DIGITAL NOTES

ON

SOFTWARE ENGINEERING

[R20A0511]

B.TECH III YEAR – I SEM (R20)
(2022-23)

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING &TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015Certified) Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, INDIA.

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

	L	T/P/D	C
III Year B.Tech IT -I SEM	**3**	**-/-/-**	**3**

SOFTWARE ENGINEERING
(R20A0511)

COURSE OBJECTIVES:

1. To provide the idea of decomposing the given problem into Analysis, Design, Implementation, Testing and Maintenance phases
2. To understand software process models such as waterfall and evolutionary models and software requirements and SRS document.
3. To understand different software design and architectural styles & software testing approaches such as unit testing and integration testing.
4. To understand quality control and how to ensure good quality software through quality assurance .
5. To gain the knowledge of how Analysis, Design, Implementation, Testing and Maintenance processes are conducted in an object oriented software projects

UNIT - I:

Introduction to Software Engineering: The evolving role of software, software characteristics, software Applications.

A Generic view of process: Software engineering- A layered technology, a process framework, The Capability Maturity Model Integration (CMMI). Process patterns, process assessment, personal and team process models.

Process models: The waterfall model, Incremental process models, Evolutionary process models, The Unified process.

UNIT – II

Software Requirements: Functional and non-functional requirements, User requirements, System requirements, Interface specification, the software requirements document.

Requirements engineering process: Feasibility studies, Requirements elicitation and analysis, Requirements validation, Requirements Management.

System models: Context Models, Behavioral models, Data models, Object models, structured methods, UML diagrams.

UNIT - III

Design Engineering: Design process and Design quality, Design concepts, the design model.
Creating an architectural design: Software architecture, Data design, Architectural styles and patterns, Architectural Design.

Performing User interface design: Golden rules, User interface analysis and design, interface

analysis, interface design steps, Design evaluation.

UNIT - IV

Testing Strategies: A strategic approach to software testing, test strategies for conventional software, Black-Box and White-Box testing, Validation testing, System testing, the art of Debugging.

Risk management: Reactive vs. Proactive Risk strategies, software risks, Risk identification, Risk projection, Risk refinement, RMMM, RMMMPlan.

UNIT - V

Quality Management: Software Quality, Quality concepts, Software quality assurance, Software Reviews, Formal technical reviews, Statistical Software quality Assurance, Software reliability, The ISO 9000 quality standards.

Case Study – ATM Management System.

TEXT BOOKS :

1. Software Engineering A practitioner's Approach, Roger S Pressman, 6th edition. McGrawHillInternationalEdition.

2. Software Engineering, Ian Somerville, 7th edition, Pearson education.

REFERENCE BOOKS :

1. Software Engineering, A Precise Approach, Pankaj Jalote, Wiley India,2010.

2. Software Engineering: A Primer, Waman S Jawadekar, Tata McGraw-Hill,2008

3. Fundamentals of Software Engineering, Rajib Mall, PHI,2005

4. Software Engineering, Principles and Practices, Deepak Jain, Oxford University Press.

5. Software Engineering1: Abstraction and modelling, Diner Bjorner, Springer International edition,2006. 6. Software Engineering2: Specification of systems and languages, Diner Bjorner, Springer International edition2006.

7. Software Engineering Foundations, Yingux Wang, Auerbach Publications,2008.

8. Software Engineering Principles and Practice, Hans Van Vliet, 3rd edition, John Wiley &SonsLtd.

9. Software Engineering3: Domains, Requirements, and Software Design, D. Bjorner, Springer International Edition.

10. Introduction to Software Engineering, R. J. Leach, CRC Press.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

INDEX

UNIT - I

INTRODUCTION:

Software Engineering is a framework for building software and is an engineering approach to software development. Software programs can be developed without S/E principles and methodologies but they are indispensable if we want to achieve good quality software in a cost effective manner.
Software is defined as:

Instructions + Data Structures + Documents

Engineering is the branch of science and technology concerned with the design, building, and use of engines, machines, and structures. It is the application of science, tools and methods to find cost effectivesolution to simple and complex problems.

SOFTWARE ENGINEERING is defined as a systematic, disciplined and quantifiable approach for the development, operation and maintenance of software.

The Evolving role of software:

The dual role of Software is as follows:
1. A Product- Information transformer producing, managing and displaying information.
2. A Vehicle for delivering a product- Control of computer(operating system),the communication of information(networks) and the creation of other programs.

Characteristics of software
- **Software is developed or engineered**, but it is not manufactured in the classical sense.
- **Software does not wear out,** but it deteriorates due to change.
- **Software is custom built** rather than assembling existing components.

THE CHANGING NATURE OFSOFTWARE
The various categories of software are
1. System software
2. Application software
3. Engineering and scientific software
4. Embedded software
5. Product-line software
6. Web-applications
7. Artificial intelligence software

- **System software.** System software is a collection of programs written to service other programs
- **Embedded software**-- resides in read-only memory and is used to control products and systems for the consumer and industrial markets.
- **Artificial intelligence software.** Artificial intelligence (AI) software makes use of nonnumeric algorithms to solve complex problems that are not amenable to computation or straightforward analysis

Engineering and scientific software. Engineering and scientific software have been characterized by "number crunching" algorithms.

LEGACY SOFTWARE

Legacy software are older programs that are developed decades ago. The quality of legacy software is

poor because it has inextensible design, convoluted code, poor and nonexistent documentation, test cases and results that are not achieved.

As time passes legacy systems evolve due to following reasons:

- The software must be adapted to meet the needs of new computing environment or technology.
- The software must be enhanced to implement new business requirements.
- The software must be extended to make it interoperable with more modern systems or database
- The software must be rearchitected to make it viable within a network environment.

SOFTWARE APPLICATIONS

System Software –

System Software is necessary to manage the computer resources and support the execution of application programs. Software like operating systems, compilers, editors and drivers, etc., come under this category. A computer cannot function without the presence of these. Operating systems are needed to link the machine-dependent needs of a program with the capabilities of the machine on which it runs. Compilers translate programs from high-level language to machine language.

Application Software –

Application software is designed to fulfill the user's requirement by interacting with the user directly. It could be classified into two major categories:- generic or customized. Generic Software is the software that is open to all and behaves the same for all of its users. Its function is limited and not customized as per the changing requirements of the user. However, on the other hand, Customized software the software products which are designed as per the client's requirement, and are not available for all.

Networking and Web Applications Software –

Networking Software provides the required support necessary for computers to interact with each other and with data storage facilities. The networking software is also used when software is running on a network of computers (such as the World Wide Web). It includes all network management software, server software, security and encryption software, and software to develop web-based applications like HTML, PHP, XML, etc.

Embedded Software –

This type of software is embedded into the hardware normally in the Read-Only Memory (ROM) as a part of a large system and is used to support certain functionality under the control conditions. Examples are software used in instrumentation and control applications like washing machines, satellites, microwaves, etc.

Reservation Software –

A Reservation system is primarily used to store and retrieve information and perform transactions related to air travel, car rental, hotels, or other activities. They also provide access to bus and railway reservations, although these are not always integrated with the main system. These are also used to relay computerized information for users in the hotel industry, making a reservation and ensuring that the hotel is not overbooked.

Business Software –

This category of software is used to support business applications and is the most widely used category of software. Examples are software for inventory management, accounts, banking, hospitals, schools, stock markets, etc.

Entertainment Software –
Education and entertainment software provides a powerful tool for educational agencies, especially those that deal with educating young children. There is a wide range of entertainment software such as computer games, educational games, translation software, mapping software, etc.

Artificial Intelligence Software –
Software like expert systems, decision support systems, pattern recognition software, artificial neural networks, etc. come under this category. They involve complex problems which are not affected by complex computations using non-numerical algorithms.

Scientific Software –
Scientific and engineering software satisfies the needs of a scientific or engineering user to perform enterprise-specific tasks. Such software is written for specific applications using principles, techniques, and formulae specific to that field. Examples are software like MATLAB, AUTOCAD, PSPICE, ORCAD, etc.

Utilities Software –
The programs coming under this category perform specific tasks and are different from other software in terms of size, cost, and complexity. Examples are anti-virus software, voice recognition software, compression programs, etc.

Document Management Software –
Document Management Software is used to track, manage and store documents in order to reduce the paperwork. Such systems are capable of keeping a record of the various versions created and modified by different users (history tracking). They commonly provide storage, versioning, metadata, security, as well as indexing and retrieval capabilities.

A GENERIC VIEW OF PROCESS SOFTWARE ENGINEERING-A LAYERED TECHNOLOGY

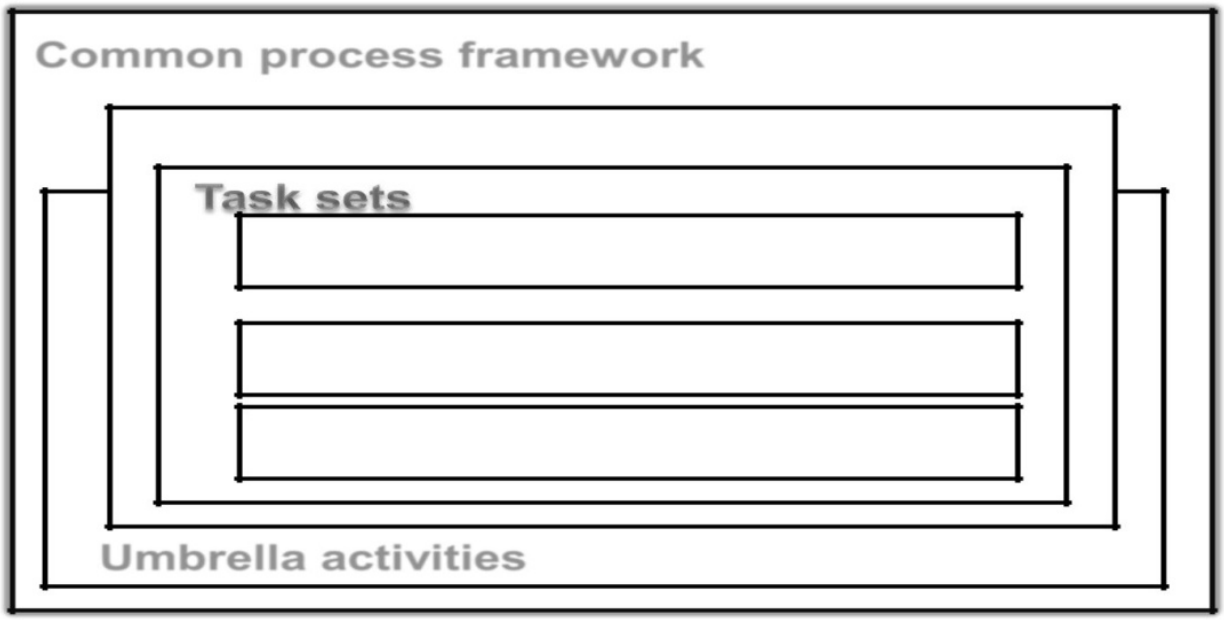
Fig: Software Engineering-A layered technology

SOFTWARE ENGINEERING - A LAYERED TECHNOLOGY

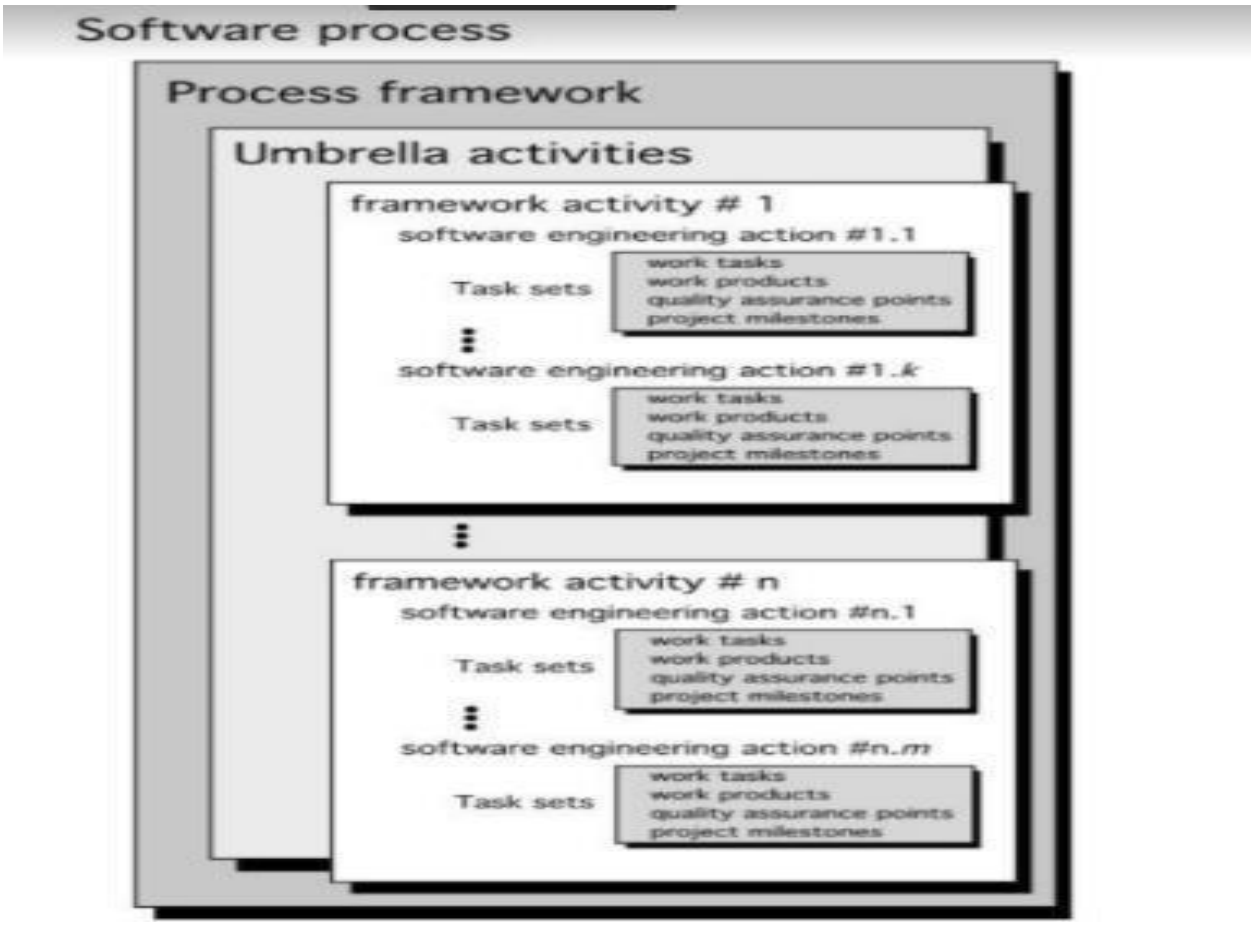
- Quality focus - Bedrock that supports Software Engineering.
- Process - Foundation for software Engineering
- Methods - Provide technical How-to's for building software
- Tools - Provide semi-automatic and automatic support to methods

A PROCESS FRAMEWORK

- Establishes the foundation for a complete software process
- Identifies a number of framework activities applicable to all software projects
- Also include a set of umbrella activities that are applicable across the entire software process.



