

**FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA,
NIGERIA**



**CENTRE FOR OPEN DISTANCE AND e-LEARNING
(CODEL)**

SYSTEM ANALYSIS AND DESIGN

(CIT 316)

COURSE DEVELOPMENT TEAM

CIT 316

SYSTEM ANALYSIS AND DESIGN

Programme Coordinator

Mrs. O. A. Abisoye
Computer Science Department
Federal University of Technology, Minna, Nigeria.

Instructional Designers

Dr. Gambari, Amosa Isiaka
Mr. Falode, Oluwole Caleb
Centre for Open Distance and e-Learning,
Federal University of Technology, Minna, Nigeria.

Editor

Chinenye Priscilla Uzochukwu
Centre for Open Distance and e-Learning,
Federal University of Technology, Minna, Nigeria.

Director

Prof. J. O. Odigure
Centre for Open Distance and e-Learning,
Federal University of Technology, Minna, Nigeria

INTRODUCTION

CPT316: System Analysis and Design is a 3 credit unit course for students studying towards acquiring a Bachelor of Science in Computer Science and other related disciplines. The course is divided into 4 modules and 10 study units. This course guide will look briefly into the concept of System Analysis and Design. Then, the course will make an in-depth review of system development life cycle. The course goes further to deal with system development methodologies, models, techniques and tools. This course also introduces such other knowledge that will enable the reader have proper understanding of how to build a successful system.

The course guide therefore gives you an overview of what the course; CPT316 is all about, the textbooks and other materials to be referenced, what you expect to know in each unit, and how to work through the course material.

What you will learn in this Course

This study guide intends to provide several fundamental concepts and practical techniques that can improve the probability of success in any information technology (IT) project.

A survey conducted by Standish Group in 2004 found that just 28% of IT projects succeed today. Outright failures—IT projects cancelled before completion—occur in 18% of all IT projects. Unfortunately, many of the systems that are not abandoned are delivered to the users significantly late, cost far more than planned, and have fewer features than originally planned.

Most of us would like to think that these problems only occur to “other” people or “other” organizations, but they happen in most companies. Significant IT project failure have occurred many times; even Microsoft has a history of failures and overdue project (e.g., Windows 1.0, Windows 95).

The overall aim of this course, CPT316 is to introduce you to basic concepts of System Analysis and Design so as to enable you to understand the basic system development methodologies, models, tools and techniques that can improve the probability of success in any IT project.

This course also look at project management and unified process, requirement discipline and detailed requirement modeling of a system development, design activities in software environment, use case realization, system access and implementation.

Course Aim

This course aims to introduce students to the basics, concepts and features of system analysis and design. It is believed the knowledge will enable the reader understand and appreciate decision processes when it comes to building a system.

It will help the reader to understand what methodology to select, what model to use, what tools will be suitable for what type of project in question. Readers will now know that building a project requires detailed requirement discipline and modeling, has design tasks and activities to be followed and implementation phase requirements some definite steps.

Course Objectives

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

However, below are overall objectives of this course. On completing this course, you should be able to:

1. Define System and Information System (IS)
2. Define System Development
3. Know the two major components of system development
4. Know the basic four phase of a system development life cycle
5. Know a system analyst
6. Roles of a System Analyst
7. Qualities of a System Analyst
8. Understand several different categories of system development methodologies and how to choose among them.
9. Understand the three fundamental system development models
10. Know the tools and techniques used in system development
11. Define a Project
12. Understand Information Technology Projects
13. Know Attributes of a Project
14. Know how to manage a project
15. Understand Project Management Triple Constraints
16. Define Unified Process
17. Know the six basic principles of Unified Process
18. Understand RUP discipline in relation to Analyst role
19. Understand Rational Unified Process Life Cycle
20. Define requirement and understand different kinds of requirements
21. Understanding requirement management
22. Know the purpose for requirement discipline
23. Differentiate Requirement Modeling from Requirement Analysis
24. Know the general overview of Requirement Modeling
25. Discover association between requirements
26. Know various tasks and activities carried out during design phase
27. Understand the design phase overview
28. Know issues to be considered during design

29. Understand the ability to review activities that took place in design phase
30. Know the meaning of use-case
31. Know the six basic first principles of use case
32. Understand use case realization.
33. Understand System Implementation
34. Know list of processes and deliverables in system implementation
35. Measure success of system access and implementation

Working through this Course

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

Course Materials

The major components of the course are:

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

Study Units

There are 10 study units and 4 modules in this course. They are:

MODULE 1

UNIT 1 SYSTEM DEVELOPMENT

UNIT 2 THE SYSTEM ANALYST

MODULE 2

UNIT 1 SYSTEM METHODOLOGIES

UNIT 2 SYSTEM MODELS, TOOLS AND TECHNIQUES

MODULE 3

UNIT 1 PROJECT MANAGEMENT

UNIT 2 RELATION UNIFIED PROCESS

UNIT 3 REQUIREMENT DISCIPLINE AND DETAILED REQUIREMENT MODELING

MODULE 4

UNIT 1 DESIGN ACTIVITIES AND ENVIRONMENT

UNIT 2 USE CASE REALIZATION

UNIT 3 SYSTEM ACCESS AND IMPLEMENTATION

Recommended Texts

Recommended text that will be of enormous benefit to you learning this course include:

Frederick P. (1986) “No Silver Bullet—Essence and Accident in Software Engineering,” Proceedings of the IFIP Tenth World Computing Conference, edited by H.-J. Kugler (1986): 1069–76.

Edward Yourdon (1989), Modern Structured Analysis, Englewood Cliffs, NJ: Yourdon Press. (*for information on classic modern process-centered methodology*)

James Martin (1989), Information Engineering, volumes 1–3, Englewood Cliffs, NJ: Prentice Hall. (*for information on data-centered methodology*)

Steve McConnell (1996), Rapid Development, Redmond, WA: Microsoft Press. (*A good reference for comparing systems development methodologies and One of the best RAD books*)

Barry Boehm (1988), “A Spiral Model of Software Development and Enhancement,” Computer, 21(5):61–72. (*for throwaway prototyping methodology and spiral model*)

John Wiley & Sons (2002) eXtreme Programming in Action: Practical Experiences from Real World Projects, New York.

The internet resource links below will be of enormous benefit to you in learning this course:

<http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>

http://en.wikipedia.org/wiki/Systems_analyst

<http://www2.accaglobal.com/pdfs/studentaccountant/bakehouse0506.pdf>

http://xa.yimg.com/kq/groups/22830576/1266190173/name/ISCA_Chap2_May-11.pdf

http://en.wikipedia.org/wiki/Unified_Process

<http://old.nios.ac.in/cca/cca1.pdf>

http://www.asapm.org/asapmag/articles/A7_AboutRUP.pdf

http://www.augsburg.edu/ppages/~schwalbe/C6919_ch01.pdf

http://sce.uhcl.edu/helm/rationalunifiedprocess/process/workflow/requirem/in_req.htm

<http://www.computer.org/comp/proceedings/re/2002/1465/00/14650006.pdf>

<http://www.erts2012.org/Site/0P2RUC89/TA-1.pdf>

<http://www.justice.gov/jmd/irm/lifecycle/ch7.htm>

<http://www.liteea.com/slscp/governance/sdlc/slm7.pdf>

http://epf.eclipse.org/wikis/openup/practice.tech.use_case_driven_dev.base/guidances/guidelines/uc_realizations_448DDA77.html

http://www.outsideininc.com/wp-content/uploads/2012/02/Use-Case-2_0_Feb14_2012.pdf

<http://www.its.ny.gov/pmmp/guidebook2/SystemImplement.pdf>

Assignment File

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

Presentation Schedule

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

Assessment

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

At the end of the course, you will need to sit for a final written examination. This examination will account for 60% of your total score.

Tutor Marked Assignments (TMAs)

There are TMAs in this course. You need to submit all the TMAs. The best 5 will therefore be counted. When you have completed each assignment, send them to your tutor as soon as possible and make certain that it gets to your tutor on or before the stipulated deadline. If for any reason you cannot complete your assignment on time, contact your tutor before the assignment is due to discuss the possibility of extension. Extension will not be granted after the deadline, unless on extraordinary cases.

Final Examination and Grading

The final examination for CPT 316 will be of last for a period of 2½ hours and have a value of 60% of the total course grade. The examination will consist of questions which reflect the self assessment exercise and tutor marked assignments that you have previously encountered. Furthermore, all areas of the course will be examined. It would be better to use the time between finishing the last unit and sitting for the examination, to revise the entire course. You might find it useful to review your TMAs and comment on them before the examination. The final examination covers information from all parts of the course.

The following are practical strategies for working through this course

1. Read the course guide thoroughly
2. Organize a study schedule. Refer to the course overview for more details. Note the time you are expected to spend on each unit and how the assignment relates to the units. Important details, e.g. details of your tutorials and the date of the first day of the semester are available. You need to gather together all these information in one place such as a diary, a wall chart calendar or an organizer.

- Whatever method you choose, you should decide on and write in your own dates for working on each unit.
3. Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they get behind with their course works. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.
 4. Turn to Unit 1 and read the introduction and the objectives for the unit.
 5. Assemble the study materials. Information about what you need for a unit is given in the table of content at the beginning of each unit. You will almost always need both the study unit you are working on and one of the materials recommended for further readings, on your desk at the same time.
 6. Work through the unit, the content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit, you will be encouraged to read from your set books
 7. Keep in mind that you will learn a lot by doing all your assignments carefully. They have been designed to help you meet the objectives of the course and will help you pass the examination.
 8. Review the objectives of each study unit to confirm that you have achieved them. If you are not certain about any of the objectives, review the study material and consult your tutor.
 9. When you are confident that you have achieved a unit's objectives, you can start on the next unit. Proceed unit by unit through the course and try to pace your study so that you can keep yourself on schedule.
 10. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor marked assignment form and also written on the assignment. Consult you tutor as soon as possible if you have any questions or problems.
 11. After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this course guide).

Tutors and Tutorials

There are 8 hours of tutorial provided in support of this course. You will be notified of the dates, time and location together with the name and phone number of your tutor as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course. You must mail your tutor marked assignment to your tutor well before the due date. At least two working days are required for this purpose. They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by telephone, e-mail or discussion board if you need help. The following might be circumstances in which you would find help necessary: contact your tutor if:

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

You should endeavour to attend the tutorials. This is the only opportunity to have face to face contact with your tutor and ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain the maximum benefit from the course tutorials, have some questions handy before attending them. You will learn a lot from participating actively in discussions.

GOODLUCK!

UNIT 1: SYSTEM DEVELOPMENT

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 Definitions
 - 3.2 What is System Development
 - 3.3 System Development Team
 - 3.4 System Development Life Cycle
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 7.0 References

1.0 Introduction

Systems are created to solve problems. One can think of the systems approach as an organized way of dealing with a problem. In this dynamic world, the subject System Analysis and Design (SAD), mainly deals with software development activities. This unit introduces the systems development life cycle, the fundamental four phase model (planning, analysis, design, and implementation) that is common to all information system development projects.

2.0 Objectives

After studying this unit, you should be able to:

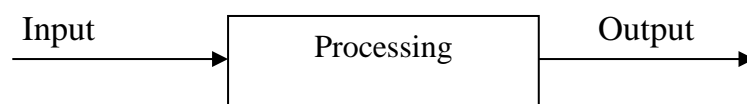
- 36. Define System and Information System (IS)
- 37. Define System Development
- 38. Know the two major components of system development
- 39. Know the basic four phase of a system development life cycle

3.0 Main Body

3.1 Definitions

System:- A system is a collection of elements or components that are organized for a common purpose. It can also be seen as a set of interacting or interdependent components forming an integrated whole or a set of elements (often called 'components') and relationships which are different from relationships of the set or its elements to other elements or sets.

Basically there are three major components in every system, namely input, processing and output.



Information System:- An information system (IS) is any combination of information technology and people's activities that support operations, management and decision making. In a very broad sense, the term information system is frequently used to refer to the interaction between people, processes, data and technology. In this sense, the term is used to refer not only to the information and communication technology (ICT) that an organization uses, but also to the way in which people interact with this technology in support of business processes.

3.2 What is System Development

System development is the process of examining a business situation to improve it through better procedures and methods. System development has two major components – system analysis and system design.

System analysis – is the process of gathering and interpreting facts, diagnosing problems, and using the information to recommend improvements to the system. Systems Analysis is a development phases in a project that primarily focus on the business problems, i.e., WHAT the system must do in terms of Data, Processes, and Interfaces, independent of any technology that can or will be used to implement a solution to that problem.

System analysis helps

- Study current operations and problems, information flow
- Assess future needs, changes required
- Analyze suitable alternatives

System design – is the process of planning a new business system or one to replace/complement existing one. Systems Design: development phases focus on the technical construction and implementation of the system (HOW technology will be used in the system.)

3.3 System Development Team

People responsible for systems development usually in large systems include:

- Top management level steering committee – group of key information system service users that acts as a review body, decides project's worth, its aims, satisfy information requirements of managers and users.
- Project management team – computer professionals and key users.
- System analyst – people who determine user requirements, design the system and assist in development and implementation activities.

- System designers – people who take lead role during design, development and implementation stages.

3.4 System Development Life Cycle

The systems development life cycle (SDLC) is the process of understanding how an information system (IS) can support business needs, designing the system, building it, and delivering it to users.

Building a system can be compared with building a house. First, the house (or the information system) starts with a basic idea. Second, this idea is transformed into a simple drawing that is shown to the customer and refined (often through several drawings, each improving on the other) until the customer agrees that the picture depicts what he or she wants. Third, a set of blueprints is designed that presents much more detailed information about the house (e.g., the type of water faucets, where the telephone jacks will be placed). Finally, the house is built following the blueprints—and often with some changes and decisions made by the customer as the house is erected.

The SDLC has a similar set of four fundamental phases: planning, analysis, design, and implementation (Figure 1.1). Different projects may emphasize different parts of the SDLC or approach the SDLC phases in different ways, but all projects have elements of these four phases. Each phase is itself composed of a series of steps, which rely on techniques that produce deliverables (specific documents and files that provide understanding about the project).

Phase	Chapter	Step	Technique	Deliverable
Planning Focus: Why build this system? How to structure the project? Primary Outputs: — System Request with Feasibility Study — Project Plan	2	Identify Opportunity	Project Identification	System Request
	2	Analyze Feasibility	Technical Feasibility Economic Feasibility Organizational Feasibility	Feasibility Study
	3	Develop Workplan	Time Estimation Timeboxing Task Identification Work Breakdown Structure PERT Chart Gantt Chart Scope Management	Project Plan — Workplan
	3	Staff Project	Project Staffing Project Charter	— Staffing Plan
Analysis Focus: Who, what, where and when for this system? Primary Output — System Proposal	3	Control and Direct Project	CASE Repository Standards Documentation Risk Management	— Standards List — Risk Assessment
	4	Develop Analysis Strategy	Business Process Automation Business Process Improvement Business Process Reengineering	System Proposal
	4	Determine Business Requirements	Interview IAD session Questionnaire Document Analysis Observation	— Requirements Definition
	5	Create Use Cases	Use Case Analysis	— Use Cases
	6 7	Model Processes Model Data	Data Flow Diagramming Entity Relationship Modeling Normalization	— Process Models — Data Model
Design Focus: How will this system work? Primary Output: — System Specification	8	Design Physical System	Design Strategy	Alternative Matrix System Specification
	9	Design Architecture	Architecture Design Hardware & Software Selection	— Architecture Report — Hardware & Software Specification
	10	Design Interface	Use Scenario Interface Structure Interface Standards Interface Prototype Interface Evaluation	— Interface Design
	11	Design Programs	Data Flow Diagramming Program Structure Chart Program Specification	— Physical Process Model — Program Design
	12	Design Databases and Files	Data Format Selection Entity Relationship Modeling Denormalization Performance Tuning Size Estimation	— Database & File Specification — Physical Data Model
Implementation Focus: Delivery and support of completed system. Primary Output: — Installed System	13	Construct System	Programming Software Testing Performance Testing	Test Plan Programs Documentation Migration Plan
	14	Install System	Conversion Strategy Selection	— Conversion Plan — Business Contingency Plan
	14	Maintain System	Training Support Selection System Maintenance Project Assessment	— Training Plan Support Plan Problem Report Change Request
	14	Postimplementation	Post-implementation Audit	Post-implementation Audit Report

Figure 1.1: System Development Life Cycle Phases

(Source:- <http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>)

For example, when you apply for admission to a university, there are several phases that all students go through: information gathering, applying, and accepting. Each of these phases has steps: information gathering includes steps like searching for schools, requesting information, and reading brochures. Students then use techniques (e.g., Internet searching) that can be applied to steps (e.g., requesting information) to create deliverables (e.g., evaluations of different aspects of universities).

Figure 1.1 suggests that the SDLC phases and steps proceed in a logical path from start to finish. In some projects, this is true, but in many projects, the project teams move through the steps consecutively, incrementally, iteratively, or in other patterns. This section describe at a very high level the phases, steps, and some of the techniques that are used to accomplish the steps.

For now, there are two important points to understand about the SDLC. First, you should get a general sense of the phases and steps that IS projects move through and some of the techniques that produce certain deliverables. Second, it is important to understand that the SDLC is a process of gradual refinement. The deliverables produced in the analysis phase provide a general idea of the shape of the new system. These deliverables are used as input to the design phase, which then refines them to produce a set of deliverables that describes in much more detailed terms exactly how the system will be built. These deliverables in turn are used in the implementation phase to produce the actual system.

Planning

The planning phase is the fundamental process of understanding why an information system should be built and determining how the project team will go about building it. It has two steps:

1. *Project Initiation*:- During project initiation, the system's business value to the organization is identified—how will it lower costs or increase revenues? Most ideas for new systems come from outside the IS area (from the marketing department, accounting department, etc.) in the form of a system request. A system request presents a brief summary of a business need, and it explains how a system that supports the need will create business value. The IS department works together with the person or department that generated the request (called the project sponsor) to conduct a feasibility analysis. The feasibility analysis examines key aspects of the proposed project:
 - a. The technical feasibility (Can we build it?)
 - b. The economic feasibility (Will it provide business value?)
 - c. The organizational feasibility (If we build it, will it be used?)

The system request and feasibility analysis are presented to an information systems approval committee (sometimes called a steering committee), which decides whether the project should be undertaken.

2. *Project Management*:- Once the project is approved, it enters project management. During project management, the project manager creates a work plan, staffs the projects, and puts techniques in place to help the project team control and direct the project through the entire SDLC. The deliverable for project management is a project plan that describes how the project team will go about developing the system.

