ST. XAVIER'S COLLEGE

(Affiliated to Tribhuvan University)

Maitighar, Kathmandu

CSC-351 Software Engineering
Class Notes

Typed and compiled by:

Siddhant Rimal

THIS NOTE IS STILL BEING WRITTEN; UPDATED WEEKLY.
Newer revisions will be made available every week.

SINCE: Sept 7th 2016 | This revision: 11.20 – Update-17

Class-Notes on Software Engineering; Current version: 11.20 – Update-17

Notes referenced from:

- Lectures by Er. Sanjay Kr. Yadav
- Class-Notes (Nuja Joshi)
- Software Engineering 6th Edition (Ian Sommerville)*
- Software Engineering 9th Edition (Ian Sommerville)*

*TU recommended Textbook

Typed and Compiled by:

- Siddhant Rimal

Other typists:

- Ayush Suwal
- Sneha Prasai
- Divya Jyoti Pokharel

Images/Figures provided by:

- Rojesh Tamrakar
- Siddhant Rimal

Contents

1. SOFTWARE ENGINEERING 7

1.1 FAQ'S ON SE 7
1.1.1 What is software? 7
1.1.2 What is software engineering? 7
1.1.3 What is the difference between software engineering and computer science? 7
1.1.4 What is the difference between software engineering and system engineering? 7
1.1.5 What is software process? 7
1.1.6 What is software process model? 8
1.1.7 What are generic process models? (WIFE) {#Important} 8
1.1.8 What are the costs of software engineering? 8
1.1.9 What are software engineering methods? 8
1.1.10 What is CASE? {#Important topic generally asked for 5 marks} 8
1.1.11 What are the attributes of good software? 9
1.1.12 What are the key challenges facing software engineering? 10
1.1.13 What are professional and ethical responsibility of software engineering? 10
1.1.14 Software engineering is an engineering discipline that is concerned with all aspects of software production. Why? 10
1.1.15 Discuss whether professional engineers should be certified in the same way as doctors or lawyers. 10
1.1.16 What are the differences between generic software product development and custom software development? 11
1.1.17 What is the difference between software process model and a software? 11
1.1.18 Suggest two mays in which a software process model might be helpful in identifying possible process improvements. 11
1.2 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS: 11
1.2.1 Functional Requirements: 11
1.2.2 Non-functional Requirements: 11
1.2.3 Domain Requirements: 11
1.2.4 Functional Requirements 11
1.3 PRODUCT REQUIREMENTS: 12
1.4 ORGANIZATIONAL REQUIREMENTS: 12
1.5 EXTERNAL REQUIREMENTS: 12
1.6 THE REQUIREMENT DOCUMENTS: 12
1.7 USERS OF REQUIREMENT DOCUMENTS: 13

2. IEEE CODE OF ETHICS OF PRINCIPLE (SOFTWARE ENGINEERING CODE OF ETHICS FOR PROFESSIONAL PRACTICE) 13

3. SYSTEM ENGINEERING 14

3.1 WHAT IS A SYSTEM? 14
3.2 PROBLEMS OF SYSTEM ENGINEERING: 14
3.3 EMERGENT PROPERTIES 14
3.3.1 Examples of emergent properties: 15
3.4 TYPES OF EMERGENT PROPERTIES 15
3.4.1 Functional Properties 15
3.4.2 Non-functional Properties 15
3.5 THE SYSTEM ENGINEERING PROCESS 15

4. SYSTEM PROCUREMENT PROCESS 15
4.1 SPECIFICATIONS AND ARCHITECTURAL DESIGN 16
4.2 PROCUREMENT ISSUES 16
4.3 CONTRACTOR AND SUBCONTRACTOR 16
5. SYSTEM RELIABILITY ENGINEERING 17
5.1 RELIABILITY 17
5.1.1 Hardware Reliability 17
5.1.2 Software Reliability 17
5.1.3 Operator Reliability 17
6. GENERIC SOFTWARE PROCESS MODELS 17
6.1 THE WATERFALL MODEL 18
6.2 FUNDAMENTAL DEVELOPMENT ACTIVITIES: 18
6.2.1 Requirement analysis and definition 18
6.2.2 System and software design 18
6.2.3 Implementation and unit testing 18
6.2.4 Integration and system testing 18
6.2.5 Operation and maintenance 18
7. SOFTWARE SPECIFICATION 19
8. REQUIREMENTS ENGINEERING PROCESS 19
8.1 FEASIBILITY STUDY 19
8.2 REQUIREMENTS ELICITATION AND ANALYSIS 20
8.2.1 Process of Requirements Elicitation and Analysis 22
8.2.2 Viewpoints 23
8.2.3 Interviewing 23
8.2.4 Scenarios 23
8.2.4.1 Use-Case and Sequence Diagram of Library 24
8.2.4.2 Use-Case and Sequence Diagram of Library (Alternate) 25
8.2.4.3 Use-Case and Sequence Diagram of Bank 26
8.2.6 Ethnography 27
8.3 REQUIREMENT SPECIFICATION 27
8.4 REQUIREMENT VALIDATION 27
8.4.1 Requirement Reviews 27
8.4.2 Requirements Checking 27
8.4.3 Requirements Validation Checking 28
8.4.4 Requirements Management 28
8.5 ENDURING REQUIREMENTS 29
8.6 VOLATILE REQUIREMENTS 29
8.7 CLASSIFICATION OF REQUIREMENTS 29
8.8 REQUIREMENTS MANAGEMENT PLANNING 29
8.9 TRACEABILITY 29
8.10 SOURCE TRACEABILITY 30
8.11 REQUIREMENTS TRACEABILITY 30
8.12 DESIGN TRACEABILITY 30
8.13 REQUIREMENTS IDENTIFICATION 30
8.14 A CHANGE MANAGEMENT PROCESS 30

8.15 REQUIREMENTS CHANGE MANAGEMENT 31
8.15.1 Problem Analysis and Change Specification 31
8.16 CHANGE ANALYSIS AND COSTING 31
8.17 CHANGE IMPLEMENTATION 31
9. SOFTWARE DESIGN AND IMPLEMENTATION: 31
9.1 ARCHITECTURAL DESIGN 32
9.2 ABSTRACT SPECIFICATION 32
9.3 INTERFACE DESIGN 32
9.4 COMPONENT DESIGN 32
9.5 DATA STRUCTURE AND DESIGN 32
9.6 ALGORITHM DESIGN 32
10. DESIGN MODEL (STRUCTURED METHOD) 32
10.1 POSSIBLE MODELS 32
10.2 PROGRAMMING & DEBUGGING 33
10.3 SOFTWARE VALIDATION (V & V MODEL) 33
10.4 THE TESTING PROCESS 34
10.4.1 Component (Unit) testing: 34
10.4.2 System testing: 34
10.4.3 Acceptance testing: 34
10.5 SOFTWARE EVALUATION 34
11. PROJECT MANAGEMENT (UNIT 1.4) 35
11.1 SOFTWARE PROJECT MANAGEMENT 35
11.2 SOFTWARE MANAGEMENT DISTINCTION 35
11.3 MANAGEMENT ACTIVITIES 35
11.4 PROJECT COSTING 36
11.5 PROJECT STAFFING 36
11.6 PROJECT PLANNING 36
11.7 TYPES OF PLAN 36
11.8 PROJECT PLAN STRUCTURE: 37
11.9 PROJECT SCHEDULING 37
11.9.1 Bar Charts and Activity Networks: 38
11.9.2 Activity Networks 39
11.10 RISK MANAGEMENT PROCESS 40
11.10.1 Risk Identification: [Text-Book 9th Edition Pg-598 for reference] 41
1. Technology risk 41
2. People risk 41
3. Organizational risk 41
4. Requirement risk 41
5. Estimation risk 41
6. Tools risk 41
11.10.2 Risk Analysis 41
12. SOFTWARE REQUIREMENT 43
12.1 WHAT IS A REQUIREMENT? 43
12.2 TYPES OF REQUIREMENTS 43
13. MODELS{STH SIMILAR HEADING} 43

13.1 REPOSITORY MODEL 43
13.2 CLIENT-SERVER ARCHITECTURE 45
13.3 ABSTRACT MACHINE MODEL 45
13.4 CONTROL MODELS 46
13.4.1 Centralized Control 46
13.5 CALL RETURN MODEL 46
13.6 MANAGER MODEL 47
13.7 MODULAR DECOMPOSITION 47
13.7.1 Object Model 47
13.7.1.1 Advantages of Object Model 48
13.7.1.2 Disadvantages of Object Model 48
13.7.2 Data Flow Model (Filter – *gets purified while moving from one step to another*) 48
13.7.2.1 Advantages of Dataflow Model 49
13.7.2.2 Disadvantages of Dataflow Model 49
13.8 DOMAIN SPECIFICATION ARCHITECTURE 49
13.8.1 Generic Model 49
13.8.2 Reference Model 49

14. FORMAL SPECIFICATION 50

14.1 FORMAL SPECIFICATION(UNIT 2.3) 50
14.2 FORMAL METHODS 50
14.3 ACCEPTANCE OF FM 50
14.4 USES OF FMS 51
14.5 SPECIFICATIONS IN S/W PROCESS 51
14.6 SPECIFICATION TECHNIQUES 51
14.6.1 Algebraic Approach 51
14.6.2 Model based approach 51
14.6.3 Use of formal specification 52
14.7 INTERFACE SPECIFICATION 52

15. OBJECT ORIENTED DESIGN 54

15.1 CHARACTERISTICS OF OOD 54
15.2 SOFTWARE COST ESTIMATION 54
15.2.1 Introduction: 54
15.2.2 Productivity 55
15.2.3 Cost and Pricing 56
15.2.4 Measurement Problems 56
15.2.5 Lines of code (LOC) 56
15.2.6 Object Points 56
15.3 ESTIMATION TECHNIQUES 57
15.3.1 Algorithmic Cost Modelling 57
15.3.2 CoCoMo Model (Constructive Cost Model) 57
15.3.2.1 Early prototyping levels (The application composition) 58
15.3.2.2 Early design levels 58
15.3.3 Expert Judgement 59
15.3.3.1 Advantage 59
15.3.3.2 Disadvantage 59
15.3.4 Estimating by Analoging 59
15.3.4.1 Advantage 59
15.3.4.2 Disadvantage 59

15.3.5 Parkinson's Law 59
15.3.5.1 Advantage 59
15.3.5.2 Disadvantage 59
15.3.6 Pricing to Win 60
15.3.6.1 Advantage 60
15.3.6.2 Disadvantage 60
15.4 INSPECTION 60
15.4.1 Inspection Procedure 60
15.4.2 Inspection Team 60
15.4.3 Inspection Checklist 60
15.4.4 Inspection Rate: 60

16. TESTING PROCESS 62

16.1 COMPONENT TESTING 62
16.2 INTEGRATION TESTING 62
16.3 DEFECT TESTING 62
16.4 TESTING PRIORITIES 62
16.5 TEST DATA AND TEST CASES 62
16.5.1 Test Data 62
16.5.2 Test Cases 62
16.6 EQUIVALENCE PARTITIONING 63
16.7 SEARCH ROUTINE INPUT PATTERN 64
16.8 STRUCTUREAL TESTING/ WHITE BOX 64
16.9 BINARY SEARCH BY EQUIVALENCE 64
16.10 PATH TESTING 65
16.11 PROGRAM FLOW GRAPH 65
16.12 INDEPENDENT PATH 66
16.13 INTEGRATION TESTING 67
16.14 APPROACHES 67
16.15 GUIDELINES TO INTERFACE TESTING 68
16.16 STRESS TESTING 68
16.17 TESTING WORKBENCHES (TEST AUTOMATION) 68
16.18 SECURITY ASSESSMENT 68

17. MANAGING PEOPLE 71

17.1.1 Selecting Staff 71
17.1.2 Motivating People 71
17.1.3 Managing Groups 71
17.1.4 The people's capability maturity model 71
17.1.5 Quality Assurance 71
17.1.6 Quality Control 71
17.1.7 Software measurement and metrics 71

UNIT-1

Introduction to Software Engineering [2hrs]

- Definition of Software (Characteristics and Types)
- Software Engineering
- Comparison between other engineering and software engineering

System Engineering [2hrs]

- Introduction to system
- System properties
- System and their environment
- System modelling (system component)

Software Process [4hrs]

- Introduction
- Software Process Model
- Process Iteration
- Software Specification
- Software Design and implementation
- Software Validation
- Software Evolution

Project Management [3hrs]

- Introduction
- Management Activities
- Project Planning
- Project Scheduling (WBS, inter-task dependency, pert chart CPM).

1.Software Engineering

- Software Engineering is concerned with theories, methods and tools for professional software development
- Software engineering is concerned with cost-effective software development
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times the development costs.

1.1 FAQ's on SE

1.1.1 What is software?

#MISSING GOOD ANSWER

1.1.2 What is software engineering?

Software Engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.

Software engineers should adopt a systematic and organized approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

1.1.3 What is the difference between software engineering and computer science?

Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing useful software.

Computer science theories are currently insufficient to act as a complete underpinning for software engineering.

1.1.4 What is the difference between software engineering and system engineering?

#MISSING GOOD ANSWER

1.1.5 What is software process?

A set of activities shoes goal is the development or evolution of software.

- Generic activities in all software processes are:
  a) **Specification**
     What the system should do and its development constraints

  b) **Development**
     Production of the software system

  c) **Validation**
     Checking the software is what the customer wants

d) **Evolution**
Changing the software in response to changing demands

1.1.6 What is software process model?

- A simplified representation of a software process, presented from a specific perspective.

- Examples of process perspective are:
  a) **Workflow perspective** –sequence of activities
  b) **Dataflow perspective** – information flow
  c) **Role/Action perspective** – who does what

1.1.7 What are generic process models? (WIFE) {#Important}

- Waterfall
- Evolutionary development
- Formal transformation
- Integration from reusable components

1.1.8 What are the costs of software engineering?

a) Roughly about 60% costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
b) Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability.
c) Distribution of costs depends on the development model that is used.

1.1.9 What are software engineering methods?

- Structured approaches to software development which include system models, notations, rules, design advice and process guidance.
  a) **Model descriptions**
     - Descriptions of graphical models which should be produced

  b) **Rules**
     - Constraints applied to system models

  c) **Recommendations**
     - Advice on good design practice

  d) **Process Guidance**
     - What activities to follow

1.1.10 What is CASE? {#Important topic generally asked for 5 marks}

- CASE is an acronym for Computer Aided Software Engineering

- Used to support software process activities such as requirements engineering, design and program development, developing and testing.
- CASE tools therefore include design editors, data and dictionaries, compilers, debuggers, system building tools and so on.

▪ Examples of activities that can be automated using CASE tools:
  1. Graphical System Models
  2. Design using a data dictionary
  3. User interface from a graphical interface
  4. Debugging through the provision of information
  5. Automated translation of programs from an old version of programming language.

▪ CASE Classification:
  Helps us understand the types of CASE tools and their role in supporting software process activity.
  1. A functional perspective where CASE tools are classified according to their specific function.
  2. A process perspective where tools are classified according to the process activities that they support.
  3. An integration: perspective where CASE tools are classified according to hoe they are organized into integral units that provide support for one or more process activities.

{Fig:Tools, workbench and Environments | Figure here}

▪ CASE systems can be classified into 3 categories:
  1. **Tools:** Support individual process tasks.
  2. **Workbench:** Support process phases or activities such as specification, design.
  3. **Environments:** Support all or at least a substantial part of software process

1.1.11 What are the attributes of good software?

Four primary attributes of software which all software product should have to deliver the required functionality and better performance are:

1. **Maintainability:** Whenever a software is built, it should evolve to meet the changing needs of customers. This is the critical attribute because software change is an inevitable consequence of a changing business environment.
2. **Dependability:** All the software must be trustworthy. It also includes reliability, security and safety. Whenever a software is developed, it should not cause physical or economic damage in the event of system failure.
3. **Efficiency:** Software should have a proper processing time, memory utilization. It should be efficient to the user. Also software should not make wasteful use of system resources such as memory and processor cycles.
4. **Usability:** Software must be easy to use to whim it is designed. Also a good software should have an appropriate user interface and adequate documentation.

1.1.12 What are the key challenges facing software engineering?

a)
   (i) Coping with legacy system
   (ii) Coping with increasing diversity
   (iii) Coping with demands for reduced delivery time

b) Legacy System
   Old valuable system must be maintained and obtained.

c) Heterogeneity
   System are distributed and include a mix of hardware and software.

d) Delivery
   There is increasing process for faster delivery of software.

1.1.13 What are professional and ethical responsibility of software engineering?

(i) Confidentiality
   Engineers should normally respect the confidentiality of their employees/clients.

(ii) Competence
   Engineers should not disrespect their level of competence.

(iii) Intellectual property rights
   Engineers should be aware of local laws governing the use of Intellectual property such as copyright.

(iv) Computer misuse
   Software engineers should not use their technical skills to misuse other people's computers.

1.1.14 Software engineering is an engineering discipline that is concerned with all aspects of software production. Why?

#MISSING GOOD ANSWER

1.1.15 Discuss whether professional engineers should be certified in the same way as doctors or lawyers.

#MISSING GOOD ANSWER

1.1.16 What are the differences between generic software product development and custom software development?

#MISSING GOOD ANSWER

1.1.17 What is the difference between software process model and a software?

#MISSING GOOD ANSWER

1.1.18 Suggest two mays in which a software process model might be helpful in identifying possible process improvements.

#MISSING GOOD ANSWER

1.2 Functional and Non-functional requirements:

1.2.1 Functional Requirements:

- They are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations,
- May also explicitly state what the system should not do

1.2.2 Non-functional Requirements:

- They are the constraints in the services or functions offered by the system
- Includes timing constraints, constraints on development process and standards.
- Often apply to the system as a whole
- Usually do not apply to individual system features or services.

