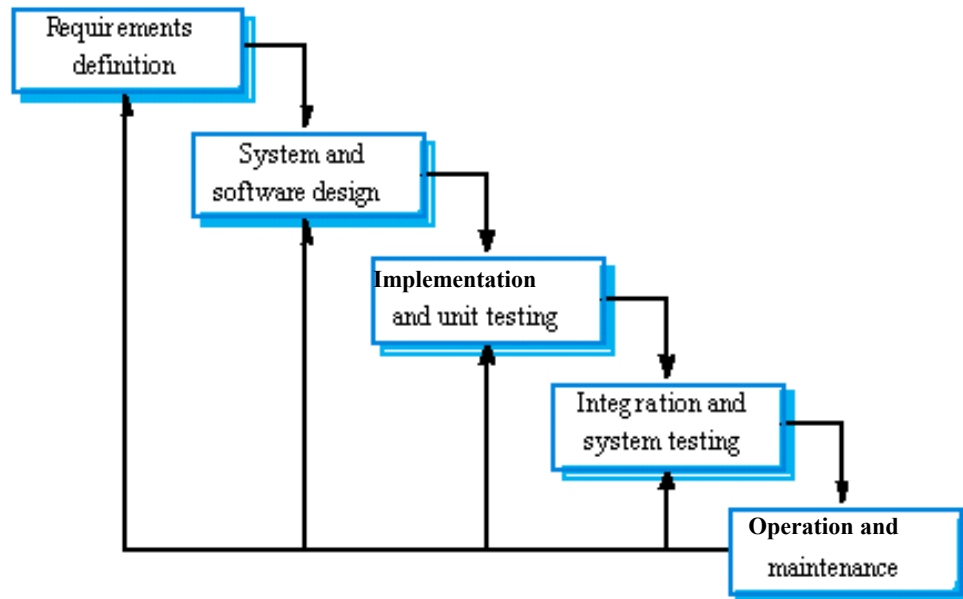


Fig. 1.0: The waterfall model

SELF ASSESSMENT EXERCISE 1

What do you understand by a waterfall model?

3.3 The Waterfall Model Phases

- a. Requirements analysis and definition
- b. System and software design
- c. Implementation and unit testing
- d. Integration and system testing
- e. Operation and maintenance.

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase.

SELF ASSESSMENT EXERCISE 2

Mention at least three problems of a waterfall model.

3.4 Waterfall Model Problems

1. Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
2. Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
3. Few business systems have stable requirements.
4. The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

4.0 CONCLUSION

In this unit you have learned about waterfall model. You have also been able to identify the phases and problems of a waterfall model.

5.0 SUMMARY

What you have learned in this unit concerns the waterfall models, their phases and drawbacks. In the next unit you shall learn about another software process model.

6.0 TUTOR-MARKED ASSIGNMENT

Outline the phases of a waterfall model.

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall,

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>

UNIT 4 EVOLUTIONARY AND INCREMENTAL DEVELOPMENT

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Objective of Evolutionary Development
 - 3.2 Problems of Evolutionary Development
 - 3.3 Applicability of Evolutionary Development
 - 3.4 Incremental Development
 - 3.5 Advantages of Incremental Development
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

In this unit we will look at the objective and applicability of evolutionary development. We will also consider the advantages of incremental development.

2.0 OBJECTIVES

By the end of this unit, the student should be able to:

- describe the objective of evolutionary development
- problems of evolutionary development
- outline the applications of evolutionary development
- point out the advantages of incremental development.

3.0 MAIN CONTENT

3.1 Objective of Evolutionary Development

The main objective of the evolutionary development is to work with customers and to evolve a final system from an initial outline specification. This model starts with well-understood requirements and adds new features as proposed by the customer.

SELF ASSESSMENT EXERCISE 1

By means of a diagram describe incremental development.

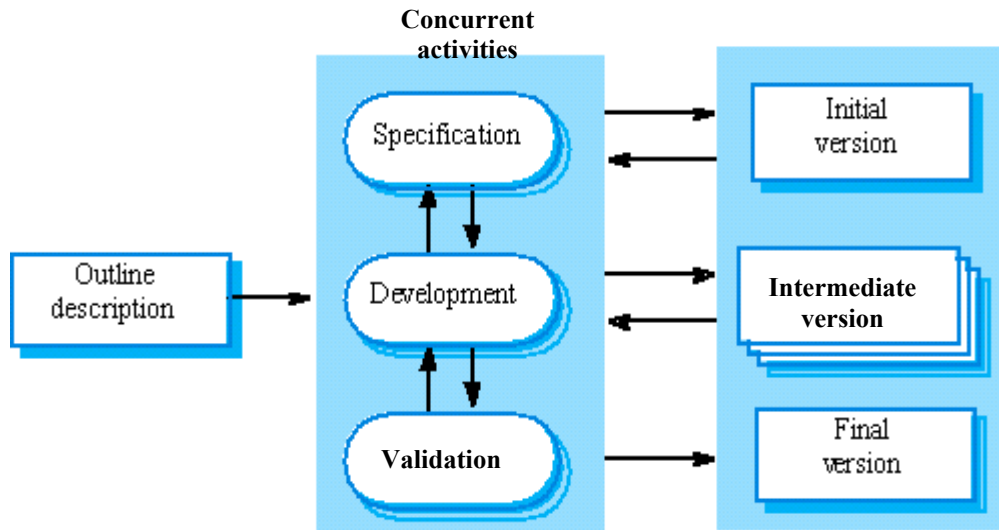


Fig 1.1: Evolutionary development

3.2 Problems of Evolutionary Development

- a. Lack of process visibility;
- b. Systems are often poorly structured
- c. Special skills (e.g. in languages for rapid prototyping) may be required.

SELF ASSESSMENT EXERCISE 2

List some problems of evolutionary development.

3.3 Applicability of Evolutionary Development

- a. For small or medium-size interactive systems
- b. For parts of large systems (e.g. the user interface)
- c. For short-lifetime systems.

3.4 Incremental Development

In the incremental development rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.

User requirements are prioritised and the highest priority requirements are included in early increments. Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

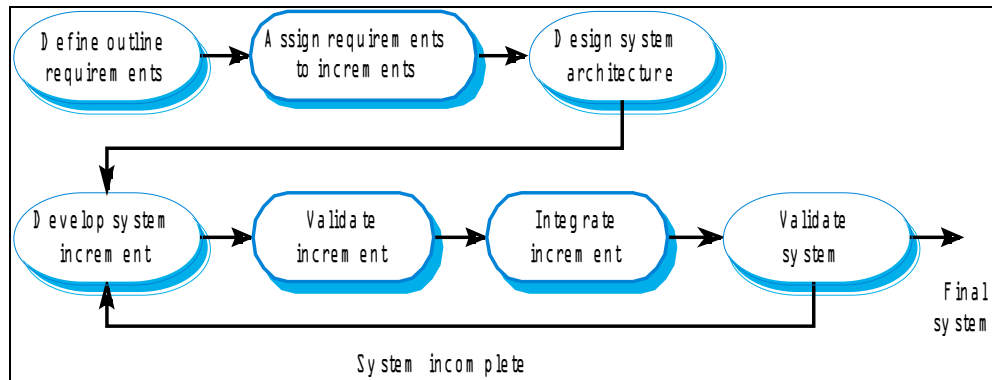


Fig. 1.2: Incremental Development

3.5 Advantages of Incremental Development

1. Customer value can be delivered with each increment so system functionality is available earlier
2. Early increments act as a prototype to help elicit requirements for later increments
3. Lower risk of overall project failure
4. The highest priority system services tend to receive the most testing.

4.0 CONCLUSION

In this unit you have learned about evolutionary and incremental development. You have also learned the advantages of incremental development and the problems of evolutionary development.

5.0 SUMMARY

What you have learned in this unit concerns evolutionary and incremental developments.

6.0 TUTOR-MARKED ASSIGNMENT

The evolutionary development is applicable in some fields. Discuss

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What Characterizes a (Software) Component?*” *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall,

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>

UNIT 5 SPIRAL DEVELOPMENT AND PROCESS ACTIVITIES

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Spiral Development
 - 3.2 Process Activities
 - 3.3 Software Specification
 - 3.4 Requirement Engineering Process
 - 3.5 Software Design and Implementation
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

In this unit the student you will gain knowledge of spiral development as well as process activities. You will also consider software specification, design and implementation.

2.0 OBJECTIVES

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

- describe spiral development
- outline process activities
- explain software specification
- describe software design and implementation.

3.0 MAIN CONTENT

3.1 Spiral Development

In the spiral development, the process is represented as a spiral rather than as a sequence of activities with backtracking. Each loop in the spiral represents a phase in the process. There are no fixed phases such as specification or design – loops. Risks are explicitly assessed and resolved throughout the process.

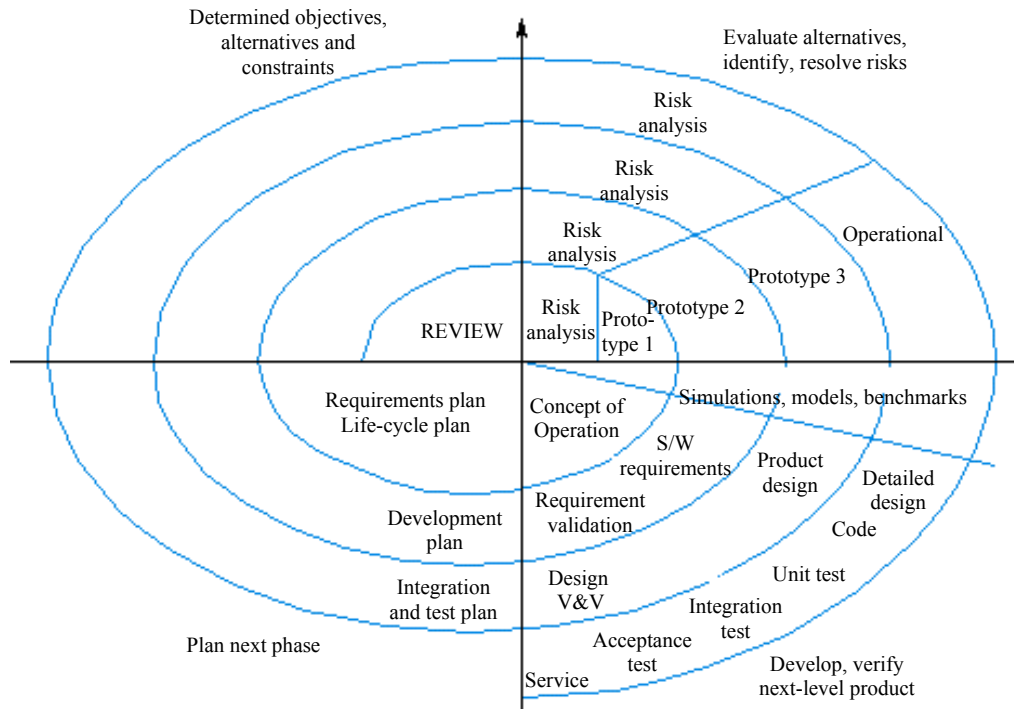


Fig. 1.0: Spiral development

3.2 Process Activities

The Process activities include:

- a. Software specification
- b. Software design and implementation
- c. Software validation
- d. Software evolution.

3.3 Software Specification

This is the process of establishing what services are required and the constraints on the system's operation and development.

3.4 Requirements Engineering Process

The requirements engineering process include:

- i. Feasibility study
- ii. Requirements elicitation and analysis
- iii. Requirements specification
- iv. Requirements validation.