A PROCESS FRAMEWORK comprises of :
Common process framework Umbrella activities Framework activities Tasks, Milestones, deliverables SQA points



A PROCESS FRAMEWORK

Used as a basis for the description of process models Generic process activities

- Communication
- Planning
- Modeling
- Construction
- Deployment

A PROCESS FRAMEWORK

Generic view of engineering complimented by a number of umbrella activities

- Software project tracking and control
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Document preparation and production
- Reusability management
- Measurement
- Risk management

CAPABILITY MATURITY MODEL INTEGRATION(CMMI)

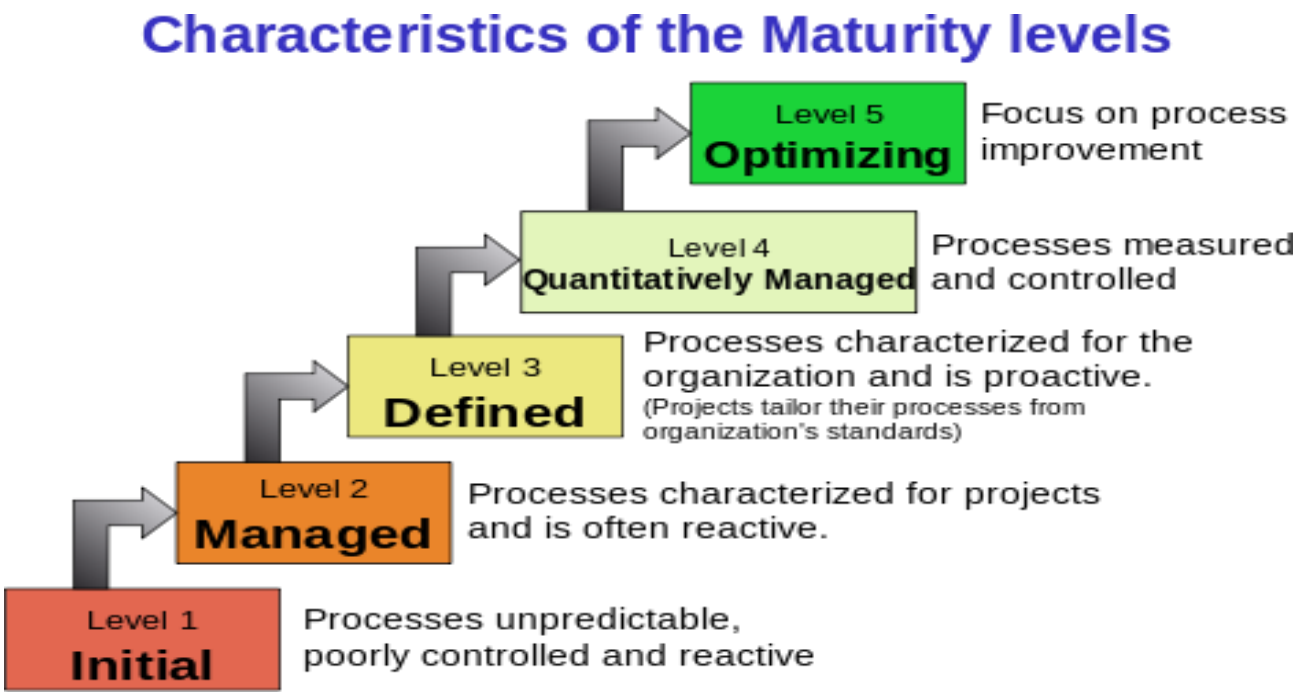
- Developed by SEI(Software Engineering institute)
- Assess the process model followed by an organization and rate the organization with different levels
- A set of software engineering capabilities should be present as organizations reach different levels of process capability and maturity.

CMMI process meta model can be represented in different ways
1.A continuous model
2.A staged model

Continuous model:
-Lets organization select specific improvement that best meet its business objectives and minimize risk-Levels are called capability levels.
-Describes a process in 2 dimensions
-Each process area is assessed against specific goals and practices and is rated according to thefollowing capability levels.

Six levels of CMMI

- Level 0:Incomplete
- Level 1:Performed
- Level 2:Managed
- Level 3:Defined
- Level 4:Quantitatively managed
- Level 5:Optimized



- Incomplete -Process is adhoc . Objective and goal of process areas are not known
- Performed -Goal, objective, work tasks, work products and other activities of software process are carried out
- Managed -Activities are monitored, reviewed, evaluated and controlled
- Defined -Activities are standardized, integrated and documented
- Quantitatively Managed -Metrics and indicators are available to measure the process and quality
- Optimized - Continuous process improvement based on quantitative feed back from the user

-Use of innovative ideas and techniques, statistical quality control and other methods for processimprovement

CMMI - Staged model
- This model is used if you have no clue of how to improve the process for quality software.
- It gives a suggestion of what things other organizations have found helpful to work first
- Levels are called maturity levels

PROCESS PATTERNS

Software Process is defined as collection of Patterns.Process pattern provides a template. It comprises of
• Process Template
-Pattern Name
-Intent
-Types
-Task pattern
- Stage pattern
-Phase Pattern
• Initial Context
• Problem
• Solution
• Resulting Context
• Related Patterns

PROCESS ASSESSMENT

Does not specify the quality of the software or whether the software will be delivered on time or will it stand up to the user requirements. It attempts to keep a check on the currentstate of the software process with the intention of improving it.
PROCESS ASSESSMENT Software Process Software Process Assessment Software Process Improvement Motivates Capability determination

APPROACHES TO SOFTWARE ASSESSMENT

• Standard CMMI assessment (SCAMPI)
• CMM based appraisal for internal process improvement
• SPICE(ISO/IEC 15504)

ISO 9001:2000 for software Personal and Team Software ProcessPersonal software process
PLANNING
HIGH LEVEL DESIGN
HIGH LEVEL DESIGN REVIEW
DEVELOPMENT
POSTMORTEM

PERSONAL AND TEAM SOFTWARE PROCESS

Team software process Goal of TSP - Build self-directed teams - Motivate the teams - Acceptance of CMM level 5 behavior as normal to accelerate software process improvement - Provide improvement guidance to high maturity organization

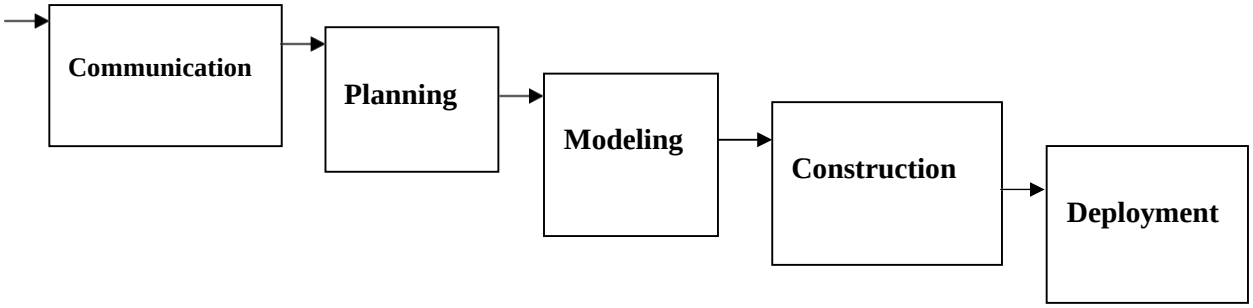
PROCESS MODELS

- Help in the software development
- Guide the software team through a set of framework activities
- Process Models may be linear, incremental or evolutionary

THE WATERFALL MODEL

- Used when requirements are well understood in the beginning
- Also called classic life cycle
- A systematic, sequential approach to Software development

• Begins with customer specification of Requirements and progresses through planning, modeling, construction and deployment.



This Model suggests a systematic, sequential approach to SW development that begins at the systemlevel and progresses through analysis, design, code and testing

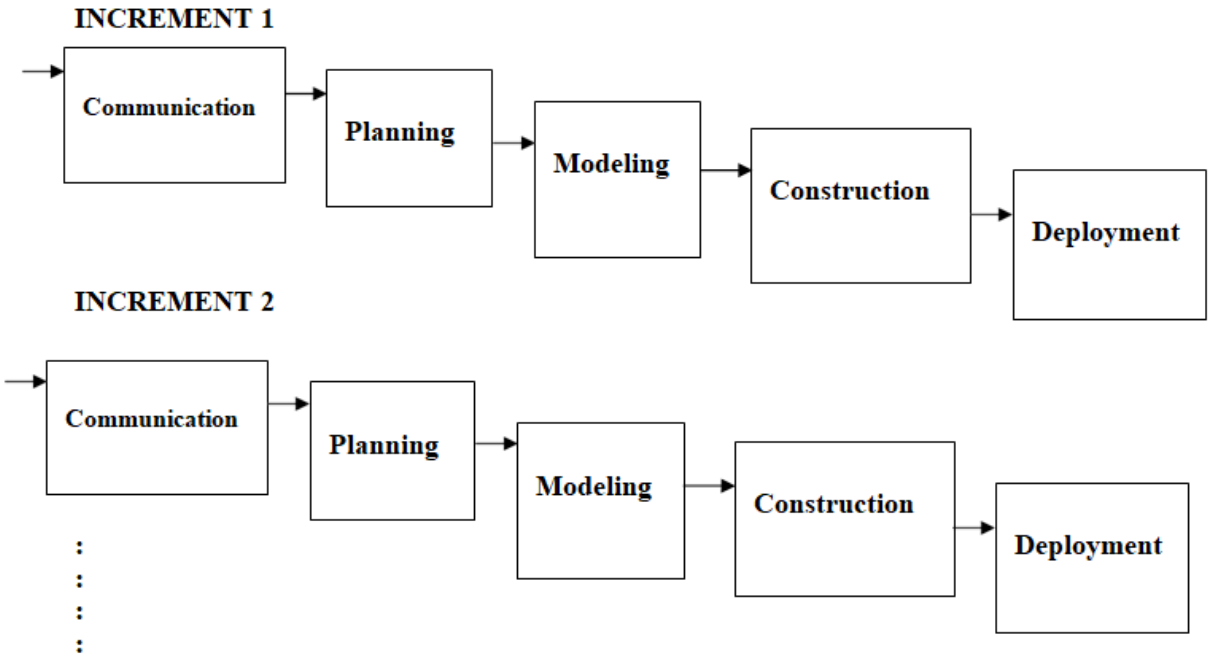
PROBLEMS IN WATERFALLMODEL

- Real projects rarely follow the sequential flow since they are always iterative
- The model requires requirements to be explicitly spelled out in the beginning, which is oftendifficult
- A working model is not available until late in the project time plan

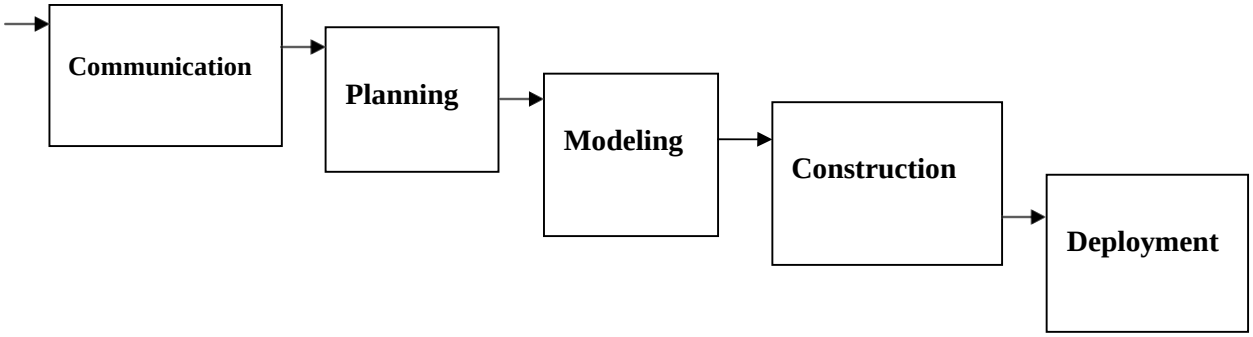
THE INCREMENTAL PROCESS MODEL

- Linear sequential model is not suited for projects which are iterative in nature
- Incremental model suits such projects
- Used when initial requirements are reasonably well-defined and compelling need to provide limited functionality quickly
- Functionality expanded further in later releases
- Software is developed in incrementsThe Incremental Model

 Communication
 Planning
 Modeling
 Construction
 Deployment



INCREMENT N



- Software releases in increments
- 1st increment constitutes Core product
- Basic requirements are addressed
- Core product undergoes detailed evaluation by the customer
- As a result, plan is developed for the next increment. Plan addresses the modification of coreproduct to better meet the needs of customer
- Process is repeated until the complete product is produced