Analysis

The analysis phase answers the questions of who will use the system, what the system will do, and where and when it will be used. See Figure 1. During this phase, the project team investigates any current system(s), identifies improvement opportunities, and develops a concept for the new system. This phase has three steps:

1. An analysis strategy is developed to guide the project team's efforts. Such a strategy usually includes an analysis of the current system (called the as-is system) and its problems, and then ways to design a new system (called the to-be system).
2. The next step is requirements gathering (e.g., through interviews or questionnaires). The analysis of this information—in conjunction with input from the project sponsor and many other people—leads to the development of a concept for a new system. The system concept is then used as a basis to develop a set of business analysis models that describes how the business will operate if the new system were developed. The set of models typically includes models that represent the data and processes necessary to support the underlying business process.
3. The analyses, system concept, and models are combined into a document called the system proposal, which is presented to the project sponsor and other key decision makers (e.g., members of the approval committee) that decide whether the project should continue to move forward.

Design

The design phase decides how the system will operate, in terms of the hardware, software, and network infrastructure; the user interface, forms, and reports that will be used; and the specific programs, databases, and files that will be needed. Although most of the strategic decisions about the system were made in the development of the system concept during the analysis phase, the steps in the design phase determine exactly how the system will operate. The design phase has four steps:

1. The design strategy must be developed. This clarifies whether the system will be developed by the company's own programmers, whether it will be outsourced to another firm (usually a consulting firm), or whether the company will buy an existing software package.
2. This leads to the development of the basic architecture design for the system that describes the hardware, software, and network infrastructure that will be used. In most cases, the system will add or change the infrastructure that already exists in the organization. The interface design specifies how the users will move through the system (e.g., navigation methods such as menus and on-screen buttons) and the forms and reports that the system will use.
3. The database and file specifications are developed. These define exactly what data will be stored and where they will be stored.
4. The analyst team develops the program design, which defines the programs that need to be written and exactly what each program will do.

This collection of deliverables (architecture design, interface design, database and file specifications, and program design) is the system specification that is handed to the programming team for implementation. At the end of the design phase, the feasibility analysis and project plan are reexamined and revised, and another decision is made by the project sponsor and approval committee about whether to terminate the project or continue. See Figure 1.

Implementation

The final phase in the SDLC is the implementation phase, during which the system is actually built (or purchased, in the case of a packaged software design). This is the phase that usually gets the most attention, because for most systems it is the longest and most expensive single part of the development process. This phase has three steps:

1. System construction is the first step. The system is built and tested to ensure it performs as designed. Since the cost of bugs can be immense, testing is one of the most critical steps in implementation. Most organizations spend more time and attention on testing than on writing the programs in the first place.
2. The system is installed. Installation is the process by which the old system is turned off and the new one is turned on. It may include a direct cutover approach (in which the new system immediately replaces the old system), a parallel conversion approach (in which both the old and new systems are operated for a month or two until it is clear that there are no bugs in the new system), or a phased conversion strategy (in which the new system is installed in one part of the organization as an initial trial and then gradually installed in others). One of the most important aspects of conversion is the development of a training plan to teach users how to use the new system and help manage the changes cause by the new system.
3. The analyst team establishes a support plan for the system. This plan usually includes a formal or informal post-implementation review, as well as a systematic way for identifying major and minor changes needed for the system.

4.0 Conclusion

In this unit, we look at an overview of the concept of system analysis and design, system development team. The course unit dealt with the fundamental four phase of system development life cycle.

5.0 Summary

All system development projects follow essentially the same fundamental process called the system development life cycle (SDLC). The SDLC starts with a planning phase in which the project team identifies the business value of the system, conducts a feasibility analysis, and plans the project. The second phase is the analysis phase, in which the team develops an analysis strategy,

gathers information, and builds a set of analysis models. In the next phase, the design phase, the team develops the physical design, architecture design, interface design, data base and file specifications, and program design. In the final phase, implementation, the system is built, installed, and maintained.

Self Assessment (A False Start)

An estate group in the Federal Government co-sponsored a data warehouse with the IT department. A formal proposal was written by IT in which costs were estimated at N100,000, the project duration was estimated to be eight months, and the responsibility for funding was defined as the business unit's. The IT department proceeded with the project before hearing whether the proposal was ever accepted. The project actually lasted two years because requirements gathering took nine months instead of one and a half, the planned user base grew from 200 to 2,500, and the approval process to buy technology for the project took a year. Three weeks prior to technical delivery, the IT Director canceled the project. This failed endeavor cost the organization N2.5 million.

Question:- Why did this system fail? Why would a company spend money and time on a project and then cancel it? What could have been done to prevent this?

6.0 Tutor Marked Assignment

1. Define a system. Explain the components of a system.
2. Describe the principal steps in the analysis phase. What are the major deliverables?

7.0 References

<http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>

http://xa.yimg.com/kq/groups/22830576/1266190173/name/ISCA_Chap2_May-11.pdf

UNIT 2: THE SYSTEM ANALYST

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 System Analyst
 - 3.2 Roles of a System Analyst
 - 3.3 Qualities of a System Analyst
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 7.0 References

1.0 Introduction

The key person in any system development is the systems analyst. The system analyst analyzes business situation, identifies opportunities for improvements, and designs an information system to implement them. Being a systems analyst is one of the most interesting, exciting, and challenging jobs around. As a systems analyst, you will work with a variety of people and learn how they conduct business. Specifically, you will work with a team of systems analysts, programmers, and others on a common mission. You will feel the satisfaction of seeing systems that you designed and developed, make a significant business impact, while knowing that your unique skills helped make that happen.

It is important to remember that the primary objective of the systems analyst is not to create a wonderful system. The primary goal is to create value for the organization, which for most companies means increasing profits (government agencies and not-for-profit organizations measure value differently). Many failed systems were abandoned because the analysts tried to build a wonderful system without clearly understanding how the system would support the organization's goals, current business processes, and other information systems to provide value.

This unit looked at who a system analyst is and the role they played in the development of a new system.

2.0 Objectives

Students at the end of this course should be able to:

1. Know a system analyst
2. Roles of a System Analyst
3. Qualities of a System Analyst

3.0 Main Body

3.1 The System Analyst

A system analyst researches problems, plans solutions, recommends software and systems, and coordinates development to meet business or other requirements. They will be familiar with a variety of programming languages, operating systems, and computer hardware platforms. Because they often write user requests into technical specifications, the systems analysts are the liaisons between vendors and information technology professionals. They may be responsible for developing cost analysis, design considerations, and implication time-lines.

A system analyst may:

- Plan a system flow from the ground up.
- Interact with customers to learn and document requirements that are then used to produce business requirements documents.
- Write technical requirements from a critical phase.
- Interact with designers to understand software limitations.
- Help programmers during system development, ex: provide use cases, flowcharts or even Database design.
- Perform system testing.
- Deploy the completed system.
- Document requirements or contribute to user manuals.
- Whenever a development process is conducted, the system analyst is responsible for designing components and providing that information to the developer.

3.2 Roles of a System Analyst

The systems analyst systematically assesses how users interact with technology and businesses function by examining the inputting and processing of data and the outputting of information with the intent of improving organizational processes. Many improvements involve better support of users' work tasks and business functions through the use of computerized information systems. This definition emphasizes a systematic, methodical approach to analyzing—and potentially improving—what is occurring in the specific context experienced by users and created by a business.

The analyst plays many roles, sometimes balancing several at the same time. The three primary roles of the systems analyst are consultant, supporting expert, and agent of change.

Systems Analyst as Consultant

The systems analyst frequently acts as a systems consultant to humans and their businesses and, thus, may be hired specifically to address information systems issues within a business. Such hiring can be an advantage because outside consultants can bring with them a fresh perspective that other people in an organization do not possess. It also means that outside analysts are at a disadvantage because the true organizational culture can never be known to an outsider.

As an outside consultant, you will rely heavily on the systematic methods discussed throughout this course guide to analyze and design appropriate information systems for users working in a particular business. In addition, you will rely on information systems users to help you understand the organizational culture from others' viewpoints.

Systems Analyst as Supporting Expert

Another role a system analyst may be required to play is that of supporting expert within a business for which you are regularly employed in some systems capacity. In this role, the analyst draws on professional expertise concerning computer hardware and software and their uses in the business. This work is often not a full-blown systems project, but rather it entails a small modification or decision affecting a single department.

As the support expert, you are not managing the project; you are merely serving as a resource for those who are. If you are a systems analyst employed by a manufacturing or service organization, many of your daily activities may be encompassed by this role.

Systems Analyst as Agent Of Change

The most comprehensive and responsible role that the systems analyst takes on is that of an agent of change, whether internal or external to the business. As an analyst, you are an agent of change whenever you perform any of the activities in the systems development life cycle and are present and interacting with users and the business for an extended period (from two weeks to more than a year).

An agent of change can be defined as a person who serves as a catalyst for change, develops a plan for change, and works with others in facilitating that change. Your presence in the business changes it. As a systems analyst, you must recognize this fact and use it as a starting point for your analysis. Hence, you must interact with users and management (if they are not one and the same) from the very beginning of your project. Without their help you cannot understand what they need to support their work in the organization, and real change cannot take place.

If change (that is, improvements to the business that can be realized through information systems) seems warranted after analysis, the next step is to develop

a plan for change along with the people who must enact the change. Once a consensus is reached on the change that is to be made, you must constantly interact with those who are changing.

As a systems analyst acting as an agent of change, you advocate a particular avenue of change involving the use of information systems. You also teach users the process of change, because changes in the information system do not occur independently but cause changes in the rest of the organization as well.

3.3 Qualities of a Systems Analyst

From the foregoing descriptions of the roles the systems analyst plays, it is easy to see that the successful systems analyst must possess a wide range of qualities. Many different kinds of people are systems analysts, so any description is destined to fall short in some way. There are some qualities, however, that most systems analysts seem to display.

A problem solver:- A system analyst is a person who views the analysis of problems as a challenge and who enjoys devising workable solutions. When necessary, the analyst must be able to systematically tackle the situation at hand through skillful application of tools, techniques, and experience.

A communicator:- System analyst must be capable of relating meaningfully to other people over extended periods of time. Systems analysts need to be able to understand humans' needs in interacting with technology, and they need enough computer experience to program, to understand the capabilities of computers, to glean information requirements from users, and to communicate what is needed to programmers.

Possession of strong personal and professional ethics:- System analysts need to possess strong personal and professional ethics to help them shape their client relationships.

Technical skills- System analyst must be creative, have questioning attitude and inquiring mind. System analyst must have knowledge of the basics of the computer and business function he/she wants to develop.

4.0 Conclusion

This unit explains who a system analyst is, the role they played in system development and qualities a system analyst must possess.

5.0 Summary

A System Analyst needs variety of skills. As organizational change agents, all analysts need to have general skills, such as technical, business, analytical,

interpersonal, management, and ethical. An analyst must possess various skills to effectively carry out this job. Specifically they may be divided into 2 categories: interpersonal and technical skills. Interpersonal skills deal with relationships and the interface of the analyst with people in business. It also include following: communication, understanding (identifying problems), teaching (educating people various prorammes), selling (ideas and promoting innovations). Technical skills include: creativity, problem solving, project management, dynamic interface, questioning attitude and inquiring mind, knowledge of the basics of the computer and business function.

Self Assessment Test

Think about your ideal system analyst position. Write a newspaper advert to hire someone for that position. What requirements would the job have? What skills and experience would be required? How would applicants demonstrate that they have the appropriate skills and experience?

6.0 Tutor Marked Assignment

1. Outline ten qualities of a good System Analyst
2. Briefly outline the responsibilities of other analysts in a system development team (infrastructure analysts, business analysts, change management analysts)

7.0 References

http://en.wikipedia.org/wiki/Systems_analyst

<http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>

http://wiki.answers.com/Q/What_qualities_do_you_need_to_become_a_systems_analyst

http://www.prenhall.com/behindthebook/0132240858/pdf/Kendall_Feature1_Why_We_Wrote_This_Book.pdf

UNIT 3: SYSTEM DEVELOPMENT METHODOLOGIES

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 System Development Methodologies
 - 3.2 Selecting appropriate System Development Methodology
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 7.0 References

1.0 Introduction

A methodology is a formalized approach to implementing the SDLC (i.e., it is a list of steps and deliverables). There are many different systems development methodologies and each one is unique because of its emphasis on processes versus data and the order and focus it places on each SDLC phase. Some methodologies are formal standards used by government agencies, while others have been developed by consulting firms to sell to clients. Many organizations have their own internal methodologies that have been refined over the years, and they explain exactly how each phase of the SDLC is to be performed in that company.

All system development methodologies lead to a representation of the system concept in terms of processes and data; however, they vary in terms of whether the methodology places primary emphasis on business processes or on the data that supports the business.

Process-centered methodologies focus first on defining the activities associated with the system, i.e., the processes. Process-centered methodologies utilize process models as the core of the system concept. Analysts concentrate initially on representing the system concept as a set of processes with information flowing into and out of the processes.

Data-centered methodologies focus first on defining the contents of the data storage containers and how the contents are organized. Data-centered methodologies utilize data models as the core of the system concept.

Object-oriented methodologies attempt to balance the focus between processes and data. Object-oriented methodologies utilize the Unified Modeling Language (UML) to describe the system concept as a collection of objects incorporating both data and processes.

Another important factor in categorizing methodologies is the sequencing of the SDLC phases and the amount of time and effort devoted to each. In the early days of computing, the need for formal and well-planned life cycle

methodologies was not well understood. Programmers tend to move directly from a very simple planning phase right into the construction step of the implementation phase; in other words, they moved directly from a very fuzzy, not-well-thought-out system request into writing code.

This is the same approach that you may sometimes use when writing programs for a programming class. It can work for small programs that require only one programmer, but if the requirements are complex or unclear, you may miss important aspects of the problem and have to start all over again, throwing away part of the program (and the time and effort spent writing it). This approach also makes teamwork difficult because members have little idea about what needs to be accomplished and how to work together to produce a final product.

This unit examines several commonly used system development methodologies, tools and techniques that differ in their focus and approach to each of the system development phase.

2.0 Objectives

At the end of this unit, students should be able to understand several different categories of system development methodologies and how to choose among them.

3.0 Main Body

3.1 System Development Methodologies

There are three major categories of systems development methodologies that have evolved over time: Structured Design, Rapid Application Development (RAD), and Agile Development. Each category represents a collection of methodologies that attempts to improve on previous practice, and varies in terms of the progression through the SDLC phases and the emphasis placed on each phase.

Structured Design

The first category of systems development methodologies is called structured design. These methodologies became dominant in the 1980s, replacing the previous ad hoc and undisciplined approach. Structured design methodologies adopt a formal step-by-step approach to the SDLC that moves logically from one phase to the next.

Structured design also introduced the use of formal modeling or diagramming techniques to describe a system's basic business processes and the data that support them. Traditional structured design uses one set of diagrams to represent the processes (process models) and a separate set of diagrams to represent data (data models). Because two sets of models are used, the systems

analyst must decide which set to develop first and use as the core of the system—process models or data models. Since each type of model is important to the system, there is much debate over whether to emphasize processes before data or vice versa. Numerous process-centered and data-centered methodologies follow the basic approach of the two structured design categories outlined next.

Waterfall Development The original structured design methodology (that is still used today) is waterfall development. With waterfall development-based methodologies, the analysts and users proceed sequentially from one phase to the next (see Figure 3.1). The key deliverables for each phase are typically voluminous (often hundreds of pages in length) and are presented to the project sponsor for approval as the project moves from phase to phase. Once the sponsor approves the work that was conducted for a phase, the phase ends and the next one begins. This methodology is called waterfall development because it moves forward from phase to phase in the same manner as a waterfall. Although it is possible to go backward in the SDLC (e.g., from design back to analysis), it is extremely difficult (imagine yourself as a salmon trying to swim upstream in a waterfall as shown in Figure 3.1).

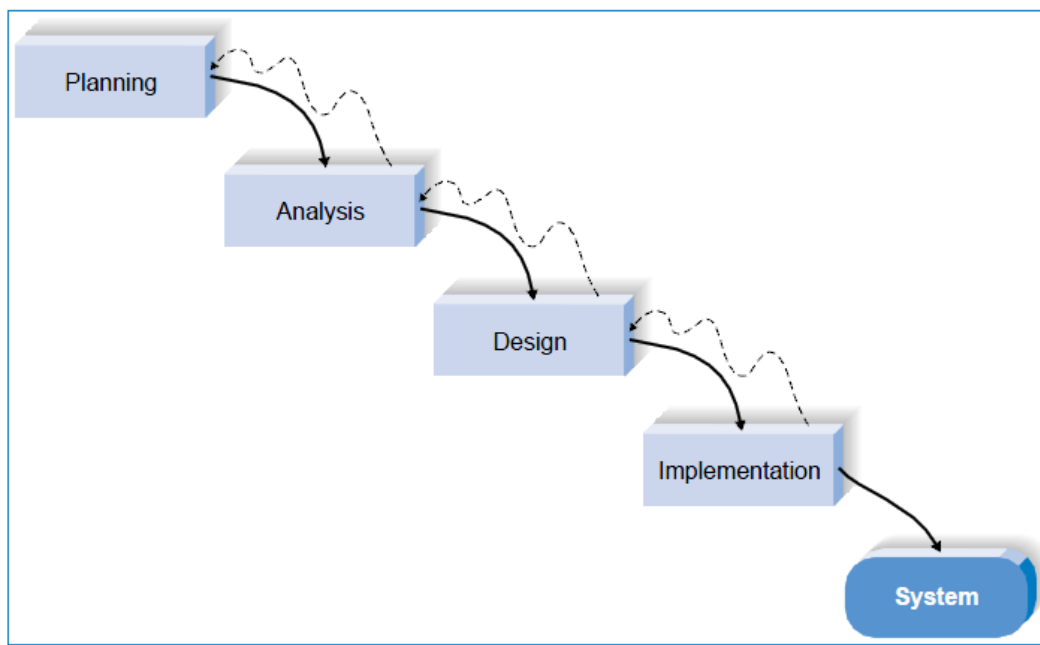


Figure 3.1: Waterfall Development based methodology

(Source:- <http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>)

The two key advantages of waterfall development-based methodologies are that system requirements are identified long before programming begins and that changes to the requirements are minimized as the project proceeds. The two key disadvantages are that the design must be completely specified before programming begins and that a long time elapses between the completion of

the system proposal in the analysis phase and the delivery of the system (usually many months or years).