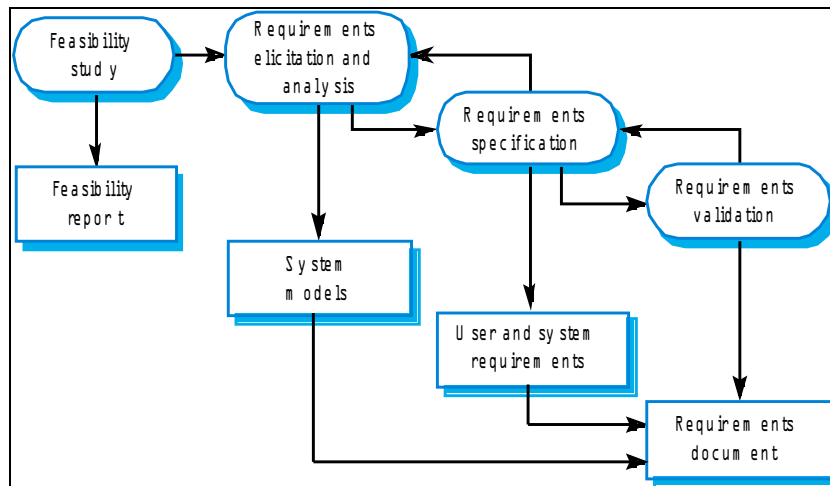


Fig. 1.1: The Requirements Engineering Process

SELF ASSESSMENT EXERCISE 1

What are the requirements for an engineering process?

3.5 Software Design and Implementation

The process of converting the system specification into an executable system is known as software design and implementation. Software design involves designing a software structure that realises the specification; while implementation involves translating this structure into an executable program. The activities of design and implementation are closely related and may be inter-leaved

SELF ASSESSMENT EXERCISE 2

Explain software design and implementation.

4.0 CONCLUSION

In this unit you have learned about the spiral development. Process activities were equally considered. You should also have learned about software specification, design and implementation.

5.0 SUMMARY

What you have learned in this unit concerns spiral development and process activities. The units that follow shall build upon issues discussed in this unit.

6.0 TUTOR-MARKED ASSIGNMENT

By means of a diagram describe the spiral development

7.0 REFERENCES/FURTHER READINGS

- Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).
- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall,

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>

MODULE 2 INTRODUCTION TO SOFTWARE ENGINEERING METHODOLOGY

Unit 1	Computer-Aided Software Engineering (Case)
Unit 2	Software Requirements
Unit 3	Functional and Non-functional Requirements
Unit 4	Requirements
Unit 5	Domain Requirements

UNIT 1 COMPUTER-AIDED SOFTWARE ENGINEERING (CASE)

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	Computer-Aided Software Engineering
3.2	CASE Technology
3.3	CASE Classification
4.0	Conclusion
5.0	Summary
6.0	Tutor-Marked Assignment
7.0	References/Further Readings

1.0 INTRODUCTION

In this unit we examine computer-aided software engineering (CASE). CASE technology and classification are equally described.

2.0 OBJECTIVES

After going through this unit, you should be able to:

- explain the basic idea of computer-aided software engineering
- describe the CASE technology
- give the CASE classification.

3.0 MAIN CONTENT

3.1 Computer-Aided Software Engineering

Computer-aided software engineering (CASE) is software to support software development and evolution processes.

3.2 CASE Technology

Case technology has led to significant improvements in the software process. However, these are not the order of magnitude of improvements that were once predicted.

Indeed, software engineering requires creative thought - this is not readily automated. Software engineering is a team activity and, for large projects, much time is spent in team interactions. CASE technology does not really support these.

3.3 CASE Classification

Classification helps us understand the different types of CASE tools and their support for process activities. CASE can be classified as follows:

- a. Functional perspective: Tools are classified according to their specific function.
- b. Process perspective: Tools are classified according to process activities that are supported.
- c. Integration perspective: Tools are classified according to their organisation into integrated units.

SELF ASSESSMENT EXERCISE 1

List the CASE classification.

SELF ASSESSMENT EXERCISE 2

What does the acronym CASE refer to?

4.0 CONCLUSION

In this unit you have learned about the computer-aided software engineering. You have also been able to understand CASE technology and its classification.

5.0 SUMMARY

What you have learned borders on the basic concept of computer-aided software engineering.

6.0 TUTOR-MARKED ASSIGNMENT

What do you understand by CASE?

7.0 REFERENCES/FURTHER READINGS

- Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).
- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall,

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>

UNIT 2 SOFTWARE REQUIREMENTS

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Requirements Engineering
 - 3.2 What is a Requirement?
 - 3.3 Requirements Imprecision
 - 3.4 Requirements Completeness and Consistency
 - 3.5 Types of Requirements
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

In this unit we consider requirements engineering as well as requirements. In addition, we consider requirements imprecision, completeness and consistency.

2.0 OBJECTIVES

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

- give a basic definition of a requirements engineering
- describe a requirement
- explain requirements imprecision
- list the types of requirements.

3.0 MAIN CONTENT

3.1 Requirements Engineering

Requirements engineering refers to the process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.

The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

3.2 What is a Requirement?

A requirement may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

This is inevitable as requirements may serve a dual function. They may be the basis for a bid for a contract - therefore must be open to interpretation; OR may be the basis for the contract itself - therefore must be defined in detail. Both these statements may be called requirements.

SELF ASSESSMENT EXERCISE 1

What do you understand by requirements engineering?

3.3 Requirements Imprecision

Problems arise when requirements are not precisely stated. Ambiguous requirements may be interpreted in different ways by developers and users.

User intention - special purpose viewer for each different document type;

Developer interpretation - Provide a text viewer that shows the contents of the document.

SELF ASSESSMENT EXERCISE 2

What is the result of requirements imprecision?

3.4 Requirements Completeness and Consistency

In principle, requirements should be both complete and consistent.

Complete: They should include descriptions of all facilities required.

Consistent: There should be no conflicts or contradictions in the descriptions of the system facilities.

In practice, it is impossible to produce a complete and consistent requirements document.

3.5 Types of Requirements

1. User Requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

2. System Requirements

- A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

3. Functional Requirements

- Statements of services the system should provide how the system should react to particular inputs and how the system should behave in particular situations.

4. Non-Functional Requirements

- Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

5. Domain Requirements

- Requirements that come from the application domain of the system and that reflect characteristics of that domain.

4.0 CONCLUSION

In this unit you have learned about requirements engineering. You have also learned about requirements imprecision. Finally, you have been able to learn about types of requirements.

5.0 SUMMARY

What you have learned in this unit is focused on requirements, the common types and requirements imprecision.

6.0 TUTOR-MARKED ASSIGNMENT

List four types of requirements.

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall,

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>

UNIT 3 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Functional Requirements
 - 3.2 Examples of Functional Requirements
 - 3.3 Non-Functional Requirements
 - 3.4 Non-Functional Classifications
 - 3.5 Non-Functional Requirement Types
 - 3.6 Requirements Interaction
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit describes functional and non-functional requirements.

2.0 OBJECTIVES

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

- describe functional requirements
- give Examples of functional requirements
- describe non-functional requirements
- classify non-functional requirements.

3.0 MAIN CONTENT

3.1 Functional Requirements

Functional requirements are requirements that describe functionality or system services. They depend on the type of software, expected users and the type of system where the software is used.

Functional user requirements may be, high-level statements of what the system should do but functional system requirements should describe the system services in detail.

3.2 Examples of Functional Requirements

1. The user shall be able to search either all of the initial set of databases or select a subset from it.
2. The system shall provide appropriate viewers for the user to read documents in the document store.
3. Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area.

3.3 Non-Functional Requirements

These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.

Process requirements may also be specified mandating a particular CASE system, programming language or development method. Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

3.4 Non-Functional Classifications

1. Product Requirements

Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.

2. Organisational Requirements

Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.

3. External Requirements

Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

3.5 Non-Functional Requirement Types

Non-functional requirements can be grouped into types as shown by the figure below:

SELF ASSESSMENT EXERCISE

Describe non-functional requirements

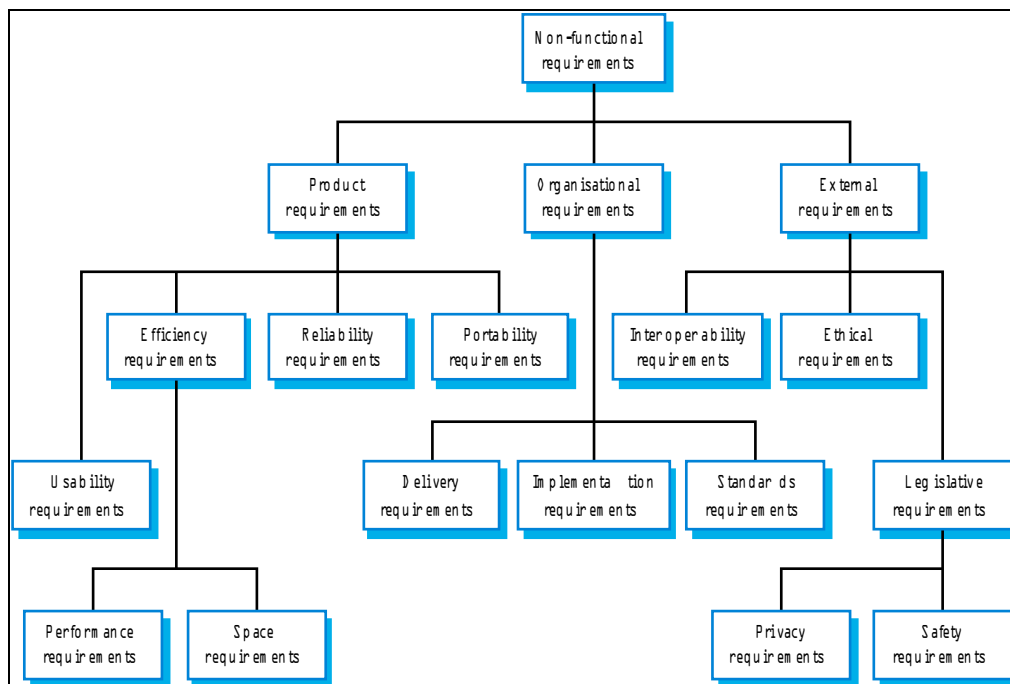


Fig. 1.0: Non-functional requirement types

3.6 Requirements Interaction

Conflicts between different non-functional requirements are common in complex systems.

Spacecraft System

To minimise weight, the number of separate chips in the system should be minimised.

To minimise power consumption, lower power chips should be used.

However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?

4.0 CONCLUSION

In this unit you have learned about functional requirements. You have also learned about non-functional requirements. Finally, you have been able to learn about non-functional requirement types.

5.0 SUMMARY

What you have learned in this unit is focused on functional and non-functional requirements. Functional requirements set out services the system should provide.

Non-functional requirements constrain the system being developed or the development process.

6.0 TUTOR-MARKED ASSIGNMENT

Discuss product requirements.

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture:

Toward Interoperability and Interchangeability,” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 4 REQUIREMENTS

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Guidelines for Writing Requirements
 - 3.2 System Requirements
 - 3.3 Requirements and Design
 - 3.4 The Requirements Document
 - 3.5 Users of a Requirements Document
 - 3.6 User Requirements
 - 3.7 IEEE Requirements Standards
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit describes requirements. It states the guidelines for writing requirements. You will learn about requirements document and the IEEE requirements standards.

2.0 OBJECTIVES

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

- list guidelines for writing requirements
- explain system requirements
- describe the requirement document
- list user requirements
- note the IEEE requirement standard.

3.0 MAIN CONTENT

3.1 Guidelines for Writing Requirements

- i. Invent a standard format and use it for all requirements.
- ii. Use language in a consistent way. Use “shall” for mandatory requirements, “should” for desirable requirements.
- iii. Use text highlighting to identify key parts of the requirement.
- iv. Avoid the use of computer jargon.