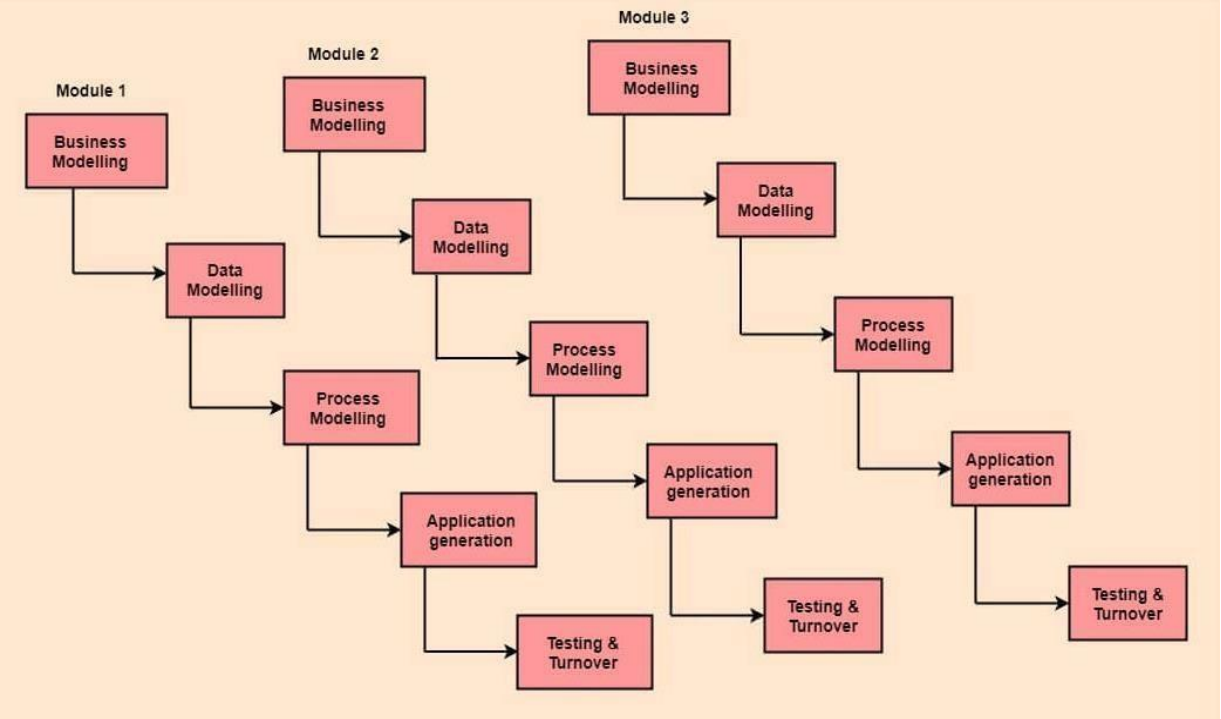
THE RAD (Rapid Application Development) MODEL

- An incremental software process model
- Having a short development cycle
- High-speed adoption of the waterfall model using a component based construction approach
- Creates a fully functional system within a very short span time of 60 to 90 days

The RAD Model consists of the following phases:

Communication Planning Construction Component reuses automatic code generation testingModeling
Business modeling
Data modeling
Process modeling

Deployment integration delivery feedback



THE RAD MODEL

- Multiple software teams work in parallel on different functions
- Modeling encompasses three major phases: Business modeling, Data modeling and processmodeling
- Construction uses reusable components, automatic code generation and testing

Problems in RAD

- Requires a number of RAD teams
- Requires commitment from both developer and customer for rapid-fire completion of activities
- Requires modularity
- Not suited when technical risks are high

EVOLUTIONARY PROCESSMODEL

- Software evolves over a period of time
- Business and product requirements often change as development proceeds making a straight-linepath to an end product unrealistic
- Evolutionary models are iterative and as such are applicable to modern day applications

Types of evolutionary models

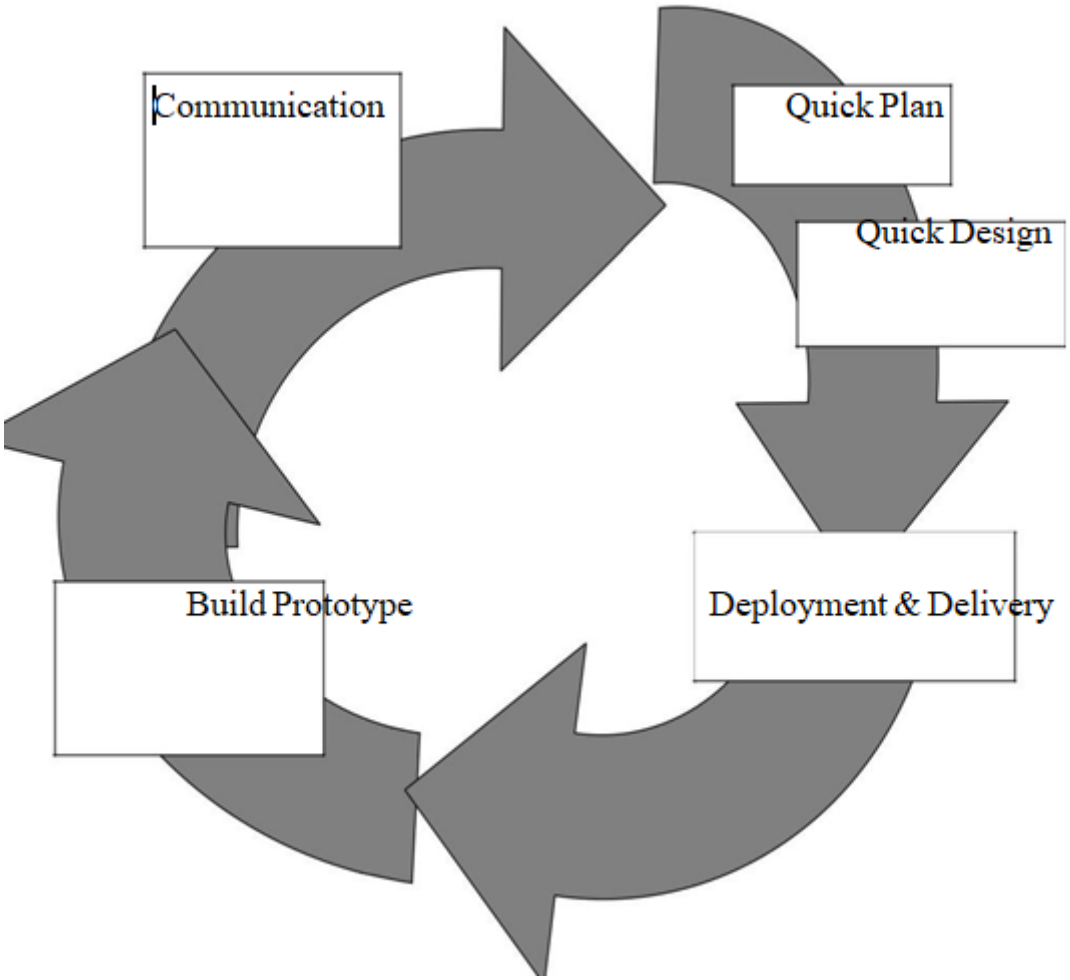
– Prototyping
– Spiral model
– Concurrent development model

PROTOTYPING

- Mock up or model(throw away version) of a software product
- Used when customer defines a set of objective but does not identify input, output, or processing requirements
- Developer is not sure of:

– efficiency of an algorithm adaptability of an operating system

– human/machine interaction



STEPS IN PROTOTYPING

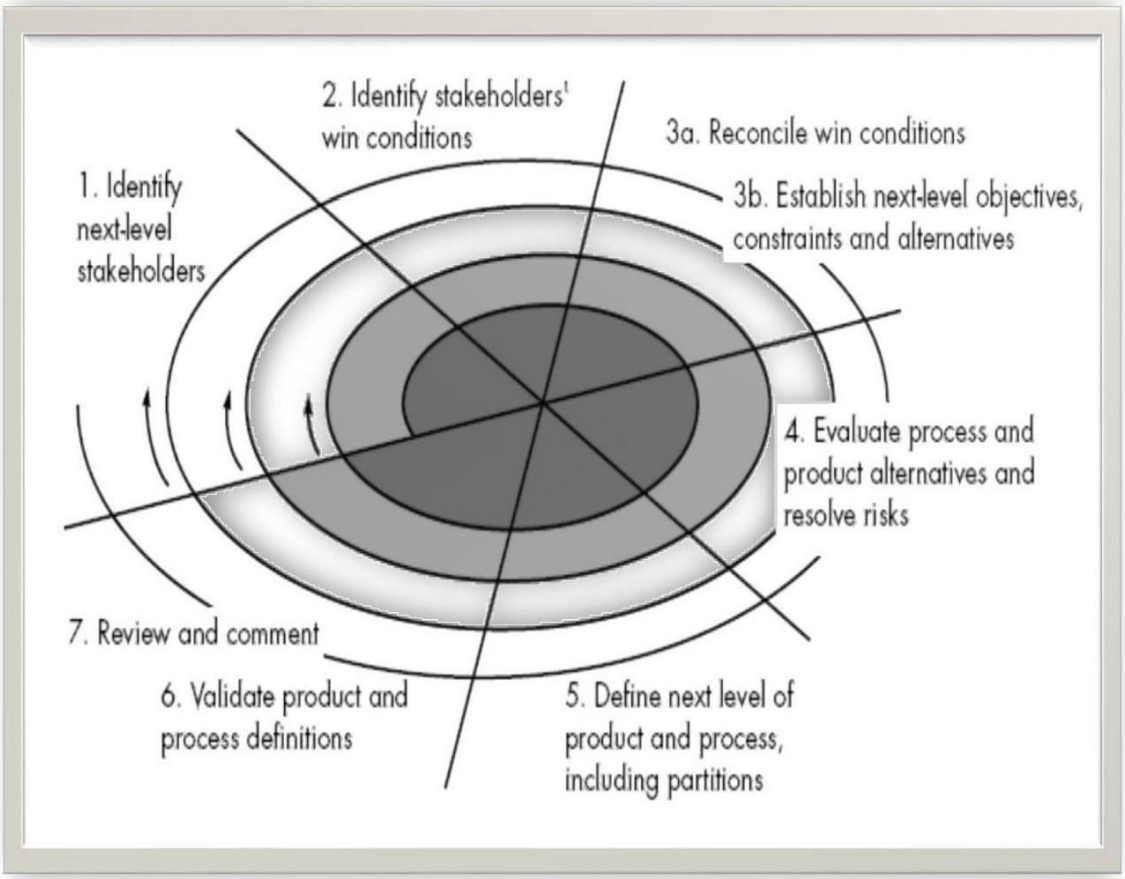
- Begins with requirement gathering
- Identify whatever requirements are known
- Outline areas where further definition is mandatory
- A quick design occur
- Quick design leads to the construction of prototype
- Prototype is evaluated by the customer
- Requirements are refined
- Prototype is turned to satisfy the needs of customer
-

LIMITATIONS OF PROTOTYPING

- In a rush to get it working, overall software quality or long term maintainability are generallyoverlooked
- Use of inappropriate OS or PL
- Use of inefficient algorithm

THE SPIRAL MODEL

An evolutionary model which combines the best feature of the classical life cycle and the iterative nature of prototype model. Include new element : Risk element. Starts in middle and continually visits the basic tasks of communication, planning, modeling, construction and deployment



THE SPIRAL MODEL

- Realistic approach to the development of large scale system and software
- Software evolves as process progresses
- Better understanding between developer and customer
- The first circuit might result in the development of a product specification
- Subsequent circuits develop a prototype
- sophisticated version of software

THE CONCURRENT DEVELOPMENT MODEL

- The concurrent development model also called concurrent engineering
- Constitutes a series of framework activities, software engineering action, tasks and their associatedstates
- All activities exist concurrently but reside in different states
- Applicable to all types of software development
- Event generated at one point in the process trigger transitions among the states

A FINAL COMMENT ON EVOLUTIONARY PROCESS

- Difficult in project planning
- Speed of evolution is not known

Does not focus on flexibility and extensibility (more emphasis on high quality)

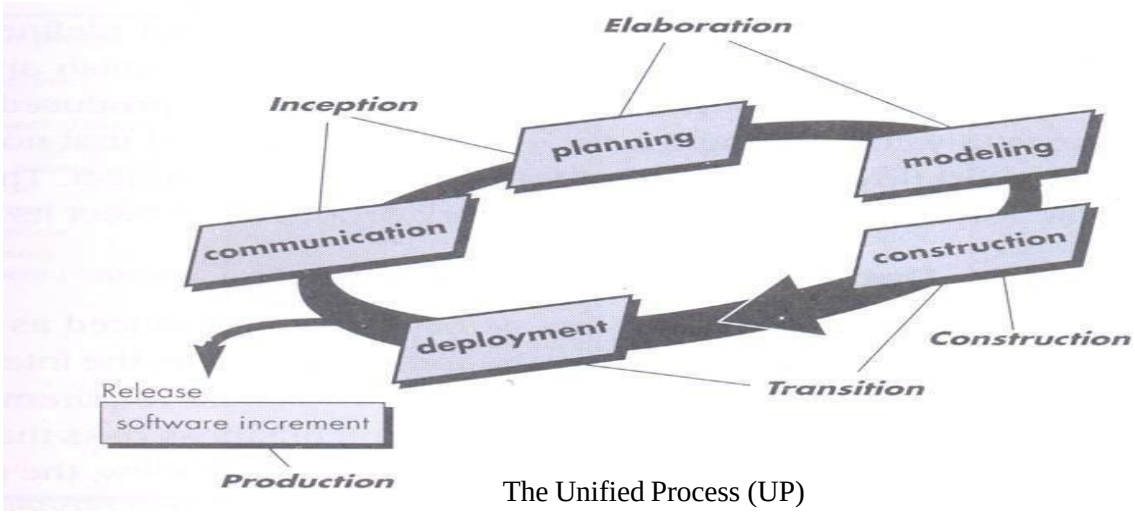
- Requirement is balance between high quality and flexibility and extensibility