The deliverables are often a poor communication mechanism, so important requirements can be overlooked in the documentation. Users rarely are prepared for their introduction to the new system, which occurs long after the initial idea for the system was introduced. If the project team misses important requirements, expensive post-implementation programming may be needed (imagine yourself trying to design a car on paper; how likely would you be to remember to include interior lights that come on when the doors open or to specify the right number of valves on the engine?).

Today's dynamic business world often imposes rapid environmental change on businesses. A system that meets existing environmental conditions during the analysis phase may need considerable rework to match the environment when it is implemented. This rework requires going back to the initial phase and making needed changes through each of the subsequent phases in turn.

Parallel Development The parallel development-based methodologies attempt to address the long time interval between the analysis phase and the delivery of the system. Instead of doing the design and implementation in sequence, a general design for the whole system is performed, then the project is divided into a series of distinct subprojects that can be designed and implemented in parallel. Once all subprojects are complete, there is a final integration of the separate pieces, and the system is delivered (Figure 3.2).

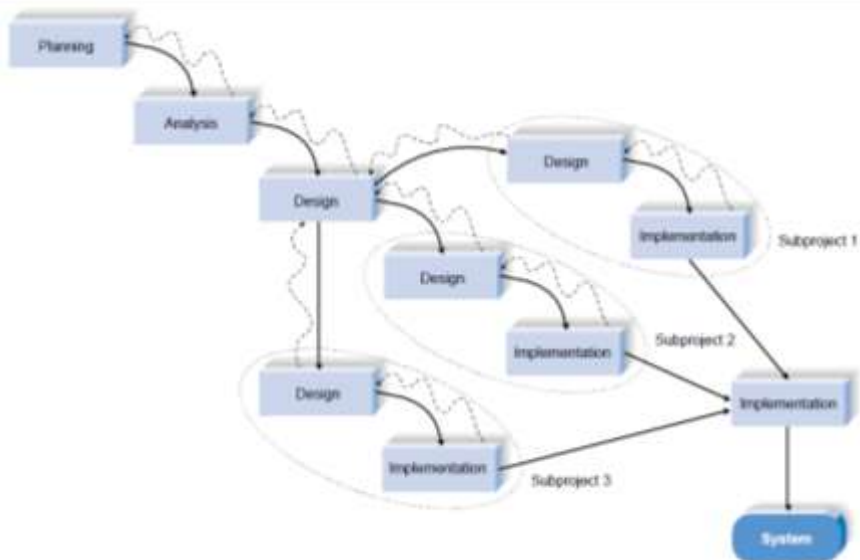


Figure 3.2: A Parallel Development-based methodology
(Source:- <http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>)

The primary advantage of these methodologies is that the schedule time required to deliver a system is shortened; thus, there is less chance of changes in the business environment causing rework. The approach still suffers from problems caused by lengthy deliverables. It also adds a new problem: sometimes the subprojects are not completely independent; design decisions made in one subproject may affect another, and the end of the project may involve significant integration challenges.

Rapid Application Development (RAD)

The second system development methodology category includes rapid application development (RAD)-based methodologies. These are a newer class of system development methodologies that emerged in the 1990s in response to both structured design methodology weaknesses. RAD-based methodologies adjust the SDLC phases to get some part of the system developed quickly and into the hands of the users. In this way, the users can better understand the system and suggest revisions that bring the system close to what is needed. There are process-centered, data-centered, and object-oriented methodologies that follow the basic approaches of the three RAD categories described in this study guide.

Most RAD-based methodologies recommend that analysts use special techniques and computer tools to speed up the analysis, design, and implementation phases, such as CASE (computer-aided software engineering) tools, JAD (joint application design), fourth-generation/visual programming languages that simplify and speed up programming (e.g., Visual Basic.NET), and code generators that automatically produce programs from design specifications. It is the combination of the changed SDLC phases and the use of these tools and techniques that improves the speed and quality of systems development.

One possible subtle problem with RAD-based methodologies, however, is managing user expectations. As systems are developed more rapidly and users gain a better understanding of information technology, user expectations may dramatically increase and system requirements expand during the project. This was less of a problem with structured design methodologies where the system requirements, once determined, were allowed only minimal change.

Phased Development The phased development-based methodologies break the overall system into a series of versions that are developed sequentially. The analysis phase identifies the overall system concept, and the project team, users, and system sponsor then categorize the requirements into a series of versions. The most important and fundamental requirements are bundled into the first version of the system. The analysis phase then leads into design and implementation, but only with the set of requirements identified for version 1 (see Figure 3.3).

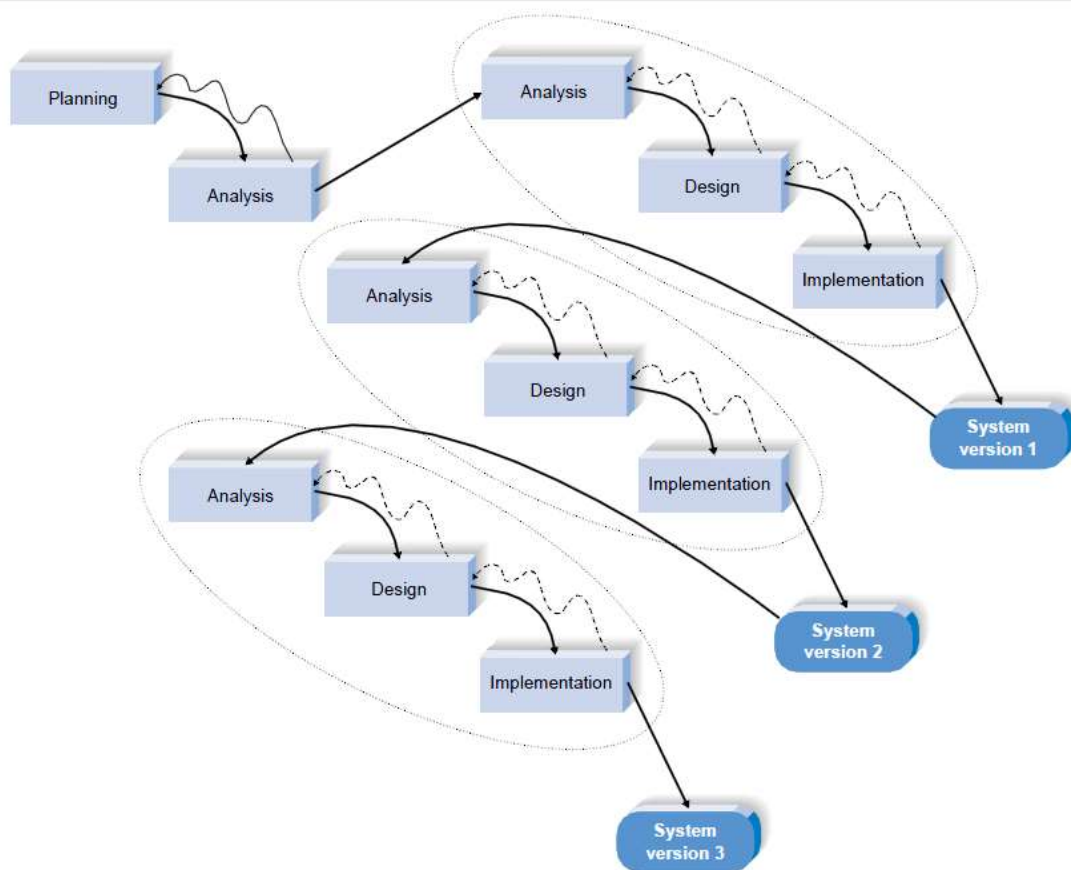


Figure 3.3: A Phased Development-based methodology

(Source:- <http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>)

Once version 1 is implemented, work begins on version 2. Additional analysis is performed on the basis of the previously identified requirements and combined with new ideas and issues that arose from users' experience with version 1. Version 2 then is designed and implemented, and work immediately begins on the next version. This process continues until the system is complete or is no longer in use.

Phased development-based methodologies have the advantage of quickly getting a useful system into the hands of the users. Although it does not perform all the functions the users need at first, it begins to provide business value sooner than if the system were delivered only after all requirements are completed, as is the case with the waterfall or parallel methodologies. Likewise, because users begin to work with the system sooner, they are more likely to identify important additional requirements sooner than with structured design situations.

The major drawback to phased development is that users begin to work with systems that are intentionally incomplete. It is critical to identify the most important and useful features and include them in the first version while managing users' expectations along the way.

Prototyping The prototyping-based methodologies perform the analysis, design, and implementation phases concurrently, and all three phases are performed repeatedly in a cycle until the system is completed. With these methodologies, a basic analysis and design are performed, and work immediately begins on a system prototype, a “quick-and-dirty” program that provides a minimal amount of features. The first prototype is usually the first part of the system that the user will use. This is shown to the users and the project sponsor, who provide reaction and comments. This feedback is used to reanalyze, redesign, and re-implement a second prototype that provides a few more features. This process continues in a cycle until the analysts, users, and sponsor agree that the prototype provides enough functionality to be installed and used in the organization. After the prototype (now called the “system”) is installed, refinement occurs until it is accepted as the new system (see Figure 3.4).

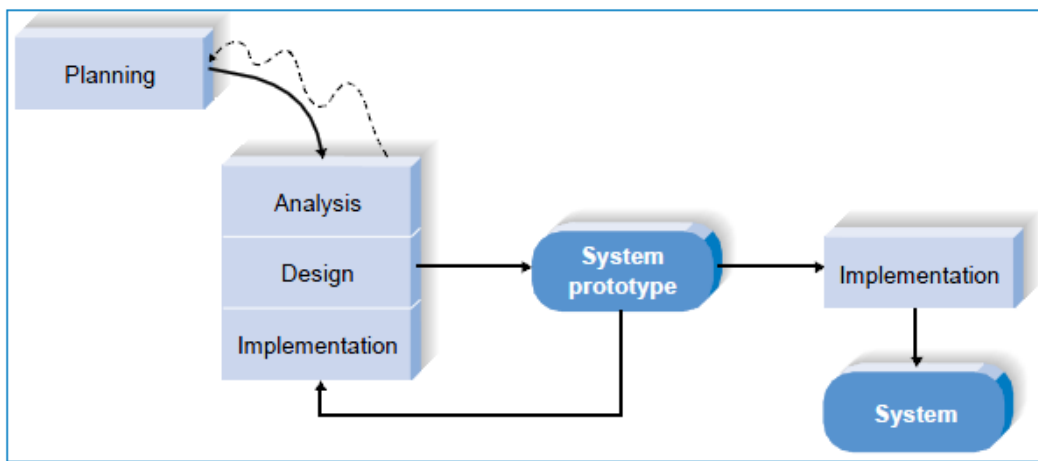


Figure 3.4: A Prototyping-based methodology

(Source:- <http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>)

The key advantage of a prototyping-based methodology is that it very quickly provides a system for the users to interact with, even if it is not initially ready for widespread organizational use. Prototyping reassures the users that the project team is working on the system (there are no long time intervals in which the users perceive little progress), and the approach helps to more quickly refine real requirements. Rather than attempting to understand system specification materials, the users can interact with the prototype to better understand what it can and cannot do.

The major problem with prototyping is that its fast-paced system releases challenge attempts to conduct careful, methodical analysis. Often the prototype undergoes such significant changes that many initial design decisions prove to be poor ones. This can cause problems in the development of complex systems because fundamental issues and problems are not recognized until well into the development process. Imagine building a car and discovering late in the prototyping process that you have to take the whole engine out to change the

oil (because no one thought about the need to change the oil until after the car had been driven 10,000 miles).

Throwaway Prototyping Throwaway prototyping-based methodologies are similar to the prototyping-based methodologies in that they include the development of prototypes; however, throwaway prototypes are done at a different point in the SDLC. These prototypes are used for a very different purpose than ones previously discussed, and they have a very different appearance (see Figure 3.5).

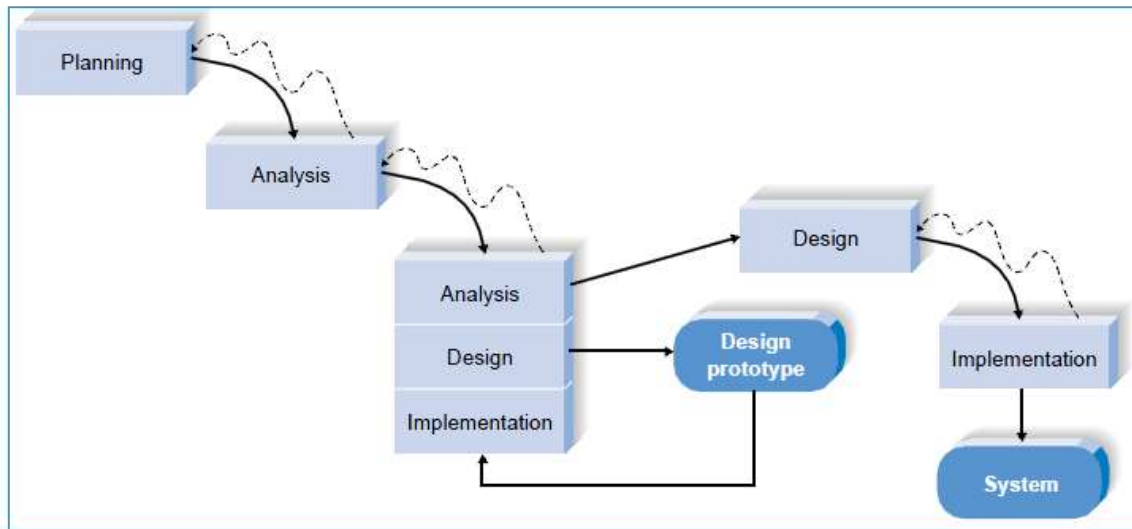


Figure 3.5: A Throwaway Prototyping-based methodology
(Source:- <http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>)

The throwaway prototyping-based methodologies have a relatively thorough analysis phase that is used to gather information and to develop ideas for the system concept. Many of the features suggested by the users may not be well understood, however, and there may be challenging technical issues to be solved. Each of these issues is examined by analyzing, designing, and building a design prototype.

A design prototype is not a working system; it is a product that represents a part of the system that needs additional refinement, and it contains only enough detail to enable users to understand the issues under consideration. For example, suppose users are not completely clear on how an order entry system should work. The analyst team might build a series of HTML pages viewed using a Web browser to help the users visualize such a system. In this case, a series of mock-up screens appear to be a system, but they really do nothing. Or, suppose that the project team needs to develop a sophisticated graphics program in Java. The team could write a portion of the program with artificial data to ensure that they could create a full-blown program successfully.

A system that is developed using this type of methodology probably uses several design prototypes during the analysis and design phases. Each of the prototypes is used to minimize the risk associated with the system by confirming that important issues are understood before the real system is built. Once the issues are resolved, the project moves into design and implementation. At this point, the design prototypes are thrown away, which is an important difference between this approach and prototyping, in which the prototypes evolve into the final system.

Throwaway prototyping-based methodologies balance the benefits of well thought-out analysis and design phases with the advantages of using prototypes to refine key issues before a system is built. It may take longer to deliver the final system as compared with prototyping-based methodologies (because the prototypes do not become the final system), but the approach usually produces more stable and reliable systems.

Agile Development

A third category of systems development methodologies is still emerging today: Agile Development. These programming-centric methodologies have few rules and practices, all of which are fairly easy to follow. They focus on streamlining the SDLC by eliminating much of the modeling and documentation overhead and the time spent on those tasks. Instead, projects emphasize simple, iterative application development. Examples of Agile Development methodologies include extreme programming, Dynamic Systems Development Method (DSDM).

Extreme Programming Extreme programming (XP) is founded on four core values: communication, simplicity, feedback, and courage. These four values provide a foundation XP developers use to create any system. First, the developers must provide rapid feedback to the end users on a continuous basis. Second, XP requires developers to follow the KISS (Keep It Simple, Stupid) principle. Third, developers must make incremental changes to grow the system and they must embrace change, not merely accept it. Fourth, developers must have a quality first mentality. XP also supports team members in developing their own skills. Three of the key principles that XP uses to create successful systems are continuous testing, simple coding performed by pairs of developers, and close interactions with end users to build systems very quickly. After a superficial planning process, project teams perform analysis, design, and implementation phases iteratively (see Figure 3.6).

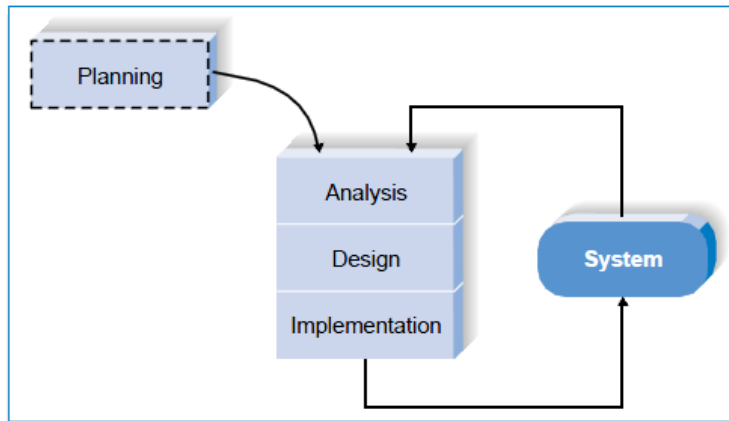


Figure 3.6: An Extreme Programming-based methodology
 (Source:- <http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>)

Testing and efficient coding practices are core to XP. In fact, each day code is tested and placed into an integrative testing environment. If bugs exist, the code is backed out until it is completely free of errors. XP relies heavily on refactoring, which is a disciplined way to restructure code to keep it simple. An XP project begins with user stories that describe what the system needs to do. Then, programmers code in small, simple modules and test to meet those needs.

Users are required to be available to clear up questions and issues as they arise. Standards are very important to minimize confusion, so XP teams use a common set of names, descriptions, and coding practices. XP projects deliver results sooner than even the RAD approaches, and they rarely get bogged down in gathering requirements for the system.

For small projects with highly motivated, cohesive, stable, and experienced teams, XP should work just fine. However, if the project is not small or the teams aren't jelled then the likelihood of a successful XP project is reduced. Consequently, the use of XP in combination with outside contractors produces a highly questionable outcome, since the outside contractors may never "jell" with insiders. XP requires a great deal of discipline to prevent projects from becoming unfocused and chaotic. Furthermore, it is only recommended for small groups of developers (not more than ten), and it is not advised for mission-critical applications. Since little analysis and design documentation is produced with XP there is only code documentation; therefore, maintenance of large systems developed using XP may be impossible. Also, since mission-critical business information systems tend to exist for a long time, the utility of XP as a business information system development methodology is in doubt. Finally, the methodology requires considerable on-site user input, something that is frequently difficult to obtain.