3.2 System Requirements

System requirements are more detailed specifications of system functions, services and constraints than user requirements. They are intended to be a basis for designing the system. They may be incorporated into the system contract.

3.3 Requirements and Design

In principle, requirements should state what the system should do and the design should describe how it does this. In practice, requirements and design are inseparable.

A system architecture may be designed to structure the requirements. The system may inter-operate with other systems that generate design requirements. The use of a specific design may be a domain requirement.

SELF ASSESSMENT EXERCISE 1

List 4 users of a requirement document.

3.4 The Requirements Document

The requirements document is the official statement of what is required of the system developers. It should include both a definition of user requirements and a specification of the system requirements. It is NOT a design document. As far as possible, it should set up WHAT the system should do rather than HOW it should do it.

SELF ASSESSMENT EXERCISE 2

Requirements and design are separable. True or False?

3.5 Users of a Requirement Document

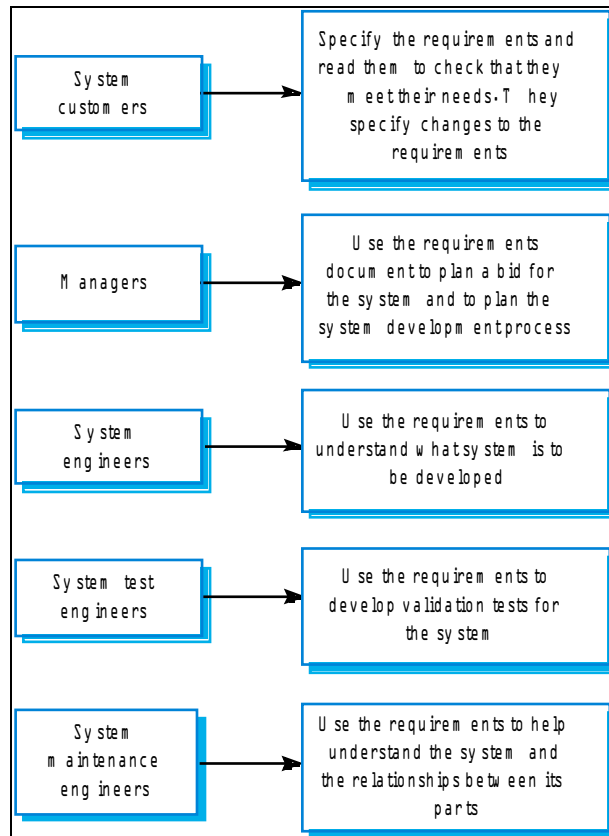


Fig. 1.0: Users of a requirement document

3.6 User Requirements

- i. Should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
- ii. User requirements are defined using natural language, tables and diagrams as these can be understood by all users.

3.7 IEEE Requirements Standards

Defines a generic structure for a requirements document that must be instantiated for each specific system.

- Introduction.
- General description
- Specific requirements
- Appendices
- Index.

4.0 CONCLUSION

Requirements set out what the system should do and define constraints on its operation and implementation. User requirements are high-level statements of what the system should do. User requirements should be written using natural language, tables and diagrams. System requirements are intended to communicate the functions that the system should provide.

A software requirements document is an agreed statement of the system requirements. The IEEE standard is a useful starting point for defining more detailed specific requirements standards.

5.0 SUMMARY

What you have learned borders on requirements; user and system requirements.

6.0 TUTOR-MARKED ASSIGNMENT

What are the guidelines for writing requirements?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 5 DOMAIN REQUIREMENTS

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Domain Requirements
 - 3.2 Library System Domain Requirements
 - 3.3 Domain Requirements Problems
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings
- 1.0 INTRODUCTION**

This unit describes domain requirements. Problems of domain requirements are equally considered.

2.0 OBJECTIVES

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

- explain domain requirements
- list the library system domain requirements
- state domain requirements problems.

3.0 MAIN CONTENT

3.1 Domain Requirements

Domain requirements are derived from the application domain and describe system characteristics and features that reflect the domain.

Domain requirements can be new functional requirements, constraints on existing requirements or define specific computations.

If domain requirements are not satisfied, the system may be unworkable.

3.2 Library System Domain Requirements

- i. There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.
- ii. Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

SELF ASSESSMENT EXERCISE

State two problems of domain requirements.

3.3 Domain Requirement Problems

i. Understandability

- a) Requirements are expressed in the language of the application domain;
- b) This is often not understood by software engineers developing the system.

ii. Implicitness

Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

4.0 CONCLUSION

In this unit you have learned domain requirements. You have also been able to learn about domain requirement problems.

5.0 SUMMARY

What you have learned borders on domain requirements.

6.0 TUTOR-MARKED ASSIGNMENT

What do you understand by domain requirements?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

MODULE 3 REQUIREMENTS ENGINEERING PROCESSES

- Unit 1 Concepts of Requirements Engineering
Unit 2 Viewpoints

Unit 3	Interviewing
Unit 4	Requirements Validation
Unit 5	System Modelling

UNIT 1 CONCEPTS OF REQUIREMENTS ENGINEERING

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	Requirements Engineering Processes
3.2	Feasibility Studies
3.3	Feasibility Study Implementation
3.4	Elicitation and Analysis
4.0	Conclusion
5.0	Summary
6.0	Tutor-Marked Assignment
7.0	References/Further Readings

1.0 INTRODUCTION

In this unit we will learn a few basic concepts of requirements engineering processes. This unit will introduce you to feasibility studies and implementation.

2.0 OBJECTIVES

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

- describe requirements engineering processes
- explain feasibility studies and its implementation
- explain elicitation and analysis.

3.0 MAIN CONTENT

3.1 Requirements Engineering Processes

The processes used for requirements engineering (RE) vary widely depending on the application domain, the people involved and the organisation developing the requirements.

However, there are a number of generic activities common to all processes:

- A. Requirements elicitation

- B. Requirements analysis
- C. Requirements validation
- D. Requirements management

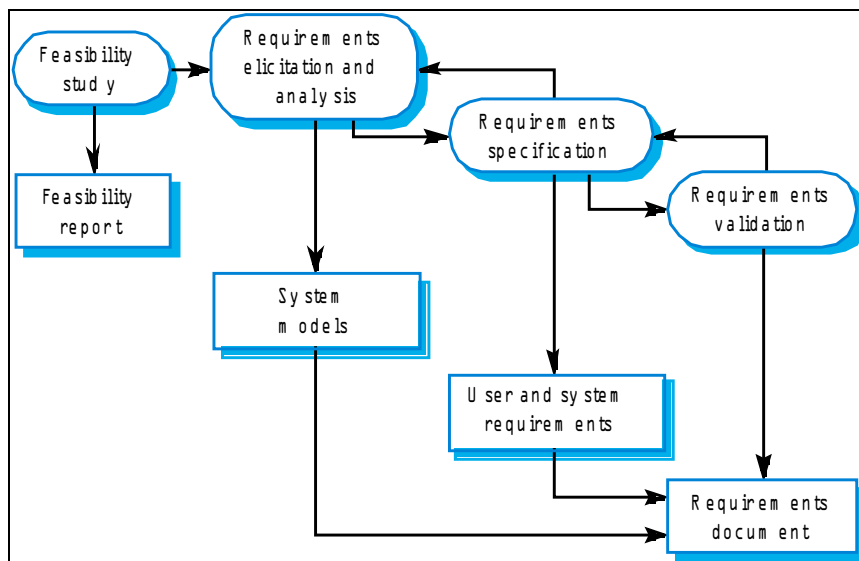


Fig 1.0: The requirements engineering process

3.2 Feasibility Studies

A feasibility study decides whether or not the proposed system is worthwhile. A feasibility study is a short, focused study that checks

- If the system contributes to organisational objectives;
- If the system can be engineered using current technology and within budget;
- If the system can be integrated with other systems that are used.

SELF ASSESSMENT EXERCISE 1

Define a class.

3.3 Feasibility Study Implementation

The feasibility study implementation is based on information assessment (what is required), information collection and report writing.

Questions for people in the organisation

- What if the system wasn't implemented?
- What are current process problems?
- How will the proposed system help?
- What will be the integration problems?
- Is new technology needed? What skills?
- What facilities must be supported by the proposed system?

SELF ASSESSMENT EXERCISE 2

What is an inheritance?

3.4 Elicitation and Analysis

Elicitation and analysis are sometimes called requirements elicitation or requirements discovery. This involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.

It may involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders*.

4.0 CONCLUSION

In this unit you have learned about requirements engineering processes and feasibility studies. You have also been able to understand elicitation and analysis.

5.0 SUMMARY

What you have learned borders on the basic concepts of requirements engineering processes.

6.0 TUTOR-MARKED ASSIGNMENT

What do you understand by object-oriented programming?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). “A Security Framework for Reflective Java Applications,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). “Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 2 VIEWPOINTS

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	Viewpoints
3.2	Types of Viewpoints
3.3	Viewpoint Identification
3.4	LIBSYS Viewpoint Hierarchy
4.0	Conclusion
5.0	Summary
6.0	Tutor-Marked Assignment
7.0	References/Further Readings

1.0 INTRODUCTION

In this unit, you will learn about viewpoints, their identification and types.

2.0 OBJECTIVES

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

- explain the concept of viewpoints
- list the types of viewpoints
- identify viewpoints.

3.0 MAIN CONTENT

3.1 Viewpoints

Viewpoints are a way of structuring the requirements to represent the perspectives of different stakeholders. Stakeholders may be classified under different viewpoints.

This multi-perspective analysis is important as there is no single correct way to analyse system requirements.

3.2 Types of Viewpoints

i. Interactor Viewpoints

People or other systems that interact directly with the system. In an ATM, the customer's and the account database are interactor VPs.

ii. Indirect Viewpoints

Stakeholders who do not use the system themselves but who influence the requirements. In an ATM, management and security staff are indirect viewpoints.

iii. Domain Viewpoints

Domain characteristics and constraints that influence the requirements. In an ATM, an example would be standards for inter-bank communications.

SELF ASSESSMENT EXERCISE

List the types of viewpoints.

3.3 Viewpoint Identification

Identify viewpoints using

- Providers and receivers of system services;
- Systems that interact directly with the system being specified;
- Regulations and standards;
- Sources of business and non-functional requirements.
- Engineers who have to develop and maintain the system;
- Marketing and other business viewpoints.

3.4 LIBSYS Viewpoint Hierarchy

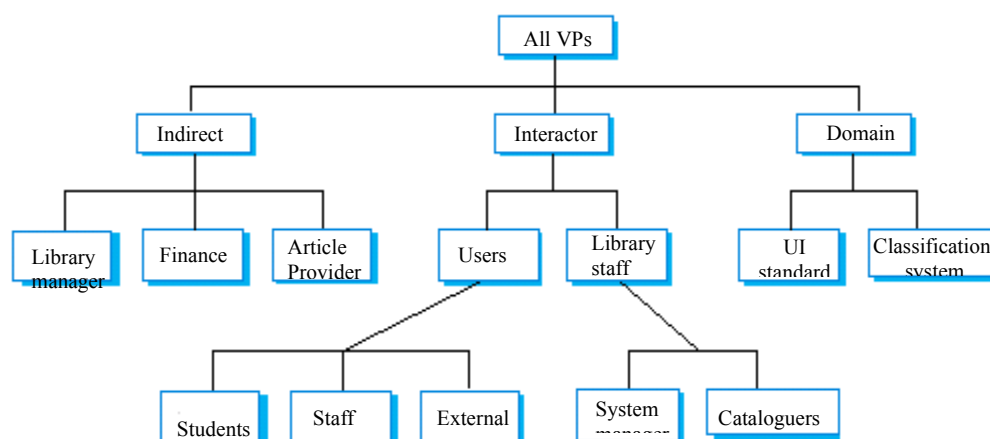


Fig. 1.0: LIBSYS viewpoint hierarchy

4.0 CONCLUSION

In this unit you have learned about viewpoints, types of viewpoints and its identification.