1.2.3 Domain Requirements:

- They are requirements that come from the application domain of the system and that reflect characteristics and constraints of that domain.
- May be functional or non-functional requirements.

1.2.4 Functional Requirements

- Describes what the system should do
- Depends on type of software being developed, expected users of the software and general approach taken by the organization when writing requirements.
- Should be both complete and consistent.
- Completeness: All services required by the user must be defined.
- Consistent: That requirements should not have contradictory definitions.

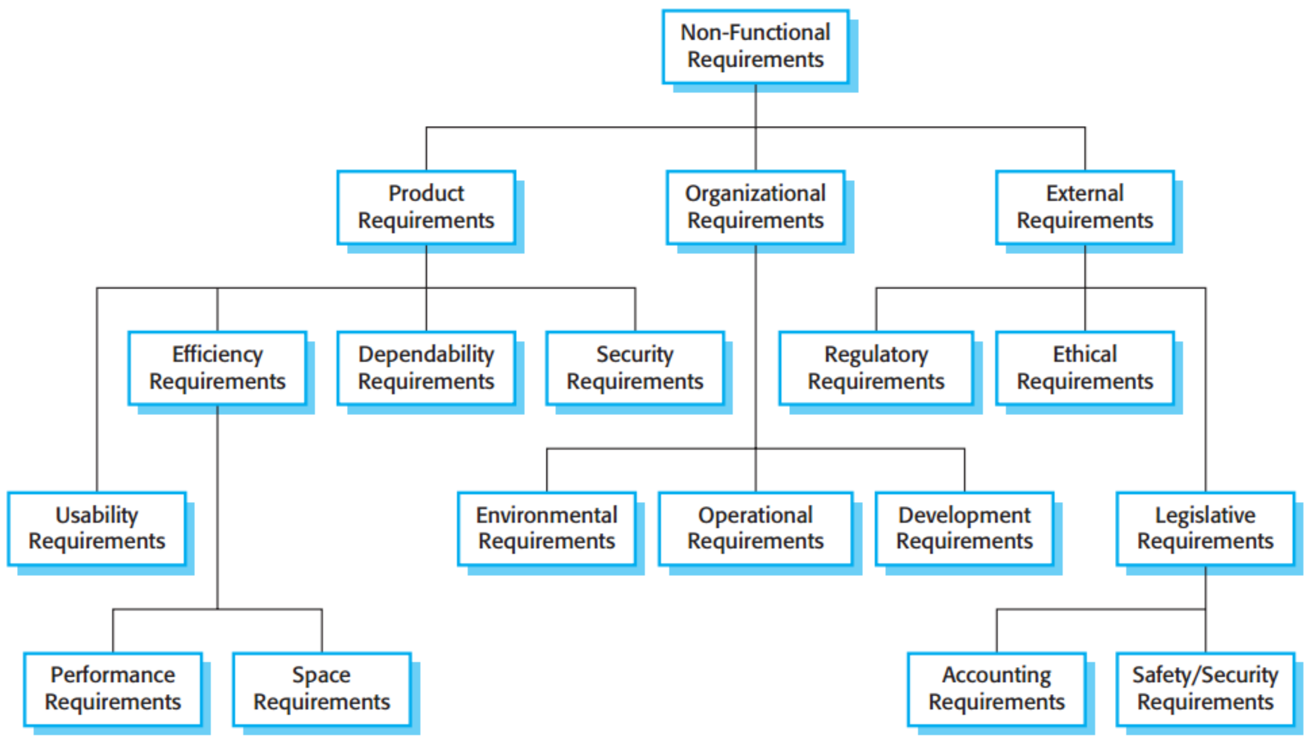


Fig: Types of non-functional requirement

1.3 Product Requirements:

- Specify behavior of product
- Examples include performance requirements on how fast the system must execute and how much memory it requires.

1.4 Organizational Requirements:

- Are derived from policies and procedures in customer's and developer's organization.

1.5 External Requirements:

- Derived from factors external to the system and is development process.
- May include interoperability requirements that define how the system interacts with systems in other organization.
- Legislative requirements that must be followed to ensure that the system operates with law.

1.6 The requirement documents:

- Official statement of what is required by system developers.
- Should include both definition and specification of requirements.
- Should set of WHAT the system should do rather than HOW it should be done.

1.7 Users of Requirement Documents:

1. System Customers
   Specify the requirements and read them to check that they meet their needs. Customers specify changes to requirement.

2. Managers
   Use the requirement document to a bid for the system and to plan the system development process.

3. System Engineers
   Use the requirements to understand what system is to be developed.

4. System Test Engineers
   Use the requirements to develop validation test for the system.

5. System Maintenance Engineers
   Use requirements to help understand the system and the relationship between its parts.

2. IEEE Code of Ethics of principle (Software Engineering code of Ethics for Professional practice)

Software engineer shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following eight principles:

(i) PUBLIC:
Software engineers shall act consistently with the public interest.

(ii) CLIENT & EMPLOYER:
Software engineers shall act in a manner that is in the best interest of their client and employer consistent with the public interest.

(iii) PRODUCT:
Software engineers shall ensure that their products and related modifications meet the highest professional standard possible.

(iv) JUDGEMENT:

Software engineers shall maintain integrity and independence in their professional judgment.

(v) MANAGEMENT:
Software engineering management and leaders shall subscribe to, and promote and ethical approach to the management of software development and maintenance.

(vi) PROFESSION:
Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

(vii) COLLEAGUES:
Software engineers shall be fair and supportive of their colleagues.

(viii) SELF:
Software engineers shall participate in life-long learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

3.System Engineering

3.1 What is a system?

- A purposeful collection of interrelated components working together towards of some common objective.
- A system may include software, electrical, electronic and mechanical hardware and be operated by people.
- System components are dependent on other system components.
- The properties and behaviors of the system components are intermingled.

3.2 Problems of System Engineering:

System engineering requires a great deal of coordination across disciplines

(i) Almost infinite possibilities for design across components
(ii) Mutual distrust and lack of understanding
(iii) System must be designed to last many years in changing environment

3.3 Emergent properties

(i) Properties if the system as a whole rather than properties that can be derived from the properties of components of a system.
(ii) Emergent properties are a consequence of the relationship between system components.

(iii) Therefore, they can only be accessed and measured once the components have been integrated into a system.

3.3.1 Examples of emergent properties:

a) The overall weight of the system
b) The reliability of a system
c) The usability of a system

3.4 Types of emergent properties

3.4.1 Functional Properties

This approach when all the parts of the system work together to achieve some objective.

3.4.2 Non-functional Properties

This is related to the behavior of the system in its operational environment. Example: reliability, performance, safety and security.

3.5 The System Engineering Process

(i) Usually follows a waterfall model because of the need for parallel development of different parts of the system.
(ii) Little scope for iteration between phases, because hardware changes are very expensive. Software may be able to compensate for hardware problems.

4.System Procurement Process

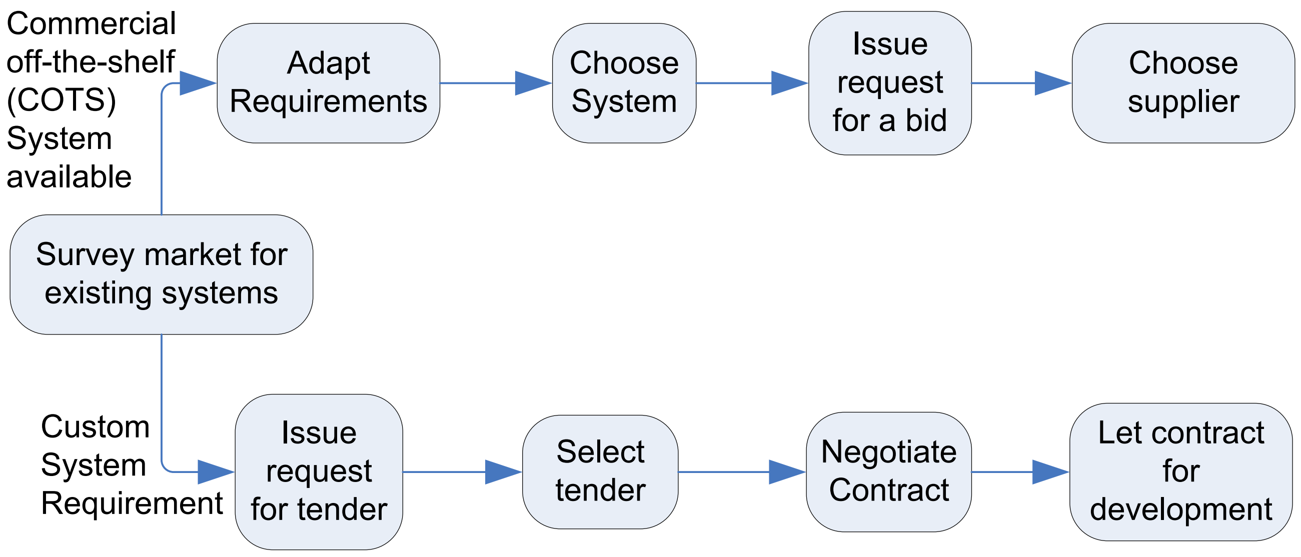

Commercial off-the-shelf (COTS) System available
Adapt Requirements
Choose System
Issue request for a bid
Choose supplier
Survey market for existing systems
Custom System Requirement
Issue request for tender
Select tender
Negotiate Contract
Let contract for development

Fig: The system procurement process

❖ Procurement refers to: Acquiring a system for an organization to meet some need.

4.1 Specifications and architectural design

Some system specification and architectural design are usually necessary before procurement:

1) You need a specification to let a contract for system development
2) The specification may allow you to buy a commercial off-the-shelf (COTS). Almost always cheaper than developing a system from scratch.

4.2 Procurement Issues

1) Requirements may have to be modified to match the compatibility of off-the-shelf components.
2) The requirement specification may be the part of the contract for the development of the system.
3) There is usually a contract negotiation period to agree changes after the contractor to build a system that has been selected.

4.3 Contractor and Subcontractor

1) The procurement of large system is based around some principle contractors.
2) Subcontractors are issued to other suppliers to supply parts of the system.
3) Customer deals with the principle contractor and doesn't deal directly with sub-contractors.

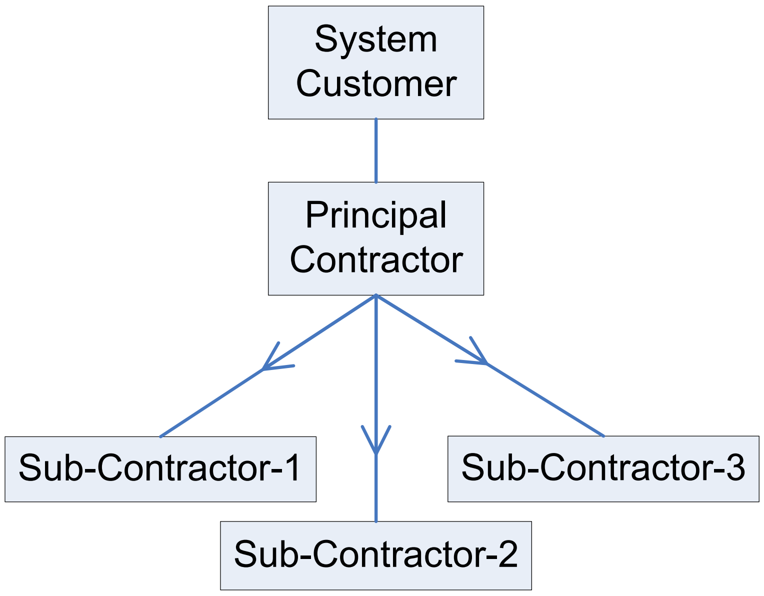


Fig: Contractor/Sub-Contractor Model

5.System Reliability Engineering

1. Because of the component interdependencies, fault can be propagated through the system
2. System failure often occurs because of the unforeseen inter-relationship between the components.
3. Software reliability measures may give a false picture of the system reliability.

5.1 Reliability

5.1.1 Hardware Reliability

What is the probability of a hardware component failing and how long does it take to repair that component.

5.1.2 Software Reliability

How likely it is a system component will produce an incorrect output. Software failure is usually distinct from hardware failure, in that, software does not wear-out.

5.1.3 Operator Reliability

How likely is that the operator of a system will make an error

6.Generic Software Process Models

1. The Waterfall model
2. Evolutionary development
3. Formal system development

4. Re-use based development

6.1 The Waterfall model

This takes the fundamental process activities of specification development, validation and evolution and represents them as separate process phase such as requirements specification, software design, implementation testing and son on.

{The SDLC figure}

The first published model of software engineering was derived from more general system engineering. Because of the cascade from one phase to another, this model is known as the waterfall model or software life cycle.

6.2 Fundamental Development Activities:

The principal stages of the model map onto fundamental development activities:

6.2.1 Requirement analysis and definition

The system's service, constraints and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

6.2.2 System and software design

The system design process partitions the requirements to their hardware or software systems. It established an overall system architecture. Software designing involves identifying and describing the fundamental software system abstraction and their relationships

6.2.3 Implementation and unit testing

During this stage, the software design is released as a set of programs or program units. Unit testing involved verifying that each unit meets its specification

6.2.4 Integration and system testing

The Integrated Program units or programs are integrated and tested as a complete system

6.2.5 Operation and maintenance

- Advantages and Disadvantages
  - **Advantages**
  - This model is only appropriate when the requirements are well understood.

  - **Disadvantages**
  - Difficulty of accommodating change after the process is underway
  - Inflexible partitioning of project into distinct stages
  - Difficult to respond to changing customer requirements

7.Software Specification

- Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.
- This process leads to the production of a requirements document that is the specification for the system.
- End-users and customers need a high-level statement of the requirements; system developers need a more detailed system specification.

8.Requirements Engineering Process

- Feasibility Study
- Requirement Elicitation and analysis
- Requirement Specification
- Requirement Validation

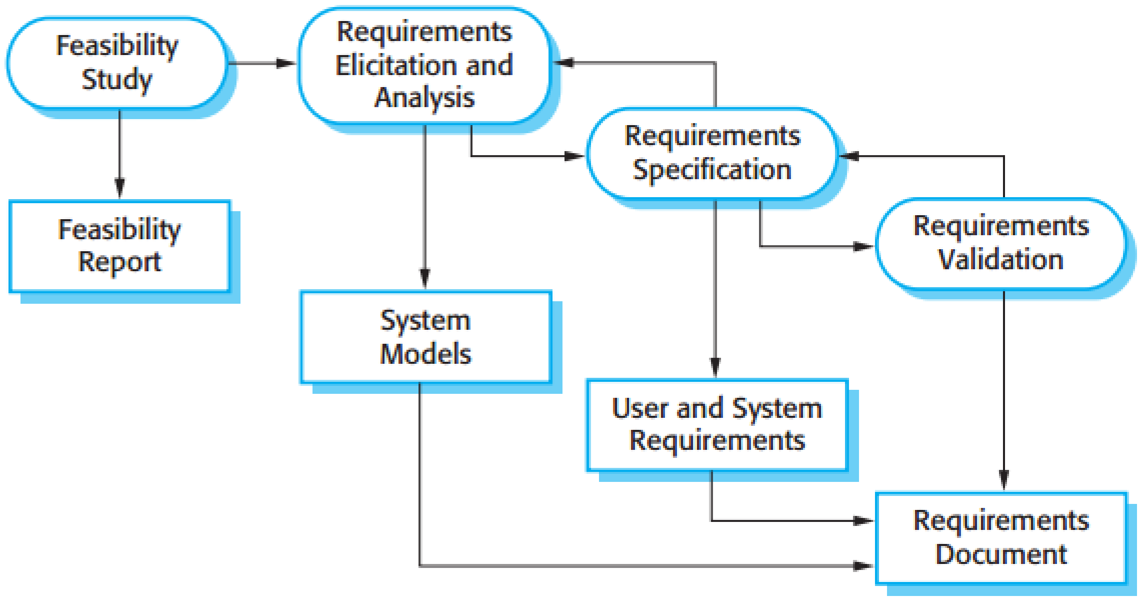


Fig: The requirements engineering process

8.1 Feasibility Study

The input to the feasibility study is a set of preliminary business, requirements, an outline description of system and how the system is intended to support business process.

The results of the feasibility study should be a report that recommends whether or not it is worth carrying on with the requirements engineering and system development process.

It is a short, focused study that aims to answer a number of questions.

1. Does the system contribute to the overall objective of the organization?
2. Can the system be implemented using current technologies and within given cost & schedule constraints?
3. Can the system be integrated with other systems which are already in place?
    a. Carrying out a feasibility, study involves information assessment, information collection and report writing.
    b. You may consult information sources such as the managers of departments where the system where the system will be used, software engineers, technology experts & end-users.
    c. Normally, it should complete in 2/3 weeks.
    d. You should recommend about whether or not the system development should continue.

8.2 Requirements elicitation and analysis

- Software engineers work with customers and system end-users to find out application domain, what services the system should provide, the required performance of system, hardware constraints and so on.
- May involve a variety of people in an organization.
- Stakeholder is referred to person who will be affected by the system directly or indirectly.
- Stakeholders include end-users who interact with the system and everyone else in an organization that may be affected by the installation.
- Other stakeholders may be engineers, business manager, domain experts, and trade union representatives.