3.2 Selecting the Appropriate Development Methodology

Since there are many methodologies, the first challenge faced by analysts is to select which methodology to use. Choosing a methodology is not simple, because no one methodology is always best (if it were, we'd simply use it everywhere!).

Many organizations have standards and policies to guide the choice of methodology. You will find that organizations range from having one "approved" methodology to having several methodology options to having no formal policies at all. Table 3.1 summarizes some important methodology selection criteria. One important item not discussed in this figure is the degree of experience of the analyst team. Many of the RAD methodologies require the use of new tools and techniques that have a significant learning curve. Often these tools and techniques increase the complexity of the project and require extra time for learning. Once they are adopted and the team becomes experienced, the tools and techniques can significantly increase the speed in which the methodology can deliver a final system.

Table 3.1: Criteria for selecting a methodology

Ability to Develop Systems	Structured Methodologies		RAD Methodologies			Agile Methodologies
	Waterfall	Parallel	Phased	Prototyping	Throwaway Prototyping	XP
with Unclear User Requirements	Poor	Poor	Good	Excellent	Excellent	Excellent
with Unfamiliar Technology	Poor	Poor	Good	Poor	Excellent	Poor
that are Complex	Good	Good	Good	Poor	Excellent	Poor
that are Reliable	Good	Good	Good	Poor	Excellent	Good
with a Short Time Schedule	Poor	Good	Excellent	Excellent	Good	Excellent
with Schedule Visibility	Poor	Poor	Excellent	Excellent	Good	Good

4.0 Conclusion

This unit dealt with system development methodologies and how and when to choose an appropriate methodology when developing a system.

5.0 Summary

System development methodologies are formalized approaches to implementing an SDLC. System development methodologies have evolved over several decades. Structured design methodologies, such as waterfall and parallel development, move logically from one phase to the next and are more focused on system processes (process-centric) or on system data (data-centric). They produce a solid, wellthought-out system but can overlook requirements because users must specify them early in the design process before seeing the actual system. RAD-based methodologies attempt to speed up development and make it easier for users to specify requirements by having parts of the system developed sooner either by producing different versions (phased development)

or by using prototypes (prototyping, throwaway prototyping). RAD-based methodologies still tend to be either process-centric or data-centric. Agile development methodologies focus on streamlining the SDLC by eliminating many of the tasks and time associated with requirements definition and documentation. The choice of a methodology is influenced by several factors: clarity of the user requirements; familiarity with the base technology; system complexity; need for system reliability; time pressures; and need to see progress on the time schedule.

Self Assessment Test

The basic methodologies discussed in this unit can be combined and integrated to form new hybrid methodologies. Suppose you were to combine throwaway prototyping with the use of parallel development. What would the methodology look like? Draw a picture (similar to Figure 3.5). How would this new methodology compare to the others?

6.0 Tutor Marked Assignment

1. What are the key factors in selecting a methodology?
2. Compare and contrast process-centered methodologies, data-centered methodologies, and object-oriented methodologies.

7.0 References

<http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>

http://en.wikipedia.org/wiki/Systems_analyst

UNIT 4: SYSTEM DEVELOPMENT MODELS, TOOLS AND TECHNIQUES

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 The System Development Models
 - 3.2 System Development Tools and Techniques
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 7.0 Reference

1.0 Introduction

Despite the fact that ‘software crisis’ has been recognized for nearly 40 years, much software development can still be characterized as a chaotic cycle of ‘code and fix’. This can work fairly well when the system is small and business requirements are well understood. However, as the system grows, adding or changing functionality becomes difficult, bugs become harder to identify and fix, and schedules become more and more difficult to manage.

This unit looked at three fundamental system development models and tools and techniques that may be used to produce systems that simple and easy to maintain.

2.0 Objectives

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

1. Understand the three fundamental system development models
2. Know the tools and techniques used in system development

3.0 Main Body

3.1 System Development Models

Systems development methodologies impose a disciplined process upon software systems development, with the aim of making it more predictable and more efficient. They do this by structuring the development process into phases, stages and deliverables, with a strong emphasis on the planning and control of the various development activities. While there are many and various methodologies available, the underlying approach taken is normally based on one of a small number of development models.

The Waterfall Lifecycle Model

The Lifecycle Model is the traditional approach to systems development, with each stage of the lifecycle defined by the products or ‘deliverables’ produced when the stage is completed. The methods and techniques that are

recommended – dataflow diagrams, entity models, state transition diagrams – all form part of an overall system development process. This is managed by setting stages in the project for the completion of particular diagrams, for the development of software systems, or for the installation of items of hardware.

This process has become known as the ‘Waterfall Model’. Different exponents of this approach employ different names for each stage but all state that a project consists of a series of stages which create forward momentum, and which make it difficult to iterate any particular stage or, indeed, to abort a project which has become unfeasible. The traditional Waterfall Model assumes that IT strategy has been formulated and that the lifecycle provides a series of stages which control and manage a specific project. Typical stages in a waterfall lifecycle include: initial study, feasibility, investigation, business requirements definition, systems design, system specification, development, testing, implementation and review.

Starting with the initial study, the business analyst might work in conjunction with company management to select those projects which could deliver the most business benefits for the company. Feasibility will be established, sometimes using investment appraisal techniques, and a detailed analysis of the existing system will be conducted. As a result of this investigation, business requirements will be defined and a new system will be designed and specified for programmers to develop and test. Implementation will take place using the appropriate method of cutover, and the process will end with a review, which might itself be the catalyst to further projects. The waterfall approach has been defined as suitable for use when:

- a. requirements are well known in advance
- b. there are no high risk, unresolved cost implications
- c. the nature of the requirements will not change significantly
- d. the requirements match the key stakeholders’ expectations
- e. the hardware and software is well known
- f. there is enough time to proceed sequentially.

If these conditions are not met, then a more evolutionary approach is called for to mitigate the risks in the project.

The Spiral Model

The forward momentum created in the Waterfall Model can cause significant problems. To overcome these, Boehm (2000) developed a model that adopted an evolutionary approach towards the development of information systems. The Spiral Model (see Figure 2.7) is a procedure, rather than a lifecycle, because it introduces activities that are concerned with process rather than product.

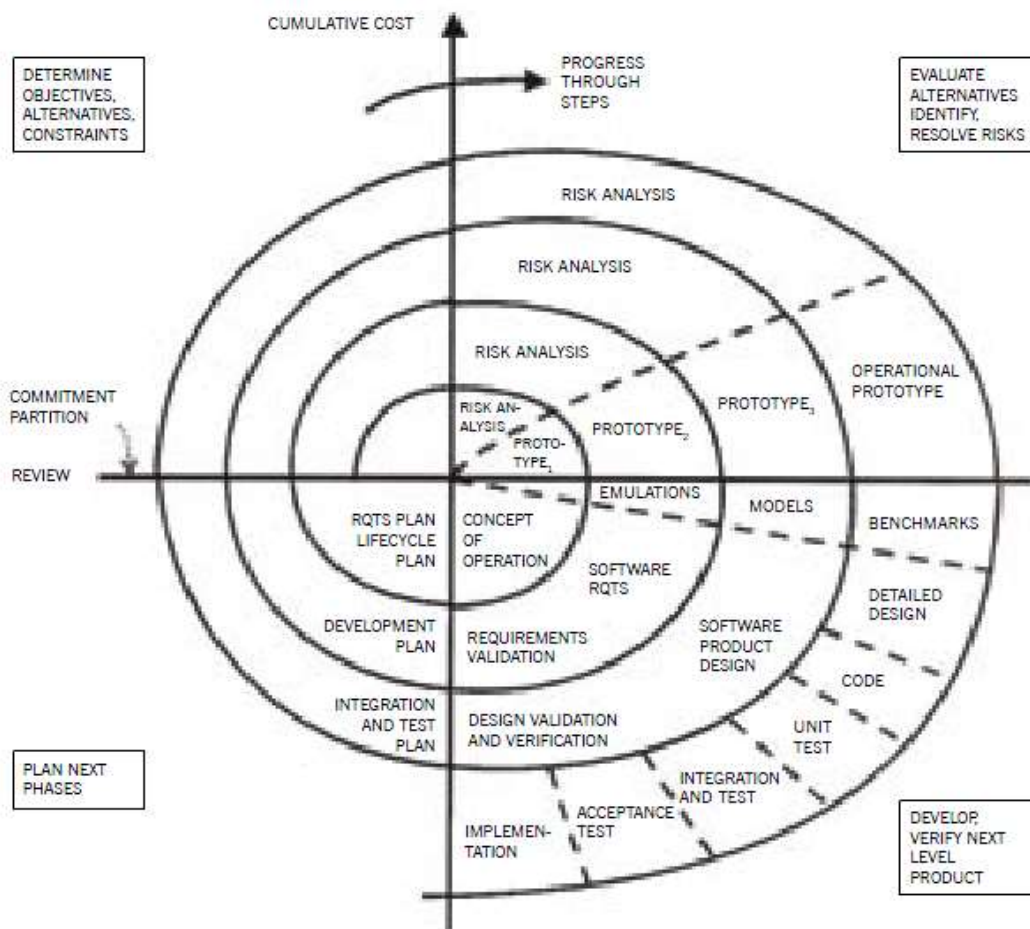


Figure 4.1: The Spiral Model

(Source:- http://xa.yimg.com/kq/groups/22830576/1266190173/name/ISCA_Chap2_May-11.pdf)

Two major features of the model are its use of risk analysis and the evolutionary nature of process. This evolutionary development is enabled by prioritizing requirements, following an initial analysis, in order to identify those features with the highest priority for the initial iteration. On completion of each iteration, user feedback indicates and prioritizes the next set of features. This enables the development team to make refinements and add more detail to each subsequent iteration.

It is the feedback and continual refinement that places this model apart from the purely incremental approach. Boehm's Spiral Model incorporates the best features of both the classic lifecycle and the prototyping approaches, by prefacing each iteration with an analysis of risk. It is an appropriate technique when developing information systems where requirements are not clear, and where a waterfall lifecycle would be inappropriate – for example, in a new development where users and IT professionals are lacking in experience. In the Spiral Model, activities are often repeated in order to clarify issues or to provide a more complete definition of user requirements. The development

process begins at the centre of the spiral even though, at this stage, the requirements are not fully defined. System requirements are refined with each cycle that is performed.

The model is divided into four quadrants. Starting from the top left, each cycle will initially identify and determine objectives, alternatives, and constraints. In the second quadrant, alternatives are evaluated and risks are analysed. In the third quadrant, the prototype is developed. Each cycle then completes in the fourth quadrant by planning for the next cycle of the spiral.

As the project progresses, the development team moves up the spiral. The first cycle results in the development of a product specification. Subsequent passes around the spiral might be used to develop a prototype, and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. The key to the model is that software evolves as the process progresses – the developer and the customer better understand and react to risks at each evolutionary level. At each level, risk analysis is performed using:

- a. risk identification – project, technical and business risks
- b. risk projection – an estimation of the risks likely to occur and of their subsequent consequences
- c. risk assessment – the prioritizing of risks, and the development of control mechanisms to eliminate/reduce the occurrence of the risk
- d. risk management and monitoring – to produce risk management and monitoring procedures.

Prototyping is used as a risk reduction mechanism. The developer can produce a prototype in each cycle of the development using the Spiral Model to define a framework for user/developer communication. Software systems projects differ from other types of development project in that the technology used is developing extremely rapidly. This has a number of implications:

- a. Customers are often unfamiliar with the latest IT developments, and so may be unable to judge whether developers are overselling a particular technology or product, or exaggerating the ease with which it can be delivered.
- b. Technological advances can make projects obsolete before they have been completed.
- c. There is a tendency, on the part of developers, to desire cutting-edge bespoke solutions, which carry greater risk, rather than use tested, commercial ‘off-the-shelf’ products.

The Agile Model

A recent survey by the British Computer Society (BCS) found that poor management of project requirements and scope were the most common causes of project failure. But for systems development projects, user requirements are often not clear at the start, for a number of reasons:

- users may be unsure of what they want

- it may be difficult to identify their tacit knowledge about day-to-day processes
- they may not have been consulted sufficiently
- user requirements may be misunderstood
- departmental and strategic requirements may be poorly defined
- external factors can cause requirements to change a 'simple' change to requirements may require a fundamental redesign of the system, with large time and cost implications.

According to the BCS study, three-quarters of IT project managers reported that, in their experience, no project had ever been delivered to the initial specifications. The most frequent criticism of the Waterfall and (to a lesser extent) the Spiral Models is that they are overly bureaucratic. There are so many things to do (that are not directly productive) in order to follow the methodology that the pace and direction of development is slow, inflexible, and difficult to change. As a reaction to these criticisms, a new approach has appeared during the past 10 years. For a while, methodologies based on this approach were known as 'lightweight', but now the generally accepted term is 'agile'. These new methods attempt a useful compromise between 'no' and 'too much' discipline, providing just enough to gain a useful payoff. The Agile Alliance was formed in 2000 with a manifesto for software development. This claims that the Alliance is 'uncovering better ways of developing software by doing it and helping others to do it', and values: individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; responding to change over following a plan.

Agile proposes 12 principles, some of which are hardly unique to the Agile approach, but an important feature is that it welcomes change and this differentiates it from Waterfall and Spiral Model developments. Agile is based on an analysis of software development practices that help companies build high quality products, and as a result, can often deliver working software in weeks rather than months. Using verbal rather than written communication, Agile is proving attractive to companies with a high degree of stakeholder agreement and that can communicate easily.

Agile methods are adaptive rather than predictive. Traditional methods attempt to plan the development process in great detail over a long period of time. This approach works well until things change, and so the temptation is to resist change. Agile methods, however, positively welcome change and constantly adapt plans, processes, workgroups and methods to respond to changed circumstances, new insights, or better understanding.

Agile methods are people-oriented rather than process-oriented. The goal of traditional development methods is to define a process that will work well whoever happens to be using it. Agile methods assert that no process will ever

make up for a lack of skill in the development team, so the primary role of any process is to support the development team in its work.

A variety of methods are available under the general heading of Agile. Varying in formality and scope they range from SCRUM (at the most 'lightweight'), through Crystal Orange and XP, to DSDM (as a RAD-like 'heavyweight' method). All are based on the frequent release of working systems to the user, and, unlike more traditional methods, all place the commitment to be on-time and on-budget above prescribed systems functionality.

3.2 System Development Tools and Techniques

System development tools and techniques help end users and system analysts to improve current IS and to develop new ones. It helps to:

- conceptualize, clarify, document and communicate the activities and recourses involved in the organization of IS.
- Analyze business operations, decision making and information processing activities.
- Propose and design new or improved IS to solve business problems or pursue business opportunities.

The major tools used for system development can be grouped into four categories based on the systems features each document has.

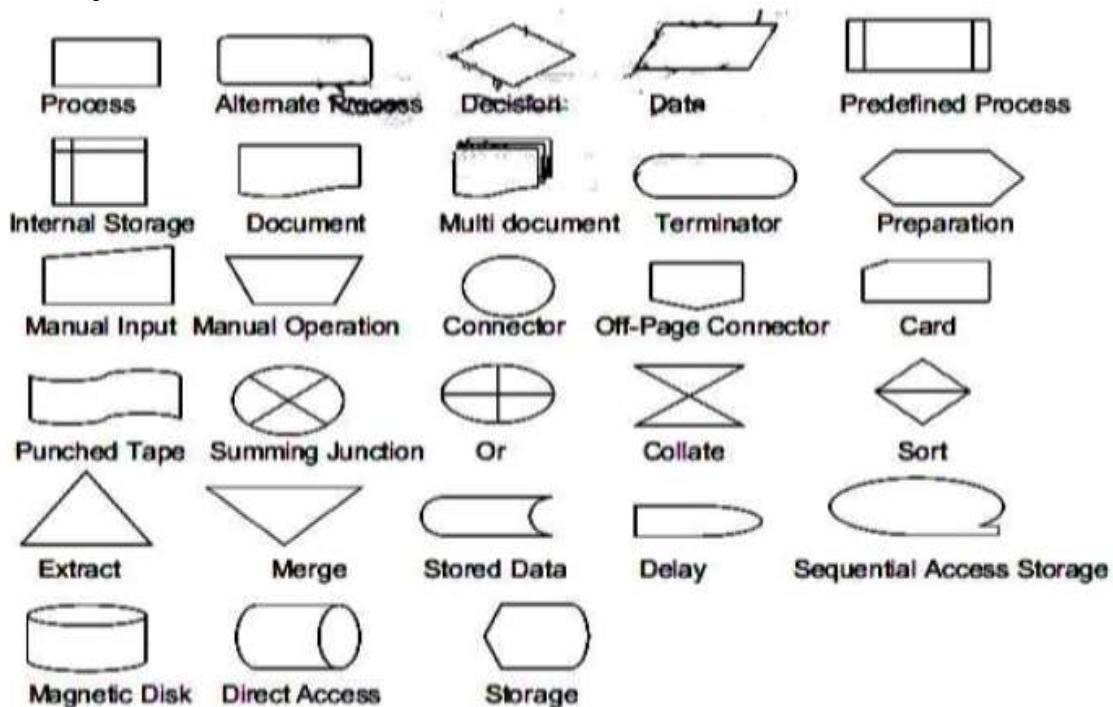
1. User Interface Tools:- help design interface between end users and the computer system.
 - a. Layout forms and screens – are used to construct the formats and contents of input/output media and responses.
 - b. Dialogue flow diagrams – analyze the flow of dialogues on screens generated by alternative user responses.
2. Data attributes and relationships tools:- define, catalogue and design data resources.
 - a. Data Dictionary – catalogue the description of the attributes (characteristics) of all data elements and their relationships to each other as well as to external systems.
 - b. Entity-relationship diagrams – help document the number and type of relationship among the entities in a system.
 - c. File layout forms – help document the type, size and names of the data elements in a system.
 - d. Grid chart – help identify the use of each type of data element in input/output or storage media of a system.
3. System components and flows Tools:- help document the data flow among the major resources and activities of an IS.
 - a. System component matrix – provides a matrix framework to document the resources, activities and information output.

- b. System flow charts – show the flow of data media as they are processed by the hardware devices and manual activities.
 - c. Data flow diagram – uses symbols to show the data flow among external entities (such as people or organizations, etc), processing activities and data storage elements.
4. Detailed system process tools:- help programmers develop detailed procedures and processes required in the design of a computer program.
- a. Decision trees and decision tables – use a network or tabular form to document the complex conditional logic involved in choosing among the information processing alternatives in a system.
 - b. Structure charts – document the purpose, structure and hierarchical relationships of the modules in a program.