5.0 SUMMARY

What you have learned in this unit is based on viewpoints and types of viewpoints.

6.0 TUTOR-MARKED ASSIGNMENT

How are viewpoints identified?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of

the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 3 INTERVIEWING

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	Interviewing
3.2	Types of Interview
3.3	Interviews in Practice
3.4	Effective Interviewers
4.0	Conclusion
5.0	Summary
6.0	Tutor-Marked Assignment
7.0	References/Further Readings

1.0 INTRODUCTION

What you will learn in this unit concerns interviewing. The types of interview and attributes of an effective interviewer will equally be discussed.

2.0 OBJECTIVES

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

- explain the term ‘interviewing’
- describe the 2 types of interview
- explain interviews in practice
- give the attributes of an effective interviewer.

3.0 MAIN CONTENT

3.1 Interviewing

In formal or informal interviewing, the requirements engineering team puts questions to stakeholders about the system that they use and the system to be developed.

3.2 Types of Interview

There are two types of interview:

- Closed interviews where a pre-defined set of questions are answered.
- Open interviews where there is no pre-defined agenda and a range of issues are explored with stakeholders.

SELF ASSESSMENT EXERCISE

List the types of interviews.

3.3 Interviews in Practice

In practice interviews are normally a mix of closed and open-ended interviewing. Indeed, interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.

Interviews are not good for understanding domain requirements

- Requirements engineers cannot understand specific domain terminology;
- Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.

3.4 Effective Interviewers

- i. Interviewers should be open-minded, willing to listen to stakeholders and should not have pre-conceived ideas about the requirements.
- ii. They should prompt the interviewee with a question or a proposal and should not simply expect them to respond to a question such as 'what do you want'.

4.0 CONCLUSION

In this unit you have learned about interviewing. You have also been able to identify the types of interview and the attributes of an effective interviewer.

5.0 SUMMARY

What you have learned in this unit concerns interviewing and the common types of interviews.

6.0 TUTOR-MARKED ASSIGNMENT

Explain the term 'interviewing'.

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What Characterizes a (Software) Component?*” *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

[http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754.](http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754)

UNIT 4 REQUIREMENTS VALIDATION

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Requirements Validation
 - 3.2 Requirements Checking
 - 3.3 Requirements Validation Techniques
 - 3.4 Requirements Reviews
 - 3.5 Requirements Management
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

Now that you understand what requirements are, it's time to learn about *requirements validation*. You will equally learn about requirements validation techniques and requirements management.

2.0 OBJECTIVES

By the end of this unit, the student should be able to:

- explain requirements validation
- describe requirements checking
- discuss requirements management.

3.0 MAIN CONTENT

3.1 Requirements Validation

Requirements validation is concerned with demonstrating that the requirements define the system that the customer really wants. Requirements error costs are high, so validation is very important.

Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

3.2 Requirements Checking

- i. **Validity:** Does the system provide the functions which best support the customer's needs?

- ii. **Consistency:** Are there any requirements conflicts?
- iii. **Completeness:** Are all functions required by the customer included?
- iv. **Realism:** Can the requirements be implemented giving the available budget and technology
- v. **Verifiability:** Can the requirements be checked?

SELF ASSESSMENT EXERCISE 1

Validity is essential in requirements checking. True or False? Discuss

3.3 Requirements Validation Techniques

- i. **Requirements reviews:** Systematic manual analysis of the requirements.
- ii. **Prototyping:** Using an executable model of the system to check requirements.
- iii. **Test-case generation:** Developing tests for requirements to check testability.

SELF ASSESSMENT EXERCISE 2

List 3 requirements validation techniques.

3.4 Requirements Reviews

Regular reviews should be held while the requirements definition is being formulated. Both client and contractor staff should be involved in reviews.

Reviews may be formal (with completed documents) or informal. Good communication between developers, customers and users can resolve problems at an early stage

3.5 Requirements Management

Requirements management is the process of managing changing requirements during the requirements engineering process and system development.

Requirements are inevitably incomplete and inconsistent

- New requirements emerge during the process as business needs change and a better understanding of the system is developed;
- Different viewpoints have different requirements and these are often contradictory.

4.0 CONCLUSION

In this unit you have learned about the requirements validation. You have also been able to identify the requirements validation techniques. You should also have learned about requirements reviews and management.

5.0 SUMMARY

What you have learned in this unit concerns validation. In the next unit you shall learn about system models.

6.0 TUTOR-MARKED ASSIGNMENT

What do you understand by requirements management?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,*” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 5 SYSTEM MODELLING

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	Relevance of System Modelling
3.2	Model Types
3.3	Context Models
3.4	Process Models
3.5	Behavioural Models
3.6	Data Processing Models
3.7	Application of Data Flow Diagrams
4.0	Conclusion
5.0	Summary
6.0	Tutor-Marked Assignment
7.0	References/Further Readings

1.0 INTRODUCTION

In this unit the student will gain knowledge of the relevance of system modelling. The unit describes the context, process, behavioural and data processing models.

2.0 OBJECTIVES

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

- state the relevance of system modelling
- list model types
- explain the context, process, behavioural and data processing models.

3.0 MAIN CONTENT

3.1 Relevance of System Modelling

System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers. Different models present the system from different perspectives

- External perspective showing the system's context or environment;
- Behavioural perspective showing the behaviour of the system;
- Structural perspective showing the system or data architecture.

3.2 Model Types

- a. Data processing model showing how the data is processed at different stages.
- b. Composition model showing how entities are composed of other entities.

- c. Architectural model showing principal sub-systems.
- d. Classification model showing how entities have common characteristics.
- e. Stimulus/response model showing the system's reaction to events.

3.3 Context Models

Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.

Social and organisational concerns may affect the decision on where to position system boundaries. Architectural models show the system and its relationship with other systems.

3.4 Process Models

- i. Process models show the overall process and the processes that are supported by the system.
- ii. Data flow models may be used to show the processes and the flow of information from one process to another.

3.5 Behavioural Model

Behavioural models are used to describe the overall behaviour of a system.

- A. Two types of behavioural model are:
 - a. Data processing models that show how data is processed as it moves through the system;
 - b. State machine models that show the system's response to events.
- B. These models show different perspectives so both of them are required to describe the system's behaviour

3.6 Data Processing Models

Data flow diagrams (DFDs) may be used to model the system's data processing. These show the processing steps as data flows through a system. Data flow diagrams are an intrinsic part of many analysis methods. DFDs have simple and intuitive notation that customers can understand. They show end-to-end processing of data.

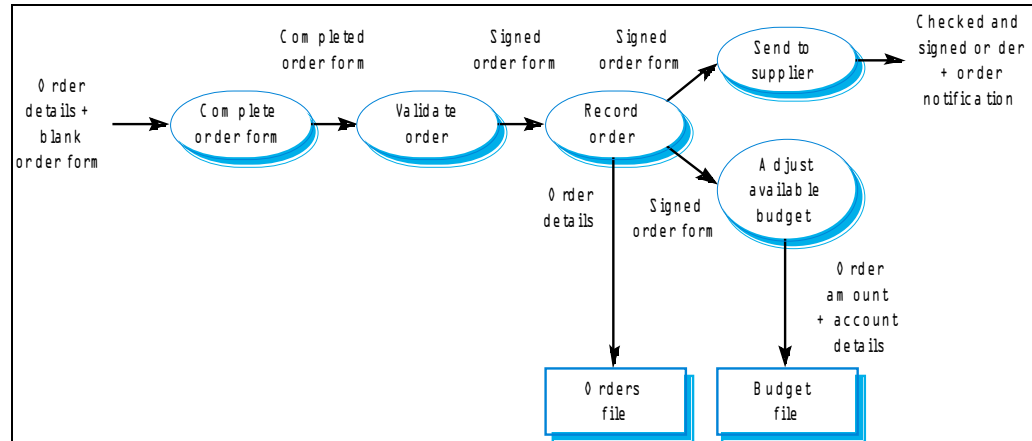


Fig. 1.0: Order processing data flow diagram

SELF ASSESSMENT EXERCISE 1

List the types of model.

3.7 Application of Data Flow Diagrams

- DFDs model the system from a functional perspective.
- Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system.
- Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment.

SELF ASSESSMENT EXERCISE 2

Give at least one application of context models.

4.0 CONCLUSION

A model is an abstract system view. Complementary types of model provide different system information. Context models show the position of a system in its environment with other systems and processes. Data flow models may be used to model the data processing in a system.

5.0 SUMMARY

What you have learned in this unit concerns system modelling.

6.0 TUTOR-MARKED ASSIGNMENT

What is the relevance of system modelling?

7.0 REFERENCES/FURTHER READINGS

- Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).
- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What Characterizes a (Software) Component?*” *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

MODULE 4 REQUIREMENTS ENGINEERING PROCESSES

Unit 1	Formal Methods
Unit 2	Specification
Unit 3	Introduction to Architectural Design
Unit 4	Models
Unit 5	Sub-Systems and Modules

UNIT 1 FORMAL METHODS

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Formal Methods
 - 3.2 Formal Specification
 - 3.3 Acceptance of Formal Methods
 - 3.4 Use of Formal Methods
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

In this unit, you will find information about formal methods. This unit also covers acceptance and use of formal methods.

2.0 OBJECTIVES

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

- enumerate formal methods
- discuss the acceptance of formal methods
- explain the use of formal methods.

3.0 MAIN CONTENT

3.1 Formal Methods

Formal methods include:

- Formal specification;
- Specification analysis and proof;
- Transformational development;
- Program verification.

3.2 Formal Specification

Formal specification is part of a more general collection of techniques that are known as ‘formal methods’.

These are all based on mathematical representation and analysis of software.

SELF ASSESSMENT EXERCISE 1

Program verification is a formal method. True or False?

3.3 Acceptance of Formal Methods

Formal methods have not become mainstream software development techniques as was once predicted. Other software engineering techniques have been successful at increasing system quality. Hence the need for formal methods has been reduced. Market changes have made time-to-market rather than software with a low error count the key factor. Formal methods do not reduce time to market and the scope of formal methods is limited. They are not well-suited to specifying and analysing user interfaces and user interaction. Formal methods are still hard to scale up to large systems.

3.4 Use of Formal Methods

The principal benefits of formal methods are in reducing the number of faults in systems. Consequently, their main area of applicability is in critical systems engineering. There have been several successful projects where formal methods have been used in this area. In this area, the use of formal methods is most likely to be cost-effective because high system failure costs must be avoided.

SELF ASSESSMENT EXERCISE 2

What is the principal benefit of formal methods?

4.0 CONCLUSION

In this unit you have learned about formal methods and their use.

5.0 SUMMARY

What you have learned borders on formal methods and their use.

6.0 TUTOR-MARKED ASSIGNMENT

Enumerate at least three formal methods.

7.0 REFERENCES/FURTHER READINGS

- Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).
- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 2 SPECIFICATION**CONTENTS**

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content

- 3.1 Specification in the Software Process
- 3.2 Use of Formal Specification
- 3.3 Specification Techniques
- 3.4 Systematic Algebraic Specification
- 3.5 Specification Operations
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit covers creating and using objects. You will learn how to instantiate an object, and, once instantiated, how to use the `dot` operator to access the object's instance variables and methods.

2.0 OBJECTIVES

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

- describe how to create objects
- write a program to create objects
- explain how to initialise objects
- describe the process of garbage collection.

3.0 MAIN CONTENT

3.1 Specification in the Software Process

Specification and design are inextricably intermingled. Architectural design is essential to structure a specification and the specification process.

Formal specifications are expressed in a mathematical notation with precisely defined vocabulary, syntax and semantics.