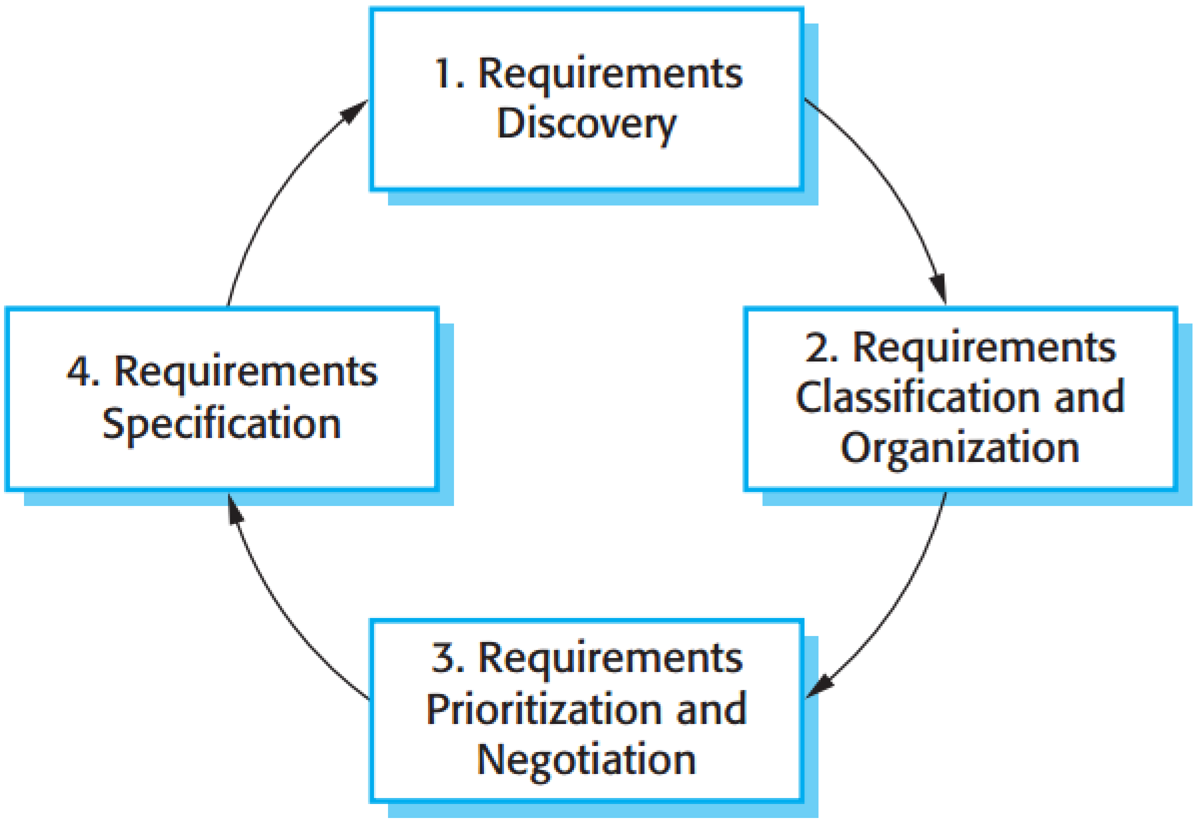

1. Requirements Discovery
2. Requirements Classification and Organization
3. Requirements Prioritization and Negotiation
4. Requirements Specification

Fig: The requirements elicitation and analysis process

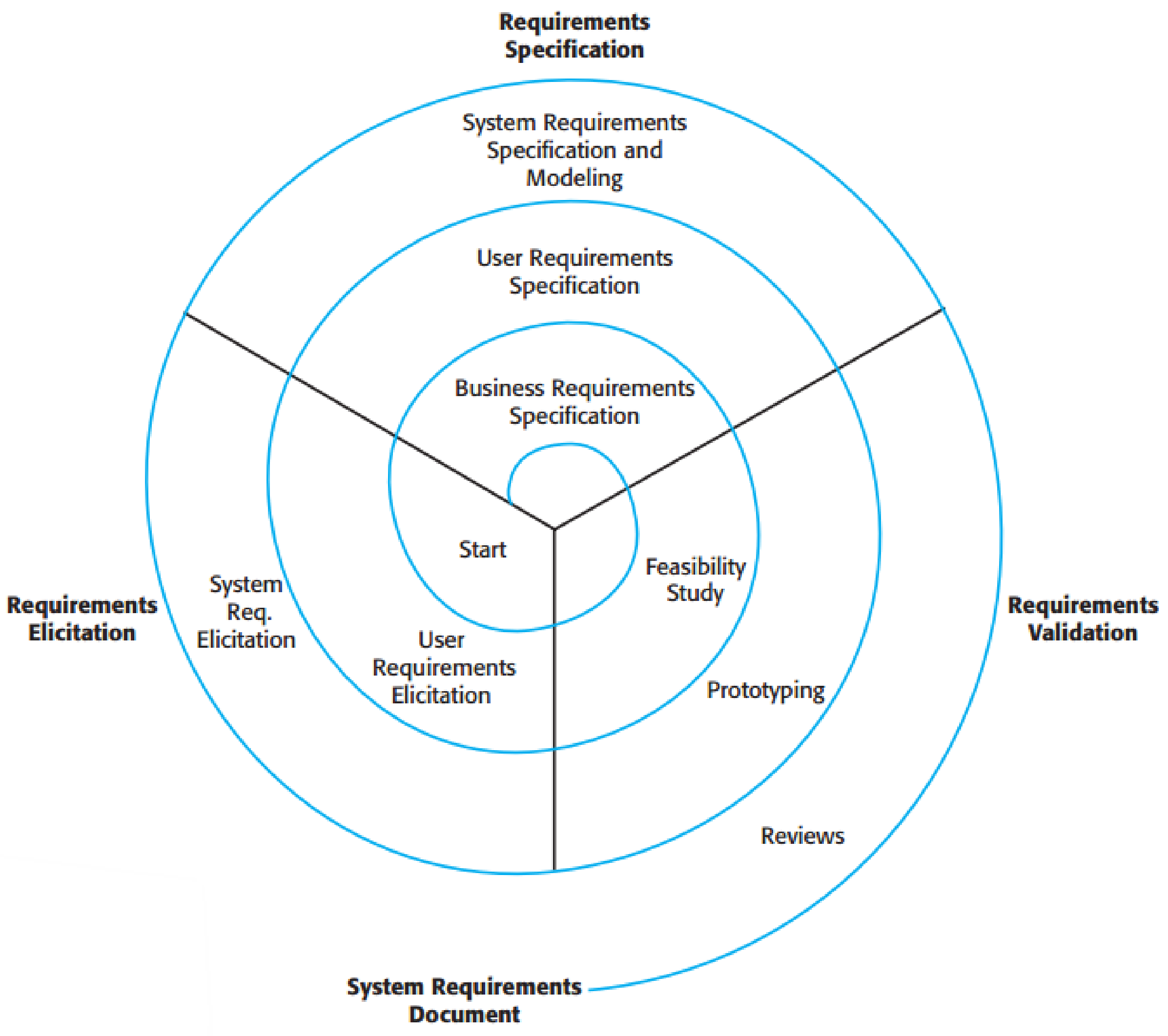

Requirements Specification
System Requirements Specification and Modeling
User Requirements Specification
Business Requirements Specification
Start
Feasibility Study
Requirements Elicitation
System Req. Elicitation
User Requirements Elicitation
Prototyping
Requirements Validation
Reviews
System Requirements Document

Fig: A spiral view of the requirements engineering process

8.2.1 Process of Requirements Elicitation and Analysis

1. **Requirements discovery**
   Process of interacting with stakeholders in the system to collect their requirements.

2. **Requirements classification and organization**
   Takes the unstructured collection of requirements groups related requirements and organizes them into coherent clusters

3. **Requirements prioritization and negotiation**
   Concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation

4. **Requirements documentation**
   The requirements are documented and input into next round of spiral.

8.2.2 Viewpoints

Stakeholders represents different ways of looking at a problem or problem viewpoints.

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

❖ Types:
  1. Interactor
  2. Indirect
  3. Domain

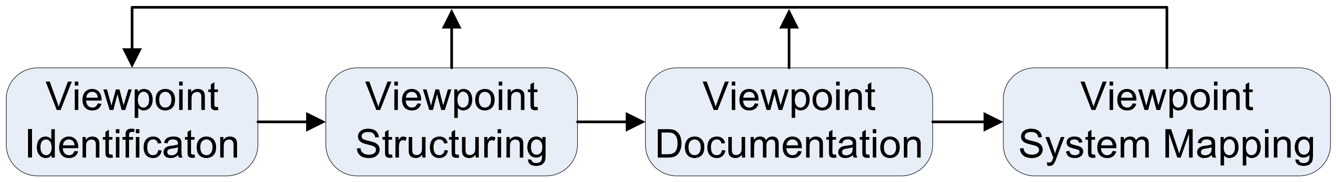


Fig: The VORD Method

8.2.3 Interviewing

The requirements engineering team puts questions to stockholders about the system that they use and system to be development.

1. **Closed interviews:** where the stakeholders answers a predefined sets of questions.
2. **Open interviews:** where there is no predefined agenda.

8.2.4 Scenarios

1. Scenarios are descriptions of how 'a system' is used in practice.
2. They are helper requirements elicitation as people can relate to these more rapidly than abstract statement of what that require from a system.
3. Are useful for adding detail to an outline requirement description.