Description of some of the major tools

Flowcharts – a graphic technique used by analyst to represent IPO of a business in a pictorial form. It represents an algorithm or process showing the steps as boxes of various kinds, and their order by connecting these with arrows. It is used in analyzing, designing, documenting or managing a process or program in various fields.

Symbols used in flowchart



Types of flowcharts

Are divided into four major categories

1. Document flowchart, showing a document flow through systems
2. Data flowchart – data flow in a system
3. System flowchart – controls at a physical or resource level

4. Program flowchart – controls in a program within a system

Advantages of Flowchart

1. Communication of system logic
2. Effective analysis
3. Proper documentation
4. Efficient coding – acts as a guide or blueprint during system analysis, program development phase
5. Proper debugging
6. Efficient program maintenance





Limitations of Flowcharts

1. Complex logic – makes flowchart complex and clumsy
2. Alterations and modifications requires complete re-drawing
3. Reproduction – flowchart symbols cannot be types, reproduction of flowchart becomes a problem
4. Flow lost amidst technical details.

Data Flow Diagram (DFD):- uses simple symbols to illustrate the flow of data among external entities such as, people or organizations. Processes activities and data storage elements.

DFD has four basic elements: data sources and destination, data flows, transformation processes and data stores. Data are processed by combining four symbols as shown in Table 4.1.

Table 4.1: DFD Symbols

Symbol	Name	Explanation (Symbol represents ____)
	Data Sources and Data Destinations / Sinks	People or organization that sends or receives data within the system. [= Square boxes called Data destinations or Data Sinks]
	Data flows	Flow of data into or out of a process. [Curved or straight lines with arrows]
	Transformation process / Bubbles	The processes that transform data from inputs to Outputs. [Circles/Bubbles]
	Data stores	Storage of data [Two horizontal lines]

Source & Destination of Data Flow Transformed into Data Stores

Decision Tree (Tree Diagram):- support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs and utility. Mostly used in operation research,

specifically in decision analysis, to help identify a strategy most likely to reach a goal and to calculate conditional probabilities.

Decision Table:- a table which may accompany a flowchart, defining possible contingencies in a program or appropriate course of action for each contingency.

Decision table is important because branches of flowchart multiply at each diamond (comparison symbol) and may easily run into hundred for larger systems. Programmers are liable to miss some of the branches in flowchart if there is no decision table.

Case Tools:- CASE (Computer-Aided-Software Engineering) means, automation of anything that humans do to develop systems and support virtually all phases of traditional system development process. DFD and system flowcharts that users review are commonly generated by system developers using the on-screen drawing modules found in CASE software packages. Ideal CASE system has integrated set of tools and features to perform all aspects in the life cycle.

Some of the features that various CASE products possess are repository/data dictionary; computer aided diagramming tools; word processing; screen and report generator; prototyping; project management; code generation and reverse engineering.

Data Dictionary:- computer file containing descriptive information about data items in the files of a business information system. Each computer record of a data dictionary contains information about a single data item used in a business information system. This information may include the identity of the;

1. Codes: length, type (alphabet, numeric), range (0 – 99)
2. Source: name of source document used to create the data item
3. Files: names of the files storing the data item
4. Programs: names of programs that modify the data item
5. Access rights: names of programs or users permitted to access/process the data item
6. Access denials: names of programs or users not permitted to access/process the data item

Data dictionary is updated when data items, data fields or programs are introduced/deleted.

Self Assessment Test

Suppose you are an analyst for the MX Company, a large consulting firm with offices around the world. The company wants to build a new knowledge management system that can identify and track the expertise of individual

consultants anywhere in the world on the basis of their education and the various consulting projects on which they have worked. Assume that this is a new idea that has never before been attempted in MX or elsewhere. MX has an international network, but the offices in each country may use somewhat different hardware and software. MX management wants the system up and running within a year. What model would you recommend MX Company to use? Why?

4.0 Conclusion

This unit dealt looked at three fundamental system development models, tools and techniques involved in building a new system.

5.0 Summary

While there are many and various methodologies available, the underlying approach taken is normally based on one of a small number of development models. There are three fundamental system development models – Waterfall Model, Spiral Model and Agile Model. The Lifecycle Model is the traditional approach to systems development, with each stage of the lifecycle defined by the products or ‘deliverables’ produced when the stage is completed. The forward momentum created in the Waterfall Model can cause significant problems. To overcome these, Boehm (2000) developed a model that adopted an evolutionary approach towards the development of information systems - the Spiral Model. Two major features of the model are its use of risk analysis and the evolutionary nature of process. Agile model proposes 12 principles, some of which are hardly unique to the Agile approach, but an important feature is that it welcomes change and this differentiates it from Waterfall and Spiral Model developments. Agile is based on an analysis of software development practices that help companies build high quality products, and as a result, can often deliver working software in weeks rather than months.

System development tools and techniques help end users and system analysts to improve current IS and to develop new ones. It helps to conceptualize, clarify, document and communicate the activities and recourses involved in the organization of IS; analyze business operations, decision making and information processing activities; propose and design new or improved IS to solve business problems or pursue business opportunities.

6.0 Tutor Marked Assignment

1. With the help of a diagram, explain waterfall life cycle model
2. List the advantages and limitations of Spiral Model

7.0 References

http://xa.yimg.com/kq/groups/22830576/1266190173/name/ISCA_Chap2_May-11.pdf

<http://www.wiley.com/college/dennis/0471073229/pdf/ch01.pdf>

<http://www2.accaglobal.com/pdfs/studentaccountant/bakehouse0506.pdf>

Boehm B, Spiral Development: Experience, Principles, and Refinements, Spiral Development Workshop (ed) Hansen W, 2000.

UNIT 5: PROJECT MANAGEMENT

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 Project Management Definition
 - 3.2 Project Management Process Group
 - 3.3 Project Management Triangle
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 7.0 References

1.0 Introduction

Many people and organizations today have a new—or renewed—interest in project management. Until the 1980s, project management primarily focused on providing schedule and resource data to top management in the military, computer, and construction industries. Today’s project management involves much more, and people in every industry and every country manage projects. New technologies have become a significant factor in many businesses. Computer hardware, software, networks, and the use of interdisciplinary and global work teams have radically changed the work environment. Today’s companies, governments, and nonprofit organizations are recognizing that to be successful, they need to be conversant with and use modern project management techniques. Individuals are realizing that to remain competitive in the workplace, they must develop skills to become good project team members and project managers. They also realize that many of the concepts of project management will help them in their everyday lives as they work with people and technology on a day-to-day basis.

This unit will look at meaning of project, some common IT projects, project attributes, triple constraints of a project and the meaning of project management.

2.0 Objectives

At the end of this unit, students should be able to

1. Define a Project
2. Understand Information Technology Projects
3. Know Attributes of a Project
4. Know how to manage a project
5. Understand Project Management Triple Constraints

3.0 Main Body

3.1 What is a Project

A project is “a temporary endeavor undertaken to create a unique product, service, or result.” Projects are different from operations in that they end when their objectives have been reached or the project has been terminated.

3.2 Information Technology Projects

Projects can be large or small and involve one person or thousands of people. They can be done in one day or take years to complete. Information technology projects involve using hardware, software, and/or networks to create a product, service, or result.

Examples of information technology projects include the following:

- * A small software development team adds a new feature to an internal software application for the finance department
- * A college campus upgrades its technology infrastructure to provide wireless Internet access across the whole campus
- * A cross-functional taskforce in a company decides what Voice-over-Internet-Protocol (VoIP) system to purchase and how it will be implemented.
- * A company develops a new system to increase sales force productivity and customer relationship management.
- * A television network implements a system to allow viewers to vote for contestants and provide other feedback on programs.
- * The automobile industry develops a Web site to streamline procurement
- * A government group develops a system to track child immunizations

3.3 Attributes of a Project

Projects come in all shapes and sizes. The following attributes help to define a project further:

- * *A project has a unique purpose.* Every project should have a well-defined objective. The unique purpose of a project would be to create a collaborative report with ideas from people throughout the company. The results would provide the basis for further discussions and projects.
- * *A project is temporary.* A project has a definite beginning and a definite end.
- * *A project is developed using progressive elaboration.* Projects are often defined broadly when they begin, and as time passes, the specific details of the project become clearer. Therefore, projects should be developed in increments. A project team should develop initial plans and then update them with more detail based on new information. For example, suppose a few people submitted ideas for the information technology collaboration project, but they did not clearly address how the ideas would support the business strategy of improving operations. The project team might decide to prepare a questionnaire for people to fill in as they submit their ideas to improve the quality of the inputs.

- * *A project requires resources, often from various areas.* Resources include people, hardware, software, and other assets. Many projects cross departmental or other boundaries to achieve their unique purposes. For the information technology collaboration project, people from information technology, marketing, sales, distribution, and other areas of the company would need to work together to develop ideas. The company might also hire outside consultants to provide input. Once the project team has selected key projects for implementation, they will probably require additional resources. And to meet new project objectives, people from other companies—product suppliers and consulting companies—may be added. Resources, however, are limited and must be used effectively to meet project and other corporate goals.
- * *A project should have a primary customer or sponsor.* Most projects have many interested parties or stakeholders, but someone must take the primary role of sponsorship. The project sponsor usually provides the direction and funding for the project. Once further information technology projects are selected, however, the sponsors for those projects would be senior managers in charge of the main parts of the company affected by the projects.
- * *A project involves uncertainty.* Because every project is unique, it is sometimes difficult to define its objectives clearly, estimate how long it will take to complete, or determine how much it will cost. External factors also cause uncertainty, such as a supplier going out of business or a project team member needing unplanned time off. This uncertainty is one of the main reasons project management is so challenging, especially on projects involving new technologies.

3.4 What is Project Management

Project management is “the application of knowledge, skills, tools and techniques to project activities to meet project requirements.” Project managers must not only strive to meet specific scope, time, cost, and quality goals of projects, they must also facilitate the entire process to meet the needs and expectations of the people involved in or affected by project activities.

3.5 Project Management Triple Constraint

Every project is constrained in different ways by its scope, time, and cost goals. These limitations are sometimes referred to in project management as the triple constraint. To create a successful project, a project manager must consider scope, time, and cost and balance these three often-competing goals. He or she must consider the following:

- *Scope:* What work will be done as part of the project? What unique product, service, or result does the customer or sponsor expect from the project? How will the scope be verified?

- *Time:* How long should it take to complete the project? What is the project's schedule? How will the team track actual schedule performance? Who can approve changes to the schedule?
- *Cost:* What should it cost to complete the project? What is the project's budget? How will costs be tracked? Who can authorize changes to the budget?

4.0 Conclusion

This unit make an elaborate view of the meaning of project, its attributes and project management triple constraints.

5.0 Summary

There is a new or renewed interest in project management today as the number of projects continues to grow and their complexity continues to increase. The success rate of information technology projects has more than doubled since 1995, but still only about a third are successful in meeting scope, time, and cost goals. Using a more disciplined approach to managing projects can help projects and organizations succeed.

A project is a temporary endeavor undertaken to create a unique product, service, or result. An information technology project involves the use of hardware, software, and/or networks. Projects are unique, temporary, and developed incrementally; they require resources, have a sponsor, and involve uncertainty. The triple constraint of project management refers to managing the scope, time, and cost dimensions of a project. Project management is the application of knowledge, skills, tools, and techniques to project activities to meet project requirements.

6.0 Tutor Marked Assignment

1. What does it take to deliver a successful project.
2. With the aid of diagram, explain project management triangle

7.0 References

<http://old.nios.ac.in/cca/cca1.pdf>

<http://www.augsburg.edu/ppages/~schwalbe/ipm3ch1.pdf>

UNIT 6: UNIFIED PROCESS

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 Unified Process
 - 3.2 Principles of a Unified Process
 - 3.3 Life Cycle of Unified Process
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 7.0 References

1.0 Introduction

Rational Unified Process (Unified Process) is a disciplined software development methodology that targets producing high quality software deliverables. The Rational unified process encourages systematic development while classifying tasks, assigning tasks and responsibilities within a development organization.

The course guide will then look at how to build your project using Rational Unified Process (RUP).

2.0 Objectives

Students at the end of this unit, should be able to

1. Define Unified Process
2. Know the six basic principles of Unified Process
3. Understand RUP discipline in relation to Analyst role
4. Understand Rational Unified Process Life Cycle

3.0 Main Body

3.1 What is Unified Process

The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP).

Unified Process is not simply a process, but rather an extensible framework which should be customized for specific organizations or projects. The Rational Unified Process is, similarly, a customizable framework. As a result it is often impossible to say whether a refinement of the process was derived from UP or from RUP, and so the names tend to be used interchangeably.

3.2 Principles of Rational Unified Process

The Rational Unified Process provides six guidelines to implement successful projects. The six basic principles are outlined in figure 6.1. The Rational Unified Process (RUP) supports an iterative approach to development that helps identify risk proactively, reduces re-factoring cost, and builds models with an easy exit strategy. Rational Unified Process recommends using use cases and scenarios to capture functional requirements. The Rational Unified Process supports component-based software development. Components are non-trivial modules, subsystems that fulfill a clear function. The Rational Unified Process provides a systematic approach to defining an architecture using new and existing components.

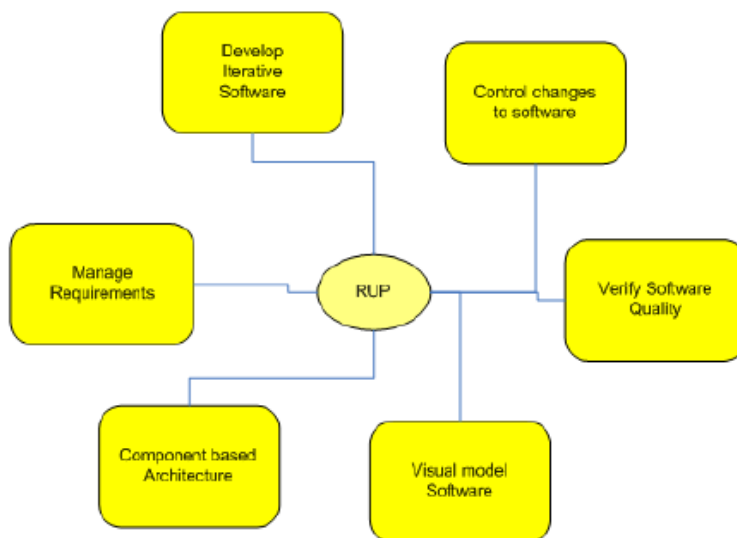


Figure 6.1: Principles of Rational Unified Process

RUP encourages visual software models to depict architectures and component. Frequent verification of quality should be reviewed with respect to the requirements based on reliability, functionality, application performance and system performance. Control changes to software are recommended by the process. The process describes how to control, track and monitor changes to enable iterative development. Project size must be optimal and must be decided based on the requirement. Process must be introduced slowly in initial part of the project and it must be implemented intensely at later parts of the project. Periodic reviews in RUP helps to identify risk proactively and continuously improve the process. Appropriation of process strength is dependent on project size, distributed teams, complexity, stakeholders and other dependent factors. More formal control is required when a project meets the following criteria:

- Project members are distributed in different places
- User community is large and distributed.
- Project is Large or Very Large.
- Many stakeholders.

Defining and understanding business needs is an important aspect of implementing RUP. Key business players must identify the business needs, and try to prioritize business and stakeholder needs.

3.2 RUP Disciplines

Rational Unified Process places structure around the activities performed and the resulting artifacts, also known as deliverables. Related to the analyst's role, RUP includes the following disciplines:

Business modeling: Provides guidance for the analyst on how to understand and visually depict a business.

Requirements: Involves finding, maintaining and managing requirements for the business application. The business models developed in the business modeling discipline are a key input to these activities.

Analysis and design: Transforms the requirements into a design of the system-to-be and adapts that design to match the implementation environment, designing it for performance. The analyst can discover flaws in design. Change requests are generated and applied. Business entities in the business modeling discipline are also an input to identifying analysis and design solutions.

Implementation: Defines the organization of code, implements classes and objects, tests the resulting implementation elements, and integrates them into an executable system. This discipline includes developer testing - that is, testing done by developers to verify that each developed element behaves as intended. This behavior derives ultimately, although often indirectly, from requirements captured by the analyst.

Test: Validates the system against (amongst other things) the requirements, ensuring that the system works properly. Requirements artifacts provided by the analyst are the basis for the definition of the evaluation activities.

Deployment: Describes the activities associated with ensuring that the software product and related materials are available for end users. The analyst produces the software requirements specification, which is one of the key inputs to development end user support and training materials.

Configuration and change management: Supports the analyst with the process of change, ensuring that changes are effectively documented and accepted during the lifecycle of the project. This also allows the analyst and those in other roles to do impact analysis.

Project management: Plans the project and each iteration and phase of the project. The requirements artifacts, particularly the requirements management plan, are important inputs to the planning activities. The driving forces behind the assessment and management activities are the requirements.

Environment: Develops and maintains the supporting artifacts that the analyst uses during requirements management and modeling.

3.3 RUP Life Cycle

RUP consists of four sequential processes or phases during design and development of a business solution. The four sequential processes includes: Inception, Elaboration, Construction and Transition. These phases are iterative in nature and yield products in each field. Iterations are between two weeks to six months. The RUP lifecycle is illustrated in figure 6.2.

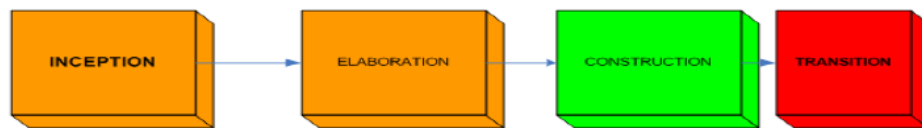


Figure 6.2: RUP Life Cycle

Inception Phase

Inception is the smallest phase in the project, and ideally it should be quite short. If the Inception Phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process.

The following are typical goals for the Inception phase.

- Establish a justification or business case for the project
- Establish the project scope and boundary conditions
- Outline the use cases and key requirements that will drive the design tradeoffs
- Outline one or more candidate architectures
- Identify risks
- Prepare a preliminary project schedule and cost estimate

Elaboration Phase

During the Elaboration phase, the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the

creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).

The architecture is validated primarily through the implementation of an Executable Architecture Baseline. This is a partial implementation of the system which includes the core, most architecturally significant, components. It is built in a series of small, timeboxed iterations. By the end of the Elaboration phase, the system architecture must have stabilized and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behavior in terms of performance, scalability and cost.

The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the Construction phase. At this point the plan should be accurate and credible, since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase.