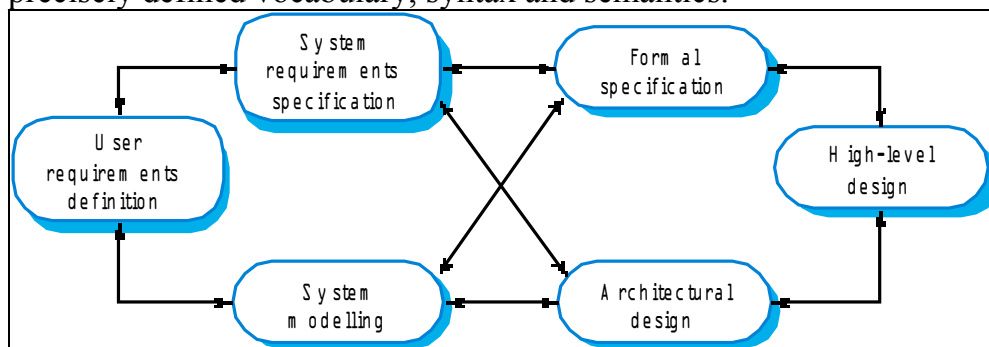


Fig. 1.0: Specification in the software process

3.2 Use of Formal Specification

Formal specification involves investing more effort in the early phases of software development.

This reduces requirements errors as it forces a detailed analysis of the requirements. Incompleteness and inconsistencies can be discovered and resolved. Hence, savings as made as the amount of rework due to requirements problems is reduced.

SELF ASSESSMENT EXERCISE

Describe the model-based specification.

3.3 Specification Techniques

- a. Algebraic specification: The system is specified in terms of its operations and their relationships.
- b. Model-based specification: The system is specified in terms of a state model that is constructed using mathematical constructs such as sets and sequences. Operations are defined by modifications to the system's state.

3.4 Systematic Algebraic Specification

Algebraic specifications of a system may be developed in a systematic way:

- Specification structuring
- Specification naming
- Operation selection
- Informal operation specification
- Syntax definition
- Axiom definition.

3.5 Specification Operations

- i. **Constructor operations:** Operations which create entities of the type being specified.
- ii. **Inspection operations:** Operations which evaluate entities of the type being specified.

To specify behaviour, define the inspector operations for each constructor operation.

4.0 CONCLUSION

Formal system specification complements informal specification techniques.

Formal specifications are precise and unambiguous. They remove areas of doubt in a specification. Formal specification techniques are most applicable in the development of critical systems and standards. Algebraic techniques are suited to interface specification where the interface is defined as a set of object classes.

5.0 SUMMARY

What you have learned in this unit is focused on formal specification.

6.0 TUTOR-MARKED ASSIGNMENT

Enumerate and explain the specification operations.

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications*

of Fingerprint, Iris, Face, Gait, and Multimodal Recognition, Proceedings of the IEEE, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 3 INTRODUCTION TO ARCHITECTURAL DESIGN

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Architectural Design
 - 3.2 Software Architecture

3.3	Architectural Models
3.4	Architectural Design Decisions
4.0	Conclusion
5.0	Summary
6.0	Tutor-Marked Assignment
7.0	References/Further Readings

1.0 INTRODUCTION

This unit describes the notion of architectural design. You will learn about software architecture and the common architectural models. This unit also covers architectural design decisions.

2.0 OBJECTIVES

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

- explain the notion of architectural design
- define software architecture
- describe the common architectural models
- list the architectural design decisions.

3.0 MAIN CONTENT

3.1 Architectural Design

The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is architectural design.

Architectural design is an early stage of the system design process. It represents the link between specification and design processes. Architectural design is often carried out in parallel with some specification activities. It involves identifying major system components and their communications.

3.2 Software Architecture

The software architecture is the fundamental framework for structuring the system. It is the output of the architectural design process

SELF ASSESSMENT EXERCISE 1

What is the main use of architectural models?

3.3 Architectural Models

Architectural models are used to document an architectural design. They include:

- a. Static structural model which shows the major system components.
- b. Dynamic process model which shows the process structure of the system.
- c. Interface model which defines sub-system interfaces.
- d. Relationships model such as a data-flow model that shows sub-system relationships.
- e. Distribution model that shows how sub-systems are distributed across computers.

3.4 Architectural Design Decisions

Architectural design is a creative process so the process differs depending on the type of system being developed. Architectural design decisions include decisions on the application architecture, the distribution and the architectural styles to be used. However, a number of common decisions span all design processes. These include:

- i. Is there a generic application architecture that can be used?
- ii. How will the system be distributed?
- iii. What architectural styles are appropriate?
- iv. What approach will be used to structure the system?
- v. How will the system be decomposed into modules?
- vi. What control strategy should be used?
- vii. How will the architectural design be evaluated?
- viii. How should the architecture be documented?

SELF ASSESSMENT EXERCISE 2

Describe at least 3 architectural models.

4.0 CONCLUSION

In this unit you have learned about architectural design. You have also learned about architectural models and design decisions.

5.0 SUMMARY

What you have learned in this unit borders on architectural design.

6.0 TUTOR-MARKED ASSIGNMENT

Discuss the architectural design.

7.0 REFERENCES/FURTHER READINGS

- Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).
- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 4 MODELS

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Domain - Specific Models
 - 3.2 Repository Models
 - 3.3 Advantages of Repository Models
 - 3.4 Disadvantages of Repository Models
- 4.0 Conclusion

- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit begins with a discussion of domain – specific models and types. It also presents the repository models, stating the advantages and disadvantages.

2.0 OBJECTIVES

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

- define domain – specific models
- explain 2 types of domain- specific models
- describe the repository models.

3.0 MAIN CONTENT

3.1 Domain-Specific Model

Architectural models which are specific to some application domain are known as domain-specific models.

Two types of domain-specific models are:

- Generic models which are abstractions from a number of real systems and which encapsulate the principal characteristics of these systems. Generic models are usually bottom-up models
- Reference models which are more abstract idealised model. They provide a means of information about that class of system and of comparing different architectures. Reference models are top-down models.

SELF ASSESSMENT EXERCISE 1

Mention two types of domain-specific models

3.2 Repository Models

When large amounts of data are to be shared, the repository model of sharing is most commonly used.

3.3 Advantages of Repository Model

- i. Sub-systems need not be concerned with how data is produced by centralised management e.g. backup, security, etc.
- ii. Efficient way to share large amount of data
- iii. Sharing model is published as repository schema

3.4 Disadvantages of Repository Model

- i. Sub-systems must agree on a repository data model.
Inevitably a compromise
- ii. Data evolution is difficult and expensive
- iii. No scope for specific management policies
- iv. Difficult to distribute efficiently.

SELF ASSESSMENT EXERCISE 2

State two advantages of repository models

4.0 CONCLUSION

In this unit you have learned about domain – specific models as well as repository models.

5.0 SUMMARY

What you have learned in this unit borders on models.

6.0 TUTOR-MARKED ASSIGNMENT

What do you understand by domain-specific models?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What*

Characterizes a (Software) Component?” Software – Concepts & Tools, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” Software – Practice & Experience (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 5 SUB-SYSTEMS AND MODULES

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 What is a Sub-System?
 - 3.2 What is a Module?
 - 3.3 Modular Decomposition
 - 3.4 Modular Decomposition Models
- 4.0 Conclusion
- 5.0 Summary

- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit will focus primarily on sub-systems and modules. You will also learn about modular decomposition.

2.0 OBJECTIVES

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

- a sub-system
- a module
- modular decomposition
- modular decomposition models.

3.0 MAIN CONTENT

3.1 What is a Sub-System?

A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems.

3.2 What is a Module?

A module is a system component that provides services to other components but would not normally be considered as a separate system

3.3 Modular Decomposition

Modular decomposition refers to the process whereby sub-systems are decomposed into modules.

3.4 Modular Decomposition Models

There are two modular decomposition models covered in this unit:

- An object model where the system is decomposed into interacting object;
- A pipeline or data-flow model where the system is decomposed into functional modules which transform inputs to outputs.

SELF ASSESSMENT EXERCISE

List the modular decomposition models.

4.0 CONCLUSION

Specifically, you learned about sub – systems and modules. You also learned about modular decomposition.

5.0 SUMMARY

What you have learned in this unit concerns sub-systems and modules.

6.0 TUTOR-MARKED ASSIGNMENT

What is a module?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What Characterizes a (Software) Component?*” *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” *Proceedings of*

the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

MODULE 5 SOFTWARE ENGINEERING

Unit 1	Software Life-Cycle Models
Unit 2	Requirements Engineering
Unit 3	Formal Specification
Unit 4	System Models
Unit 5	Software Design

UNIT 1 SOFTWARE LIFE-CYCLE MODELS

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content

3.1	Software Life-Cycle Models
3.2	Waterfall Model
3.3	Prototyping Model
3.4	Spiral Model
4.0	Conclusion
5.0	Summary
6.0	Tutor-Marked Assignment
7.0	References/Further Readings

1.0 INTRODUCTION

This unit considers software life-cycle models. You will consider the most commonly used models, their benefits and drawbacks.

2.0 OBJECTIVES

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

- describe the concept of software life-cycle model
- list the factors which determine the suitable life-cycle models
- explain the waterfall model
- describe the prototyping model
- explain the spiral model.

3.0 MAIN CONTENT

3.1 Software Life-Cycle Models

A software product usually begins as a vague concept. Once the need for a software product has been established, the product goes through a series of development phases. Typically, the product is specified, designed, and then implemented. If the client is satisfied, the product is installed, and while it is operational it is maintained. When the product finally comes to the end of its useful life, it is decommissioned. The series of steps through which a product progresses is called the life-cycle model.

The best life-cycle model for a given product may be different. The factors which determine the appropriate model include the size of the project, the complexity, the required development time, the degree of risk, the degree of certainty as to what the customer wants, and the degree to which the customer requirements may change.

The two most widely used life-cycle models are the waterfall model and the prototyping model. In addition, the spiral model is now receiving

considerable attention. The strengths and weaknesses of these models will be examined here.

3.2 Waterfall Model

The following activities occur during the waterfall life cycle paradigm:

- Requirements analysis and definition. The system's services, constraints and goals are established by consultation with the customers and users. These are then defined in a manner which is understandable by both customers/users and development staff.
- Specification Phase. From the requirements, a specifications document is produced which states exactly what the product is to do (but not how it will be done).
- System and Software Design. The systems design process partitions the requirements to either hardware or software systems. Software design is actually a multi-step process which focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. In contrast to the specifications document that specifies what requirements will be met, the design documents contain representations that describe how the product will meet them.
- Implementation and unit testing. During this stage, the software design is realized as a set of programs or modules. Unit testing involves verifying that each unit meets its specification.
- Integration and system testing. The individual program units are integrated and tested as a complete system to ensure that the software requirements have been met.
- Acceptance testing. The purpose of acceptance testing is for the client to determine whether the product satisfies its specifications as claimed by the developer. During acceptance testing, the product is evaluated for its correctness, robustness, performance, and documentation.
- Operations and maintenance. The operations and maintenance phase involves the re-application of each of the preceding activities for existing software. The re-application may be required to correct an error in the original software, to adapt the software to changes in its external environment (e.g., new hardware, operating system), or to provide enhancement to function or performance requested by the customer. This is generally the longest life-cycle phase.

These stages are shown diagrammatically below. Normal development is shown by the solid green arrows. Maintenance occurs along the path of the dashed arrows.