8.2.4.1 Use-Case and Sequence Diagram of Library

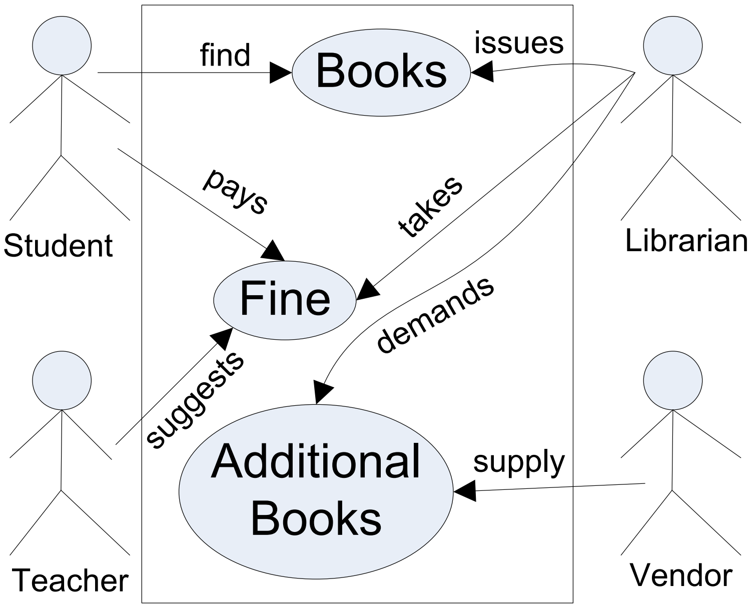


Fig: Use-Case Diagram of Library System

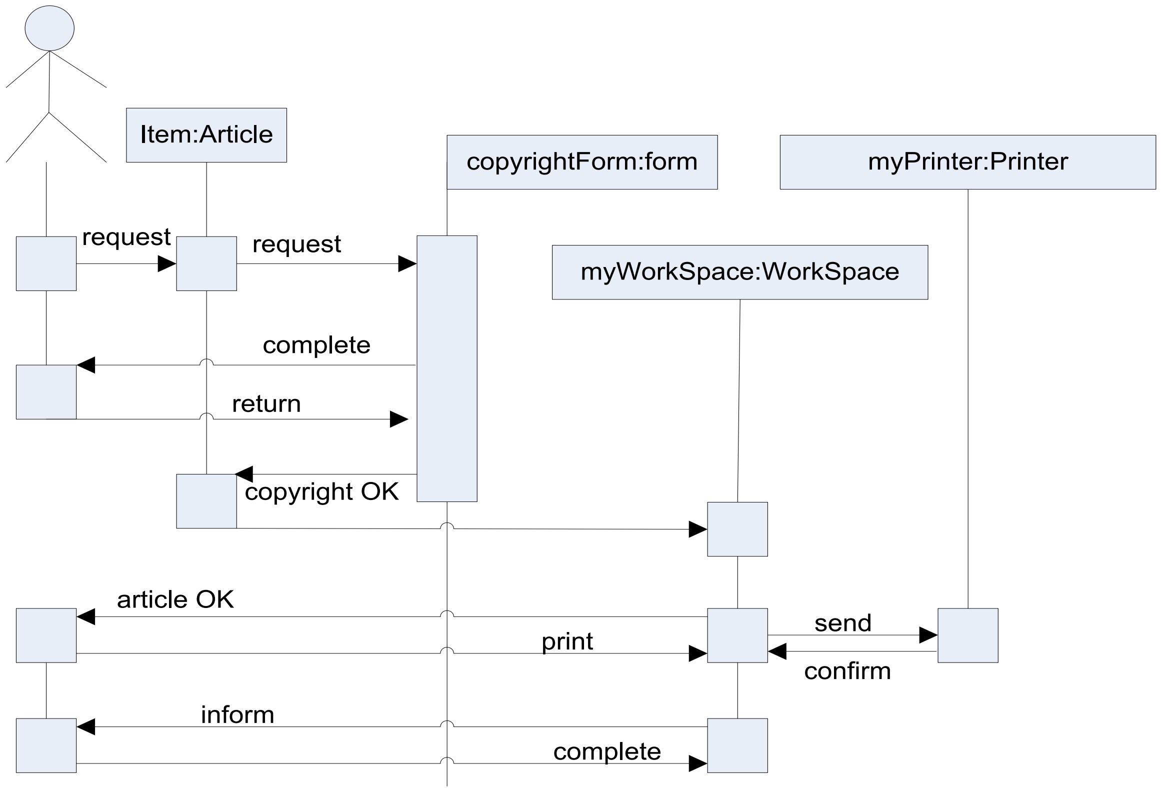


Fig: Sequence Diagram of Library

8.2.4.2 Use-Case and Sequence Diagram of Library (Alternate)

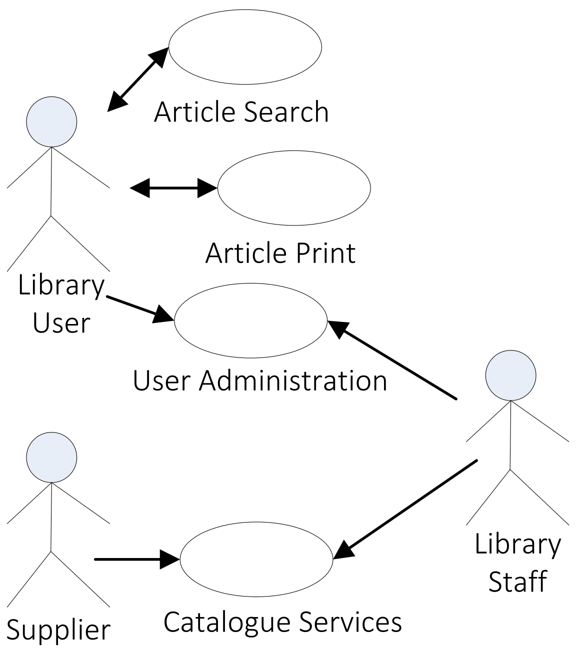


Fig: Use-Case Diagram of Library System

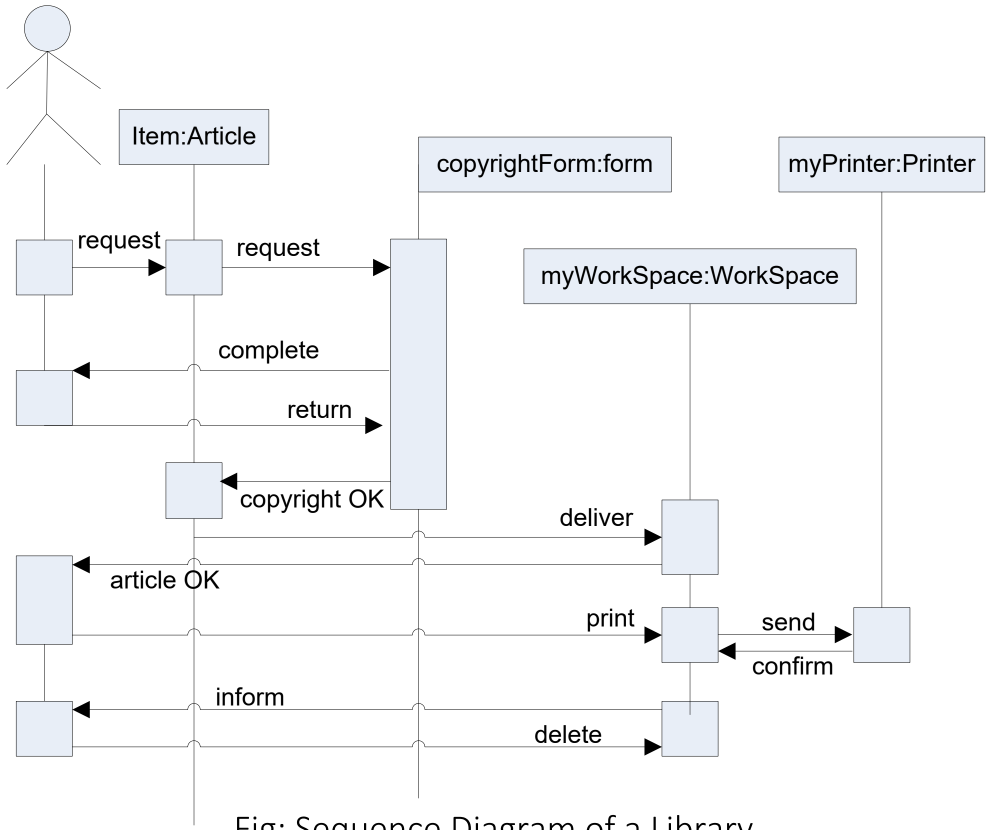


Fig: Sequence Diagram of a Library

8.2.4.3 Use-Case and Sequence Diagram of Bank

{Draw Use-Case diagram yourself}

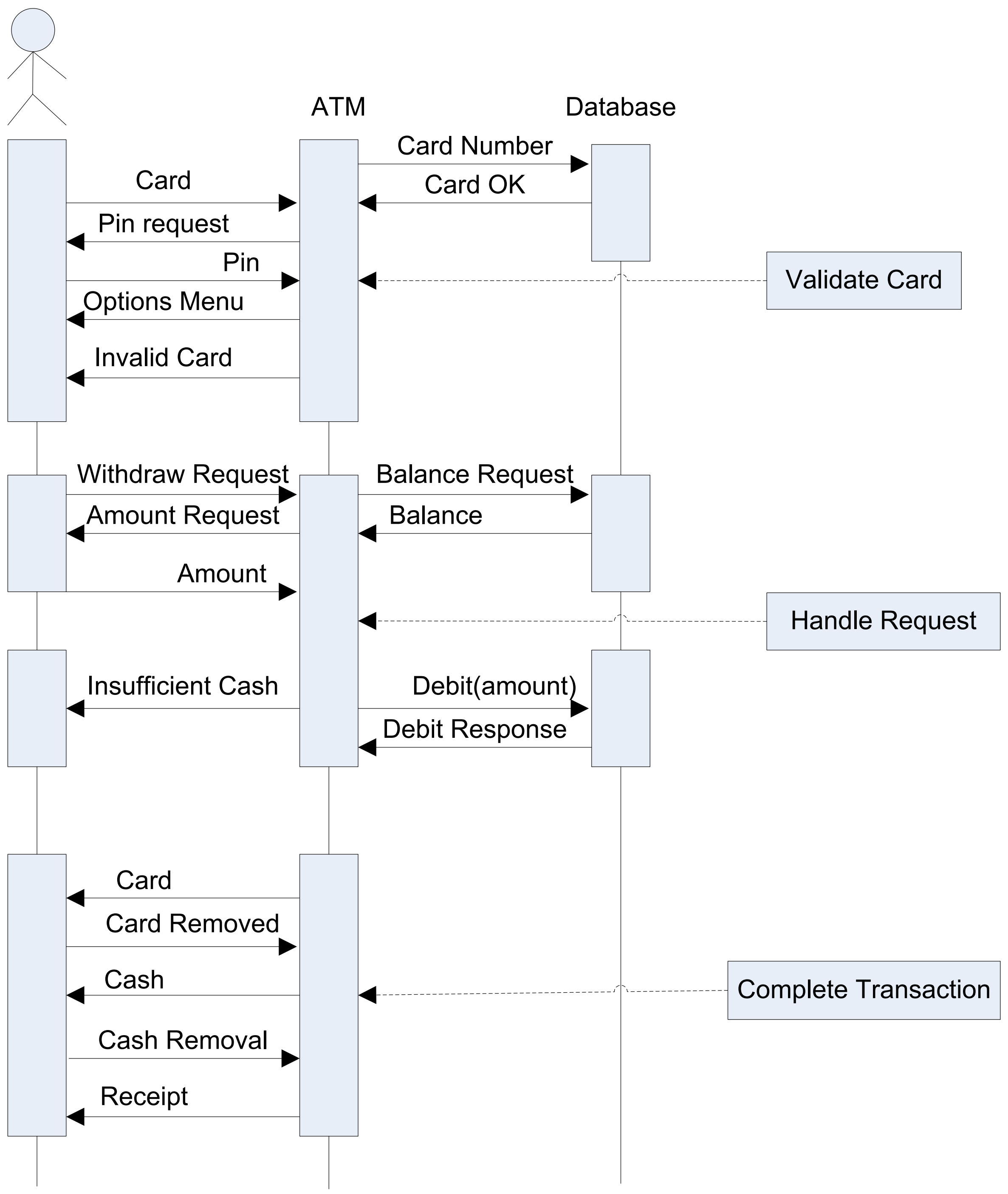


Fig: Sequence Diagram of Bank

8.2.6 Ethnography

- A social scientist spends a considerable time observing and analyzing how people actually work.
- People do not have to explain or articulate their work.
- Social and organizational factors of importance may be observed
- Ethnographic studies haves shown that work is usually richer and more complex than suggested by simple system models.

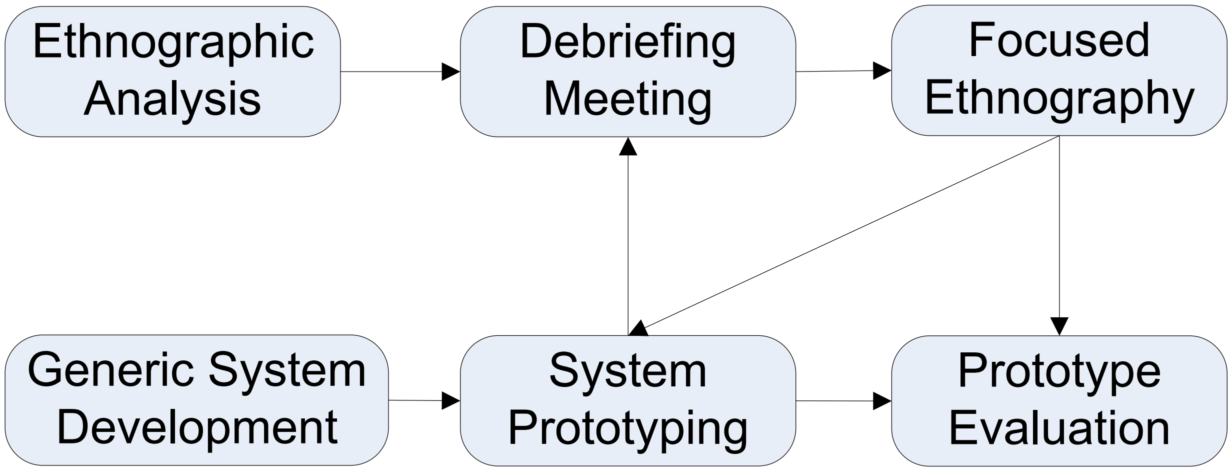


Fig:Ethnography

8.3 Requirement Specification

The activity of translating the information gathered during the analysis activity into a document that defines a set of requirements.

8.4 Requirement Validation

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

8.4.1 Requirement Reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor should be involved in reviews.
- Reviews may be formal or informal. Good communication between developers, customers and users can resolve problems at an early stage.

8.4.2 Requirements Checking

1. **Validation**
   Does the system provides the functions which best support the customer's needs?

2. **Consistency**
   Are there any requirements conflicts?

3. **Completeness**
   Are all functions required by the customer included?

4. **Realism**
   Can the requirements be implemented given available budget and technology.

5. **Verifiability**
   Can the requirements be checked?

8.4.3 Requirements Validation Checking

1. Requirement Review
2. Prototyping(executable code)
3. Test-case generation
4. Automated Consistency Analysis

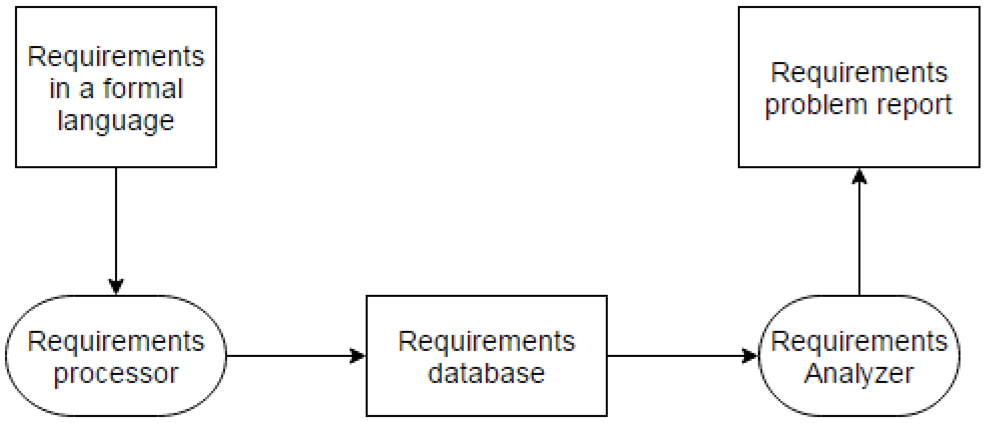


Fig: Automated Consistency Analysis

8.4.4 Requirements Management

- Requirements for large software are changing.

1. Large systems usually have a diverse user community where users have different requirements and priorities.
2. People who pay for a system and the users of system are rarely the same.
3. Business and technical environment of system changes after installation.

- Requirement management is the process of understanding and controlling changes to system requirements.

- It should start soon as a draft version of requirements document is available.

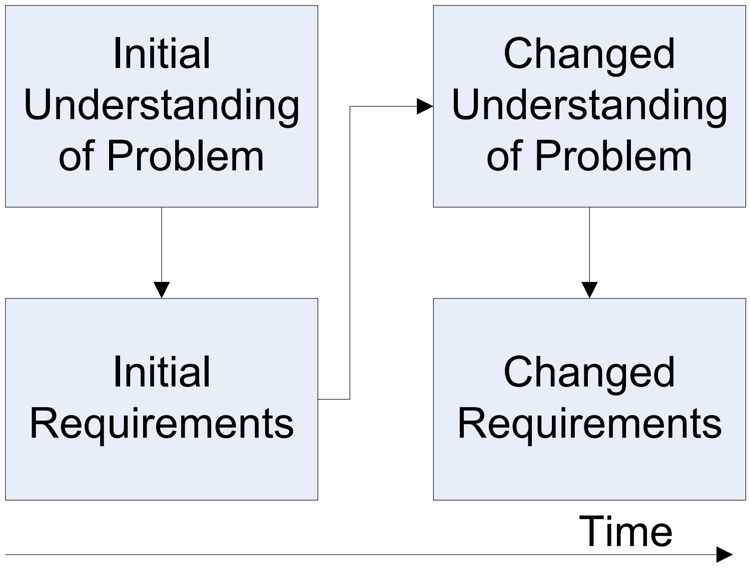

Initial Understanding of Problem
Changed Understanding of Problem
Initial Requirements
Changed Requirements
Time

Fig: Requirements Evolution

8.5 Enduring Requirements

Stable requirements derived from the core activity of the customer organization. May be derived from domain models.

8.6 Volatile Requirements

Requirements which changes during development or when the system is in use.

8.7 Classification of Requirements

1. Mutable Requirements
2. Emergent Requirements
3. Consequential Requirements
4. Compatibility Requirements

8.8 Requirements management Planning

- Requirements Identification
- A change management process
- Traceability policies
- CASE Tool Support

8.9 Traceability

- Concerned with the relationships between requirements, their sources and the system design.

8.10 Source Traceability

- Links from requirements to stakeholders who proposed these requirements.

8.11 Requirements Traceability

- Links between dependent requirements.

8.12 Design Traceability

- Links from requirements to design.

Traceability Matrix

Req^{d} ID	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D			R	R	D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

- Blank → no relation
- D→1.1 is dependent on 1.2
- R→weaker relationship

8.13 Requirements Identification

- Each requirement must be uniquely identified.
- It can be cross-referenced by other requirements so that it can be used in traceability.

8.14 A change management process

- Is set of activities that assess the impact and cost of changes.
- Requirements management involves the processing of large amounts of information about requirements.

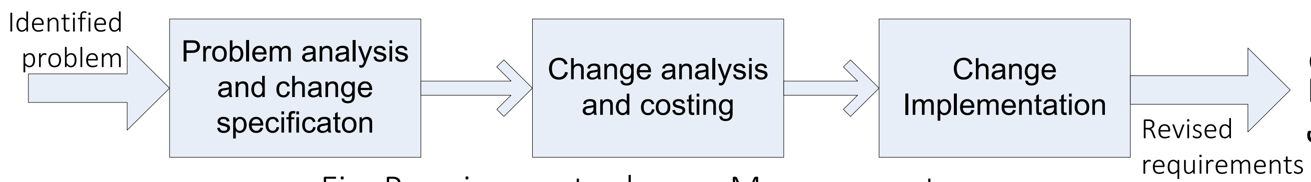

Identified problem
Problem analysis and change specificaton
Change analysis and costing
Change Implementation
Revised requirements

Fig: Requirements change Management

8.15 Requirements change Management

- Should apply to all proposed changes to the requirements
- Principal Stages

8.15.1 Problem Analysis and Change Specification

- Process starts with an identified requirements problem with a specific change proposal.

8.16 Change Analysis and Costing

- Effect of proposed change is assessed using traceability information and general knowledge of system requirement.
- Costs of making change is estimated in terms of modifications to requirements documents.

8.17 Change Implementation

- Requirements document, system design and implementation are modified
- Should organize requirements documents so that changes can be made without extensive rewriting or reorganization.

9.Software design and Implementation:

- The process of converting the system specification into an executable system.
- Software design means design a software structure that realizes the specification.
- Implement means translate this structure to an executable program.
- The activities of design and implementation are closely related and may be intentional.

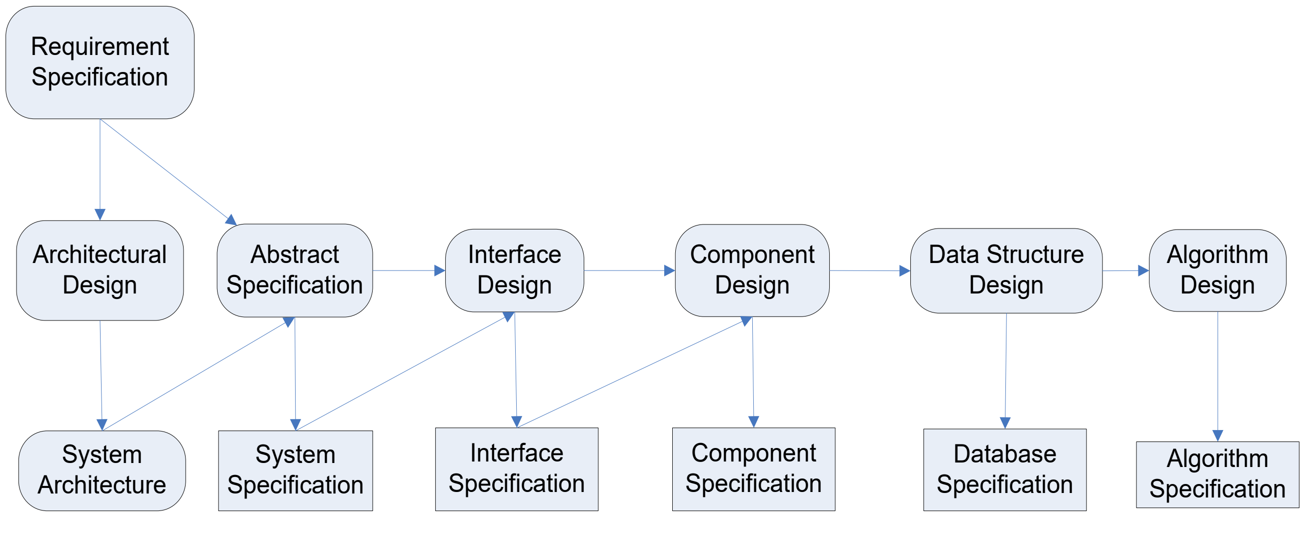


Fig: The software design process

9.1 Architectural design

A subsystem making up the system and their relationships are identified and documented.

9.2 Abstract Specification

For each sub-system, an abstract specification of its services and the constraints under which it must operate is produced.

9.3 Interface Design

For each sub-systems, its interface with other sub-system, are designed and documented.

9.4 Component Design

Services are allocated to components and the interface of these component are designed.

9.5 Data structure and Design

The data structure used in system implementation are design in detail and specified.

9.6 Algorithm Design

The algorithm used to provide services are designed in detail and specified.

10. Design Model (Structured Method)

- Systematic approaches to developing a software design.
- The design is usually documented as a set of graphical models.

10.1 Possible models

(i) Data Flow Model:
Where the system is modelled using the data transformations that take place as it processed.

(ii) Sequence Model:
That shows how object in the system interact when system is executing

(iii) Structural Model:
Where the system components and their aggregations are documented.

(iv) Object Model:
Shows the object classes used in the system and their dependencies.

(v) **State Transition Model:**
Show system states and the triggers from transitions from one state to another

10.2 Programming & Debugging

1. Translating a design into a program and removing errors from that program.
2. Programming is a personal activity - there is no generic programming process.
3. Programmers carry out some program testing to discover faults in the program and remove these faults in debugging process.

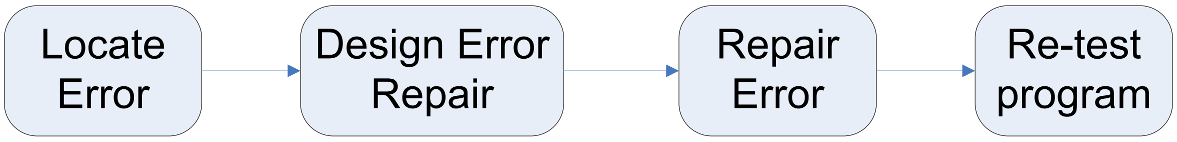


Fig: The debugging process

10.3 Software Validation (V & V Model)

- Verification and validation is intended to show that a system confirms to its specifications and meets the requirements of the system customer.
- Involves checking & review processes and system testing
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- A system confirms to its specifications and that the system meets the expectations of the customer buying the system.
- It involves checking process, such as inspections and reviews at each stage of the software process.