Construction Phase

Construction is the largest phase in the project (see Figure 12 & 13). In this phase, the remainder of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, timeboxed iterations. Each iteration results in an executable release of the software. It is customary to write full text use cases during the construction phase and each one becomes the start of a new iteration. Common UML (Unified Modelling Language) diagrams used during this phase include Activity, Sequence, Collaboration, State (Transition) and Interaction Overview diagrams.

Transition Phase

The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training.

4.0 Conclusion

This unit explained the meaning of unified process, six principles of Rational Unified Process, RUP disciplines and RUP Life Cycle.

4.0 Summary

Rational Unified Process can be adopted as a whole or in parts. RUP has been successfully implemented across many organizations and it is well supported by Rational tools. RUP can be tailored to make the process agile. RUP's process-based approach helps to develop a robust system

6.0 Tutor Marked Assignment

1. What are the characteristics of Rational Unified Process known to you.
2. Does RUP contribute major significance in Project Management? How?

7.0 References

http://www.asapm.org/asapmag/articles/A7_AboutRUP.pdf

http://www.augsburg.edu/ppages/~schwalbe/C6919_ch01.pdf

http://en.wikipedia.org/wiki/Unified_Process

UNIT 7: REQUIREMENT DISCIPLINE AND DETAILED REQUIREMENT MODELLING

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 What is Requirement
 - 3.2 The Purpose of Requirement Discipline
 - 3.3 Requirement Management
 - 3.4 Requirement Modeling Vs Requirement Analysis
 - 3.5 Overview of Requirement Modeling
 - 3.6 Discovering Requirement Associations
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 7.0 References

1.0 Introduction

With no end in sight, system complexity continues to increase, worsening the challenges that have already plagued systems designers for years. Incomplete or volatile requirements, poorly specified and managed interfaces, integration testing finding problems at the very last stages of a program, development and analysis in disconnected domains, multi-domain expertise in short supply, and coordinating work and status across multilevel supply chains – these are all problems commonly discussed among systems developers. Modern software development processes, like the Rational Unified Process, prescribe iterative approach to software development. One of the fundamental assumptions of an iterative process is that system requirements don't have to be completely understood to commence development. At first glance the assumption that one can start developing a system without completely understanding its requirements seems paradoxical. However, upon closer inspection, it is the understanding of these requirements, the associated development risks, and the system architecture that drive the early iterations of system development.

In the scope of systems and software engineering, requirement modeling is increasingly recognized as a separate activity. Its importance grows with the size and complexity of the intended system. To carry out requirement modeling, a number of different approaches have been developed, many of which are supported by dedicated CASE tools (to name but a few, Caliber RM, Rational Requisite Pro, Catalyze, etc.). This unit outlines the meaning of requirement, requirement management and purpose for requirement discipline. The unit also differentiate requirement modeling from requirement analysis,

dealt with requirement modeling approach as well as requirement decomposition.

2.0 Objectives

At the end of this unit, students should be able to

1. Define requirement and understand different kinds of requirements
2. Understanding requirement management
3. Know the purpose for requirement discipline
4. Differentiate Requirement Modeling from Requirement Analysis
5. Know the general overview of Requirement Modeling
6. Discover association between requirements

3.0 Main Body

3.1 What is Requirement

A requirement is defined as "a condition or capability to which a system must conform". There are many different kinds of requirements. One way of categorizing them is described as the FURPS + model, using the acronym FURPS to describe the major categories of requirements with subcategories as shown below.

- Functionality
- Usability
- Reliability
- Performance
- Supportability

The "+" in FURPS+ reminds you to include such requirements as:

- design constraints
- implementation requirements
- interface requirements
- physical requirements.

3.2 Requirement Management

Requirements management is a systematic approach to finding, documenting, organizing and tracking the changing requirements of a system.

A formal definition of requirements management is that it is a systematic approach to:

- eliciting, organizing, and documenting the requirements of the system, and
- establishing and maintaining agreement between the customer and the project team on the changing requirements of the system.

Keys to effective requirements management include maintaining a clear statement of the requirements, along with applicable attributes for each requirement type and traceability to other requirements and other project artifacts.

Collecting requirements may sound like a rather straightforward task. In real projects, however, you will run into difficulties because:

- Requirements are not always obvious, and can come from many sources.
- Requirements are not always easy to express clearly in words.
- There are many different types of requirements at different levels of detail.
- The number of requirements can become unmanageable if not controlled.
- Requirements are related to one another and also to other deliverables of the software engineering process.
- Requirements have unique properties or property values. For example, they are neither equally important nor equally easy to meet.
- There are many interested parties, which means requirements need to be managed by cross-functional groups of people.
- Requirements change.

So, what skills do you need to develop in your organization to help you manage these difficulties? The following skills are important to master:

- Problem analysis
- Understanding stakeholder needs
- Defining the system
- Managing scope of the project
- Refining the system definition
- Managing changing requirements

Problem Analysis

Problem analysis is done to understand problems, initial stakeholder needs, and propose high-level solutions. It is an act of reasoning and analysis to find "the problem behind the problem". During problem analysis, agreement is gained on the real problem(s), and who the stakeholders are. Also, you define what from a business perspective are the boundaries of the solution, as well as business constraints on the solution. You should also have analyzed the business case for the project so that there is a good understanding of what return is expected on the investment made in the system being built.

Understanding Stakeholder Needs

Requirements come from many sources, examples would be customers, partners, end users, and domain experts. You need to know how to best determine what the sources should be, get access to those sources, and also how to best elicit information from them. The individuals who provide the

primary sources for this information are referred to as stakeholders in the project. If you're developing an information system to be used internally within your company, you may include people with end user experience and business domain expertise in your development team. Very often you will start the discussions at a business model level rather than a system level. If you're developing a product to be sold to a market place, you may make extensive use of your marketing people to better understand the needs of customers in that market.

Elicitation activities may occur using techniques such as interviews, brainstorming, conceptual prototyping, questionnaires, and competitive analysis. The result of the elicitation would be a list of requests or needs that are described textually and graphically, and that have been given priority relative one another.

Defining the System

To define the system means to translate and organize the understanding of stakeholder needs into a meaningful description of the system to be built. Early in system definition, decisions are made on what constitutes a requirement, documentation format, language formality, degree of requirements specificity (how many and in what detail), request priority and estimated effort (two very different valuations usually assigned by different people in separate exercises), technical and management risks, and initial scope. Part of this activity may include early prototypes and design models directly related to the most important stakeholder requests. The outcome of system definition is a description of the system that is both natural language and graphical.

Managing the Scope of the Project

To efficiently run a project, you need to carefully prioritize the requirements, based on input from all stakeholders, and manage its scope. Too many projects suffer from developers working on so called "Easter eggs" (features the developer finds interesting and challenging), rather than early focusing on tasks that mitigate a risk in the project or stabilize the architecture of the application. To make sure that you resolve or mitigate risks in a project as early as possible, you should develop your system incrementally, carefully choosing requirements to for each increment that mitigates known risks in the project. To do so, you need to negotiate the scope (of each iteration) with the stakeholders of the project. This typically requires good skills in managing expectations of the output from the project in its different phases. You also need to have control of the sources of requirements, of how the deliverables of the project look, as well as the development process itself.

Refining the System Definition

The detailed definition of the system needs to be presented in such a way that your stakeholders can understand, agree to, and sign off on them. It needs to cover not only functionality, but also compliance with any legal or regulatory requirements, usability, reliability, performance, supportability, and maintainability. An error often committed is to believe that what you feel is complex to build needs to have a complex definition. This leads to difficulties in explaining the purpose of the project and the system. People may be impressed, but they will not give good input since they don't understand. You should put lots of effort in understanding the audience for the documents you are producing to describe the system. You may often see a need to produce different kinds of description for different audiences.

We have seen that the use-case methodology, often in combination with simple visual prototypes, is a very efficient way of communicating the purpose of the system and defining the details of the system. Use cases help put requirements into a context, they tell a story of how the system will be used.

Another component of the detailed definition of the system is to state how the system should be tested. Test plans and definitions of what tests to perform tells us what system capabilities will be verified.

Managing Changing Requirements

No matter how careful you are about defining your requirements, there will always be things that change. What makes changing requirements complex to manage is not only that a changed requirement means that more or less time has to be spent on implementing a particular new feature, but also that a change to one requirement may have an impact on other requirements. You need to make sure that you give your requirements a structure that is resilient to changes, and that you use traceability links to represent dependencies between requirements and other artifacts of the development lifecycle. Managing change include activities like establishing a baseline, determining which dependencies are important to trace, establishing traceability between related items, and change control.

3.3 Purpose for Requirement Discipline

The purpose of the Requirements discipline is:

- To establish and maintain agreement with the customers and other stakeholders on what the system should do.
- To provide system developers with a better understanding of the system requirements.
- To define the boundaries of (delimit) the system.
- To provide a basis for planning the technical contents of iterations.

- To provide a basis for estimating cost and time to develop the system.
- To define a user-interface for the system, focusing on the needs and goals of the users.

To achieve these goals, it is important, first of all, to understand the definition and scope of the problem which we are trying to solve with this system. The Business Rules, Business Use-Case Model and Business Object Model developed during Business Modeling will serve as valuable input to this effort. Stakeholders are identified and Stakeholder Requests are elicited, gathered and analyzed.

3.4 Requirement Modeling Vs Requirement Analysis

The term "requirement analysis" is often used in the systems and software engineering to refer to a process of building up a specification of the system to be built.

Typically, this term refers to a process of building such specification of the system that:

1. Satisfies the customer's demands with respect to the system in question.
2. Provides sufficient information to build the system.

The requirement analysis often employs formal or semi-formal modeling techniques and notations, such as business process modeling, use case modeling, class or data modeling, etc. This, generally, means that the results of requirement analysis are not directly presentable to a customer, who is usually an expert in his domain area but has little understanding of these notations.

The term "requirement modeling" is somewhat similar – it refers to a process of building up a specification of the system which has the same two properties as listed above, but, in addition, is centered about the third, most important property:

3. The resulting specification must be understood in the same manner by both customer (who wants the system to be built and pays for the development) and developer (who is responsible for actually building it).

The approach described here makes a clear distinction between requirement modeling and requirement analysis:

- Requirement modeling results in a requirement specification, the main purpose of which is to allow the customer and the developer to agree on what is being developed.
- The main goal of analysis (specifically, requirement analysis) is to provide a formal or semi-formal description of the system being built (in general, the main goal of analysis is to provide a formal or semi-formal description of the problem to be solved, while design provides a formal or semi-formal description of the intended solution).

3.5 Overview of Requirement Modeling

The general process flow of the requirement modeling is shown on the Figure 7.1. This represents the general case; depending on the specifics of a system being developed, some activities may become trivial or may be bypassed altogether.

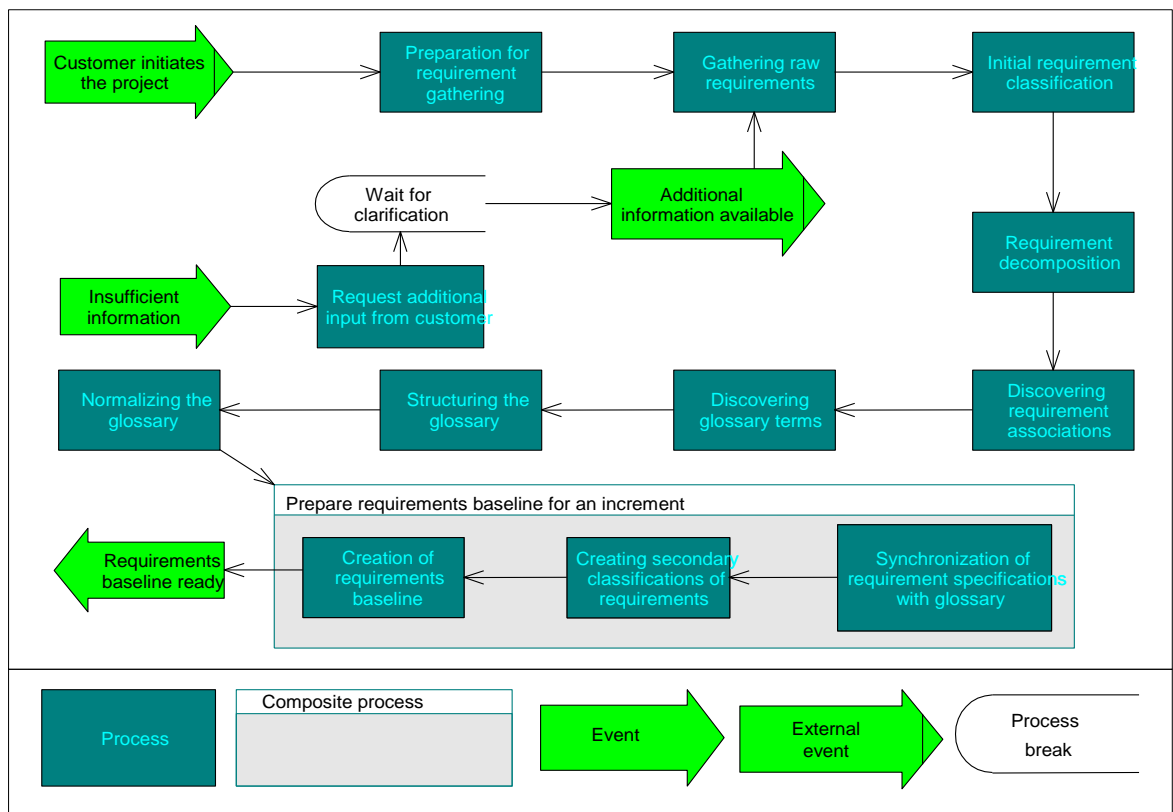


Figure 7.1: Requirement Modeling Process

Individual elements of the requirement modeling process are examined in more detail below.

Participants

In the scope of this course guide, several different parties (also known as actors or roles) are involved into the requirement modeling:

1. Customer – wants the system in question to be developed and pays for the development.
2. Developer – a person or a team who will do the actual development and deliver the system to the customer.
3. Requirement modeler – a person or a team who facilitates the communication between the customer and the developer. An important part of this communication is constituted by requirement specification.

Of course, in real projects the same person can play more than one of the above roles. This, however, does not affect the general requirement modeling process.

Initiating requirement modeling

The process of requirement gathering is always initiated by the customer. The customer's decision "it's time to start building a solution" is usually accompanied by some initial information about what the customer wants. Of course, this information rarely provides enough to build the system in question. Usually the customer will later volunteer more demands (or change existing ones) and provide clarifications to the gray areas when a developer needs it.

The preparation for requirement gathering largely depends on what tools are being used for requirement modeling. The most important part of this preparation is capturing and categorizing all information provided by the customer in its raw form - this is what requirement modeling will work with.

In the scope of our approach we use the term project document library to refer to a collection of raw information provided by the customer for the specific project. The individual documents in this library are project visions, relevant standards, customer interview logs, etc. The main idea here is that a project document library will contain all raw information relevant for the project and nothing else.

Gathering raw requirements

A single requirement is, basically, a constraint on the system being developed. Each of these constraints typically addresses a single specific aspect of the system and can be positive ("the system must do X"), negative ("the system must not do X") or some shade in between ("It would be nice for the system to do X, but we can live without it just as well").

The process of gathering raw requirements basically involves going through the project document library, finding such constraints and creating a separate requirement for each constraint found. At this point we don't care about things like classifying these requirements or whether they conflict with each other. All we need to do is to collect the requirements and tag each one with the importance / urgency (Must, Should, Could, Must not, etc.).

As the requirement modeling diagram (Figure 4.1) suggests, the process of gathering raw requirements is not usually performed in one go. Each time customer provides more information (either voluntary or answering a request from a requirement modeler or developer) this new information must be scanned for additional requirements.

Initial classification of requirements

Once requirements are identified, it's a good idea to classify them. This breaks up a large set of requirements into logically related subsets (requirement classes), which greatly simplifies working with requirements. Note, that requirement classification is usually not visible to the customer, but is introduced to make life easier for the developer.

How to classify requirement is largely project- and developer- specific. The traditional separation of requirements into functional and non-functional is well known and can be further refined if necessary. For example, non-functional requirements may be further subdivided into requirements dealing with performance, requirements dealing with security, etc. It is also often beneficial to have several independent classifications of the same set of requirements, thus providing different views of the same requirement set for different purposes.

3.6 Discovering Requirement Associations

Requirements are not independent of each other. It is up to the requirement modeler to decide what associations between requirements are relevant and shall be tracked. Specific examples of such associations include:

- Requirement dependency: if a requirement A depends upon the requirement B, then it is impossible to implement A without implementing B first. The information about requirement dependency is important for both change impact analysis and deciding what requirements to include in a specific requirement baseline.
- Requirement conflict: it is not uncommon for the customer to make contradictory demands, especially when these demands come from more than one source. When these demands become requirements, it is

essential that the information about their incompatibility be captured. Conflicting requirements can never be implemented in the same system and, as a result, always require clarification from the customer.

- Requirement equivalence: two equivalent requirements describe the same constraint on the system in different terms. Clearly, only one of these should be retained so, again, a clarification from the customer is needed.
- Requirement correlation: it is often the case that a number of requirements must be treated as a group, i.e. if one of these requirements makes it into the release, then the entire group must do the same. For example, when a composite requirement is split into simple requirements, the resulting simple requirements are correlated, because, to the customer, they are still one requirement (the original composite one; decomposition is just a way to make life easier for the developer).

4.0 Conclusion

Something has got to change. It is becoming too costly and risky to develop complex, multi-discipline systems using processes of the past. Working sequentially and in isolation, waiting for late stage testing to validate/verify the product, and relying on static documents to drive the process are outdated methods that are yielding late, costly, and malfunctioning systems. This unit uncover the meaning of system development requirements and it various kinds, the purpose for requirement discipline, requirement management, difference between requirement modeling and requirement analysis and how to discover requirement associations.

5.0 Summary

A requirement is a condition or capability to which a system must conform. There are many different kinds of requirements. One way of categorizing them is described as the FURPS + model. Requirements management is a systematic approach to finding, documenting, organizing and tracking the changing requirements of a system. Skills needed to develop to help manage requirement difficulties include problem analysis, understanding stakeholder needs, defining the system, managing scope of the project, refining the system definition and managing changing requirements. One of the purpose of requirements discipline is to establish and maintain agreement with the customers and other stakeholders on what the system should do.

6.0 Tutor Marked Assignment

1. Explain five purpose for requirement discipline.
2. With the aid of a well labeled diagram, explain requirement modeling.