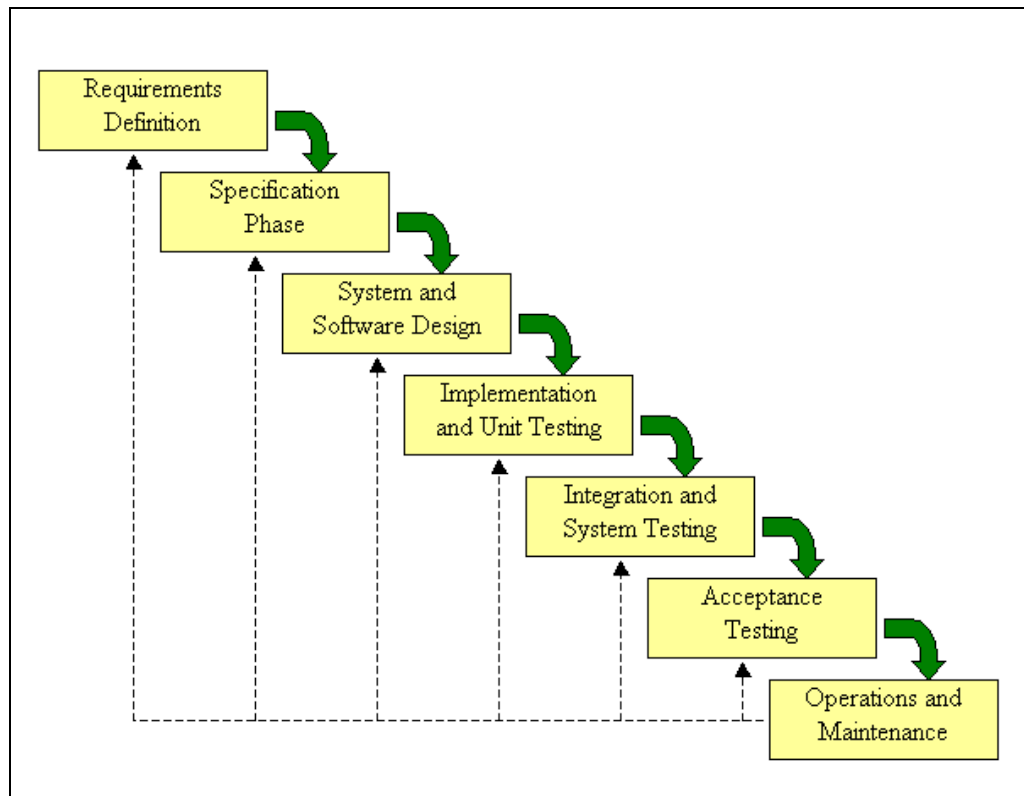


Fig. 1.0: The waterfall model

The waterfall model is the most widely used in software engineering. It leads to systematic, rational software development, but like any generic model, the life-cycle paradigm can be problematic for the following reasons:

1. The rigid sequential flow of the model is rarely encountered in real life. Iteration can occur causing the sequence of steps to become muddled.
2. It is often difficult for the customer to provide a detailed specification of what is required early in the process. Yet this model requires a definite specification as a necessary building block for subsequent steps.
3. Much time can pass before any operational elements of the system are available for customer evaluation. If a major error in implementation is made, it may not be uncovered until much later.

Do these potential problems mean that the life-cycle paradigm should be avoided? Absolutely not! They do mean, however, that the application of this software engineering paradigm must be carefully managed to ensure successful results.

SELF ASSESSMENT EXERCISE 1

Describe the two most widely used life-cycle models.

3.3 Prototyping Model

A prototyping moves the developer and customer toward a "quick" implementation. Prototyping begins with requirements gathering. Meetings between developer and customer are conducted to determine overall system objectives and functional and performance requirements. The developer then applies a set of tools to develop a quick design and build a working model (the "prototype") of some element(s) of the system. The customer or user "test drives" the prototype, evaluating its function and recommending changes to better meet customer needs. Iteration occurs as this process is repeated, and an acceptable model is derived. The developer then moves to "productize" the prototype by applying many of the steps described for the classic life cycle.

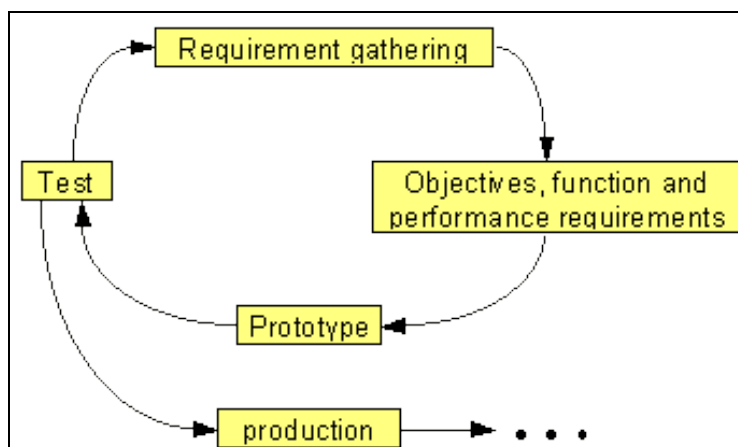


Fig. 1.1: The prototyping model

In object-oriented programming a library of reusable objects (data structures and associated procedures) the software engineer can rapidly create prototypes and production programs.

The benefits of prototyping are:

1. a working model is provided to the customer/user early in the process, enabling early assessment and bolstering confidence,
2. the developer gains experience and insight by building the model, thereby resulting in a more solid implementation of "the real thing"
3. the prototype serves to clarify otherwise vague requirements, reducing ambiguity and improving communication between developer and user.

But prototyping also has a set of inherent problems:

1. The user sees what appears to be a fully working system (in actuality, it is a partially working model) and believes that the prototype (a model) can be easily transformed into a production system. This is rarely the case. Yet many users have pressured developers into releasing prototypes for production use that have been unreliable, and worse, virtually unmaintainable.
2. The developer often makes technical compromises to build a "quick and dirty" model. Sometimes these compromises are propagated into the production system, resulting in implementation and maintenance problems.
3. Prototyping is applicable only to a limited class of problems. In general, a prototype is valuable when heavy human-machine interaction occurs, when complex output is to be produced or when new or untested algorithms are to be applied. It is far less beneficial for large, batch-oriented processing or embedded process control applications.

SELF ASSESSMENT EXERCISE 2

What are the benefits of prototyping?

3.4 Spiral Model

There is almost always risk involved in the development of software. For example:

- key personnel may resign before the product has been adequately documented,
- the manufacturer of hardware on which the product is critically dependent may go bankrupt,
- too little (or too much) time may be invested in testing,
- technological breakthroughs may render the product obsolete,
- a lower-priced, functionally equivalent product may come to market.

For obvious reasons, software developers try to minimize risks whenever possible. A product built using the waterfall model may be subject to substantial risk because of its linear development cycle. The prototyping model is quite effective at minimising risk, allowing a periodic reassessment of the requirements.

The idea of minimising risks via the use of prototypes and other means is the underlying concept of the *spiral model* [8]. A simplistic way of looking at the spiral model is as a series of waterfall models, each preceded by a risk analysis. Before commencing each phase, an attempt

is made to control (or resolve) the risks. If it is impossible to adequately resolve all the significant risks at a given stage, the project is immediately terminated. Prototypes can be used to provide information about certain classes of risk. For example, timing constraints can be tested by constructing a prototype and measuring whether the prototype can achieve the necessary performance.

The spiral model is shown in the figure below. The radial dimension represents cumulative cost to date, the angular dimension represents progress through the spiral. Each cycle of the spiral corresponds to a development phase.

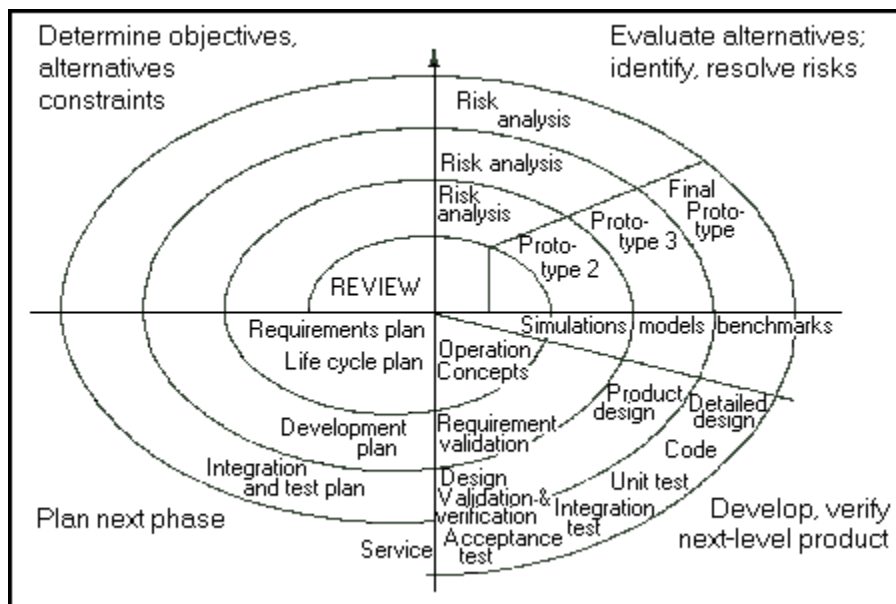


Fig. 1.2: The spiral model

A phase begins (in the top left quadrant) by determining objectives of that phase, alternatives for achieving those objectives, and constraints imposed on those alternatives. Next, that strategy is analyzed from the viewpoint of risk. Attempts are made to resolve every potential risk, in some cases by building a prototype. If certain risks cannot be resolved, the project may be terminated or scaled down. If all risks are resolved, the next development step is started. This quadrant of the spiral model corresponds to the pure waterfall model. Finally, the results of that phase are evaluated and the next phase is planned.

The advantages of the spiral model are:

- The primary advantage is that the spiral model has a wide range of options to accommodate the good features of other life-cycle models. It becomes equivalent to another lifecycle model in appropriate situations. Also the risk-avoidance approach keeps from having additional difficulties.

- The spiral model focuses its early attention on the option of reusing existing software.
- It prepares for life-cycle evolution, growth, and changes of the software product. Major sources of this change are included in the product objectives.
- It incorporates software quality objectives into software product development. Emphasis is placed on identifying all objectives and constraints during each round.
- The risk analysis and validation steps eliminate errors early on.
- Maintenance is included as another cycle of the spiral; there is essentially no distinction between maintenance and development. This helps to avoid underestimation of resources needed for maintenance.

The weaknesses include:

- The risk-driven model is dependent on the developers' ability to identify project risk. The entire product depends on the risk assessment skills of the developer. If those skills are weak then the product could be a disaster. A design produced by an expert may be implemented by non-experts. In a case such as this, the expert does not need a great deal of detailed documentation, but must provide enough additional documentation to keep the non-experts from going astray.
- The process steps need to be further elaborated to make sure that the software developers are consistent in their production. It is still fairly new compared to other models, so it has not been used significantly and therefore the problems associated with it haven't been widely tested and solved.

4.0 CONCLUSION

In this unit you have learned about the software life-cycle models. You have also been able to understand the factors that determine the appropriate life-cycle model. You would have learned of the waterfall, prototyping and spiral models

5.0 SUMMARY

What you have learned borders on software life-cycle models.

6.0 TUTOR-MARKED ASSIGNMENT

What do you understand by the software life-cycle model?

7.0 REFERENCES/FURTHER READINGS

- Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).
- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 2 REQUIREMENTS ENGINEERING**CONTENTS**

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Requirements Engineering
 - 3.2 The Requirements Documents
 - 3.3 Requirements Document Structure
 - 3.4 Writing Requirements
 - 3.5 User Requirements
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

In this unit, you will learn about requirements engineering. Requirements documents and user requirements will also be considered.

2.0 OBJECTIVES

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

- explain requirements engineering
- define the requirements document
- discuss the requirements document structure
- describe the user requirements.