THE UNIFIED PROCESS

Evolved by Rumbaugh, Booch, Jacobson. Combines the best features their OO models. Adopts additional features proposed by other experts. Resulted in Unified Modeling Language (UML). Unified process developed Rumbaugh and Booch. A framework for Object-Oriented Software Engineering using UML

PHASES OF UNIFIED PROCESS

- INCEPTION PHASE
- ELABORATION PHASE
- CONSTRUCTION PHASE
- TRANSITION PHASE



The Unified Process (UP)

UNIFIED PROCESS WORK PRODUCTS

Tasks which are required to be completed during different phases

1. Inception Phase

*Vision document
*Initial Use-Case model
*Initial Risk assessment
*Project Plan

2. Elaboration Phase

*Use-Case model
*Analysis model
*Software Architecture description
*Preliminary design model
*Preliminary model

3. Construction Phase

*Design model

*System components
*Test plan and procedure
*Test cases
*Manual

4. Transition Phase

*Delivered software increment
*Beta test results
*General user feedback

UNIT-II

SOFTWARE REQUIREMENTS

IEEE defines Requirement as :

1. A condition or capability needed by a user to solve a problem or achieve an objective
2. A condition or capability that must be met or possessed by a system or a system component to satisfy constract, standard, specification or formally imposed document
3. A documented representation of a condition nor capability as in 1 or 2

SOFTWARE REQUIREMENTS

- Encompasses both the User's view of the requirements(the external view) and the Developer's view(inside characteristics)

User's Requirements

--Statements in a natural language plus diagram, describing the services the system is expected toprovide and the constraints

- System Requirements --Describe the system's function, services and operational condition

SOFTWARE REQUIREMENTS

- System Functional Requirements

--Statement of services the system should provide
--Describe the behavior in particular situations
--Defines the system reaction to particular inputs

- Nonfunctional Requirements
- Constraints on the services or functions offered by the system

--Include timing constraints, constraints on the development process and standards
--Apply to system as a whole

- Domain Requirements

--Requirements relate to specific application of the system
--Reflect characteristics and constraints of that system

FUNCTIONAL REQUIREMENTS

- Should be both complete and consistent
- Completeness

-- All services required by the user should be defined

- Consistent

-- Requirements should not have contradictory definition

- Difficult to achieve completeness and consistency for large system

NON-FUNCTIONALREQUIREMENTS

Types of Non-functional Requirements

1.Product Requirements

-Specify product behavior
-Include the following

- Usability
- Efficiency
- Reliability
- Portability

2. Organizational Requirements

--Derived from policies and procedures
--Include the following:
- Delivery
- Implementation
- Standard

3. External Requirements

-- Derived from factors external to the system and its development process
--Includes the following
- Interoperability
- Ethical
- Legislative
-

PROBLEMS FACED USING THE NATURAL LANGUAGE

1. Lack of clarity-- Leads to misunderstanding because of ambiguity of natural language
2. Confusion-- Due to over flexibility, sometime difficult to find whether requirements are same or distinct.
3. Amalgamation problem-- Difficult to modularize natural language requirements

STRUCTURED LANGUAGESPECIFICATION
- Requirements are written in a standard way
- Ensures degree of uniformity
- Provide templates to specify system requirements
- Include control constructs and graphical highlighting to partition the specification

SYSTEM REQUIREMENTS STANDARD FORM
- Function
- Description
- Inputs
- Source
- Outputs
- Destination
- Action
- Precondition
- Post condition
- Side effects

Interface Specification
- Working of new system must match with the existing system
- Interface provides this capability and precisely specified

Three types of interfaces

1. Procedural interface-- Used for calling the existing programs by the new programs
2. 2.Data structures--Provide data passing from one sub-system to another
3. 3.Representations of Data-- Ordering of bits to match with the existing system

--Most common in real-time and embedded system

The Software Requirements document

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

The Software Requirements document

Suggests that there are 6 requirements that requirement document should satisfy. It should

- specify only external system behavior
- Specify constraints on the implementation.
- Be easy to change
- Serve as reference tool for system maintainers
- Record forethought about the life cycle of the system.
- Characterize acceptable responses to undesired events

Purpose of SRS

- Communication between the Customer, Analyst, system developers, maintainers,
- firm foundation for the design phase
- support system testing activities
- Support project management and control
- controlling the evolution of the system

IEEE requirements standard

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

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

IEEE requirements standard

1.IntroductionPurpose
Scope
Definitions, Acronyms and AbbreviationsReferences
Overview

4. General description Product perspective Product function summary User characteristics General constraints

Assumptions and dependencies

5. Specific Requirements
- Functional requirements

6. External interface requirements
- Performance requirements
- Design constraints
- Attributes eg. security, availability, maintainability, transferability/conversion
- Other requirements
- Appendices
- Index

REQUIREMENTS ENGINEERING PROCESS

To create and maintain a system requirement document. The overall process includes four high level requirements engineering sub-processes:

1. Feasibility study

--Concerned with assessing whether the system is useful to the business

2.Elicitation and analysis

--Discovering requirements

3.Specifications

--Converting the requirements into a standard form

4.Validation

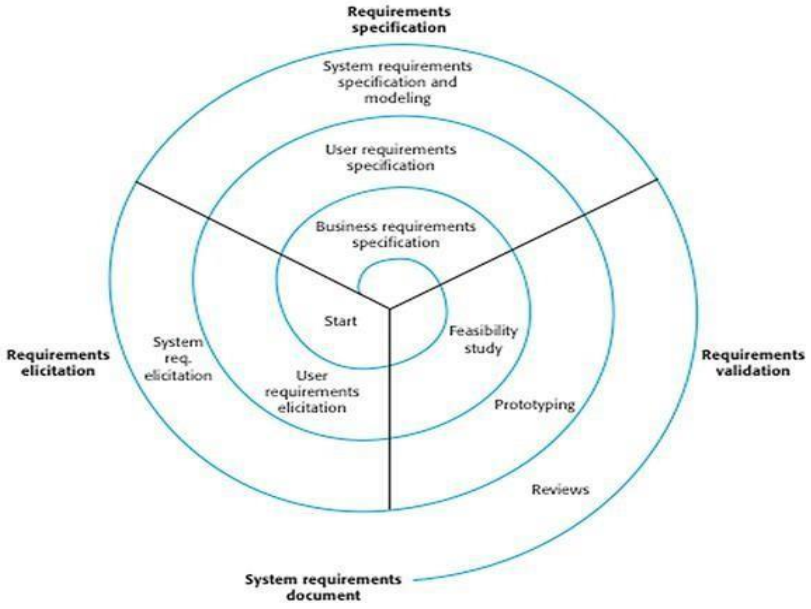
-- Checking that the requirements actually define the system that the customer wants

SPIRAL REPRESENTATION OF REQUIREMENTSENGINEERING PROCESS

Process represented as three stage activity. Activities are organized as an iterative process around a spiral. Early in the process, most effort will be spent on understanding high-level business and the use requirement. Later in the outer rings, more effort will be devoted to system requirements engineering and system modeling

Three level process consists of: 1.Requirements elicitation

2. Requirements specification
3. Requirements validation



FEASIBILITY STUDIES

Starting point of the requirements engineering process

- Input: Set of preliminary business requirements, an outline description of the system and how the system is intended to support business processes
- Output: Feasibility report that recommends whether or not it is worth carrying out further Feasibilityreport answers a number of questions:

1. Does the system contribute to the overall objective
2. Can the system be implemented using the current technology and within given cost and schedule
3. Can the system be integrated with other system which are already in place.

REQUIREMENTS ELICITATION ANALYSIS

Involves a number of people in an organization.

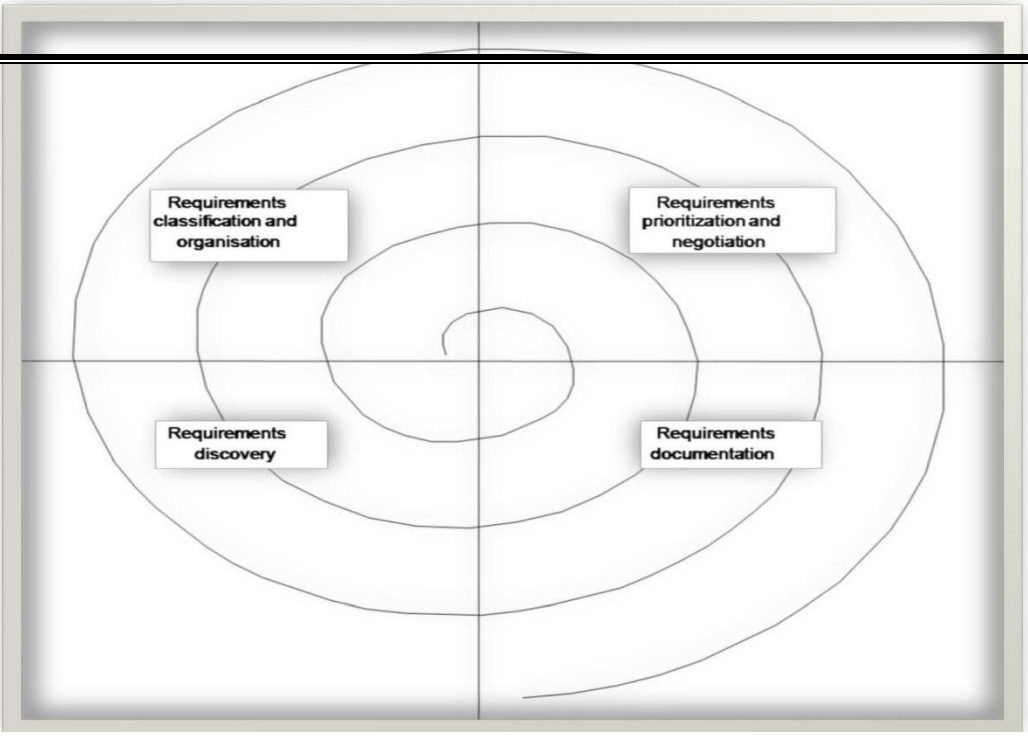
Stakeholder definition-- Refers to any person or group who will be affected by the system directly or indirectly i.e. End-users, Engineers, business managers, domain experts.

Reasons why eliciting is difficult

1. Stakeholder often don't know what they want from the computer system.
2. Stakeholder expression ofrequirements in natural language is sometimes difficult toUnderstand.
3. Different stakeholders express requirements differently

4.Influences of political factors Change in requirements due to dynamic environments.

REQUIREMENTS ELICITATION PROCESS



Process activities

1. Requirement Discovery -- Interaction with stakeholder to collect their requirements including domain and documentation
2. Requirements classification and organization -- Coherent clustering of requirements from unstructured collection of requirements
3. Requirements prioritization and negotiation -- Assigning priority to requirements

--Resolves conflicting requirements through negotiation

4. Requirements documentation -- Requirements be documented and placed in the next round of spiral
5. The spiral representation of Requirements Engineering

REQUIEMENTS DICOVERY TECHNIQUES

1. View points --Based on the viewpoints expressed by the stake holder

--Recognizes multiple perspectives and provides a framework for discovering conflicts in the requirements proposed by different stakeholders

Three Generic types of viewpoints

1. Interactor viewpoint--Represents people or other system that interact directly with the system
2. Indirect viewpoint--Stakeholders who influence the requirements, but don't use the system

3.Domain viewpoint--Requirements domain characteristics and constraints that influence the requirements.

2. Interviewing--Puts questions to stakeholders about the system that they use and the system to be developed. Requirements are derived from the answers.

Two types of interview

– Closed interviews where the stakeholders answer a pre-defined set of questions.
– Open interviews discuss a range of issues with the stakeholders for better understanding their needs.

Effective interviewers

a) Open-minded: no pre-conceived ideas
b) Prompter: prompt the interviewee to start discussion with a question or a proposal

3. Scenarios --Easier to relate to real life examples than to abstract description. Starts with an outline

of the interaction and during elicitation, details are added to create a complete description of that interaction

Scenario includes:

1. Description at the start of the scenario
2. Description of normal flow of the event
3. Description of what can go wrong and how this is handled
4. Information about other activities parallel to the scenario
5. Description of the system state when the scenario finishes

LIBSYS scenario

- **Initial assumption**: The user has logged on to the LIBSYS system and has located the journal containing the copy of the article.
- **Normal**: The user selects the article to be copied. He or she is then prompted by the system to either provide subscriber information for the journal or to indicate how they will pay for the article. Alternative payment methods are by credit card or by quoting an organizational account number.
- The user is then asked to fill in a copyright form that maintains details of the transaction and they then submit this to the LIBSYS system.
- The copyright form is checked and, if OK, the PDF version of the article is downloaded to the LIBSYS working area on the user's computer and the user is informed that it is available. The user is asked to select a printer and a copy of the article is printed

LIBSYS scenario

What can go wrong: The user may fail to fill in the copyright form correctly. In this case, the form should be re-presented to the user for correction. If the resubmitted form is still incorrect then the user's request for the article is rejected.

- The payment may be rejected by the system. The user's request for the article is rejected.
- The article download may fail. Retry until successful or the user terminates the session..

Other activities: Simultaneous downloads of other articles.

System state on completion: User is logged on. The downloaded article has been deleted from LIBSYS workspace if it has been flagged as print-only.