7.0 References

http://sce.uhcl.edu/helm/rationalunifiedprocess/process/workflow/requirement/in_req.htm

<http://www.computer.org/comp/proceedings/re/2002/1465/00/14650006.pdf>

<http://www.erts2012.org/Site/OP2RUC89/TA-1.pdf>

UNIT 8: DESIGN ACTIVITIES AND ENVIRONMENTS

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 Tasks and Activities in Design Phase
 - 3.2 Design phase overview
 - 3.3 Design phase issues for consideration
 - 3.4 Design phase review activities
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 7.0 References

1.0 Introduction

The objective of the design phase in system development is to transform the detailed, defined requirements into complete, detailed specifications for the system to guide the work of the development phase. The decisions made in this phase address, in detail, how the system will meet the defined functional, physical, interface, and data requirements. Design Phase activities may be conducted in an iterative fashion, producing first a general system design that emphasizes the functional features of the system, then a more detailed system design that expands the general design by providing all the technical detail.

This unit discuss the tasks and activities that are performed during the design phase and an overview of the design phase environment.

2.0 Objectives

Students at the end of this unit should be able to:

1. Know various tasks and activities carried out during design phase
2. Understand the design phase overview
3. Know issues to be considered during design
4. Understand the ability to review activities that took place in design phase

3.0 Main Body

3.1 Tasks and Activities in Design Phase

The following tasks are performed during the Design Phase. The tasks and activities actually performed depend on the nature of the project.

Establish the Application Environment

Identify/specify the target, the development and the design and testing environment. How and where will the application reside. Describe the architecture where this application will be developed and tested and who is responsible for this activity.

Design the Application

In the system design, first the general system characteristics are defined. The data storage and access for the database layer need to be designed. The user interface at the desktop layer needs to be designed. The business rules layer or the application logic needs to be designed.

Establish a top-level architecture of the system and document it. The architecture shall identify items of hardware, software, and manual-operations. All the system requirements should be allocated among the hardware configuration items, software configuration items, and manual operations.

Transform the requirements for the software item into an architecture that describes its top-level structure and identifies the software components. Ensure that all the requirements for the software item are allocated to its software components and further refined to facilitate detailed design. Develop and document a top-level design for the interfaces external to the software item and between the software components of the software item.

Develop Maintenance Manual

Develop the maintenance manual to ensure continued operation of the system once it is completed.

Develop Operations Manual

Develop the Operations Manual for mainframe systems/applications and the System Administration Manual for client/server systems/applications.

Conduct Preliminary Design Review

This is an ongoing interim review of the system design as it evolves through the Design Phase. This review determines whether the initial design concept is consistent with the overall architecture and satisfies the functional, security, and technical requirements in the Functional Requirements Document.

Design Human Performance Support (Training)

Identify the users and how they will be trained on the new system. Be sure to address the users with disabilities so as to ensure equal access to all individuals.

Design Conversion/Migration/Transition Strategies

If current information needs to be converted/migrated/transitioned to the new system, plans need to be designed for those purposes, especially if converting means re-engineering existing processes.

Conduct a Security Risk Assessment

Conduct a security risk assessment by addressing the following components: assets, threats, vulnerabilities, likelihood, consequences and safeguards. The risk assessment evaluates compliance with baseline security requirements, identifies threats and vulnerabilities, and assesses alternatives for mitigating or accepting residual risks.

Conduct Critical Design Review

The Project Manager and System Proponent conduct the critical design review and approve/disapprove the project into the Development Phase. This review is conducted at the end of the Design Phase and verifies that the final system design adequately addresses all functional, security, and technical requirements and is consistent with the overall architecture.

Revise Previous Documentation

Review documents from the previous phases and assess the need to revise them during the Design Phase. The updates should be signed off by the Project Manager.

3.2 Design Phase Overview

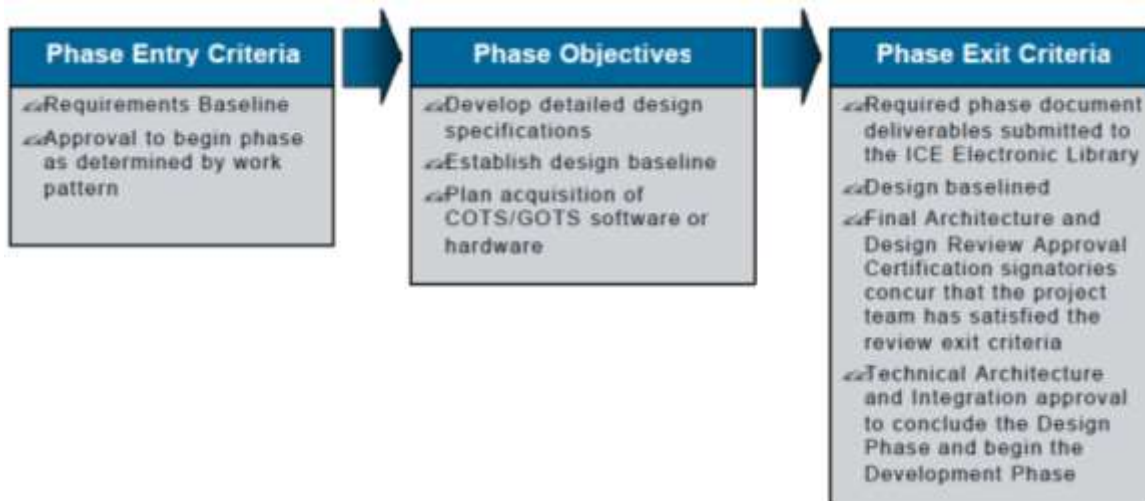


Figure 8.1: Design Phase Overview

The design environment has the project team who typically performs the key activities identified in Figure 8.1. Each activity is subsequently decomposed into the main tasks that the project team should execute to ensure that it successfully and efficiently accomplishes the activity. The activities and tasks are listed in the recommended order of completion; however, the actual order will depend on the particular project.

3.3 Design Phase issues for consideration

Project Decision Issues

The decisions of this phase re-examine in greater detail many of the parameters addressed in previous phases. The design prepared in this phase will be the basis for the activities of the Development Phase. The overall objective is to establish a complete design for the system. The pre-requisites for this phase are the Project Plan, Functional Requirements Document, and Test Plan. A number of project approach, project execution, and project continuation decisions are made in this phase.

Project approach decisions include

- Identifying existing or COTS components that can be used, or economically modified, to satisfy validated functional requirements.

- Using appropriate prototyping to refine requirements and enhance user and developer understanding and interpretation of requirements.
- Selecting specific methodologies and tools to be used in the later life cycle phases, especially the Development and Implementation Phases.
- Determining how user support will be provided, how the remaining life cycle phases will be integrated, and newly identified risks and issues handled.

Project execution decisions include

- Modifications that must be made to the initial information system need
- Modifications that will be made to current procedures
- Modifications that will be made to current systems/databases or to other systems/databases under development
- How conversion of existing data will occur

Project continuation decisions include

- The continued need of the information system to exist
- The continued development activities based on the needs addressed by the design
- Availability of sufficient funding and other required resources for the remainder of the systems life cycle

The system user community shall be included in the Design Phase actions as needed. It is also in the Design Phase that new or further requirements might be discovered that are necessary to accommodate individuals with disabilities.

Security Issues

The developer shall obtain the requirements from the System Security Plan and allocate them to the specific modules within the design for enforcement purposes. For example, if a requirement exists to audit a specific set of user actions, the developer may have to add a work flow module into the design to accomplish the auditing.

Detailed security requirements provide users and administrators with instructions on how to operate and maintain the system securely. They should address all applicable computer and telecommunications security requirements, including: system access controls; marking, handling, and disposing of magnetic media and hard copies; computer room access; account creation, access, protection, and capabilities; operational procedures; audit trail requirements; configuration management; processing area security; employee check-out; and emergency procedures. Security operating procedures may be created as separate documents or added as sections or appendices to the User and Operations Manuals. This activity should be conducted during the Design Phase.

3.4 Design Phase Review Activity

Upon completion of all design phase tasks and receipt of resources for the next phase, the Project Manager, together with the project team should prepare and present a project status review for the decision maker and project stakeholders. The review should address: (1) Design Phase activities status, (2) planning status for all subsequent life cycle phases (with significant detail on the next phase, to include the status of pending contract actions), (3) resource availability status, and (4) acquisition risk assessments of subsequent life cycle phases given the planned acquisition strategy.

4.0 Conclusion

This unit looked into the tasks and activities involved in design phase of a system development life cycle. An overview of the design phase environment and issues to be considered during the design phase was dealt with. Once the design phase is completed, there is a need to review all activities carried out before moving to the development phase, this was discussed in the last topic of this unit.

5.0 Summary

During the design phase activities, the project team transforms the baselined requirements into a detailed system design. At the conclusion of the design phase, the project team should have a design that is aligned with the architecture standards, accommodates the estimated system workload to ensure that sufficient system resources are available before the system is deployed, and can be supported within the existing infrastructure environment. Any subsequent changes to the approved system design during the development of the system or when the system is operational first must be approved by Technical Architecture and Integration before the project team may make any system changes implementing the proposed design changes.

Upon completion of all design phase tasks and receipt of resources for the next phase, the Project Manager, together with the project team should prepare and present a project status review for the decision maker and project stakeholders.

6.0 Tutor Marked Assignment

1. Explain five security requirements that are needed to be addressed during the design phase of a system development.
2. What do we mean by establishing application environment in the design phase of a system development.

7.0 References

<http://www.justice.gov/jmd/irm/lifecycle/ch7.htm>

<http://www.liteea.com/slgcp/governance/sdlc/slm7.pdf>

UNIT 9: USE CASE REALIZATION

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 Definition of Use Case
 - 3.2 First Principles of Use Case
 - 3.3 Use Case Realization
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 8.0 References

1.0 Introduction

Use cases make it clear what a system is going to do and, by intentional omission, what it is not going to do. They enable the effective envisioning, scope management and incremental development of systems of any type and any size. They have been used to drive the development of software systems. Over the years they have become the foundation for many different methods and an integral part of the Unified Modeling Language. They are used in many different contexts and environments, and by many different types of team. For example use cases can be beneficial for both small agile development teams producing user-intensive applications and large projects producing complex systems of interconnected systems, such as enterprise systems, product lines, and systems in the cloud.

This unit looked at the meaning of use case, six basic principles at the heart of successful application of use cases and how they can be used to introduce the concept of use-case modeling and use-case driven development. It also dealt with use case realization.

2.0 Objectives

Students at the end of this unit should be able to:

1. Know the meaning of use-case
2. Know the six basic first principles of use case
3. Understand use case realization.

3.0 Main Body

3.1 What is Use-Case

A use case is all the ways of using a system to achieve a particular goal for a particular user. Taken together the set of all the use cases gives you all of the useful ways to use the system, and illustrates the value that it will provide.

Use-case approach has a much broader scope than just requirements capture. They can and should be used to drive the development, which means that the supports the analysis, design, planning, estimation, tracking and testing of

systems. It does not prescribe how you should plan or manage your development work, or how you should design, develop or test your system. It does however provide a structure for the successful adoption of your selected management and development practices.

A use case analysis is the most common technique used to identify the requirements of a system (normally associated with software/process design) and the information used to both define processes used and classes (which are a collection of actors and processes) which will be used both in the use case diagram and the overall use case in the development or redesign of a software system or program. The use case analysis is the foundation upon which the system will be built.

3.2 First Principles of Use Cases

There are six basic principles at the heart of any successful application of use cases:

1. Keep it simple by telling stories
2. Understand the big picture
3. Focus on value
4. Build the system in slices
5. Deliver the system in increments
6. Adapt to meet the team's needs

Principle 1: Keep it simple by telling stories

Storytelling is how cultures survive and progress; it is the simplest and most effective way to pass knowledge from one person to another. It is the best way to communicate what a system should do, and to get everybody working on the system to focus on the same goals.

The use cases capture the goals of the system. To understand a use case we tell stories. The stories cover how to successfully achieve the goal, and how to handle any problem that may occur on the way. Use cases provide a way to identify and capture all the different but related stories in a simple but comprehensive way. This enables the system's requirements to be easily captured, shared and understood. As a use case is focused on the achievement of a particular goal, it provides a focus for the storytelling. Rather than trying to describe the system in one go we can approach it use case by use case. The results of the storytelling are captured and presented as part of the use-case narrative that accompanies each use case.

When using storytelling as a technique to communicate requirements it is essential to make sure that the stories are captured in a way that makes them actionable and testable. A set of test cases accompanies each use-case narrative to complete the use case's description. The test cases are the most important part of a use case's description, more important even than the use-case

narrative. This is because they make the stories real, and their use can unambiguously demonstrate that the system is doing what it is supposed to do. It is the test cases that define what it means to successfully implement the use case.

Principle 2: Understand the big picture

Whether the system you are developing is large or small, whether it is a software system, a hardware system or a new business, it is essential that you understand the big picture. Without an understanding of the system as a whole you will find it impossible to make the correct decisions about what to include in the system, what to leave out of it, what it will cost, and what benefit it will provide. This doesn't mean capturing all the requirements up front. You just need to create something that sums up the desired system and lets you understand scope and progress at a system level.

A use-case diagram is a simple way of presenting an overview of a system's requirements. Figure 9.1 shows the use-case diagram for a simple telephone system. From this picture you can see all the ways the system can be used, who starts the interaction, and any other parties involved. For example a Calling Subscriber can place a local call or a long-distance call to any of the system's Callable Subscribers. You can also see that the users don't have to be people but can also be other systems, and in some cases both (for example the role of the Callable Subscriber might be played by an answering machine and not a person).

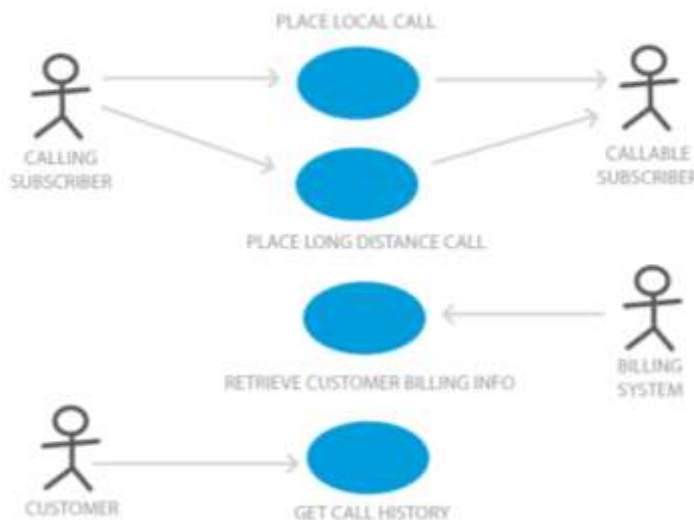


Figure 9.1: Use-Case Diagram for a simple Telephone System

A use-case diagram is a view into a use-case model. Use-case models acknowledge the fact that systems support many different goals from many different stakeholders. In a use-case model the stakeholders that use the system and contribute to the completion of the goals are modeled as actors, and the ways that the system will be used to achieve these goals are modeled as use cases. In this way the use-case model provides the context for the system's

requirements to be discovered, shared and understood. It also provides an easily accessible big picture of all the things the system will do. In a use-case diagram, such as Figure 9.1, the actors are shown as stick-men and the use cases as ellipses. The arrowheads indicate the initiator of the interaction (an Actor or the System) allowing you to clearly see who starts the use case.

A use-case model is a model of all the useful ways to use a system. It allows you to very quickly scope the system – what is included and what is not – and give the team a comprehensive picture of what the system will do. It lets you do this without getting bogged down in the details of the requirements or the internals of the system. With a little experience it is very easy to produce use-case models for even the most complex systems, creating an easily accessible big picture that makes the scope and goals of the system visible to everyone involved.

Principle 3: Focus on value

When trying to understand how a system will be used it is always important to focus on the value it will provide to its users and other stakeholders. Value is only generated if the system is actually used; so it is much better to focus on how the system will be used than on long lists of the functions or features it will offer.

Use cases provide this focus by concentrating on how the system will be used to achieve a specific goal for a particular user. They encompass many ways of using the system; those that successfully achieve the goals, and those that handle any problems that may occur. To make the value easy to quantify, identify and deliver you need to structure the use-case narrative. To keep things simple start with the simplest possible way to achieve the goal. Then capture any alternative ways of achieving the goal and how to handle any problems that might occur whilst trying to achieve the goal. This will make the relationships between the ways of using the system clear. It will enable the most valuable ways to be identified and progressed up front, and allow the less valuable ones to be added later without breaking or changing what has gone before.

Figure 9.2 shows a use-case narrative structured. The simplest way of achieving the goal is described by the basic flow. Others are presented as alternative flows. In this way you create a set of flows that structure and describe the stories, and help us to find the test cases that complete their definition.

BASIC FLOW	ALTERNATIVE FLOWS
1. Insert Card	A1 Invalid Card
2. Validate Card	A2 Non-Standard Amount
3. Select Cash Withdrawal	A3 Receipt Required
4. Select Account	A4 Insufficient Funds in ATM
5. Confirm Availability of Funds	A5 Insufficient Funds in Acct
6. Return Card	A6 Would Cause Overdraft
7. Dispense Cash	A7 Card Stuck
	A8 Cash Left Behind
	etc..

Figure 9.2: The Structure of a Use-Case Narrative

Figure 9.2 shows the narrative structure for the Withdraw Cash use case for a cash machine. The basic flow is shown as a set of simple steps that capture the interaction between the users and the system. The alternative flows identify any other way of using the system to achieve the goal such as asking for a non-standard amount, any optional facilities that may be offered to the user such as dispensing a receipt, and any problem that could occur on the way to achieving the goal such as the card getting stuck.

You don't need to capture all of the flows at the same time. Whilst recording the basic flow, it is natural to think about other ways of achieving the goal, and what could go wrong at each step. You capture these as Alternative Flows, but concentrate on the Basic Flow. You can then return to complete the alternative flows later as and when they are needed.

Principle 4: Build the system in slices

Most systems require a lot of work before they are usable and ready for operational use. They have many requirements, most of which are dependent on other requirements being implemented before they can be fulfilled and value delivered. It is always a mistake to try to build such a system in one go. The system should be built in slices, each of which has clear value to the users.

This is quite simple. First, identify the most useful thing that the system has to do and focus on that. Then take that one thing, and slice it into thinner slices. Decide on the test cases that represent acceptance of those slices. Some of the slices will have questions that can't be answered. Put those aside for the moment. Choose the most central slice that travels through the entire concept from end to end, or as close to that as possible. Estimate it as a team (estimates don't have to be "right", they're just estimates), and start building it.

Principle 5: Deliver the system in increments

Most software systems evolve through many generations. They are not produced in one go; they are constructed as a series of releases each building on the one before. Even the releases themselves are often not produced in one go, but are evolved through a series of increments. Each increment provides a demonstrable or usable version of the system. Each increment builds on the previous increment to add more functionality or improve the quality of what has come before. This is the way that all systems should be produced.