3.0 MAIN CONTENT

3.1 Requirements Engineering

Requirements engineering entails establishing *what* the customer requires from a software system. It is the process of establishing the:

- services that the customer requires from a system
- constraints under which the system operates
- constraints under which the system is developed.

3.2 The Requirements Documents

The requirements document is the official statement of *what* is required of the system developers.

It should include both a definition and a specification of requirements. *It is NOT a design document.* As far as possible, it should set out WHAT the system should do rather than HOW it should do it.

SELF ASSESSMENT EXERCISE

Explain the notion of ‘requirements engineering’.

3.3 Requirements Document Structure

The requirements document structure consists of 6 parts:

i. Introduction (Requirements Definition)

Describe need for the system and how it fits with business objectives.

ii. Functional Requirements

Describe the services to be provided in detail.

iii. **Non-functional Requirements**

Define constraints on the system and the development process.

iv. **System Evolution**

Define fundamental assumptions on which the system is based and anticipated changes.

v. **Glossary**

Define technical terms used.

vi. **Index**

3.4 **Writing Requirements**

Natural language is typically used when writing requirements definitions. This is universally understandable but three types of problems can arise:

Lack of clarity. Precision is difficult without making the document difficult to read.

Requirements confusion. Functional and non-functional requirements tend to be mixed-up

Requirements amalgamation. Several different requirements may be expressed together.

3.5 **User Requirements**

Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge.

User requirements are defined using natural language, tables and diagrams.

4.0 **CONCLUSION**

In this unit you have learned about requirements engineering. You have also learned about requirements documents and requirements documents structure as well as user requirements.

5.0 **SUMMARY**

You have considered requirements engineering as well as requirements documents and user requirements.

6.0 TUTOR-MARKED ASSIGNMENT

What is a requirements document?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 3 FORMAL SPECIFICATION

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Formal Specification
 - 3.2 Advantages of Formal Specification
 - 3.3 Specification Development
 - 3.4 Formal Specification Approaches
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings
- 1.0 INTRODUCTION**

What you will learn in this unit borders on formal specification. The advantages of formal specification as well as formal specification approaches will equally be discussed.

2.0 OBJECTIVES

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

- describe the term ‘formal specification’
- list the advantages of formal specification
- explain the formal specification approaches.

3.0 MAIN CONTENT

3.1 Formal Specification

Formal specification refers to techniques for the unambiguous specification of software.

Formal specifications are expressed in a mathematical notation with precisely defined vocabulary, syntax and semantics.

3.2 Advantages of Formal Specification

- i. It provides insights into the software requirements and the design.
- ii. Formal specifications may be analysed mathematically for consistency.
- iii. It may be possible to prove that the implementation satisfies the specification.
- iv. Formal specifications may be used to guide the tester of the component in identifying appropriate test cases.
- v. Formal specifications may be processed using software tools. It may be possible to animate the specification to provide a software prototype.

SELF ASSESSMENT EXERCISE 1

List the steps in developing a specification.

3.3 Specification Development

The following steps are taken in developing a specification:

1. Establish the bounds of the input parameters. Specify this as a predicate.
2. Specify a predicate defining the condition which must hold on the result of the function if it computes correctly.
3. Establish what changes are made to the input parameters by the function and specify these as a predicate.

4. Combine the predicates into pre and post conditions.

3.4 Formal Specification Approaches

i. Algebraic Approach

The system is described in terms of interface operations and their relationships.

ii. Model-based Approach

A model of the system acts as a specification. This model is constructed using well-understood mathematical entities such as sets and sequences.

SELF ASSESSMENT EXERCISE 2

State the two formal specification approaches.

4.0 CONCLUSION

In this unit you have learned about formal specification. You have also been able to identify the advantages of formal specification as well as the steps to be adopted in developing a specification.

5.0 SUMMARY

What you have learned in this unit concerns formal specification.

6.0 TUTOR-MARKED ASSIGNMENT

Describe the term ‘formal specification’.

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What Characterizes a (Software) Component?*” *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 4 SYSTEM MODELS

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 What are System Models?
 - 3.2 System Modelling
 - 3.3 Structured Methods
 - 3.4 The Unified Modelling Language
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit introduces the divide-and-conquer algorithm as a design technique. It explains the phases involved in this technique of design.

2.0 OBJECTIVES

By the end of this unit, the student should be able to:

- explain what system models are
- state the usefulness of system modelling to a system analyst
- discuss the unified modelling language.

3.0 MAIN CONTENT

3.1 What are System Models?

System models are abstract descriptions of systems whose requirements are being analysed.

3.2 System Modelling

System modelling helps the system analyst to understand the functionality of the system and models are used to communicate with customers.

Different models present the system from different perspectives

- External perspective showing the system's context or environment
- Behavioural perspective showing the behaviour of the system
- Structural perspective showing the system or data architecture

SELF ASSESSMENT EXERCISE

Describe at least one application of system modelling.

3.3 Structured Methods

Structured methods incorporate system modelling as an inherent part of the method. Methods define a set of models, a process for deriving these models and rules and guidelines that should apply to the models. CASE tools support system modelling as part of a structured method

3.4 The Unified Modelling Language (UML)

The unified modelling language was devised by the developers of widely used object-oriented analysis and design methods. It has become an effective standard for object-oriented modelling.

Notation

- Object classes are rectangles with the name at the top, attributes in the middle section and operations in the bottom section
- Relationships between object classes (known as associations) are shown as lines linking objects
- Inheritance is referred to as generalisation and is shown ‘upwards’ rather than ‘downwards’ in a hierarchy.

4.0 CONCLUSION

In this unit you have learned about divide-and-conquer algorithm. You have also gained knowledge of binary and sequential search.

5.0 SUMMARY

What you have learned in this unit concerns divide-and-conquer algorithm.

6.0 TUTOR-MARKED ASSIGNMENT

What are system models?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What Characterizes a (Software) Component?*” *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications,*” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition,* Proceedings of the IEEE, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,*” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method,* Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design,* Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 5 SOFTWARE DESIGN

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Software Design
 - 3.2 Stages of Design
 - 3.3 Design Phases
 - 3.4 Top-Down Design Technique
 - 3.5 Design Strategies
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit introduces you to software design. It explains the design stages and phases and describes the top-down design technique.

2.0 OBJECTIVES

By the end of this unit, the student should be able to:

- describe software design
- identify the design stages
- state the design phases
- explain the concept of top-down design technique
- identify the design strategies.

3.0 MAIN CONTENT

3.1 Software Design

Software design is a process of problem-solving and planning for a [software](#) solution. It entails deriving a solution which satisfies software requirements.

3.2 Stages of Design

A. Problem understanding

- Look at the problem from different angles to discover the design requirements.

B. Identify one or more solutions

- Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources.

C. Describe solution abstractions

- Use graphical, formal or other descriptive notations to describe the components of the design.

D. Repeat process for each identified abstraction

3.3 Design Phases

- i. The Architectural design: Identify sub-systems.
- ii. Abstract specification: Specify sub-systems.
- iii. Interface design: Describe sub-system interfaces.
- iv. Component design: Decompose sub-systems into components.
- v. Data structure design: Design data structures to hold problem data.
- vi. Algorithm design: Design algorithms for problem functions

3.4 Top-Down Design Technique

In principle, top-down design involves starting at the uppermost components in the hierarchy and working down the hierarchy level by level.

In practice, large systems design is never truly top-down. Some branches are designed before others. Designers reuse experience (and sometimes components) during the design process.

SELF ASSESSMENT EXERCISE 1

Describe the top-down design technique.

3.5 Design Strategies

i. Functional Design

- The system is designed from a functional viewpoint. The system state is centralized and shared between the functions operating on that state.

ii. Object-Oriented Design

- The system is viewed as a collection of interacting objects. The system state is decentralized and each object manages its own state. Objects may be instances of an object class and communicate by exchanging methods.

SELF ASSESSMENT EXERCISE 2

State the design phases.

4.0 CONCLUSION

In this unit you have learned about software design. You have also gained insight of design phases as well as top-down design technique.

5.0 SUMMARY

What you have learned in this unit concerns software design and design strategies.

6.0 TUTOR-MARKED ASSIGNMENT

Explain the concept of software design.

7.0 REFERENCES/FURTHER READINGS

- Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).
- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

MODULE 6 SOFTWARE ENGINEERING

Unit 1	Software Testing
Unit 2	Software Inspection
Unit 3	Software Reliability
Unit 4	Software Re-use
Unit 5	Software Prototyping

UNIT 1 SOFTWARE TESTING

CONTENTS

1.0	Introduction
2.0	Objectives
3.0	Main Content
3.1	Verification and Validation
3.2	Verification and Validation Process
3.3	Software Inspection
3.4	Testing and Debugging
3.5	Testing Stages
4.0	Conclusion
5.0	Summary

- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

In this unit the student will be equipped with the knowledge of verification and validation and its principal objectives. The unit describes testing and debugging as well.

2.0 OBJECTIVES

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

- describe the terms verification and validation
- understand the principal objective of verification and validation
- distinguish between verification and validation
- distinguish between testing and debugging
- list the testing stages.

3.0 MAIN CONTENT

3.1 Verification and Validation

In software testing, and software engineering, **Verification and Validation (V&V)** entail the process of checking that a software system meets specifications and that it fulfils its intended purpose.

Verification ensures that the final product satisfies or matches the original design (low-level checking) – i.e., you built the product right. This is done through static testing.

Validation checks that the product design satisfies or fits the intended usage (high-level checking) – i.e., you built the right product. This is done through dynamic testing and other forms of review.

3.2 Verification and Validation Process

Verification and validation process is a whole life-cycle process which must be applied at each stage in the software process.

Verification and validation has two principal objectives:

- i. The discovery of defects in a system.
- ii. The assessment of whether or not the system is usable in an operational situation.

3.3 Software Inspections

Software inspections involve people examining the source representation with the aim of discovering anomalies and defects. Software inspections do not require execution of a system so may be used before implementation. They may be applied to any representation of the system (requirements, design, test data, etc.). It is a very effective technique for discovering errors.

3.4 Testing and Debugging

Defect testing and debugging are distinct processes. Defect testing is concerned with confirming the presence of errors, while debugging is concerned with locating and repairing these errors.

Debugging involves formulating hypotheses about program behaviour then testing these hypotheses to find the system error.

SELF ASSESSMENT EXERCISE 1

Distinguish between testing and debugging.

3.5 Testing Stages

i. Unit testing

- testing of individual components

ii. Module testing

- testing of collections of dependent components

iii. Sub-system testing

- testing collections of modules integrated into sub-systems

iv. System testing

- testing the complete system prior to delivery

v. Acceptance testing

- testing by users to check that the system satisfies requirements. It is sometimes called alpha testing.

SELF ASSESSMENT EXERCISE 2

List the testing stages.

4.0 CONCLUSION

Verification and validation were considered in this unit. You have also learned about testing and debugging.

5.0 SUMMARY

What you have learned in this unit concerns verification, validation, testing and debugging.

6.0 TUTOR-MARKED ASSIGNMENT

What are the principal objectives of verification and validation?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What Characterizes a (Software) Component?*” *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

- Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 2 SOFTWARE INSPECTION

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Software Inspections
 - 3.2 Inspections and Testing
 - 3.3 Inspection Pre-conditions
 - 3.4 Inspection Checklists
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit considers software inspections. It delves into the inspection pre-conditions and inspection checklists.

2.0 OBJECTIVES

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

- explain the aim of software inspections
- identify inspection pre-conditions
- enumerate inspection checklists.

3.0 MAIN CONTENT

3.1 Software Inspections

Software inspections involve people examining the source representation with the aim of discovering anomalies and defects. Software inspections do not require execution of a system so may be used before implementation.

This may be applied to any representation of the system (requirements, design, test data, etc.). It is a very effective technique for discovering errors.