4. Use cases -- scenario based technique for requirement elicitation. A fundamental feature of UML, notation for describing object-oriented system models. Identifies a type of interaction and the actors involved. Sequence diagrams are used to add information to a Use case

Article printing use-case Article printing LIBSYS use cases Article printing Article search
User administration Supplier Catalogue services Library

User Library Staff

REQUIREMENTS VALIDATION

Concerned with showing that the requirements define the system that the customer wants. Important because errors in requirements can lead to extensive rework cost

Validation checks

1. Validity checks --Verification that the system performs the intended function by the user
2. Consistencycheck --Requirements should not conflict
3. Completeness checks --Includes requirements which define all functions and constraints intended bythe system user
4. Realism checks --Ensures that the requirements can be actually implemented
5. Verifiability -- Testable to avoid disputes between customer and developer.

VALIDATION TECHNIQUES

1. REQUIREMENTS REVIEWS

Reviewers check the following:

(a) Verifiability: Testable
(b) Comprehensibility
(c) Traceability
(d) Adaptability

2. PROTOTYPING

3. TEST-CASE GENERATIONRequirements management

Requirements are likely to change for large software systems and as such requirements management process is required to handle changes.

Reasons for requirements changes

(a) Diverse Users community where users have different requirements and priorities
(b) System customers and end users are different
(c) Change in the business and technical environment after installation Two classes of requirements
(a) **Enduring requirements:** Relatively stable requirements
(b) **Volatile requirements:** Likely to change during system development process or during operation

Requirements management planning

An essential first stage in requirement management process. Planning process consists of the following

1. Requirements identification -- Each requirement must have unique tag for cross reference and traceability
2. Change management process -- Set of activities that assess the impact and cost of changes

3.Traceability policy -- A matrix showing links between requirements and other elements of software development

4. CASE tool support --Automatic tool to improve efficiency of change management process. Automatedtools are required for requirements storage, change management and traceability management

Traceability

Maintains three types of traceability information.

1. Source traceability--Links the requirements to the stakeholders
2. Requirements traceability--Links dependent requirements within the requirements document
3. Design traceability-- Links from the requirements to the design module

Req. id	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D			D		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

A traceability matrix Requirements change managementconsists of three principal stages:

1. Problem analysis and change specification-- Process starts with a specific change proposal and analysed to verify that it is valid

2. Change analysis and costing--Impact analysis in terms of cost, time and risks
3. Change implementation--Carrying out the changes in requirements document, system design and itsimplementation

SYSTEM MODELS

Used in analysis process to develop understanding of the existing system or new system. Excludesdetails. An abstraction of the system
Types of system models
1.Context models
2. Behavioural models
3.Data models
4.Object models
5.Structured models

CONTEXT MODELS

A type of architectural model. Consists of sub-systems that make up an entire system
First step: Toidentify the subsystem.
Represent the high level architectural model as simple block diagram

- Depict each sub system a named rectangle
- Lines between rectangles indicate associations between subsystems Disadvantages

--Concerned with system environment only, doesn't take into account other systems, which may takedata or give data to the model

The context of an ATM system consists of the following Auto-teller systemSecurity system Maintenance system Account data base Usage database Branch accounting system Branch counter system

Behavioral models

Describes the overall behaviour of a system. Two types of behavioural model
1.Data Flow models
2.State machine models

Data flow models --Concentrate on the flow of data and functional transformation on that data. Show the processing of data and its flow through a sequence of processing steps. Help analyst understand what is going on

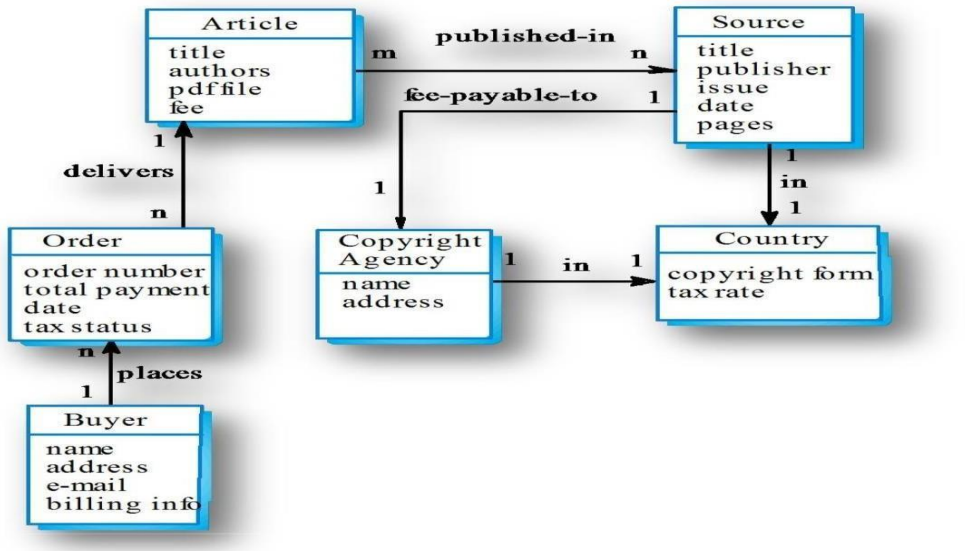
Advantages
-- Simple and easily understandable
-- Useful during analysis of requirements

State machine models

Describe how a system responds to internal or external events. Shows system states and events that cause transition from one state to another. Does not show the flow of data within the system. Used formodeling of real time systems
Exp: Microwave oven
Assumes that at any time, the system is in one of a number of possible states. Stimulus triggers atransition from on state to another state
Disadvantage
-- Number of possible states increases rapidly for large system models

DATA MODELS

Used to describe the logical structure of data processed by the system. An entity-relation- attribute model sets out the entities in the system, the relationships between these entities and the entity attributes.Widely used in database design. Can readily be implemented using relational databases. No specific notation provided in the UML but objects and associations can be used.



OBJECT MODELS

An object oriented approach is commonly used for interactive systems development. Expresses the systems requirements using objects and developing the system in an object oriented PL such as c++ A object class: An abstraction over a set of objects that identifies common attributes. Objects are instances of object class. Many objects may be created from a single class.

Analysis process

-- Identifies objects and object classes Object class in UML

--Represented as a vertically oriented rectangle with three sections

(a) The name of the object class in the top section

(b) The class attributes in the middle section

(c) The operations associated with the object class are in lower section.

OBJECT MODELS INHERITANCE MODELS

A type of object oriented model which involves in object classes attributes. Arranges classes into an inheritance hierarchy with the most general object class at the top of hierarchy Specialized objects inherit their attributes and services

UML notation

-- Inheritance is shown upward rather than downward

--Single Inheritance: Every object class inherits its attributes and operations from a single parent class

--Multiple Inheritance: A class of several of several parents.

Name	Description	Type	Date
Article	Details of the published article that may be ordered by people using LIBSYS.	Entity	30.12.2002
authors	The names of the authors of the article who may be due a share of the fee.	Attribute	30.12.2002
Buyer	The person or organisation that orders a co py of the article.	Entity	30.12.2002
fee-payable-to	A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee.	Relation	29.12.2002
Address (Buyer)	The address of the buyer. This is used to any paper billing information that is required.	Attribute	31.12.2002

UML Diagrams

The UML is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML gives you a standard way to write a system's blueprints, covering conceptual things, such as business processes and system functions, as well as concrete things, such as classes written in a specific programming language, database schemas, and reusable software components.

Model

A model is a simplification of reality. A model provides the blueprints of a system. A model may be structural, emphasizing the organization of the system, or it may be behavioral, emphasizing the dynamics of the system.

Why do we model

We build models so that we can better understand the system we are developing.

Through modeling, we achieve four aims.

1. Models help us to visualize a system as it is or as we want it to be.
2. Models permit us to specify the structure or behavior of a system.
3. Models give us a template that guides us in constructing a system.
4. Models document the decisions we have made.

We build models of complex systems because we cannot comprehend such a system in its entirety.

Principles of Modeling

There are four basic principles of model

1. The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.
2. Every model may be expressed at different levels of precision.
3. The best models are connected to reality.
4. No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.

An Overview of UML

- The Unified Modeling Language is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system.
- The UML is appropriate for modeling systems ranging from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems. It is a very expressive language, addressing all the views needed to develop and then deploy such systems.

The UML is a language for

- Visualizing
- Specifying
- Constructing
- Documenting

- **Visualizing** The UML is more than just a bunch of graphical symbols. Rather, behind each symbol in the UML notation is a well-defined semantics. In this manner, one developer can write a model in the UML, and another developer, or even another tool, can interpret that model unambiguously
- **Specifying** means building models that are precise, unambiguous, and complete.

- **Constructing** the UML is not a visual programming language, but its models can be directly connected to a variety of programming languages
- **Documenting** a healthy software organization produces all sorts of artifacts in addition to raw executable code. These artifacts include
  - Requirements
  - Architecture
  - Design
  - Source code
  - Project plans
  - Tests
  - Prototypes
  - Releases

To understand the UML, you need to form a **conceptual model of the language**, and this requires learning three major elements:

1. Things
2. Relationships
3. Diagrams

Things in the UML

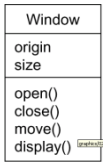
There are four kinds of things in the UML:

Structural things
Behavioral things
Grouping things
Annotational things

Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. In all, there are seven kinds of structural things.

1. Classes
2. Interfaces
3. Collaborations
4. Use cases
5. Active classes
6. Components
7. Nodes

Class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations.



Interface

Interface is a collection of operations that specify a service of a class or component.
An interface therefore describes the externally visible behavior of that element.
An interface might represent the complete behavior of a class or component or only a part of that behavior.

An interface is rendered as a circle together with its name. An interface rarely stands alone. Rather, it is typically attached to the class or component that realizes the interface

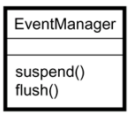
Collaboration defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements. Therefore, collaborations have structural, as well as behavioral, dimensions. A given class might participate in several collaborations.

Graphically, a collaboration is rendered as an ellipse with dashed lines, usually including only its name

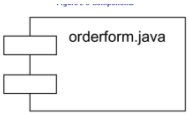
Usecase

- Use case is a description of set of sequence of actions that a system performs that yields an observable result of value to a particular actor
- Use case is used to structure the behavioral things in a model.
- A use case is realized by a collaboration. Graphically, a use case is rendered as an ellipse with solid lines, usually including only its name

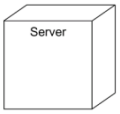
Active class is just like a class except that its objects represent elements whose behavior is concurrent with other elements. Graphically, an active class is rendered just like a class, but with heavy lines, usually including its name, attribu erations



Component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. Graphically, a component is rendered as a rectangle with tabs



Node is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability. Graphically, a node is rendered as a cube, usually including only its na



Behavioral Things are the dynamic parts of UML models. These are the verbs of a model, representing behavior over time and space. In all, there are two primary kinds of behavioral things

Interaction
state machine

Interaction

Interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose

An interaction involves a number of other elements, including messages, action sequences and links

Graphically a message is rendered as a directed line, almost always including the name of its operation

State Machine

State machine is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events

State machine involves a number of other elements, including states, transitions, events and activities

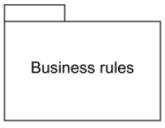
Graphically, a state is rendered as a rounded rectangle, usually including its name and its substates

Grouping Things:-

1. are the organizational parts of UML models. These are the boxes into which a model can be decomposed
2. There is one primary kind of grouping thing, namely, packages.

Package:-

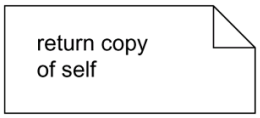
- A package is a general-purpose mechanism for organizing elements into groups. Structural things, behavioral things, and even other grouping things may be placed in a package
- Graphically, a package is rendered as a tabbed folder, usually including only its name and, sometimes, its contents



Annotational things are the explanatory parts of UML models. These are the comments you may apply to describe about any element in a model.

A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.

Graphically, a note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment



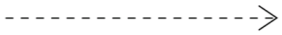
Relationships in the UML: There are four kinds of relationships in the UML:

1. Dependency
2. Association
3. Generalization
4. Realization

Dependency:-

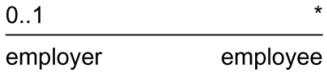
Dependency is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing

Graphically a dependency is rendered as a dashed line, possibly directed, and occasionally including a label



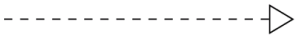
Association is a structural relationship that describes a set of links, a link being a connection among objects.

Graphically an association is rendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and role names



Aggregation is a special kind of association, representing a structural relationship between a whole and its parts. Graphically, a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent

Realization is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. Graphically a realization relationship is rendered as a cross between a generalization and a dependency relationship



Diagrams in the UML

- **Diagram** is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships).
- In theory, a diagram may contain any combination of things and relationships.
- For this reason, the UML includes nine such diagrams:
    - Class diagram
    - Object diagram
    - Use case diagram
    - Sequence diagram
    - Collaboration diagram
    - Statechart diagram
    - Activity diagram
    - Component diagram
    - Deployment diagram

Class diagram

A class diagram shows a set of classes, interfaces, and collaborations and their relationships.

Class diagrams that include active classes address the static process view of a system.

Object diagram

- Object diagrams represent static snapshots of instances of the things found in class diagrams
- These diagrams address the static design view or static process view of a system
- An object diagram shows a set of objects and their relationships

Use case diagram

- A use case diagram shows a set of use cases and actors and their relationships
- Use case diagrams address the static use case view of a system.
- These diagrams are especially important in organizing and modeling the behaviors of a system.