The use cases themselves can also be too much to consider delivering all at once. For example, we probably don't need all the ways of placing a local call in the very first increment of a telephone system. The most basic facilities may be enough to get us up and running. The more optional or niche ways of placing a local call such as reversing the charges or redialing the last number called can be added in later increments. By slicing up the use cases we can achieve the finer grained control required to maximize the value in each release.

Figure 9.3 shows the incremental development of a release of a system. The first increment only contains a single slice: the first slice from use case 1. The second increment adds another slice from use case 1 and the first slice from use case 2. Further slices are then added to create the third and fourth increments. The fourth increment is considered complete and useful enough to be released.

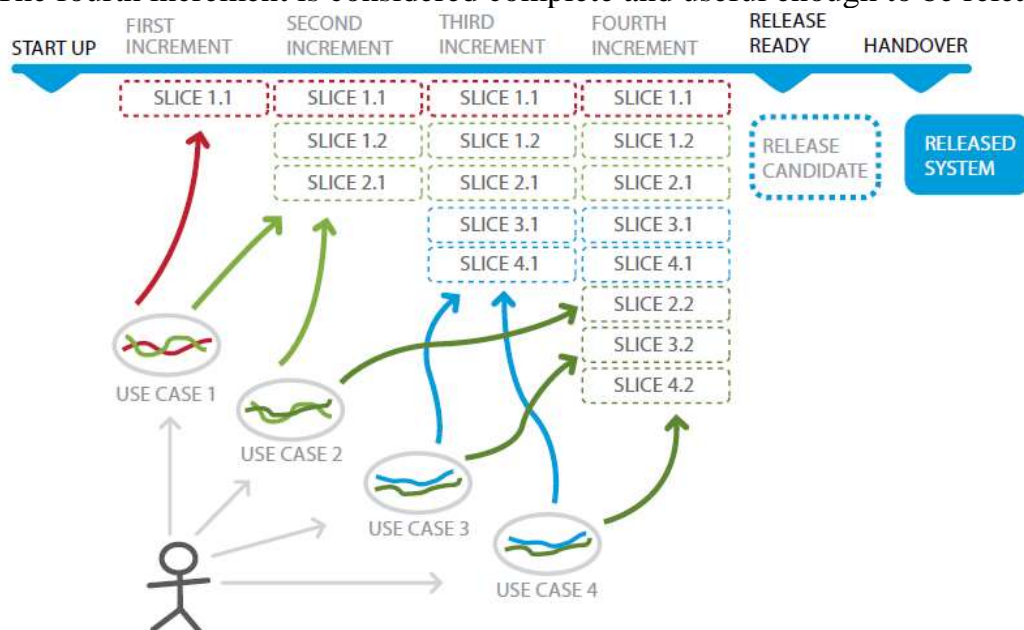


Figure 9.3: Use Cases, Use-Case Slices, Increments, and Releases

Use cases are a fabulous tool for release planning. Working at the use-case level allows whole swathes of related requirements to be deferred until the later releases. By making decisions at the use-case level you can quickly sketch out

the big picture and use this to focus in on the areas of the system to be addressed in the next release.

Use-case diagrams, showing which use cases are to be addressed in this release and which are to be left until a later release, are a great tool for illustrating the team's goals. They clearly show the theme of each release and look great pinned up on the walls of your war-room for everybody to see.

Use-case slices are a fabulous tool for building smaller increments on the way to a complete release. They allow you to target independently implementable and testable slices onto the increments ensuring that each increment is larger than, and builds on, the one before.

Principle 6: Adapt to meet the team's needs

There is no 'one size fits all' solution to the challenges of software development; different teams and different situations require different styles and different levels of detail. Regardless of which practices and techniques you select, you need to make sure that they are adaptable enough to meet the ongoing needs of the team. This applies to the practices you select to share the requirements and drive the software development as much as any other. For example lightweight requirements are incredibly effective when there is close collaboration with the users, and the development team can get personal explanations of the requirements and timely answers to any questions that arise. If this kind of collaboration is not possible, because the users are not available, then the requirements will require more detail and will inevitably become more heavyweight.

There are many other circumstances where a team might need to have more detailed requirements as an input to development. However, what's important is not listing all of the possible circumstances where a lightweight approach might not be suitable but to acknowledge the fact that practices need to scale.

3.3 Use-Case Realization

A use-case realization represents how a use case will be implemented in terms of collaborating objects. This artifact can take various forms. It can include, for example, a textual description (a document), class diagrams of participating classes and subsystems, and interaction diagrams (communication and sequence diagrams) that illustrate the flow of interactions between class and subsystem instances.

The reason for separating the use-case realization from its use case is that doing so allows the use cases to be managed separately from their realizations. This is particularly important for larger projects, or families of systems where the same use cases can be designed differently in different products within the product family. Consider the case of a family of telephone switches which have many

use cases in common, but which design and implement them differently according to product positioning, performance and price.

For larger projects, separating the use case and its realization allows changes to the design of the use case without affecting the baselined use case itself.

In a model, a use-case realization is represented as a UML collaboration that groups the diagrams and other information (such as textual descriptions) that form part of the use-case realization.

UML diagrams that support use-case realizations can be produced in an analysis context, a design context, or both, depending on the needs of the project. For each use case in the use-case model, there can be a use-case realization in the analysis/design model with a realization relationship to the use case. In UML this is shown as a dashed arrow, with an arrowhead like a generalization relationship, indicating that a realization is a kind of inheritance, as well as a dependency.

The Use-Case Model The Analysis/Design Model

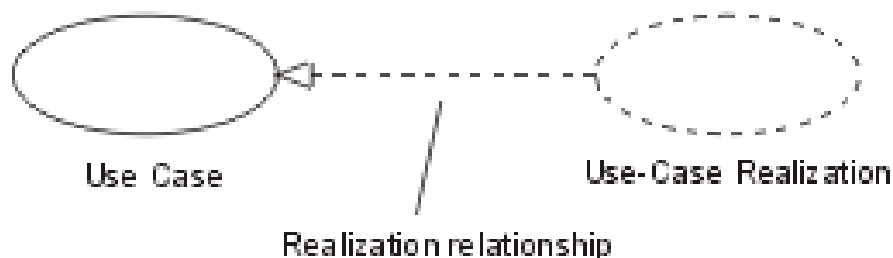


Figure 9.4: Relationship between Use Case Model and Use Case Realization

A use-case realization in the design can be traced to a use case in the use-case model.

Class Diagrams Owned by a Use-Case Realization

For each use-case realization there can be one or more class diagrams depicting its participating classes. A class and its objects often participate in several use-case realizations. It is important while designing to coordinate all the requirements on a class and its objects that different use-case realizations can have. Figure 9.5 shows an analysis class diagram for the realization of the Receive Deposit Item use case. Note the use of boundary-control-entity stereotypes to represent analysis classes.

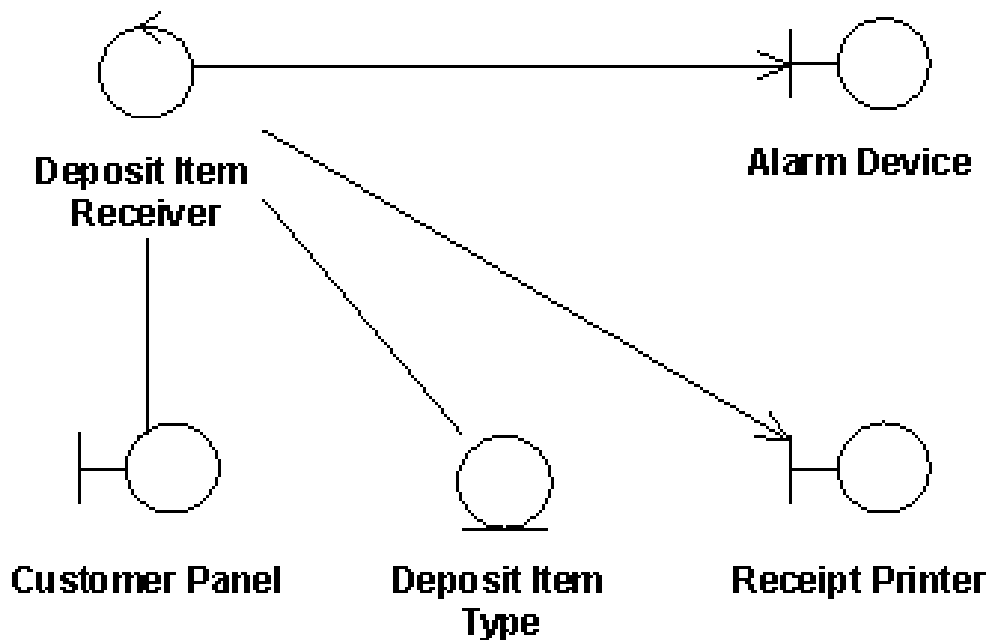


Figure 9.5 The use case Receive Deposit Item and its analysis-level class diagram.

Communication and Sequence Diagrams Owned by a Use-Case Realization

For each use-case realization there can be one or more interaction diagrams depicting its participating objects and their interactions. There are two types of interaction diagrams: sequence diagrams and communication diagrams. They express similar information, but show it in different ways. Sequence diagrams show the explicit sequence of messages and are better when it is important to visualize the time ordering of messages, whereas communication diagrams show the communication links between objects and are better for understanding all of the effects on a given object and for algorithm design.

Realizing use cases through interaction diagrams helps to keep the design simple and cohesive. Assigning responsibilities to classes on the basis of what the use-case scenario explicitly requires encourages the design to contain the following:

- Only the functionality actually used in support of a use case scenario,
- Functionality that can be tested through an associated test case,
- Functionality that is more easily traceable to requirements and changes,
- Explicitly declared class dependencies that are easier to manage.

These factors help improve the overall quality of the system.

4.0 Conclusion

This unit presents the essentials of use-case driven development as an accessible and re-usable practice. It also provides an introduction to the idea of

use cases and their application. It is not a comprehensive guide to all aspects of use cases, or a tutorial on use-case modeling. It may not be sufficient for you to adopt the practice.

5.0 Summary

The purpose of a test case is to provide a clear definition of what it means to complete a slice of the requirements. A test case defines a set of test inputs and expected results for the purpose of evaluating whether or not a system works correctly. Test cases provide the building blocks for designing and implementing tests, provide a mechanism to complete and verify the requirements, allow tests to be specified before implementation start and provide a way to assess system quality.

A use-case model is a model of all of the useful ways to use a system, and the value that they will provide. The purpose of a use-case model is to capture all of the useful ways to use a system in an accessible format that captures a system's requirements and can be used to drive its development and testing. A use-case model allows teams to agree on the required functionality and characteristics of a system, clearly establishes the boundary and scope of the system by providing a complete picture of its actors (being outside the system) and use cases (being inside the system) and enables agile requirements management.

The purpose of a use-case narrative is to tell the story of how the system and its actors work together to achieve a particular goal. Use-case narratives outline the stories used to explore the requirements and identify the use-case slices, describe a sequence of actions, including variants that a system and its actors can perform to achieve a goal, are presented as a set of flows that describe how an actor uses a system to achieve a goal, and what the system does for the actor to help achieve that goal, capture the requirements information needed to support the other development activities.

The purpose of a use-case realization is to show how the system's elements, such as components, programs, stored procedures, configuration files and database tables, collaborate together to perform a use case. Use-case realizations identify the system elements involved in the use cases, capture the responsibilities of the system elements when performing the use case, describe how the system elements interact to perform the use case and translate the business language used in the use-case narratives into the developer language used to describe the system's implementation.

6.0 Tutor Marked Assignment

1. Explain in detail, what you understand by use-case model and use-case narrative.
2. Of what significance is UML in use-case realization?

7.0 References

http://epf.eclipse.org/wikis/openup/practice.tech.use_case_driven_dev.base/guidances/guidelines/uc_realizations_448DDA77.html

<http://www.outsideininc.com/wp-content/uploads/2012/02/Use-Case-2-0-Feb14-2012.pdf>

UNIT 10: SYSTEM ACCESS AND IMPLEMENTATION

Content

- 1.0 Introduction
- 2.0 Objectives
- 3.0 Main Body
 - 3.1 Meaning of System Implementation
 - 3.2 System Implementation Processes
 - 3.3 Measurement of System Success
- 4.0 Conclusion
- 5.0 Summary
- 6.0 Tutor Marked Assignment
- 7.0 References

1.0 Introduction

The purpose of system implementation can be summarized as making the new system available to a prepared set of users (the deployment), and positioning on-going support and maintenance of the system within the performing organization (the transition). At a finer level of detail, deploying the system consists of executing all steps necessary to educate the consumers on the use of the new system, placing the newly developed system into production, confirming that all data required at the start of operations is available and accurate, and validating that business functions that interact with the system are functioning properly. This unit discussed the system implementation phase of system development life cycle and describe how a system can be successfully deployed for access by the organization.

2.0 Objectives

Students at the end of this unit, should be able to:

1. Understand System Implementation
2. Know list of processes and deliverables in system implementation
3. Measure success of system access and implementation

3.0 Main Body

3.1 Meaning of System Implementation

System implementation is a realization of a technical specification of a system. A key difference between System Implementation and all other phases of the lifecycle is that all project activities up to this point have been performed in safe, protected, and secure environments, where project issues that arise have little or no impact on day-to-day business operations. Once the system goes live, however, this is no longer the case. Any miscues at this point will almost certainly translate into direct operational and/or financial impacts on the Performing Organization. It is through the careful planning, execution, and management of system implementation activities that the Project Team can minimize the likelihood of these occurrences, and determine appropriate contingency plans in the event of a problem.

3.2 System Implementation Processes

This phase consists of the following processes:

1. *Prepare for System Implementation*, where all steps needed in advance of actually deploying the application are performed, including preparation of both the production environment and the Consumer communities.
2. *Deploy System*, where the full deployment plan, initially developed during System Design and evolved throughout subsequent lifecycle phases, is executed and validated.
3. *Transition to Performing Organization*, where responsibility for and ownership of the application are transitioned from the Project Team to the unit in the Performing Organization that will provide system support and maintenance.

Table 10.1 illustrates all of the processes and deliverables of this phase in the context of the system development lifecycle.

Table 10.1: List of Process and Deliverables

Processes	Techniques	Process Deliverables (Outcomes)
Prepare for System Implementation	Interviews Distribution of Materials Coordination of Training Logistics	<i>Established Team and Environment for System Implementation</i>
Deploy System	Training Sessions Manual Business Operations Parallel Operations	<i>Migrated and Initialized Data Operational System</i>
Transition to Performing Organization	Training Sessions Phased Ownership	<i>Ownership of System by Performing Organization</i>

Preparing for System Implementation

The purpose of preparing for system implementation is to take all possible steps to ensure that the upcoming system deployment and transition occurs smoothly, efficiently, and flawlessly.

In the implementation of any new system, it is necessary to ensure that the consumer community is best positioned to utilize the system once deployment efforts have been validated. Therefore, all necessary training activities must be scheduled and coordinated. As this training is often the first exposure to the system for many individuals, it should be conducted as professionally and competently as possible. A positive training experience is a great first step towards customer acceptance of the system.

Deploying the System

The purpose of the deploying system process is to perform all activities required to successfully install the new system and make it available to the consumers.

Deploying the system is the culmination of all prior efforts – where all of the meetings, planning sessions, deliverable reviews, prototypes, development, and testing pay off in the delivery of the final system. It is also the point in the project that often requires the most coordination, due to the breadth and variety of activities that must be performed. Depending upon the complexity of the system being implemented, it may impact technical, operational, and cultural aspects of the organization. A representative sample of high-level activities might include the installation of new hardware, increased network capabilities, deployment and configuration of the new system software, a training and awareness campaign, activation of new job titles and responsibilities, and a completely new operational support structure aimed at providing Consumer-oriented assistance during the hours that the new system is available for use (to name a few).

Transition to Performing Organization

The purpose of transition to performing organization process is to successfully prepare the organization to assume responsibility for maintaining and supporting the new application.

In many organizations, the team of individuals responsible for the long-term support and maintenance of a system is different from the team initially responsible for designing and developing the application. Often, the two teams include a comparable set of technical skills. The responsibilities associated with supporting an operational system, however, are different from those associated with new development. In order to effect this shift of responsibilities, the Project Team must provide those responsible for system support in the organization with a combination of technical documentation, training, and hands-on assistance to enable them to provide an acceptable level of operational support to the consumers. This system transition is one element (albeit a major one) of the overall Project Implementation and Transition Plan, developed as part of the Project Management Lifecycle. The Project Manager should review the transition plan to confirm that all defined actions have been successfully completed.

3.3 Measurement of System Access and Implementation Success

System access and implementation serves as its own measurement of success; indeed, a smooth implementation culminates – and validates – the entire system development effort. Nevertheless, even before the final turnover, the Project Manager can utilize the measurement criteria listed Table 10.2 to assess how successfully the implementation is proceeding. More than one “No” answer indicates a serious risk to the success of this phase – and the entire project.

Table 10.2: Measurement of System Success

Process	Measurements of Success	Yes	No
Prepare for System Implementation	Has anyone verified that every Consumer has the right level of system access and security?		
	Is there a checklist of all system components that can be used to verify that all the right versions of all components of the system are in the production environment?		
	Do the managers of Technical Services and Technical Support agree with your estimate of extra work for their units associated with new system deployment?		
Deploy System	Do your team members agree that their part of the effort as outlined in the Project Implementation and Transition Plan is reasonable and achievable?		
	Do the training evaluation forms filled out by Consumers and Customers being trained in the new system reflect scores equal or higher to those anticipated in the Project Implementation and Transition Plan?		
	Have you had to "freeze" or "fall back" in system deployment activities no more than originally anticipated in the deployment plan?		
	Is the volume of support calls within the range originally anticipated in the deployment plan?		
Transition to Performing Organization	Has the Performing Organization agreed to transition all of the remaining defects along with the system itself?		

4.0 Conclusion

This unit described system implementation phase of system development life cycle. It explained list of processes and deliverables in a system implementation and how to measure success of any new system.

5.0 Summary

Transitioning the system support responsibilities involves changing from a system development to a system support and maintenance mode of operation, with ownership of the new system moving from the Project Team to the Performing Organization.

6.0 Tutor Marked Assignment

Does a system need to be perfect before deployment?

7.0 References

- <http://en.wikipedia.org/wiki/Implementation>
- <http://www.its.ny.gov/pmmp/guidebook2/SystemImplement.pdf>