3.2 Inspections and Testing

Inspections and testing are complementary and not opposing verification techniques. Both should be used during the verification and validation process.

Inspections can check conformance with a specification but not conformance with the customer's real requirements. Inspections cannot check non-functional characteristics such as performance, usability etc.

SELF ASSESSMENT EXERCISE 1

When are inspection and testing required?

3.3 Inspection Pre-conditions

- A precise specification must be available
- Team members must be familiar with the organisation's standards
- Syntactically correct code must be available
- An error checklist should be prepared
- Management must accept that inspection will increase costs early in the software process
- Management must not use inspections for staff appraisal.

3.4 Inspection Procedure

- System overview presented to inspection team
- Code and associated documents are distributed to inspection team in advance
- Inspection takes place and discovered errors are noted
- Modifications are made to repair discovered errors
- Re-inspection may or may not be required.

3.5 Inspection Checklists

- Checklist of common errors should be used to drive the inspection
- Error checklist is programming language dependent
- The 'weaker' the type of checking, the larger the checklist
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

SELF ASSESSMENT EXERCISE 2

Enumerate the inspections checklists.

4.0 CONCLUSION

In this unit you have learned about software inspection. You have also been able to identify inspection pre-conditions and checklists.

5.0 SUMMARY

What you have learned borders on software inspection.

6.0 TUTOR-MARKED ASSIGNMENT

What is the aim of software inspection?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What

Characterizes a (Software) Component?" Software – Concepts & Tools, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetchbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 3 SOFTWARE RELIABILITY

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Software Dependability
 - 3.2 Fault Minimisation
 - 3.3 Fault-Free Software Development
 - 3.4 Reliable Software Process
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit covers software dependability and fault minimisation.

2.0 OBJECTIVES

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

- explain the notion of software dependability
- describe fault minimisation
- discuss the reliable software processes.

3.0 MAIN CONTENT

3.1 Software Dependability

In general, software customers expect all software to be dependable. However, for non-critical applications, they may be willing to accept some system failures. Some applications, however, have very high dependability requirements and special programming techniques must be used to achieve this.

3.2 Fault Minimisation

Current methods of software engineering now allow for the production of fault-free software. Fault-free software means software which conforms to its specification. It does NOT mean software which will always perform correctly as there may be specification errors. The cost of producing fault-free software is very high. It is only cost-effective in exceptional situations.

SELF ASSESSMENT EXERCISE 1

When is software said to be fault-free?

3.3 Fault-Free Software Development

- i.Fault-free software development needs a precise (preferably formal) specification.
- ii.It requires an organisational commitment to quality.
- iii.Information hiding and encapsulation in software design is essential
- iv.A programming language with strict typing and run-time checking should be used
- v.Error-prone constructs should be avoided
- vi.Dependable and repeatable development process

3.4 Reliable Software Processes

To ensure a minimal number of software faults, it is important to have a well-defined, repeatable software process. A well-defined repeatable process is one that does not depend entirely on individual skills; rather can be enacted by different people. For fault minimisation, it is clear that the process should include significant verification and validation.

SELF ASSESSMENT EXERCISE 2

What must be put in place to ensure minimal software faults.

4.0 CONCLUSION

In this unit you have learned about software dependability. You have also learned about fault minimisation and fault-free software development.

5.0 SUMMARY

What you have learned in this unit borders on software reliability and fault-free software development.

6.0 TUTOR-MARKED ASSIGNMENT

Explain the notion of ‘software dependability’.

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What Characterizes a (Software) Component?*” *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, Proceedings of the IEEE, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” Proceedings of the 2001 Winter Simulation Conference (WSC 2001), pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 4 SOFTWARE RE-USE

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Software Re-use
 - 3.2 Benefits of Re-use
 - 3.3 Requirements for Re-use
 - 3.4 Software Development with Re-use
 - 3.5 Software Development for Re-use
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit considers the concept of software re-use. You will equally learn about the benefits and the requirements for re-use.

2.0 OBJECTIVES

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

- describe the concept of software re-use
- enumerate the benefits of re-use
- explain software development with re-use
- list the requirements for re-use.

3.0 MAIN CONTENT

3.1 Software Re-use

In most engineering disciplines, systems are designed by composing existing components that have been used in other systems. Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need to adopt a design process that is based on systematic re-use.

3.2 Benefits of Re-use

i. Increased reliability

- Components exercised in working systems

ii. Reduced process risk

- Less uncertainty in development costs

iii. Effective use of specialists

- Re-use components instead of people

iv. Standards compliance

- Embed standards in reusable components

v. Accelerated development

- Avoid original development and hence speed-up production

SELF ASSESSMENT EXERCISE 1

What are the benefits of re-use?

3.3 Software Development with Re-use

Software development with re-use attempts to maximise the use of existing components. These components may have to be adapted in a new application.

Fewer components need be specified, designed and coded. Overall development costs should therefore be reduced.

3.4 Requirements for Re-use

i. It must be possible to find appropriate reusable components in a component data base.

ii. Component re-users must be able to understand components and must have confidence that they will meet their needs.

iii. The components must have associated documentation discussing HOW they can be re-used and the potential costs of re-use.

SELF ASSESSMENT EXERCISE 2

Enumerate the requirements for re-use.

3.5 Software Development for Re-use

Software components are not automatically reusable. They must be modified to make them usable across a range of applications. Software development for re-use is a development process which takes existing components and aims to generalize and document them for re-use.

4.0 CONCLUSION

In this unit you have learned about software re-use. You have also learned about the benefits of re-use and the requirements for re-use.

5.0 SUMMARY

What you have learned in this unit borders on software re-use, its benefits and requirements for re-use.

6.0 TUTOR-MARKED ASSIGNMENT

Describe the concept of software re-use.

7.0 REFERENCES/FURTHER READINGS

- Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).
- Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.
- Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). "What Characterizes a (Software) Component?" *Software – Concepts & Tools*, vol. 19, pp. 49-56.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.
- Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.
- Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.
- Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.
- Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability," *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington: VA.
- Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.
- Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.
- Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>.

UNIT 5 SOFTWARE PROTOTYPING

CONTENTS

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Content
 - 3.1 Software Prototyping
 - 3.2 Prototyping Steps
 - 3.3 Types of Prototyping
 - 3.3.1 Throwaway Prototyping
 - 3.3.2 Evolutionary Prototyping
 - 3.4 Advantages of Prototyping
 - 3.5 Disadvantages of Prototyping
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor-Marked Assignment
- 7.0 References/Further Readings

1.0 INTRODUCTION

This unit will focus primarily on software prototyping. It gives an outline of steps involved in the prototyping process. We will also consider types of prototyping as well as the advantages and disadvantages of prototyping.

2.0 OBJECTIVES

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

- describe software prototyping
- list the prototyping steps
- identify and explain the types of prototyping
- state the advantages and disadvantages of prototyping.

3.0 MAIN CONTENT

3.1 Software Prototyping

Software prototyping is the creation of [prototypes](#), i.e., incomplete versions of the [software program](#) being developed. The purpose of a prototype is to allow [users](#) of the software to evaluate proposals for the design of the eventual product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions.

3.2 Prototyping Steps

The process of prototyping involves the following steps

1. Identify Basic [Requirements](#)

Determine basic requirements including the input and output information desired. Details, such as security, can typically be ignored.

2. Develop Initial Prototype

The initial prototype is developed that includes only user interfaces.

3. Review

The customers, including end-users, examine the prototype and provide feedback on additions or changes.

4. Revise and Enhancing the Prototype

Using the feedback both the specifications and the prototype can be improved. Negotiation about what is within the scope of the contract/product may be necessary.

SELF ASSESSMENT EXERCISE 1

List the prototyping steps.

3.3 Types of Prototyping

Software prototyping has many variants. However, all the methods are in some way based on two major types of prototyping: Throwaway Prototyping and Evolutionary Prototyping.

3.3.1 Throwaway Prototyping

Throwaway or Rapid Prototyping refers to the creation of a model that will eventually be discarded rather than becoming part of the finally delivered software. After preliminary requirements gathering is accomplished, a simple working model of the system is constructed to visually show the users what their requirements may look like when they are implemented into a finished system.

The most obvious reason for using Throwaway Prototyping is that it can be done quickly. If the users can get quick feedback on their requirements, they may be able to refine them early in the development of the software. Another strength of Throwaway Prototyping is its ability to construct interfaces that the users can test. The user interface is what the user sees as the system, and by seeing it in front of them, it is much easier to grasp how the system will work.

3.3.2 Evolutionary Prototyping

Evolutionary Prototyping (also known as [breadboard](#) prototyping) is quite different from Throwaway Prototyping. The main goal when using Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. "The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built. When developing a system using Evolutionary Prototyping, the system is continually refined and rebuilt."

Evolutionary Prototyping have an advantage over Throwaway Prototyping in that they are functional systems. Although they may not have all the features the users have planned, they may be used on an interim basis until the final system is delivered. Evolutionary Prototyping have an advantage over Throwaway Prototyping in that they are functional systems. Although they may not have all the features the users have planned, they may be used on an interim basis until the final system is delivered.

3.4 Advantages of Prototyping

- i. **Reduced time and costs:** Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of what the user really wants can result in faster and less expensive software.
- ii. **Improved and increased user involvement:** Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications. The presence of the prototype being examined by the user prevents many misunderstandings and miscommunications that occur when each side believes the other understands what they said. Since users know the [problem domain](#) better than anyone on the development team does, increased interaction can result in final product that has greater tangible and intangible quality. The final product is more likely to satisfy the user's desire for look, feel and performance.

SELF ASSESSMENT EXERCISE 2

What is the advantage of evolutionary prototyping over throwaway prototyping.

3.5 Disadvantages of Prototyping

- i. **Evolutionary insufficient analysis:** The focus on a limited prototype can distract developers from properly analysing the complete project. This can lead to overlooking better solutions, preparation of incomplete specifications or the conversion of limited prototypes into poorly engineered final projects that are hard to [maintain](#). Further, since a prototype is limited in functionality it may not scale well if the prototype is used as the basis of a final deliverable, which may not be noticed if developers are too focused on building a prototype as a model.
- ii. **User confusion of prototype and finished system:** Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished. (They are, for example, often unaware of the effort needed to add error-checking and security features which a prototype may not have). This can lead them to expect the prototype to accurately model the performance of the final system when this is not the intent of the developers. Users can also become attached to features that were included in a prototype for consideration and then removed from the specification for a final

system. If users are able to require all proposed features be included in the final system this can lead to [feature creep](#).

- iii. **Developer attachment to prototype:** Developers can also become attached to prototypes they have spent a great deal of effort producing; this can lead to problems like attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture. (This may suggest that throwaway prototyping, rather than evolutionary prototyping, should be used.)
- iv. **Excessive development time of the prototype:** A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex. When the prototype is thrown away the precisely developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. Users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product.

4.0 CONCLUSION

Specifically, you learned about software prototyping. You would have also learned about steps involved in the prototyping process. The types of prototyping as well as the advantages and disadvantages of prototyping were equally considered.

5.0 SUMMARY

What you have learned in this unit is focused on software prototyping.

6.0 TUTOR-MARKED ASSIGNMENT

What is the main goal of evolutionary prototyping?

7.0 REFERENCES/FURTHER READINGS

Somerville, I. (2002). *Software Engineering Methodology*, McGraw-Hill (5th Edition).

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition, Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). “*What Characterizes a (Software) Component?*” *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers, San Francisco: CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience* (John Wiley & Sons, Inc.), vol. 33, No. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Eds.), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition*, *Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,” *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington: VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs: NJ.

Constantine L. L. and Lockwood, L. A. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition), New Jersey: Prentice Hall.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

[http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754.](http://www.freetechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754)