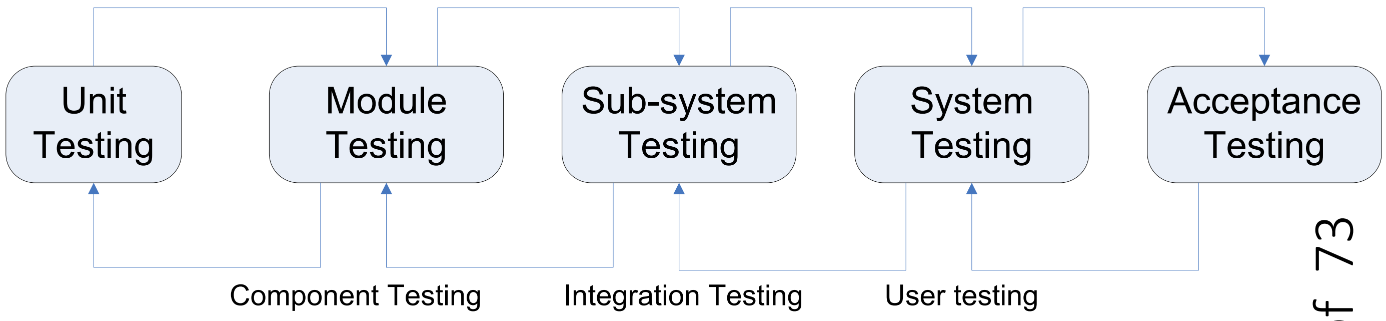


Fig: The testing process

10.4 The Testing process

10.4.1 Component (Unit) testing:

- Individual components are tested to ensure that they operate correctly
- Each component is tested independently without other system components.
- Components may be simple entities such as functions or object classes or may be coherent grouping of these entities.

10.4.2 System testing:

- The components are integrated to make up a system.
- This process is concerned with finding error that results from unanticipated interactions between components and component interface.
- Also concerned with validation that the system meets its functional and non-functional requirements and testing the emergent system properties.

10.4.3 Acceptance testing:

- Is the final stage in the testing process before the system is accepted for operational use.
- The system is tested with data supplied by the system customer
- it may reveal errors and omissions in the system requirements definition, because the real data exercise the system in different ways from the test data

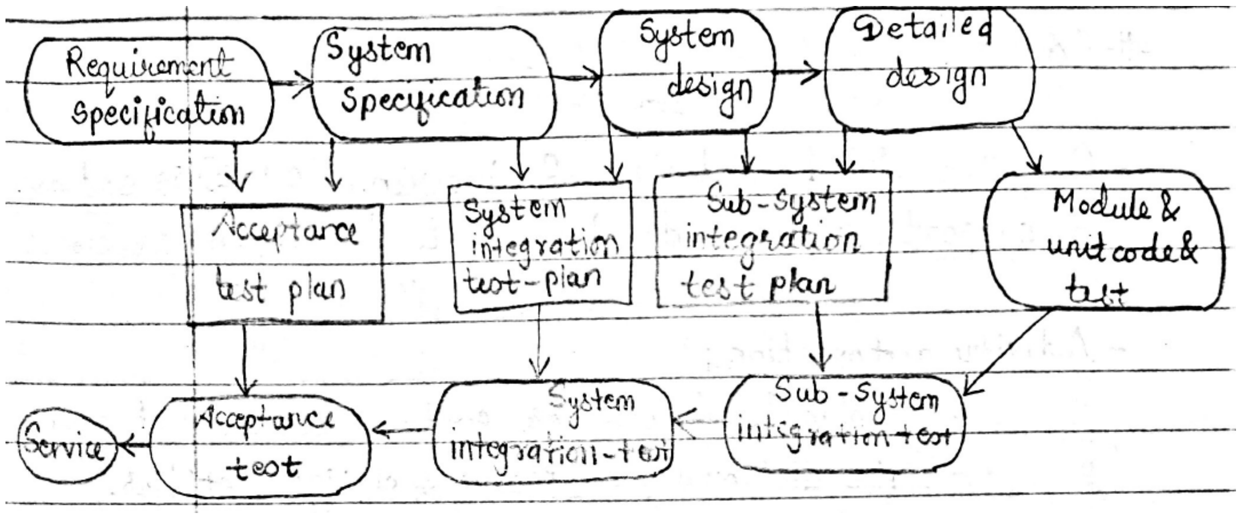

Requirement Specification
System Specification
System design
Detailed design
Acceptance test plan
System integration test-plan
Sub-system integration test plan
Module & unit code & test
Service
Acceptance test
System integration-test
Sub-system integration test

Fig: Testing phases in the software process

10.5 Software Evaluation

- Software is inherently flexible and can change
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.

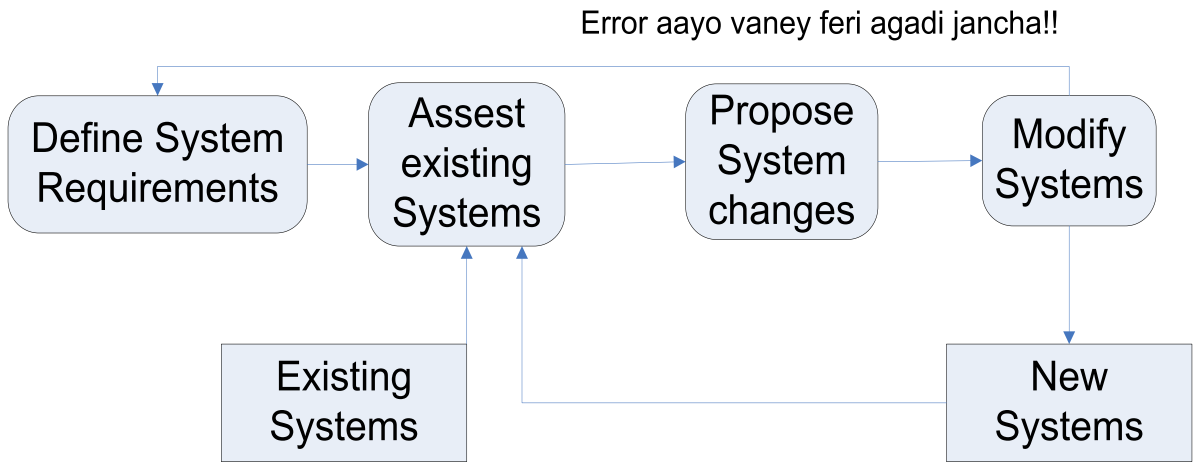


Fig: System Evolution

11. Project Management (Unit 1.4)

11.1 Software Project Management

- Concerned with activities involved in ensuring that software is delivered in time and on schedule in accordance with the requirements of the organizations developing and processing the software.
- Project management is needed because software development is always subject to budget and schedule constraints that are set by the organization developing the software.

11.2 Software management distinction

1) The project is intangible
2) The product is uniquely flexible.
3) Software engineering is not required as an engineering discipline same as mechanical, electrical, engineering etc.
4) The software development process is not standardized.
5) Many software projects are "one-off" projects.

11.3 Management Activities

1) Proposal Writing
2) Project Planning and Scheduling
3) Project Casting
4) Project monitoring and reviews.
5) Personal selection and evaluation.
6) Report writing and presentation.

11.4 Project Costing

1) Project monitoring and reviews
2) Personal selection and evaluation
3) Report writing and presentation.

11.5 Project Staffing

1) Project budget may not allow for the use of highly paid staff.
2) Staff with appropriate experience may not be available.
3) An organization may wish to develop employee skills on a software projects.
4) Managers have to work within these constraints specially when there is an international shortage of skilled IT staff.

11.6 Project Planning

1) Probably the most time-consuming project management activity
2) Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
3) Various types of plan may be developed to support the main software project plan that is concerned with schedule and budget.

11.7 Types of Plan

(i) Quality plan
Describe the quality procedures and standards that will be used in a project.

(ii) Validation plan
Describes the approach, resources and schedule for system validation.

(iii) Configuration Management plan
Describes the configuration management procedures and structures to be used.

(iv) Maintenance plan (time + cost)
Predicts the maintenance requirements of system, maintenance costs and effort required.

(v) Staff development plan
Describes how the skills and experience of project team members will be developed.

11.8 Project Plan Structure:

Sets out the resources available to the project, the work breakdown and a schedule for carrying out work.

(i) Introduction:
Briefly desires the objectives of project and sets out the constraints (e.g.: budget, time, etc.) that affect the project management.

(ii) Project Organization:
Describes the way in which the development team is organized, the people involved, and their roles in team.

(iii) Risk Analysis:
Describes possible risks, the likelihood of these risks arising and risk reduction strategies that are proposed.

(iv) Hardware and software resource requirements
Specifies the hardware and the supporting software required to carry out development.

(v) Work breakdown
Sets out the breakdown of project into activities and identifies the milestones and deliverables associated with each activity.

(vi) Project Schedule
Shows the dependencies between activities, the estimated time required to reach each milestone and allocation of the people to activities.

(vii) Monitoring and reporting mechanisms
Defines the management reports that should be produced, and when project monitoring mechanisms are to be used.

11.9 Project Scheduling

1) Split projects into task and estimate time and resources required to complete each task.
2) Organize tasks concurrently to make optimal use of workforce
3) Minimalize task dependencies to avoid delays caused by one task waiting for another to complete.
4) Dependent on project manager's intuition and experience.

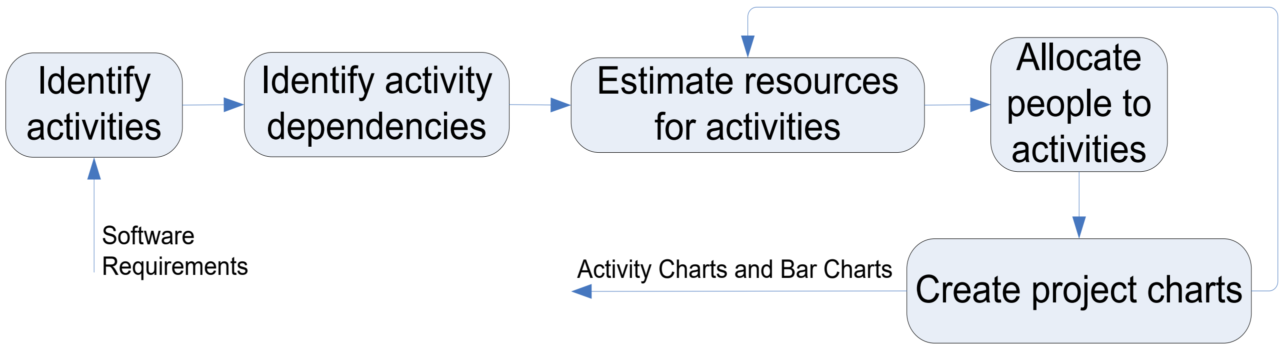

Identify activities
Identify activity dependencies
Estimate resources for activities
Allocate people to activities
Create project charts
Software Requirements
Activity Charts and Bar Charts

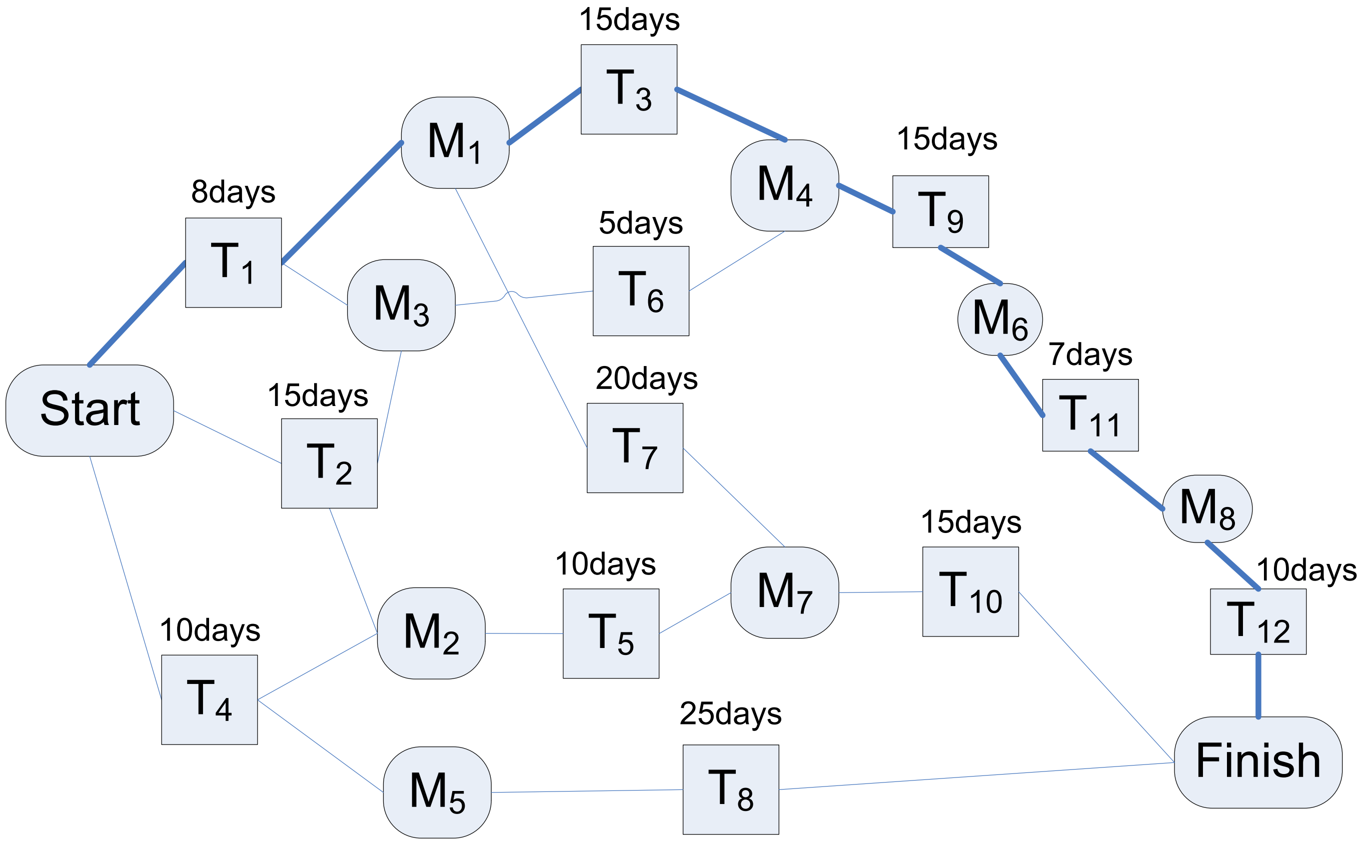
Fig: The Project Scheduling Process

11.9.1 Bar Charts and Activity Networks:

Bar Charts and activity charts are graphical notations that are used to illustrate the project schedule. Bar charts shows who is responsible for each activity and when the activity is scheduled to begin and end. Activity Network shows the dependency between the different activities, making up a project. Bar chart and activity charts can be generated automatically from a database of project information using a project management tools.

Task	Duration (Days)	Dependencies
T_1	8	
T_2	15	
T_3	15	$T_1(M_1)$
T_4	10	
T_5	10	$T_2,T_4(M_2)$
T_6	5	T_1, $T_2(M_3)$
T_7	20	$T_1(M_1)$
T_8	25	$T_4(M_5)$
T_9	15	$T_3,T_6(M_4)$
T_{10}	15	$T_5,T_7(M_7)$
T_{11}	7	$T_9(M_6)$
T_{12}	10	$T_{11}(M_8)$

11.9.2 Activity Networks



The minimum time required to finish the process can be estimated by considering the longest path in the activity graph (critical path).

Task	Duration (Days)	Dependencies
T_1	10	
T_2	15	$T_1(M_1)$
T_3	10	$T_1,T_2(M_2)$
T_4	20	
T_5	10	
T_6	15	T_3, T_4 (M_3)
T_7	20	T_3 (M_4)
T_8	35	T_7 (M_5)
T_9	15	T_6 (M_6)1
T_{10}	5	T_5, T_9 (M_7)
T_{11}	10	T_9 (M_8)
T_{12}	20	T_{10} (M_9)
T_{13}	35	T_3, T_4 (M_{10})
T_{14}	10	T_8,T_9 (M_{11})
T_{15}	20	T_{12}, T_{14} (M_{12})
T_{16}	10	T_{15} (M_{13})

11.10 Risk Management Process

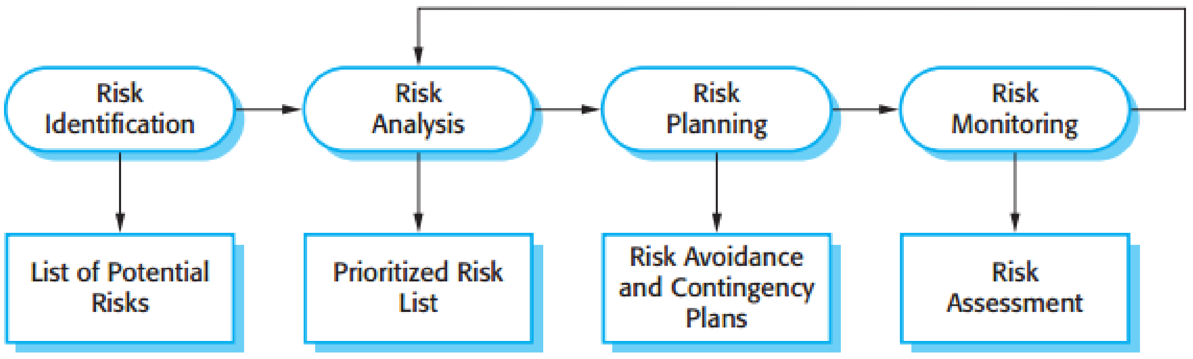


Fig: Risk Management

Risk Management is increasingly seen as one of the main jobs of project managers. It involves anticipating risk that might affect the project schedule or the quality of software being developed and taking action to avoid these risks. The result of the risk analysis should be documented in the project plan along with an analysis of the consequences of the risk occurring. Effective risk management makes it easier to cope with problems and to ensure that these do not lead to unacceptable budget or schedule slippage.