Interaction Diagrams

Both sequence diagrams and collaboration diagrams are kinds of interaction diagrams

Interaction diagrams address the dynamic view of a system

A sequence diagram is an interaction diagram that emphasizes the time-ordering of messages

A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages

Sequence diagrams and collaboration diagrams are isomorphic, meaning that you can take one and transform it into the other

Statechart diagram

- A statechart diagram shows a state machine, consisting of states, transitions, events, and activities
- Statechart diagrams address the dynamic view of a system
- They are especially important in modeling the behavior of an interface, class, or collaboration and emphasize the event-ordered behavior of an object

Activity diagram

An activity diagram is a special kind of a statechart diagram that shows the flow from activity to activity within a system

Activity diagrams address the dynamic view of a system

They are especially important in modeling the function of a system and emphasize the flow of control among objects

Component diagram

- A component diagram shows the organizations and dependencies among a set of components.
- Component diagrams address the static implementation view of a system
- They are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations

Deployment diagram

- A deployment diagram shows the configuration of run-time processing nodes and the components that live on them
- Deployment diagrams address the static deployment view of an architecture

Rules of the UML

The UML has semantic rules for

1. Names — What you can call things, relationships, and diagrams
2. Scope — The context that gives specific meaning to a name
3. Visibility — How those names can be seen and used by others
4. Integrity — How things properly and consistently relate to one another
5. Execution — What it means to run or simulate a dynamic model

Structural Diagrams

- The UML's four structural diagrams exist to visualize, specify, construct, and document the static aspects of a system.
- The UML's structural diagrams are roughly organized around the major groups of things you'll find when modeling a system.
    - Class diagram : Classes, interfaces, and collaborations
    - Object diagram : Objects
    - Component diagram : Components
    - Deployment diagram : Nodes

Class Diagram

- We use class diagrams to illustrate the static design view of a system.
- Class diagrams are the most common diagram found in modeling object-oriented systems.
- A class diagram shows a set of classes, interfaces, and collaborations and their relationships.
- Class diagrams that include active classes are used to address the static process view of a system.

Object Diagram

- Object diagrams address the static design view or static process view of a system just as do class diagrams, but from the perspective of real or prototypical cases.
- An object diagram shows a set of objects and their relationships.
- You use object diagrams to illustrate data structures, the static snapshots of instances of the things found in class diagrams.

Component Diagram

- We use component diagrams to illustrate the static implementation view of a system.
- A component diagram shows a set of components and their relationships.
- Component diagrams are related to class diagrams in that a component typically maps to one or more classes, interfaces, or collaborations.

Deployment Diagram

- We use deployment diagrams to illustrate the static deployment view of an architecture.
- A deployment diagram shows a set of nodes and their relationships.
- Deployment diagrams are related to component diagrams in that a node typically encloses one or more components.

Behavioral Diagrams

- The UML's five behavioral diagrams are used to visualize, specify, construct, and document the dynamic aspects of a system.
- The UML's behavioral diagrams are roughly organized around the major ways you can model the dynamics of a system.
    - Use case diagram : Organizes the behaviors of the system
    - Sequence diagram : Focused on the time ordering of messages
    - Collaboration diagram : Focused on the structural organization of objects that send and receive messages
    - Statechart diagram : Focused on the changing state of a system driven by events
    - Activity diagram : Focused on the flow of control from activity to activity

Use Case Diagram

- A use case diagram shows a set of use cases and actors and their relationships.
- We apply use case diagrams to illustrate the static use case view of a system.
- Use case diagrams are especially important in organizing and modeling the behaviors of a system.

Sequence Diagram

- We use sequence diagrams to illustrate the dynamic view of a system.
- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.
- A sequence diagram shows a set of objects and the messages sent and received by those objects.
- The objects are typically named or anonymous instances of classes, but may also represent instances of other things, such as collaborations, components, and nodes.

Collaboration Diagram

- We use collaboration diagrams to illustrate the dynamic view of a system.
- A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages.
- A collaboration diagram shows a set of objects, links among those objects, and messages sent and received by those objects.
- The objects are typically named or anonymous instances of classes, but may also represent instances of other things, such as collaborations, components, and nodes.

***** Sequence and collaboration diagrams are isomorphic, meaning that you can convert from one to the other without loss of information.

Statechart Diagram

We use statechart diagrams to illustrate the dynamic view of a system.
They are especially important in modeling the behavior of an interface, class, or collaboration.
A statechart diagram shows a state machine, consisting of states, transitions, events, and activities.
Statechart diagrams emphasize the event-ordered behavior of an object, which is especially useful in modeling reactive systems.

Activity Diagram

We use activity diagrams to illustrate the dynamic view of a system.
Activity diagrams are especially important in modeling the function of a system.
Activity diagrams emphasize the flow of control among objects.
An activity diagram shows the flow from activity to activity within a system.
An activity shows a set of activities, the sequential or branching flow from activity to activity, and objects that act and are acted upon.

Class Diagrams

- A class diagram shows a set of classes, interfaces, and collaborations and their relationships.
- Graphically, a class diagram is a collection of vertices and arcs.

Common Properties

Contents

- Class diagrams commonly contain the following things:
  - Classes
  - Interfaces
  - Collaborations
  - Dependency, generalization, and association relationships

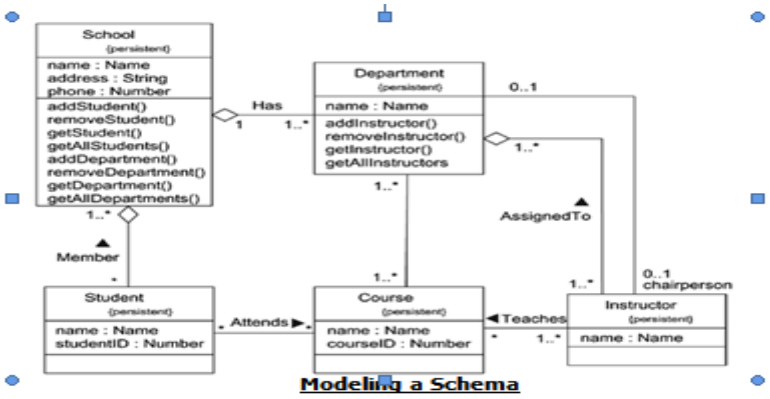
- Like all other diagrams, class diagrams may contain notes and constraints
- Class diagrams may also contain packages or subsystems

Note: Component diagrams and deployment diagrams are similar to class diagrams, except that

instead of containing classes, they contain components and nodes

Common Uses

- You use class diagrams to model the static design view of a system. This view primarily supports the functional requirements of a system
  - .



Modeling a Schema

Object Diagram

- An object diagram is a diagram that shows a set of objects and their relationships at a point in time.
- Graphically, an object diagram is a collection of vertices and arcs
- An object diagram is a special kind of diagram and shares the same common properties as all other diagrams—that is, a name and graphical contents that are a projection into a model

Contents

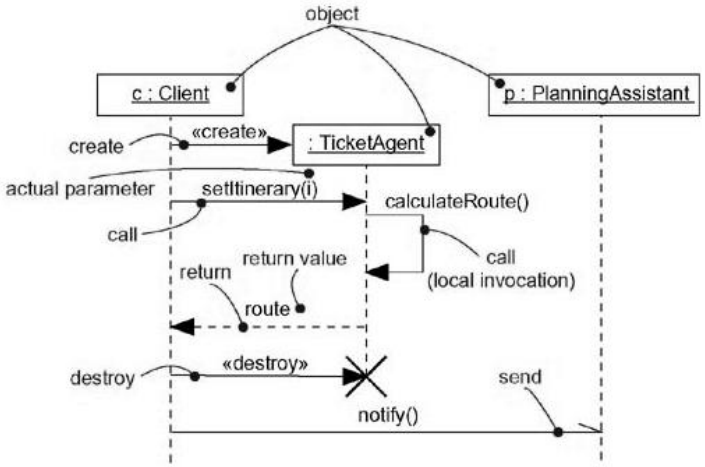
- Object diagrams commonly contain
  - Objects
  - Links
- Like all other diagrams, object diagrams may contain notes and constraints.
- Object diagrams may also contain packages or subsystems
-

Interactions

- An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose.
- A message is a specification of a communication between objects that conveys information with the expectation that activity will ensue.

Context

- We can use interactions to visualize, specify, construct, and document the semantics of a class
- We may find an interaction wherever objects are linked to one another.
- We'll find interactions in the collaboration of objects that exist in the context of your system or subsystem.
- We will also find interactions in the context of an operation.
- We might create interactions that show how the attributes of that class collaborate with one another
- Finally, you'll find interactions in the context of a class.

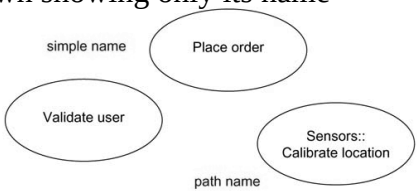


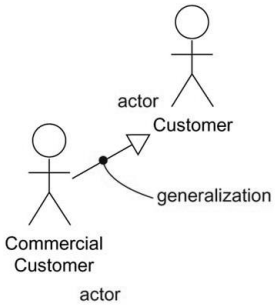
Use Cases

- A use case is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor.
- Graphically, a use case is rendered as an ellipse.

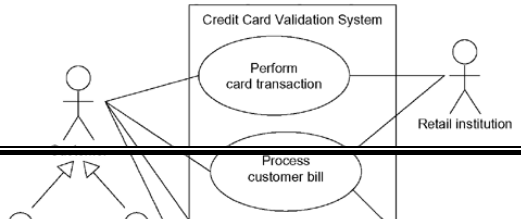
Names

- Every use case must have a name that distinguishes it from other use cases. A name is a textual string.
- That name alone is known as **a simple name**; **a path name** is the use case name prefixed by the name of the package in which that use case lives.
- A use case is typically drawn showing only its name





Actors

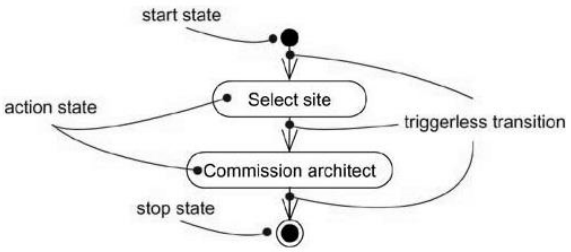


Activity Diagrams

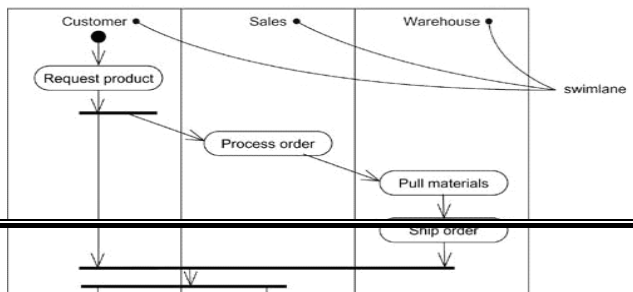
- An activity diagram shows the flow from activity to activity. An is an ongoing nonatomic execution within a state machine.
- Activities ultimately result in some action, which is made up of executable atomic computations that result in a change in state of the system or the return of a value.
- Actions encompass calling another operation, sending a signal, creating or destroying an object, or some pure computation, such as evaluating an expression.
- Graphically, an activity diagram is a collection of vertices and arcs.

Contents

- Activity diagrams commonly contain
  - Activity states and action states
  - Transitions
  - Objects
- Like all other diagrams, activity diagrams may contain notes and constraints.



Triggerless Transitions



State Machines

- **A state machine** is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events
- **A state** is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.
- **An event** is the specification of a significant occurrence that has a location in time and space.
- In the context of state machines, an event is an occurrence of a stimulus that can trigger a state transition
- **A transition** is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied.
- **An activity** is ongoing nonatomic execution within a state machine
- **An action** is an executable atomic computation that results in a change in state of the model or the return of a value

Context

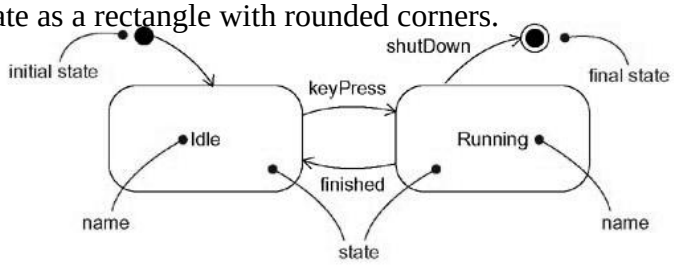
- Every object has a lifetime. On creation, an object is born; on destruction, an object ceases to exist.
- In between, an object may act on other objects (by sending them messages), as well as be acted on (by being the target of a message).
- In other kinds of systems, you'll encounter objects that must respond to signals, which are asynchronous stimuli communicated between instances.
- The behavior of objects that must respond to asynchronous stimulus or whose current behavior depends on their past is best specified by using a state machine
- This encompasses instances of classes that can receive signals, including many active objects. In fact, an object that receives a signal but has no state machine will simply ignore that signal.
- We'll also use state machines to model the behavior of entire systems, especially reactive systems, which must respond to signals from actors outside the system.

States

- A state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.
- An object remains in a state for a finite amount of time.
- When an object's state machine is in a given state, the object is said to be in that state. A state has several parts.

1. Name	A textual string that distinguishes the state from other states; a state may be anonymous, meaning that it has no name
2. Entry/exit actions	Actions executed on entering and exiting the state, respectively
3. Internal transitions	Transitions that are handled without causing a change in state
4. Substates	The nested structure of a state, involving disjoint (sequentially active) or concurrent (concurrently active) substates
5. Deferred events	A list of events that are not handled in that state but, rather, are postponed and queued for handling by the object in another state

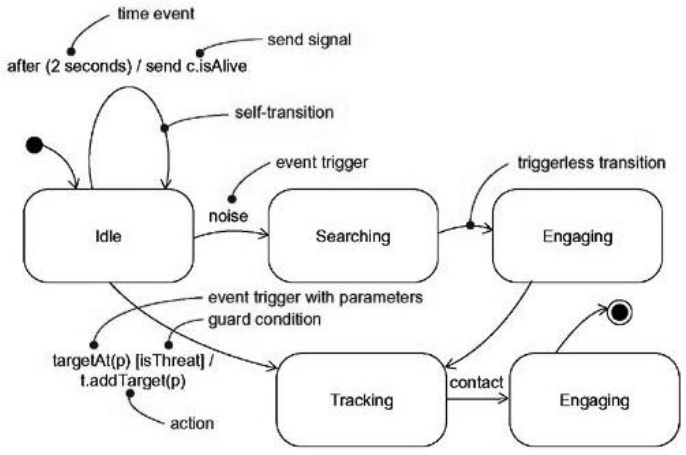
We represent a state as a rectangle with rounded corners.