1. **Risk Identification:** Risk identification possible project, product and business risk are identified.
2. **Risk Analysis:** The likelihood and consequences of these risks are assessed.
3. **Risk Planning:** Plans to address the risk either by avoiding it or minimizing its effects on the project are drawn up.
4. **Risk Monitoring:** The risk is constantly assessed and plans for risk mitigation are revised as more information about the risk becomes available.

There are therefore, three related category of risk.

1. **Project Risk:** Project risk are risks that affect the project schedule or resources. An example might be the loss of an experienced designer.
2. **Product Risk:** They are risks that affect the quality or performance of the software being developed. An example might be the failure of a purchased component to perform as expected.
3. **Business Risk:** They are risks that affect the organization developing or purchasing the software. For example: A competitor introducing a new product is a business risk.

11.10.1 Risk Identification: [Text-Book 9th Edition Pg-598 for reference]

1. Technology risk
2. People risk
3. Organizational risk
4. Requirement risk
5. Estimation risk
6. Tools risk

11.10.2 Risk Analysis

During the risk analysis process, you have to consider each identified risk and make a judgment about the probability and the scenarios of it. There is no easy way to do this. You must rely on your own judgment and experience, which is why experienced project managers are generally the best people to help with risk management.

1. The probability of the risk might be assessed as very low (<10%), low (10% to 25%), moderate (25% to 50%), high (50% to 75%), or very high (>75%).
2. The effects of the risk might be accessed as catastrophic, serious, tolerable, or insignificant.

UNIT-2

Software Requirements [6hrs]

- Introduction
- Types of requirements(functional and non-functional)
- Requirements engineering process (Feasibility study, requirements elicitation and analysis, requirement validation, requirement management)

Software Prototyping [3hrs]

- Introduction
- Prototyping in the software process
- Rapid prototyping techniques
- User interface prototyping

Formal Specification [3hrs]

- Introduction
- Formal specification in software process
- Interface specification
- Behavioral specification

12.Software Requirement

- The process of establishing the services that the customer requires form a system and the constraints under which it operates and is developed.
- The requirement themselves are the descriptions of the system services and constraints that are generated during requirements engineering process.

12.1 What is a requirement?

1) It may range from a high level abstract statement of service of a system constraint to a detailed mathematical functional specification.
2) This is unavoidable as a requirement may serve a dual function
    - Maybe the basis for a bid for a contract. Therefore, must be open to interpretation.
3) Maybe the basis for a contract – itself. Therefore, must be defined in detail.

12.2 Types of requirements

(i) **User requirement**

Statements in natural language plus diagram of the services the system provides and its operational constraints written for customers.

(ii) **System requirements**

A structured document setting out detailed description of the system services written as a contract between client and contractor.

(iii) **Software specification**

A detailed software description which can serve as a basis for a design or implementation, written for developers.

#MISSING MISSSING 1 Day's Note. Around Social Camp. Sick

13.Models{Sth similar heading}

13.1 Repository Model

- Sub-systems making up the system must exchange information so that they can work together effectively
- 2 fundamental ways in which this can be done:
    (i) All shared data is held in a central database.
    (ii) Each subsystems maintains its own database

- A system model based on a shared database is also called a repository model.
- The majority systems that use a large amount of date are organized around a shared database/repository.
- Suited to applications where data is generated by one sub-system and used by another.
- Example: CAD Systems
    (i) CASE toolsets
    (ii) Management Information Systems
    (iii) Command and Control Systems

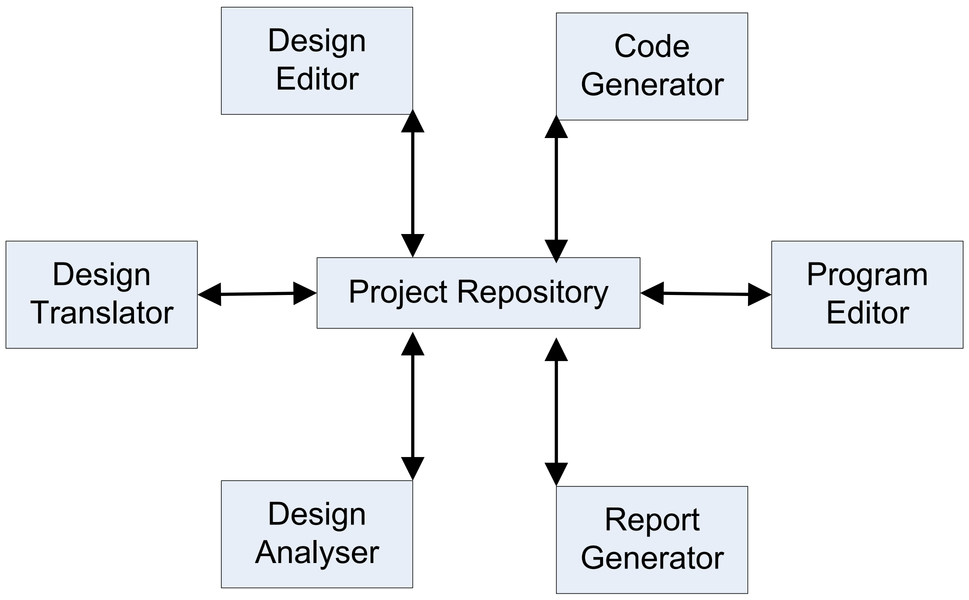


Fig: Architercture of an integrated CASE toolset

- Advantages and Disadvantages
    (i) Efficient way to share large amount of data
    (ii) Compromise between specific needs of each tool which adversely effects the performance
    (iii) May be difficult or impossible to integrate new sub-systems if their data models; do not fit the agreed schema
    (iv) Evolution may be difficult as a large volume of information is generated according to an agreed data model
    (v) Activities such as backup, security, access control and recovery from error are centralized
    (vi) Different sub-systems may have different requirements for security, recovery and backup policies
    (vii) Model of sharing is visible through the repository schema. Easy to integrate new tools given that they are compatible with agreed data model.
    (viii) May be difficult to distribute the repository over a number of machines.

13.2 Client-Server Architecture

- Distributed system model which shows how date and processing is distributed across a range of components.
- Set of stand-alone servers which provide specific activities such as printing, data management etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

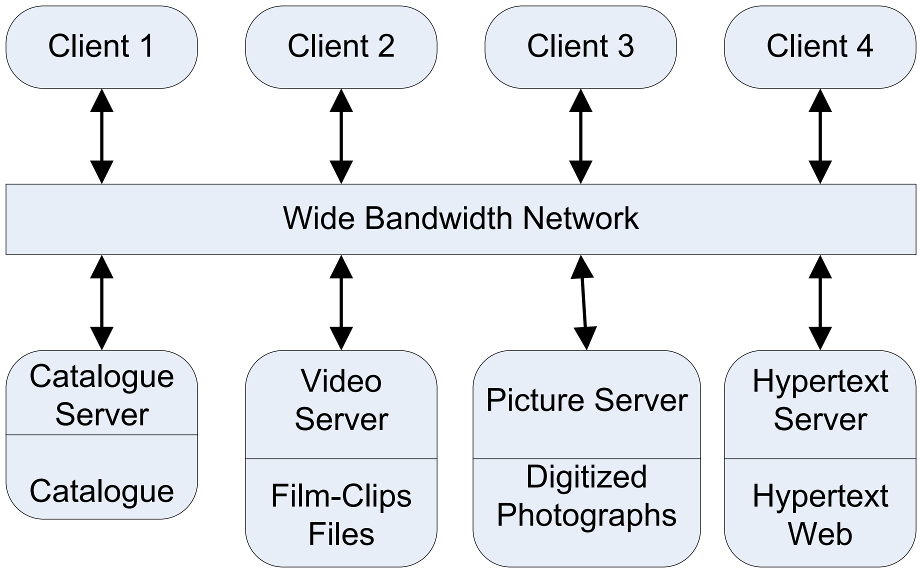


Fig: Film & Picture Library

- Advantages
  - (i) Distribution of data is straight-forward
  - (ii) Makes effective use of networked system
  - (iii) May require cheaper hardware
  - (iv) Easy to add new servers or upgrade existing servers
- Disadvantages
  - (i) No shared data model so sub-system may b\use different data organization , data interchange may be efficient
  - (ii) Redundant management in each server
  - (iii) No central register of names and services – it may be hard to find out what servers and services are available.

13.3 Abstract Machine Model

- Organizes the system into a set of layers each of which provide a set of services
- Supports the incremental development of sub-systems in different layers. When a layer interface changes only the adjacent layer is affected.

- However, often difficult to structure systems in this way.

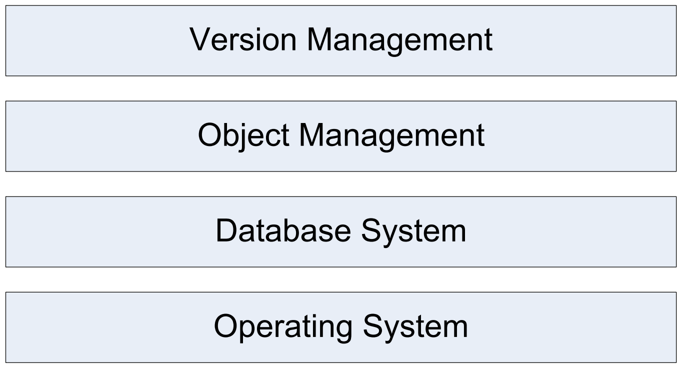


Fig: Version Management System

13.4 Control Models

- Are concerned with the control flow between sub-systems
- Centralized control
- Event-based Control

13.4.1 Centralized Control

- A control sub-systems takes responsibility for managing the execution of other sub-systems.

13.5 Call Return Model

- Top-down sub-routine model where control starts at the top of the sub-routine hierarchy and moves downwards.
- Applicable to sequential systems.

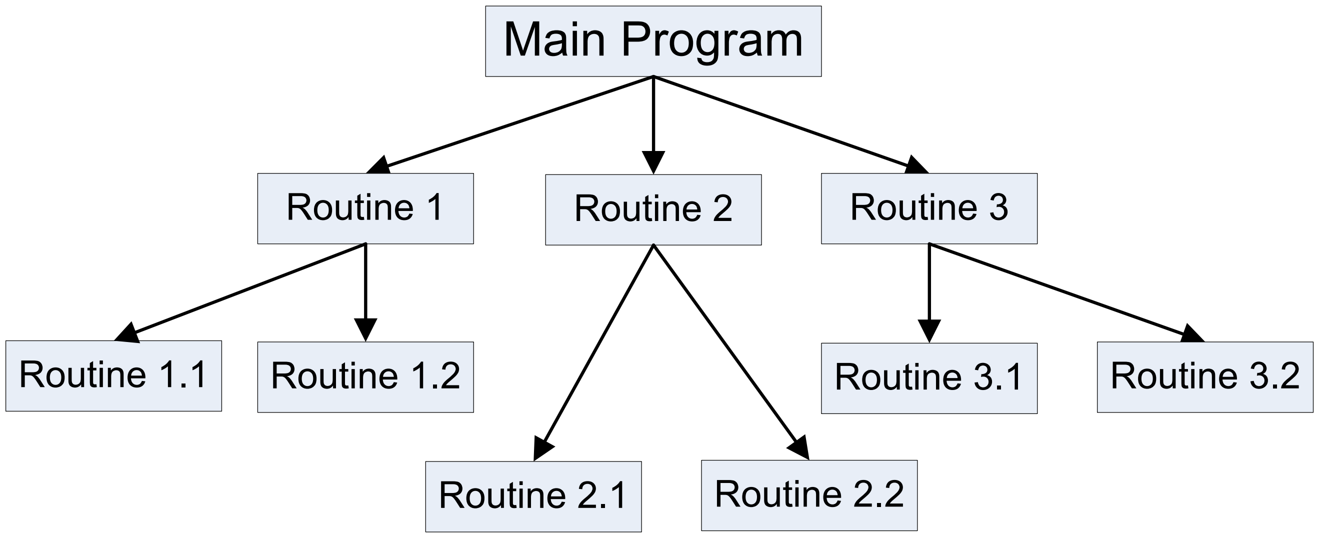


Fig: Call Return Model

13.6 Manager Model

- Applicable to concurrent systems. One system component controls the stopping, starting and coordination of other system processes.

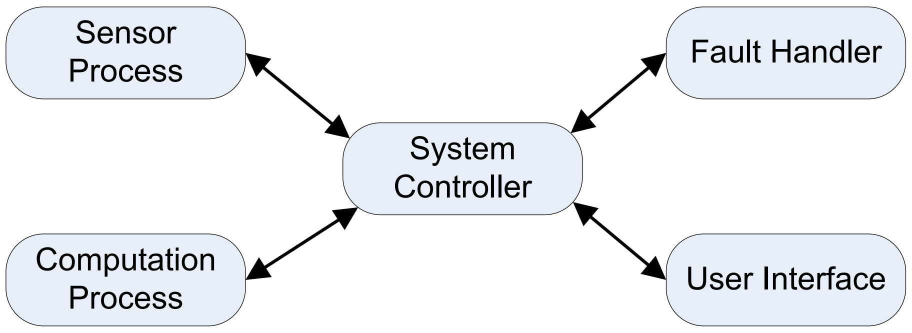


Fig: Real time System Control

13.7 Modular Decomposition

- Another structural level where sub-systems are decomposed into models.
- 2 modular decomposition models
  - Object Model
  - Data flow Model

13.7.1 Object Model

- Structure the system into a set of loosely coupled objects with well-defined interfaces

- Object Oriented decomposition is concerned with identifying object classes, their attributes and operations
- When implemented objects are created from these classes and come control model used to co –ordinate object operation
- TABLE

13.7.1.1 Advantages of Object Model

(i) The implementation of objects can be modified without affecting other objects because objects are loosely coupled
(ii) Objects are often representing of real world entities so the structure of system is readily understandable
(iii) Because these real world entities are used in different systems, objects can be reused

13.7.1.2 Disadvantages of Object Model

(i) To use services, objects must explicitly reference the name and interface of other objects
(ii) If an interface change is required to satisfy proposed system changes the effect of that change on all users of the changed object must be evaluated
(iii) More complex entities are sometimes difficult to represent as objects

13.7.2 Data Flow Model (Filter – *gets purified while moving from one step to another*)

- In a data flow model/function oriented pipeline functional transformations process their inputs and produce outputs
- Data flows from one to another and is transferred as it moves through the sequence
- The transformation may execute sequentially or in parallel and each processing step is implemented as a transform

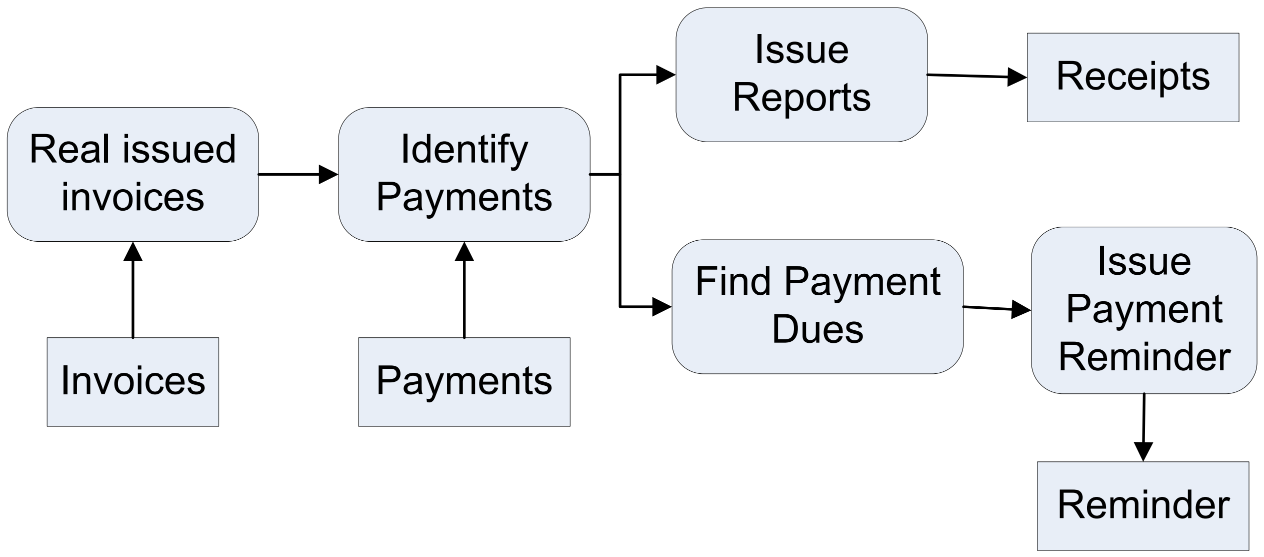


Fig: Data flow model for Invoice Processing System

13.7.2.1 Advantages of Dataflow Model

(i) It supports the reuse of transformation
(ii) It is intuiting in that many people think of their work in terms of input and output process
(iii) Evolving the system by adding new transformation is usually straight-forward
(iv) It is simple to implement either as concurrent or sequential system

13.7.2.2 Disadvantages of Dataflow Model

(i) There has to be a common format for data transfer that can be recognized by all transformation.
(ii) Each transformation must either agree with its communicating transformation on the format of the data that will be processed with a standard format for all data communicated must be imposed.