States

Initial and Final States

- There are two special states that may be defined for an object's state machine.
- **initial state**, which indicates the default starting place for the state machine or substate.
- An initial state is represented as a filled black circle
- **final state**, which indicates that the execution of the state machine or the enclosing state has been completed.
- A final state is represented as a filled black circle surrounded by an unfilled circle.
- .



UNIT III

DESIGN PROCESS AND DESIGN QUALITY

Encompasses the set of principles, concepts and practices that lead to the development of high quality system or product. Design creates a representation or model of the software. Design model provides details about S/W architecture, interfaces and components that are necessary to implement the system. Quality is established during Design. Design should exhibit firmness, commodity and design. Design sits at the kernel of S/W Engineering. Design sets the stage for construction.

QUALITY GUIDELINES

• Uses recognizable architectural styles or patterns
• Modular; that is logically partitioned into elements or subsystems
• Distinct representation of data, architecture, interfaces and components
• Appropriate data structures for the classes to be implemented
• Independent functional characteristics for components
• Interfaces that reduces complexity of connection
• Repeatable method

QUALITY ATTRIBUTES

FURPS quality attributes

• Functionality
* Feature set and capabilities of programs
* Security of the overall system
• Usability
* user-friendliness
* Aesthetics
* Consistency
* Documentation
• Reliability
* Evaluated by measuring the frequency and severity of failure
* MTTF
• Supportability
* Extensibility
* Adaptability
* Serviceability

DESIGN CONCEPTS

1. Abstractions
2. Architecture
3. Patterns
4. Modularity
5. Information Hiding
6. Functional Independence
7. Refinement
8. Re-factoring
9. Design Classes

DESIGN CONCEPTS

ABSTRACTION

Many levels of abstraction.

Highest level of abstraction: Solution is slated in broad terms using the language of the problem environment Lower levels of abstraction: More detailed description of the solution is provided

• Procedural abstraction-- Refers to a sequence of instructions that a specific and limited function
• Data abstraction-- Named collection of data that describe a data object

DESIGN CONCEPTS

ARCHITECTURE--Structure organization of program components (modules) and their interconnection Architecture Models

(a) Structural Models-- An organized collection of program components
(b) Framework Models-- Represents the design in more abstract way

(c) Dynamic Models-- Represents the behavioral aspects indicating changes as a function of external events

(d). Process Models-- Focus on the design of the business or technical process

PATTERNS

Provides a description to enables a designer to determine the followings:

(a). whether the pattern isapplicable to the current work

(b). Whether the pattern can be reused

(c). Whether the pattern can serve as a guide for developing a similar but functionally or structurally different pattern

MODULARITY

Divides software into separately named and addressable components, sometimes called modules. Modules are integrated to satisfy problem requirements. Consider two problems p1 and p2. If the complexity of p1 iscp1 and of p2 is cp2 then effort to solve p1=cp1 and effort to solve p2=cp2 If cp1>cp2 then ep1>ep2

The complexity of two problems when they are combined is often greater than the sum of the perceived complexity when each is taken separately.

• Based on Divide and Conquer strategy : it is easier to solve a complex problem when broken into sub-modules

INFORMATION HIDING

Information contained within a module is inaccessible to other modules who do not need such information. Achieved by defining a set of Independent modules that communicate with one another only that information necessary to achieve S/W function. Provides the greatest benefits when modifications are required during testing and later. Errors introduced during modification are less likely to propagate to other location within the S/W.

FUNCTIONAL INDEPENDENCE

A direct outgrowth of Modularity. abstraction and information hiding. Achieved by developing a module with single minded function and an aversion to excessive interaction with other modules. Easier to develop and have simple interface. Easier to maintain because secondary effects caused b design or code modification are limited, error propagation is reduced and reusable modules are possible. Independence is assessed by two quantitative criteria:

(1) Cohesion

(2) Coupling

Cohesion -- Performs a single task requiring little interaction with other components Coupling--Measure of interconnection among modules. Coupling should be low and cohesion should be high for good design.

REFINEMENT & REFACTORING

REFINEMENT -- Process of elaboration from high level abstraction to the lowest level abstraction. High level abstraction begins with a statement of functions. Refinement causes the designer to elaborate providing more and more details at successive level of abstractions Abstraction and refinement are complementary concepts.

Refactoring -- Organization technique that simplifies the design of a component without changing its function or behavior. Examines for redundancy, unused design elements and inefficient or unnecessary algorithms.

DESIGN CLASSES

Class represents a different layer of design architecture. Five types of Design Classes

1. User interface class -- Defines all abstractions that are necessary for human computer interaction 2. Business domain class -- Refinement of the analysis classes that identity attributes and services to implement some of business domain

3. Process class -- implements lower level business abstractions required to fully manage the business domain classes

4. Persistent class -- Represent data stores that will persist beyond the execution of the software

5. System class -- Implements management and control functions to operate and communicate within the computer environment and with the outside world.

THE DESIGN MODEL

Analysis viewed in two different dimensions as process dimension and abstract dimension. Process dimension indicates the evolution of the design model as design tasks are executed as part of software process. Abstraction dimension represents the level of details as each element of the analysis model is transformed into design equivalent

Data Design elements

-- Data design creates a model of data that is represented at a high level of abstraction

-- Refined progressively to more implementation-specific representation for processing by the computer base system

-- Translation of data model into a data base is pivotal to achieving business objective of a system

THE DESIGN MODEL

Architectural design elements. Derived from three sources

(1) Information about the application domain of the software

(2) Analysis model such as dataflow diagrams or analysis classes.

(3) Architectural pattern and styles Interface Design elements Set of detailed drawings constituting: (1) User interface

(2) External interfaces to other systems, devices etc

(3) Internal interfaces between various components

THE DESIGN MODEL

Deployment level design elements. Indicates how software functionality and subsystem will be allocated within the physical computing environment. UML deployment diagram is developed and refined Component level design elements Fully describe the internal details of each software component. UML diagram can be used

CREATING AN ARCHITECTURAL DESIGN

What is SOFTWARE ARCHITECTURE… The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationship among them.

Software Architecture is not the operational software. It is a representation that enables a software engineer to

• Analyze the effectiveness of the design in meeting its stated requirements.

• • consider architectural alternative at a stage when making design changes is still relatively easy

. • Reduces the risk associated with the construction of the software.

Why Is Architecture Important? Three key reasons

--Representations of software architecture enable communication and understanding between stakeholders

--Highlights early design decisions to create an operational entity.

--constitutes a model of software components and their interconnection

Data Design

The data design action translates data objects defined as part of the analysis model into data structures at the component level and database architecture at application level when necessary.

DATA DESIGN AT ARCHITECTURE LEVEL

• Data structure at programming level

• Data base at application level

• Data warehouse at business level.

DATA DESIGN AT COMPONENT LEVEL

Principles for data specification:

1. Proper selection of data objects and data and data models

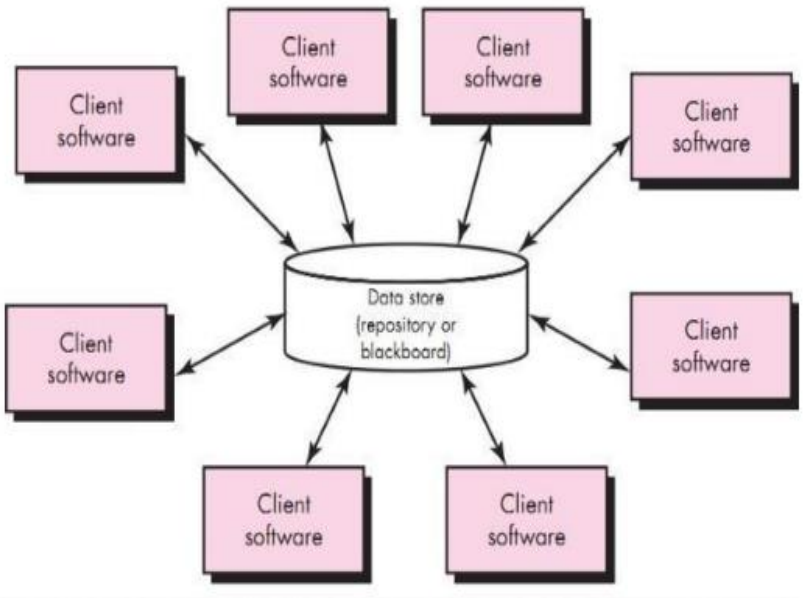
2. Identification of attribute and functions and their encapsulation of these within a class 3.Mechanism for representation of the content of each data object. Class diagrams may be used 4. Refinement of data design elements from requirement analysis to component level design. 5.Information hiding

6. A library of useful data structures and operations be developed.

7. Software design and PL should support the specification and realization of abstract data types.

ARCHITECTURAL STYLES

Describes a system category that encompasses:
(1) a set of components
(2) a set of connectors that enables "communication and coordination
(3) Constraints that define how components can be integrated to form the system
(4) Semantic models to understand the overall properties of a system



Data-flow architectures
Shows the flow of input data, its computational components and output data. Structure is also called pipe and Filter. Pipe provides path for flow of data. Filters manipulate data and work independent of its neighboring filter. If data flow degenerates into a single line of transform, it is termed as batch sequential.
Call and return architectures
Achieves a structure that is easy to modify and scale .Two sub styles
(1) Main program/sub program architecture
-- Classic program structure
- Main program invokes a number of components, which in turn invoke still other components
(2) Remote procedure call architecture
-- Components of main program/subprogram are distributed across computers over network
Object-oriented architectures
The components of a system encapsulate data and the operations. Communication and coordination between components is done via message
Layered architectures
A number of different layers are defined Inner Layer(interface with OS)
• Intermediate Layer Utility services and application function) Outer Layer (User interface)

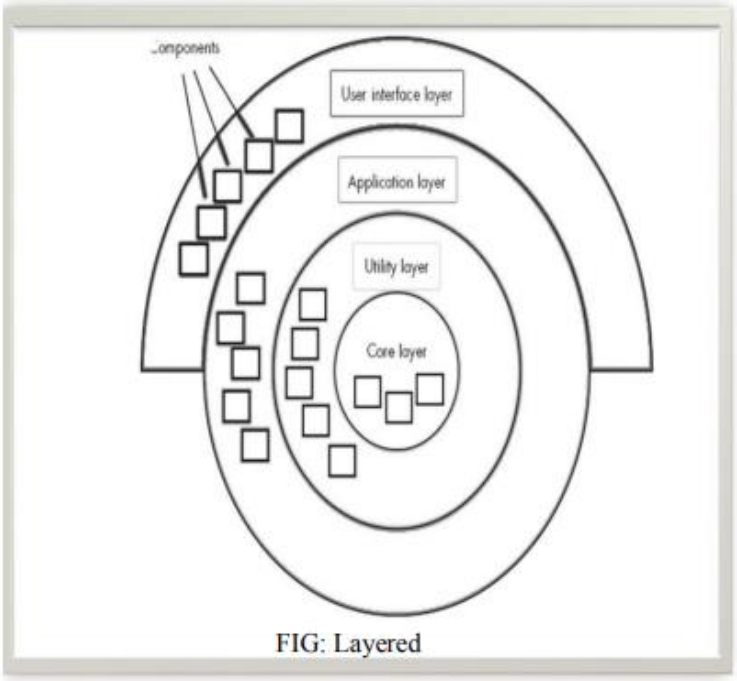


FIG: Layered

ARCHITECTURAL PATTERNS

A template that specifies approach for some behavioral characteristics of the system Patterns are imposed on the architectural styles

Pattern Domains

1.Concurrency

--Handles multiple tasks that simulate parallelism.

--Approaches (Patterns)

(a) Operating system process management pattern

(b) A task scheduler pattern

2.Persistence

--Data survives past the execution of the process

--Approaches (Patterns)

(a) Data base management system pattern

(b) Application Level persistence Pattern(word processing software)

3. Distribution

-- Addresses the communication of system in a distributed environment

--Approaches (Patterns)

(a) Broker Pattern

-- Acts as middleman between client and server.

Object-Oriented Design: Objects and object classes, An Object-Oriented design process, Design evolution.

• Performing User interface design: Golden rules, User interface analysis and design, interface analysis, interface design steps, Design evaluation.

Object and Object Classes

• Object: An object is an entity that has a state and a defined set of operations that operate on that state.

• An object class definition is both a type specification and a template for creating objects.

• It includes declaration of all the attributes and operations that are associated with object of that class.

Object Oriented Design Process

There are five stages of object oriented design process

1) Understand and define the context and the modes of use of the system.

2) Design the system architecture

3) Identify the principle objects in the system.

4) Develop a design models

5)Specify the object interfaces

Systems context and modes of use. It specifies the context of the system. it also specify the relationships between the software that is being designed and its external environment.

• If the system context is a static model it describes the other system in that environment.

• If the system context is a dynamic model then it describes how the system actually interact with the environment.

System Architecture

Once the interaction between the software system that being designed and the system environment have been defined. We can use the above information as basis for designing the System Architecture

Object Identification--This process is actually concerned with identifying the object classes. We can identify the object classes by the following

1) Use a grammatical analysis
2) Use a tangible entities
3) Use a behavioral approach
4) Use a scenario based approach

Design model

Design models are the bridge between the requirements and implementation. There are two type of design models

1) Static model describe the relationship between the objects.
2) Dynamic model describe the interaction between the objects

Object Interface Specification

It is concerned with specifying the details of the interfaces to objects. Design evolution. The main advantage OOD approach is to simplify the problem of making changes to the design. Changing the internal details of an object is unlikely to effect any other system object.

Golden Rules

1. Place the user in control
2. Reduce the user's memory load
3. Make the interface consistent

Place the User in Control

• Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
• Provide for flexible interaction.
• Allow user interaction to be interruptible and undoable.
• Streamline interaction as skill levels advance and allow the interaction to be customized.
• Hide technical internals from the casual user.
• Design for direct interaction with objects that appear on the screen.