13.8 Domain Specification Architecture

- Architectural Models which are specific to some application domain
- 2 types:
    a) Generic Model
    b) Reference Model

13.8.1 Generic Model

- Are abstractions form a number of real systems
- They encapsulate the principle characteristics of these systems

13.8.2 Reference Model

- Are more abstract and describes a larger classes of systems
- They are a way of informing designers about the general structure of that class of system

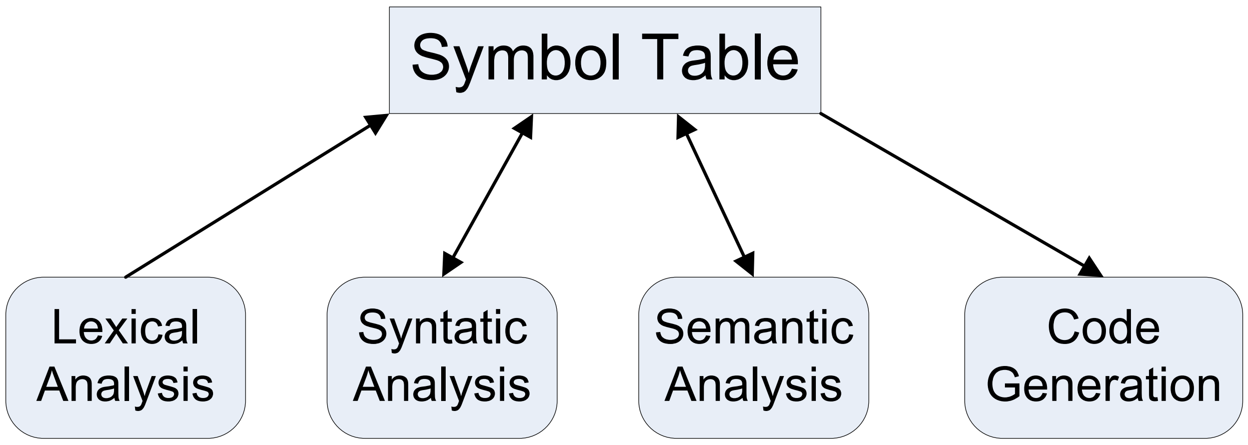


Fig: Compiler Model Representing Generic Models

OSI Reference Model				
Application				Application
Presentation				Presentation
Session				Session
Transport				Transport
Network		Network		Network
Data link		Data link		Data link
Physical		Physical		Physical

Fig : Communication Model

14. Formal Specification

14.1 Formal Specification(Unit 2.3)

Technique for the unambiguous specification of software

14.2 Formal Methods

Formal Specification is a part of a more general collection of techniques known as 'formal methods'
They are all based on mathematical representation and analysis of software

Includes:
(i) Formal Specification
(ii) Specification analysis and proof
(iii) Transformational Development
(iv) Program Verification

14.3 Acceptance of FM

FMs have not become mainstram software development technique as was once predicted

1. Other SE techniques have been successful have been successful at increasing system quality. Hence need for formal methods has been reduced.
2. Market changes have made time to market rather than s/w with a low error count the key factor.

   F.M do not reduces time to market
   - The scope of F.M is limited. They are not well suited to specifying and analyzing user interfaces and user interactions.
   - F.Ms are hard to scale up to large systems.

14.4 Uses of FMs

1. FMs have limited practical applicability
2. Their principle benefits are in reducing the no. of errors in systems so their main area of applicability is critical systems.
3. In this area, the use of FMs is most likely to be cost effective.

14.5 Specifications in s/w process

- Specification and design are inextricably intermingled.
- Architectural design is essential to structure a specification.
- Formal specification are expressed in a mathematical notation with precisely defined vocabulary, syntax and semantics.

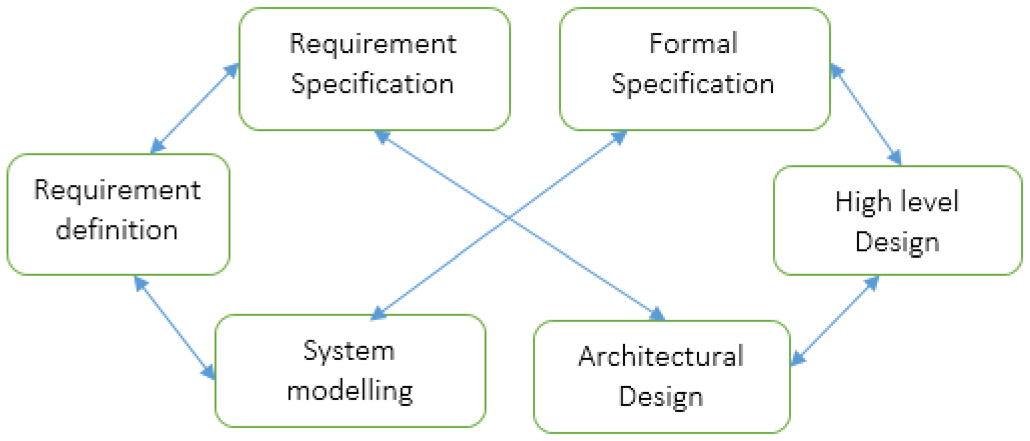


Fig: Formal specification in s/w process

14.6 Specification Techniques

14.6.1 Algebraic Approach

- The systems specified in terms of its operations and their relationships

14.6.2 Model based approach

- The system is specified in terms of a state model that is constructed using mathematical constructs such as sets and sequence. Operations are defined by modifications to system's state

14.6.3 Use of formal specification

- Involves investing more effort in the early process of software development
- This reduces requirements errors as it forces a detailed analysis of requirements
- Incompleteness and inconsistency can be discovered and resolved
- Hence, saving are made as a the amount of rework due to requirements problem reduction

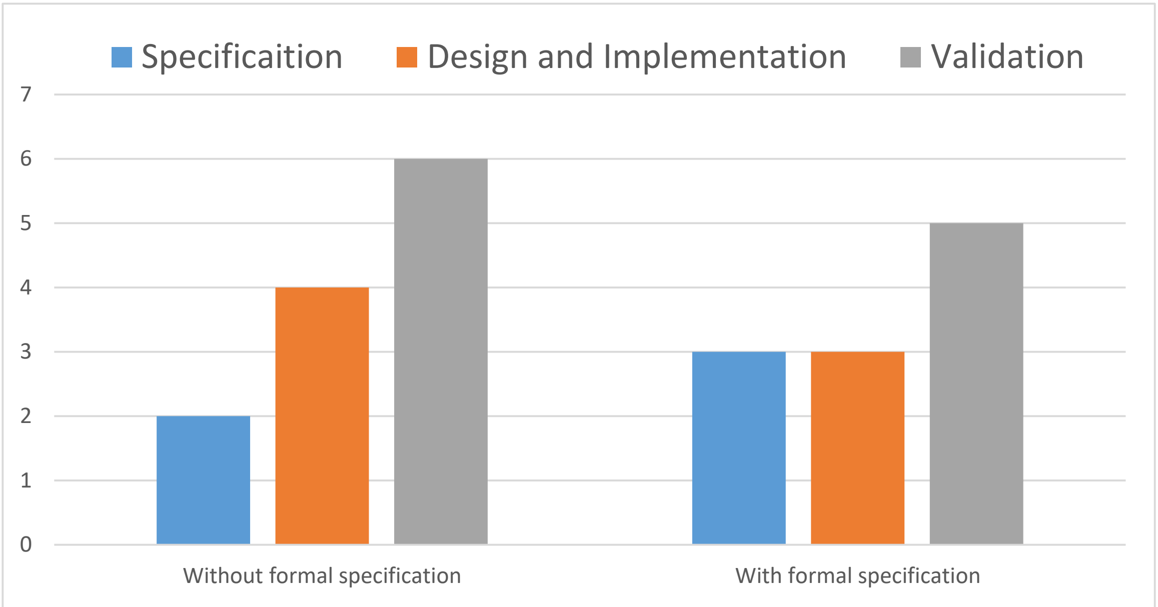


Fig:Development Costs with & without formal specification

14.7 Interface Specification

- Large systems are decomposed into subsystems with well-defined interfaces between these subsystems
- Specification of subsystem intefaces allows independenet development of different subsystems
- Interfaces may be defined as abstract data types or object class
- The algebraic approcach to formal specification is particularly well suited to interface specification

UNIT-3

Architectural Design [3hrs]

- Introduction
- System Structuring()

Object Oriented Design [3hrs]

- Introduction

15.Object Oriented Design

15.1 Characteristics of OOD

(i) Objects are abstraction of real world or system entities and manage themselves
(ii) Objects are independent and encapsulate state and representation information
(iii) System functionality is expressed in terms of object services
(iv) Shared data areas are eliminated. Objects communicate by message passing
(v) Objects may be distributed and may execute sequentially or in parallel

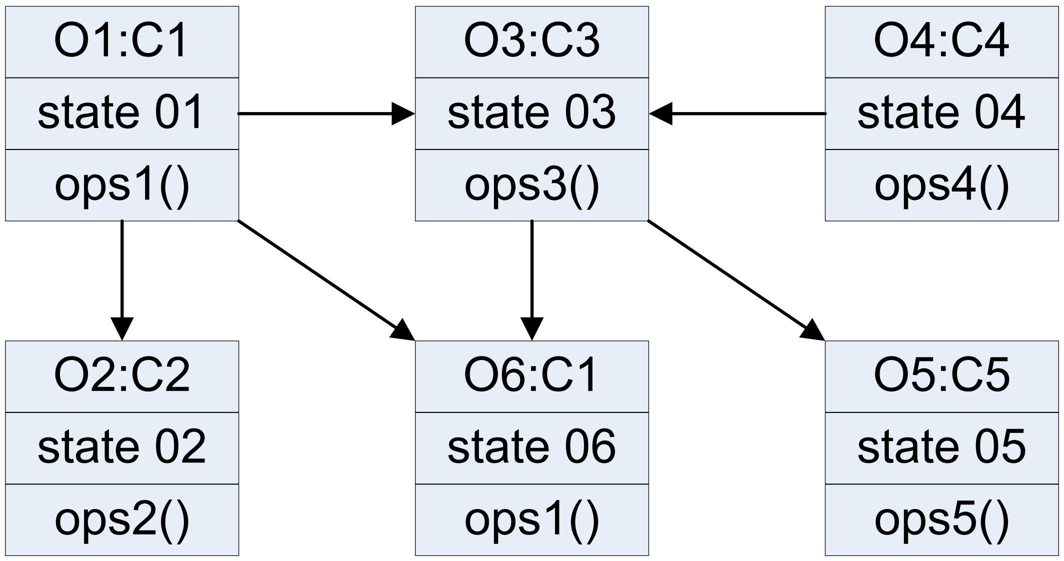


Fig: Interacting Objects

{ #MISSING A whole lotta Notes here. Ask DJ, Ayush for typed notes from11.03 11.13}

15.2 Software Cost Estimation

15.2.1 Introduction:

Project Cost Estimation and project scheduling are normally carried out together

There are three parameters involved in computing the total cost of a software development project.

1. Hardware and software costs including maintenance
2. Travel and training costs
3. Effort cost(The cost of paying software engineers)

Organizations compute costs in terms of overhead costs where they take the total cost of running the organization and divide this by the number of productive staff.

The following costs are all part of the total effort cost.

(i) Cost of providing heating and lighting office space
(ii) Cost of support staff such as accountants, administrators, system managers, cleaners and technicians
(iii) Cost of networking and communications
(iv) Cost of central facilities such as a library or recreational facilities
(v) Cost of social security and employ benefits such as pensions and health insurance

15.2.2 Productivity

Productivity estimates are usually based on measuring attributes of the software and dividing this by the total effort required for development.

There are two types of metrics that have been used:

(i) Size related matrix
(ii) Function related matrix

Factors

(i) Market opportunity
(ii) Cost estimation uncertainty
(iii) Contractual terms
(iv) Requirements volatility
(v) Financial help

A function point is not a single characteristic but it is computed by combining several different measurements or estimated. The total number of function points in a program by measuring or estimating the following program features.

(i) External I/O
(ii) User Interaction
(iii) External Interfaces
(iv) File used by the system

Unadjusted function point count (UFC) by multiplying each initial count by the estimate weight and summing all values:

$$UFC = \sum (number\ of\ element\ of\ given\ type) \times (weight)$$

❖ The number of object points in a program is a weighted estimate of
(i) The number of separate screens that are displayed.
(ii) The number of reports that are produced

(iii) The number of modules imperative programming language such as Java or C++

❖ Factors affecting software engineering productivity

(i) Application domain experience
(ii) Process quality
(iii) Project size
(iv) Technology support
(v) Working environment

15.2.3 Cost and Pricing

(i) Estimates are made to discover the cost to the developer of producing a software system
(ii) There is not a simple relationship between the development and the pricing charged to the customer
(iii) Organizational, economic, political and business considerations influence the price changes
(iv) Pricing is simply cost plus profit

15.2.4 Measurement Problems

(i) Estimating the size of measure
(ii) Estimating total number of programmer months which has elapsed
(iii) Estimating documentation team productivity and incorporating this estimate in overall estimate

15.2.5 Lines of code (LOC)

(i) The measure was first proposed when programmers were typed in word with one line per card
(ii) How does this correspond to statements as in Java which can span several lines where there can be several statements in one line?

LOC=AVC*no. of function points

Where,

AVC= Average number of LOC

15.2.6 Object Points

(i) Object points are an alternative function related major to function point when database language are used for development.
(ii) Object points are not the same as object classes
(iii) The number of object points in a program is a weighted estimation of:
   a. The number of screen that are displayed
   b. MATHI CHA… YA DEKHI.. GET NOTES FROM THERE…

15.3 Estimation Techniques

There are five types of

(i) Algorithmic Cost Modelling
(ii) Expert Judgment
(iii) Estimation by Analogue
(iv) Parkinson's Law
(v) Pricing to win

15.3.1 Algorithmic Cost Modelling

(i) A formulaic approach based on historical cost information which is generally based on the size of the software.
(ii) Cost is estimated as a mathematical function of product and process attributes those values are estimated by project managers
Effort=A x Size^B x M
A$\rightarrow$Organizational dependent Constant
B$\rightarrow$Lines between 1 and 15
M$\rightarrow$Multiplier reflecting product, process and people attribute

15.3.2 CoCoMo Model (Constructive Cost Model)

(i) An empirical model based on project experience.
(ii) Well documented independent model which is not tied to a specific software vendor
(iii) Long history from initial version published in 1981 (CoCoMo-81) through various instantiations to CoCoMo-2

CoCoMo-81		
Project Complexity	Formula	Description
Simple	PM = 2.4 (KDSI)$^{1.05}$ x M	Well understood applications developed by small teams
Moderate	PM = 3.0 (KDSI)$^{1.12}$ x M	More complex projects where team members may have limited experience of related systems
Embedded	PM = 3.6 (KDSI)$^{1.20}$ x M	Complex projects where the software is a part of a strongly coupled complex of hardware, software regulations and operational procedures
KDSI = Thousands of lines of source code		

CoCoMo-2

CoCoMo-2 supports a spiral model of development and embeds several sub-models that produce increasingly detailed estimates. This can be used in successive rounds of the development spiral. CoCoMo-2 is a 3 layered model that allows increasingly detailed estimate to be prepared as development progresses.

(i) Early prototyping levels (The application composition)
(ii) Early design levels
(iii) Post Architectural Levels

15.3.2.1 Early prototyping levels (The application composition)

(i) Supports prototyping projects and projects where there is extensive reuse
(ii) Based on standard estimates of developer productivity in object points per month
(iii) Case tool used into account text
(iv) Formula is:
PM= (NOP x (1-%reuse/1000))/PROD
where,
NOP = No of Object Points
PROD=productivity
reuse=percentage reuse is an estimation of the amount of reused code in the development

15.3.2.2 Early design levels

(i) Estimates can be made after the requirements have been agreed.
(ii) Based on standard formula for algorithmic model, formula is:
$PM=A \times Size^{B} \times PM_{m}$
M= PERS x RCPX x RUSE x PDI x PREX x FCIL x SCED
$PM_{m}=(ASLOC \times (AT/100))/ATPOROD$
Where,
A=2.5
B=1.1 to 1.24
M= Multiplier
PERS = Personal Capabilty
ASLOC = No of lines of code in the components that have to be adopted
ESLOC = The equivalent number of lines of new source code
ATPROD = Productivity of Engineers in Integrating such code
RCPH = Product Reliability and Complexity
RUSE = Reuse Required
PDIF = Platform Difficulty
PREX = Personal Experience
FCIL = Support Facilities
AD = Percentage of Adapted code that is automatically generated

(iii) Post Architecture Level
Using Same formula as early design estimates of size is adjusted to take into account a formula is:
ESLOC= ASLOC x (AA + SU + 0.4 DM + 0.3 CM + 0.3 IM)/100
where,
ESLOC = The equivalent number of lines of new source code
ASLOC = No of lines of reusable code
DM = Design Modify
CM = Code Modify
IM = Integration Effort
SU = Cost of Software Understanding
AA = Assessment Cost

15.3.3 Expert Judgement
One or more experts in

Use their experience to predict software costs.

15.3.3.1 Advantage
Relatively cheap estimation method

Can be accurate if experts have direct experience of similar system

15.3.3.2 Disadvantage
Very inaccurate if there is no expert

15.3.4 Estimating by Analoging
The cost of project is computed by comparing the project to a similar project in a same application domain

15.3.4.1 Advantage
Accurate if project data available

15.3.4.2 Disadvantage
Impossible if no comparable project has been tackled

15.3.5 Parkinson's Law
The project cost whatever resources are available

15.3.5.1 Advantage
No over speed

15.3.5.2 Disadvantage
System is usually unfinished

15.3.6 Pricing to Win
The project cost whatever the customer has to spend on it.

15.3.6.1 Advantage
You get the contract

15.3.6.2 Disadvantage
The probability that the customer gets the system s/he wants is small

15.4 Inspection

15.4.1 Inspection Procedure
(i) System Overview presented to inspection team
(ii) Code and associated documents are distributed o inspection team in advance
(iii) Inspection takes place and discovered errors are noted
(iv) Mopdifications are made to repair discovered errors
(v) Reinspection may or may not be required

15.4.2 Inspection Team
(i) Made up of atleast 4 moderators
(ii) Author of the coding being inspected
(iii) Inspector who finds error, omission, and inconsistency
(iv) Reader who reads code to the team
(v) Reader who reads code to the team
(vi) Moderator who cheers meeting and notes discovered error
(vii) Other roles are scribe and chief moderator

15.4.3 Inspection Checklist
(i) Checklist of common error should be used to drive the inspection
(ii) Error checklist is programming language dependent
(iii) The weaker the typechecking, the larger the checklist

15.4.4 Inspection Rate:
(i) 500 statements per hour during overview
(ii) 125 source statements/hour during individual preparation
(iii) 90-125 statements/hour can be inspected during inspection meeting
(iv) Inspection is therefore an expensive process

UNIT-4

Architectural Design [3hrs]

- Introduction
- System Structuring()

Object Oriented Design [3hrs]

- Introduction

16. Testing Process

16.1 Component Testing

Testing of individual program components. Usually the responsibility of component developer

16.2 Integration Testing

Testing of groups of components integrated to create a system or sub-system. The responsibility of an independent testing team:

Component testing → Integration testing

(S/W Developer) (Independent testing)

16.3 Defect Testing

- Goal: TO discover defects in program
- A successful defect test is a test which cause a program to behave in anomalous manner
- Test shows the presence not the absence of defects

16.4 Testing Priorities

- Test should exercise a system capabilities rather than its component
- Testing old capabilities is more important than boundary value cases

16.5 Test data and test cases

16.5.1 Test Data

Inputs which have been decided to test the system

16.5.2 Test Cases

Inputs to test the system and the predicted O/P's from these inputs if the syss operates according to its specification

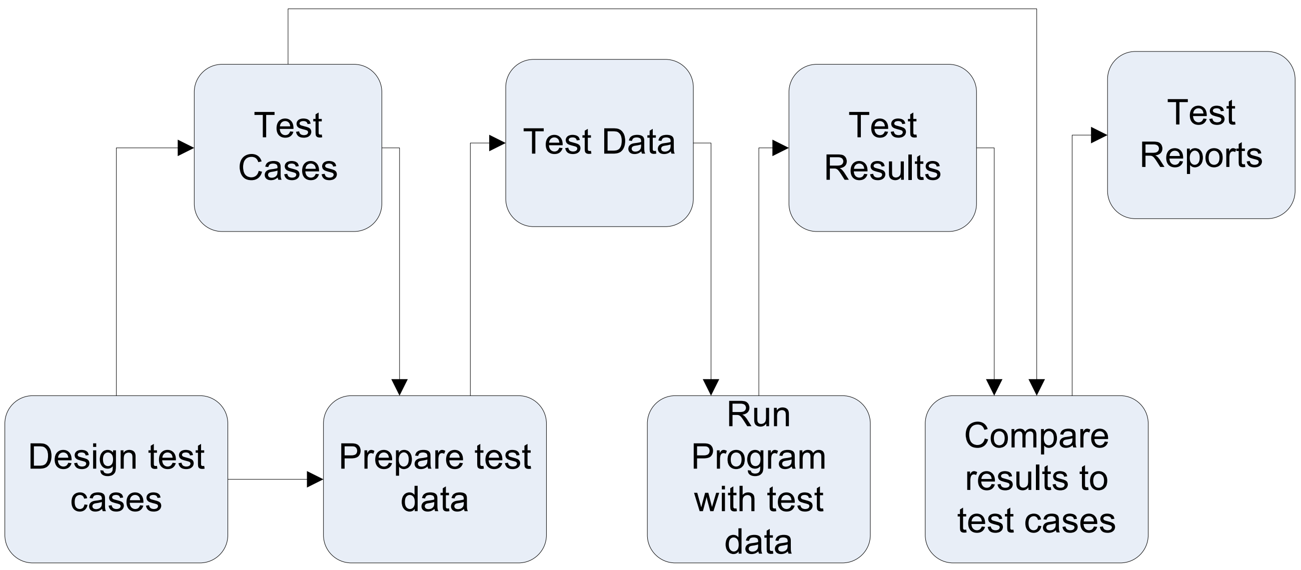


Fig: Testing Process

BLACK BOX TESTING{NOTE FROM SOMMERVILLE #MISSING }

16.6 Equivalence Partitioning

- Input data and output results often fall into different class where all member of class are related
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member

{ #MISSING FIGURE HERE}

- Test cases should be chosen for each partition
- For example: Partition system inputs and outputs into equivalent sets: If the input is a 5-digit integer between 10,000 and 99,999 equivalence partitionare less thsn 10000, 10000-99999 and greater than 99999

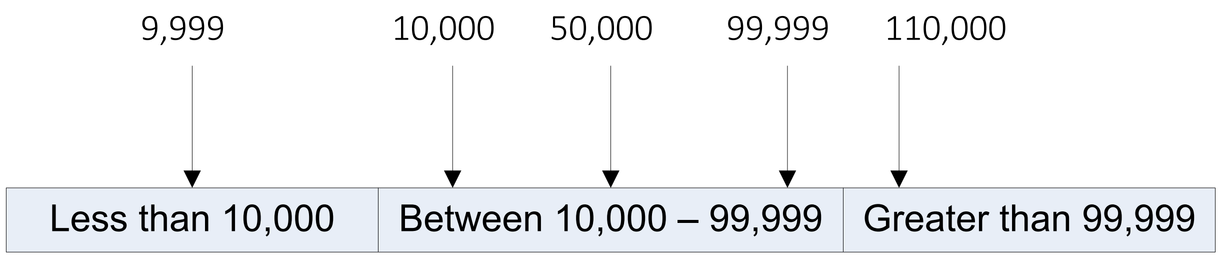


Fig: Equivalence Partitioning

16.7 Search Routine Input Pattern

- Inputs which conforms to pre-conditions
- Inputs where a pre-conditioin doesn't hold
- Input where the key element is the member of the array
- Input where key element is not a member of the array

Input Sequence	Key	Output
17	17	TRUE, 0
17	0	FALSE, ??
17, 29, 21, 23	17	TRUE, 0
41, 18, 9, 31, 30, 16, 45	45	TRUE, 6
17, 18, 21, 23, 29, 41	23	TRUE, 3
21, 23, 29, 33, 38	25	FALSE, ??

16.8 Structureal Testing/ White Box

- Derivation of test cases according to program structure
- Knowladge of program is used to identify additional test cases
- Objective is to exercise all program statements

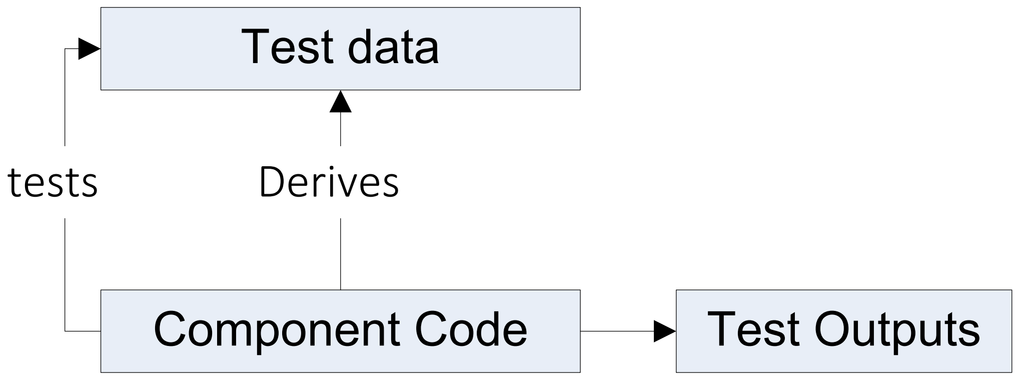


Fig: Structural Testing

16.9 Binary Search by Equivalence

- Pre-condition satisfied , Key element in array
- Pre-condition satisfied , Key element not in array
- Pre-condition unsatisfied , Key element in array
- Pre-condition unsatisfied , Key element not in array
- Input array has single value
- Input array has even number of values
- Input array has a number odd number of value

16.10 Path testing

- The objective of path testing is to ensure that the set of test cases is such that each path through a program is executed atleast once
- The starting point for path test is a program flow graph that shows nodes representing program decisions and arc representing floe of control
- Statements with conditions are therefore nodes in flow graph

16.11 Program flow graph

- Describes program control flow
- Each branch is shown as a separate path and loops are shown by arrows looping back to the loop condition node
- Used as a basis for computing cyclomatic complexity/number
- The flow graph is constructed by replacing program control statements by equivalent diagrams
- If there are no goto statements in a program it is a simple process to derive its flow graph
- In a conditional statements, each branch shown as a separate path. An arrow loping back to the condition node denotes a loop.

Public static void Search(int key, int[] elemArray, Result r)	
{	
1	int bottom=0;
2	int top = elemArray.lenght-1;
	int mid;
	r.found=false;;
4	r.index=-1;
5	while(bottom<=top)
	{
6	mid=(top+boottom)/2;
7	If(elemArray[mid]==key)
	{
8	r.index=mid;
9	r.found=true;
10	return;
	}//if part
	else
	{
11	If(elemArray[mid]<key)
12	Bottom=mid+1;
	Else
13	Top=mid-1;
	}

	}//while loop
14	}//search

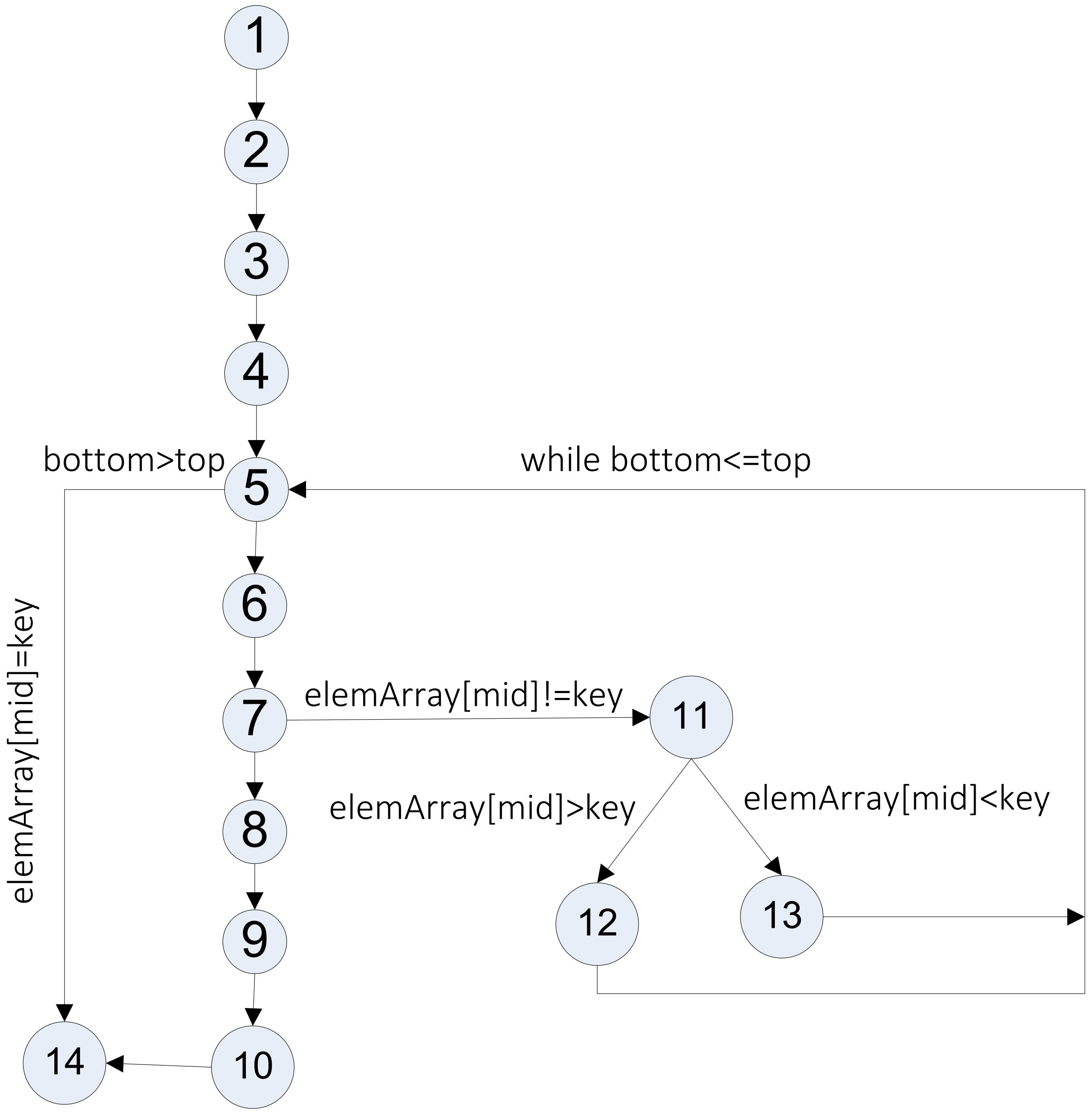


Fig: Flow graph of binary search routine

16.12 Independent Path

1,2,3,4,5,6,7,8,9,10,14

1,2,3,4,5,14

1,2,3,4,5,6,7,11,12,5,14

1,2,3,4,5,6,7,11,12,5,6,7,8,9,10,14

1,2,3,4,5,6,7,11,13,5,14

1,2,3,4,5,6,7,11,13,5,6,7,8,9,10,14

Cyclomatic complexity = condition +1 = 3+1 =4

Eg: If there are 6 statements and a while loop and all conditional expression are simple then

Cyclomatic complexity = 7+1 =8

Eg: If there are 6 statements and a while loop and if one conditional expression is a compound expression such as "if A & B or C", then cyclomatic complexity = 9+1 =10

If all of these paths are executed, we can be sure that every statement in the method has been executed atleast once and that every branch has been exercised for true and false condition.

You can find the number of independent paths in a program by computing the cyclomatic complexity of the program flow graph for programs without goto statements, the value of cyclomatic complexity is one more than number of conditions.

A simple condition is logical expression without 'and' or 'or'connectors. If program includes compound condition, which are logical expressions including 'and' or 'or' connectors, then you count the number of simple condition in compound condition where calculating the cyclomatic complexity

16.13 Integration Testing

- Test complete system or sub-system compound of integrated component
- Integration testing should be black box testing with test derive from specifications
- Many difficulties are localizing error
- Incremental integration testing reduces this process

16.14 Approaches

(i) Top down testing: Start with high level system and integration from top down replacing individual components by stubs(receipts/ticket) where appropriate

(ii) Bottom up testing: Integrate individual components in levels until the complete system is created

    - In practice, most integration involves a combination of these strategies

{#MISSING Missing note including two figures//or one??? Idk}

16.15 Guidelines to Interface testing

(i) Design test so that parameters to a called procedure are at the extreme ends of their ranges
(ii) Always test pointer parameters with null pointer
(iii) Design test which cause component to fail
(iv) Use stress-testing in message passing system
(v) In shared memory system, vary the order in which components are activated.

16.16 Stress testing

(i) Exercises system beyond its maximum design load. Stressing the system often causes defects to come to light.
(ii) Stressing the system test failure behavior. System should not fall catastrophically. Stress shecking tests checks for unacceptable loss of services or data.

16.17 Testing workbenches (Test Automation)

(i) Testing is an expensive process phase. Testing workbenches provide a range of tools to reduce the time required and total testing cost.
(ii) Most testing workbenches are open system because testing needs are organization specific.

{#MISSING Figure and almost a period worth of notes missing because my battery went dead}

16.18 Security Assessment

- Becoming increasingly important as more and more critical systems can be accessed by anyone with a network connection.
- V & V processes for web-based systems must focus on security assessment, where ability of system to resist different attacks are tested.
- This type of assessment is very difficult to carry out
- Consequently, system are often deployed with security loopholes that attackers use to gain access to or damage these systems.
- Four complementary approaches for security checking
  (i) Experience based validation
    - System is analyzed against types of attack that are known
    - Usually carried out in conjunction with tool based validation.
  (ii) Tool based validation:
    - In this case, various security tools such as password checkers are used to analyze system.
  (iii) Tiger teams:
    - A team is set up and given the objective of breaching system security

  - Simulate attacks in system and use their ingenuity to discover ways to compromise system security

(iv) Formal Verification:

- A system can be verified against a formal security specification.
- Very difficult for end-user of system to verify.
- Not widely used.

EXTRA – UNIT

This UNIT is outside of the curriculum set by IOST

[i.e. Informational but not relevant for exam]

- Managing People
-

17. Managing People

17.1.1 Selecting Staff

17.1.2 Motivating People

17.1.3 Managing Groups

17.1.4 The people's capability maturity model

17.1.5 Quality Assurance

17.1.6 Quality Control

17.1.7 Software measurement and metrics

11.20 – Update-17 - END