Make the Interface Consistent. Allow the user to put the current task into a meaningful context. Maintain consistency across a family of applications. If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

USER INTERFACE ANALYSISAND DESIGN

The overall process for analyzing and designing a user interface begins with the creation of different models of system function. There are 4 different models that is to be considered when a user interface is to be analyzed and designed.

User Interface Design Models

User model —Establishes a profile of all end users of the system

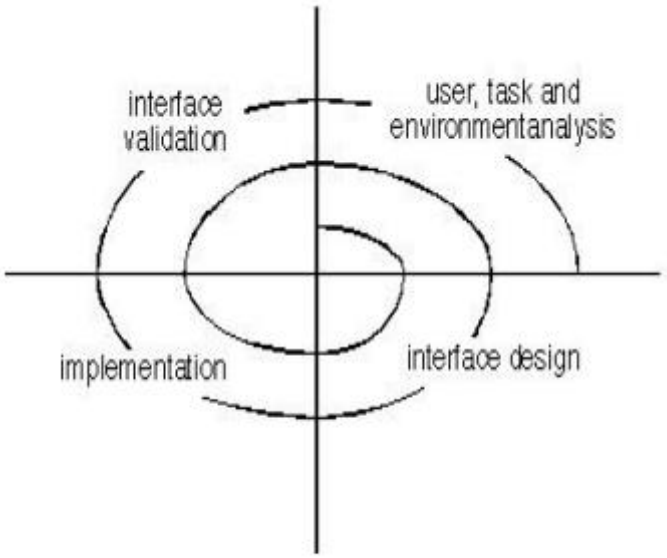
Design model — A design model of the entire system incorporates data, architectural, interface and procedural representation of the software.

A design realization of the user model User's Mental model (system perception). the user's mental image of what the interface is Implementation model — the interface "look and feel" coupled with supporting information that describe interface syntax and semantics

Users can be categorized as

1. Novice – No syntactic knowledge of the system and little semantic knowledge of the application or computer usage of the system
2. Knowledgeable, intermittent users- Reasonable semantic knowledge of the application but low recallof

syntactic information to use the system
3. Knowledgeable, frequent users- Good semantic and syntactic knowledge User interface analysis and design process
• The user interface analysis and design process is an iterative process and it can be represented as a spiral model
It consists of 5 framework activities 1.User, task and environment analysis 2.Interface design3.Interface construction 4.Interface validation



User Interface Design Process

Interface analysis
-Understanding the user who interacts with the system based on their skill levels.i.e, requirement gathering
-The task the user performs to accomplish the goals of the system are identified, described and elaborated. Analysis of work environment.
Interface design
In interface design, all interface objects and actions that enable a user to perform all desired task are defined
Implementation
A prototype is initially constructed and then later user interface development tools may be used to complete the construction of the interface.
• Validation
The correctness of the system is validated against the user requirement
Interface Analysis
Interface analysis means understanding
– (1) the people (end-users) who will interact with the system through the interface;
– (2) the tasks that end-users must perform to do their work,
– (3) the content that is presented as part of the interface
– (4) the environment in which these tasks will be conducted.
User Analysis
• Are users trained professionals, technician, clerical, o manufacturing workers?
• What level of formal education does the average user have?
• Are the users capable of learning from written materials or have they expressed a desire for classroom training?
• Are users expert typists or keyboard phobic?
• What is the age range of the user community?
• Will the users be represented predominately by one gender?

• How are users compensated for the work they perform?
• Do users work normal office hours or do they work until the job is done?

Task Analysis and Modeling

Analysis Techniques

• Use-cases define basic interaction
• Task elaboration refines interactive tasks
• Object elaboration identifies interface objects(classes)
• Workflow analysis defines how a work process is completed when several people (and roles) are involved
– What work will the user perform in specific circumstances?

Interface Design Steps

• Using information developed during interface analysis define interface objects and actions (operations).
• Define events (user actions) that will cause the state of the user interface to change. Model this behavior.
• Depict each interface state as it will actually look to the end-user.
• Indicate how the user interprets the state of the system from information provided through the interface.

Interface Design Patterns. Patterns are available for

– The complete UI
– Page layout
– Forms and input
– Tables
– Direct data manipulation
– Navigation
– Searching
– Page elements
– e-Commerce

Design Issues

• Response time
• Help facilities
• Error handling
• Menu and command labeling
• Application accessibility
• Internationalization

Design Evaluation Cycle: Steps:

Preliminary design Build prototype #1
Interface evaluation is studied by designer Design modifications are made
Build prototype # n
Interface
User evaluate's interface Interface design is complete

Golden Rules

1. Place the user in control
2. Reduce the user's memory load
3. Make the interface consistent

Place the User in Control

• Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
• Provide for flexible interaction.
• Allow user interaction to be interruptible and undoable.
• Streamline interaction as skill levels advance and allow the interaction to be customized.
• Hide technical internals from the casual user.
• Design for direct interaction with objects that appear on the screen.

Make the Interface Consistent. Allow the user to put the current task into a meaningful context. Maintain consistency across a family of applications. If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

USER INTERFACE ANALYSISAND DESIGN

The overall process for analyzing and designing a user interface begins with the creation of different models of system function. There are 4 different models that is to be considered when a user interface isto be analyzed and designed.

User Interface Design Models

User model —Establishes a profile of all end users of the system

Design model — A design model of the entire system incorporates data, architectural, interface and procedural representation of the software.

A design realization of the user model User's Mental model (system perception). the user's mental image of what the interface is Implementation model — the interface "look and feel" coupled with supporting information that describe interface syntax and semantics

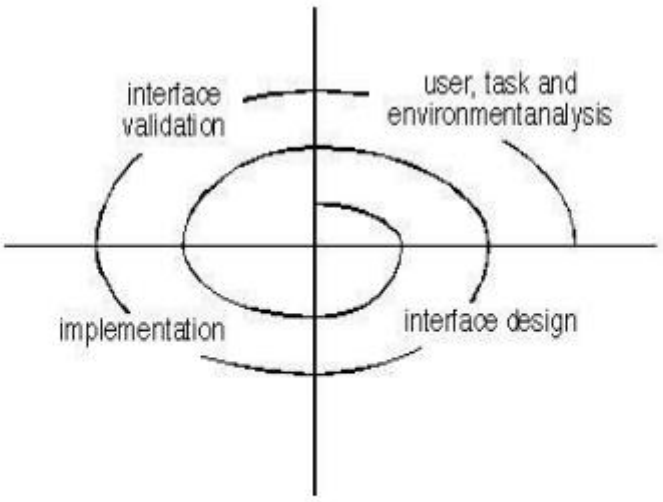
Users can be categorized as

1. Novice – No syntactic knowledge of the system and little semantic knowledge of the application orcomputer usage of the system
2. Knowledgeable, intermittent users- Reasonable semantic knowledge of the application but lowrecallof syntactic information to use the system
3. Knowledgeable, frequent users- Good semantic and syntactic knowledge User interface analysis and design process •

The user interface analysis and design

process is an iterative process and it can be represented as aspiral model It consists of 5 framework activities

1.User, task and environment analysis
2.Interface design
3.Interface construction
4.Interface validation



User Interface Design Process

Interface analysis -Understanding the user who interacts with the system based on their skill levels.i.e, requirement gathering -The task the user performs to accomplish the goals of the system are identified, described and elaborated. Analysis of work environment.

Interface design In interface design, all interface objects and actions that enable a user to perform all desired task are defined

Implementation A prototype is initially constructed and then later user interface development tools may be used to complete the construction of the interface.

Validation The correctness of the system is validated against the user requirement

Interface Analysis

Interface analysis means understanding –
(1) the people (end-users) who will interact with the system through the interface;
(2) the tasks that end-users must perform to do their work,
(3) the content that is presented as part of the interface –
(4) the environment in which these tasks will be conducted.

User Analysis

• Are users trained professionals, technician, clerical, o manufacturing workers?
• What level of formal education does the average user have?
• Are the users capable of learning from written materials or have they expressed a desire for classroom training?
• Are users expert typists or keyboard phobic?
• What is the age range of the user community?
• Will the users be represented predominately by one gender?
• How are users compensated for the work they perform?
• Do users work normal office hours or do they work until the job is done?

TASK ANALYSIS AND MODELING

Analysis Techniques

• Use-cases define basic interaction
• Task elaboration refines interactive tasks
• Object elaboration identifies interface objects(classes)
• Workflow analysis defines how a work process is completed when several people (and roles) areinvolved – What work will the user perform in specific circumstances?

Interface Design Steps

Using information developed during interface analysis define interface objects and actions
• (operations). Define events (user actions) that will cause the state of the user interface to change. Model
• thisbehavior. Depict each interface state as it will actually look to the end-user
.• Indicate how the user interprets the state of the system from information provided through theinterface.

Interface Design Patterns.

Patterns are available for
– The complete UI
– Page layout
– Forms and input
– Tables
– Direct data manipulation
– Navigation
– Searching
– Page elements
– e-Commerce

Design Issues
• Response time
• Help facilities
• Error handling
• Menu and command labeling
• Application accessibility
• Internationalization

Design Evaluation Cycle: Steps:

Preliminary design Build prototype #1
• Interface evaluation is studied by designer
• Design modifications are made
• Build prototype # n

- Interface User evaluate's interface
- Interface design is complete

UNIT– IV

Testing Strategies

Software is tested to uncover errors introduced during design and construction. Testing often accounts for More project effort than other s/e activity. Hence it has to be done carefully using a testing strategy.

The strategy is developed by the project manager, software engineers and testing specialists. Testing is the process of execution of a program with the intention of finding errors Involves 40% of total project cost

Testing Strategy provides a road map that describes the steps to be conducted as part of testing. It should incorporate test planning, test case design, test execution and resultant data collection and execution

Validation refers to a different set of activities that ensures that the software is traceable to theCustomer requirements. V&V encompasses a wide array of Software Quality Assurance

A strategic Approach for Software testing

Testing is a set of activities that can be planned in advance and conducted systematically. Testing strategy Should have the following characteristics:

-- usage of Formal Technical reviews(FTR)

-- Begins at component level and covers entire system

-- Different techniques at different points

-- conducted by developer and test group

-- should include debugging

Software testing is one element of verification and validation.

Verification refers to the set of activities that ensure that software correctly implements a specific function. (Ex: Are we building the product right?)

Validation refers to the set of activities that ensure that the software built is traceable to customer requirements. (Ex: Are we building the right product ?)

Testing Strategy

Testing can be done by software developer and independent testing group. Testing and debugging are different activities. Debugging follows testing Low level tests verifies small code segments. High level tests validate major system functions against customer requirements

Test Strategies for Conventional Software:

Testing Strategies for Conventional Software can be viewed as a spiral consisting of four levels of testing:

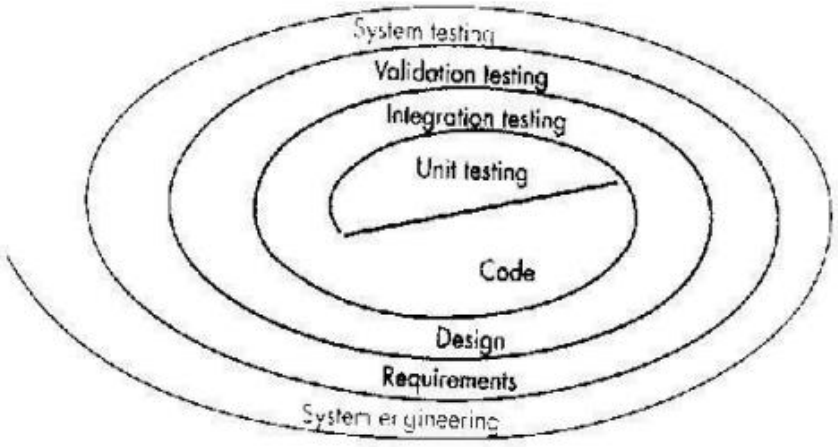
1) Unit Testing

2)Integration Testing

3)Validation Testing and

4) System Testing

Spiral Representation of Testing for Conventional Software



Unit Testing begins at the vortex of the spiral and concentrates on each unit ofsoftware in source code. It uses testing techniques that exercise specific paths in a component and its controlstructure to ensure complete coverage and maximum error detection. It focuses on the internal processing logic and data

structures. Test cases should uncover errors.

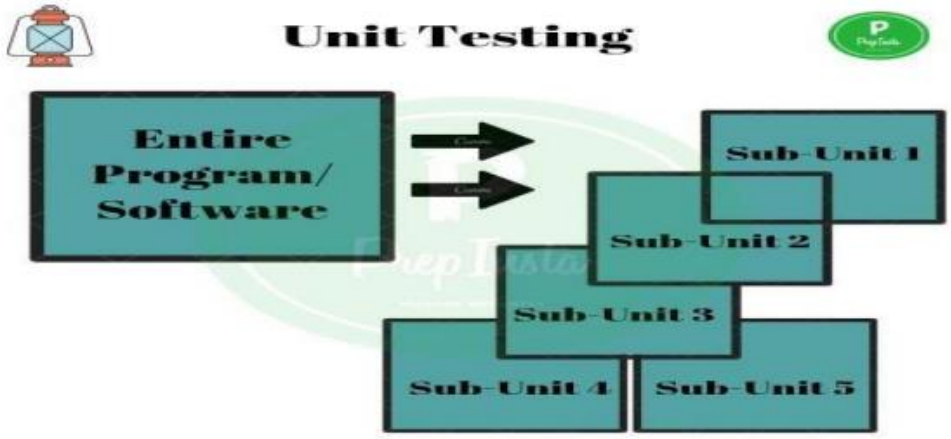


Fig: Unit Testing

Boundary testing also should be done as s/w usually fails at its boundaries. Unit tests can be designed before coding begins or after source code is generated.

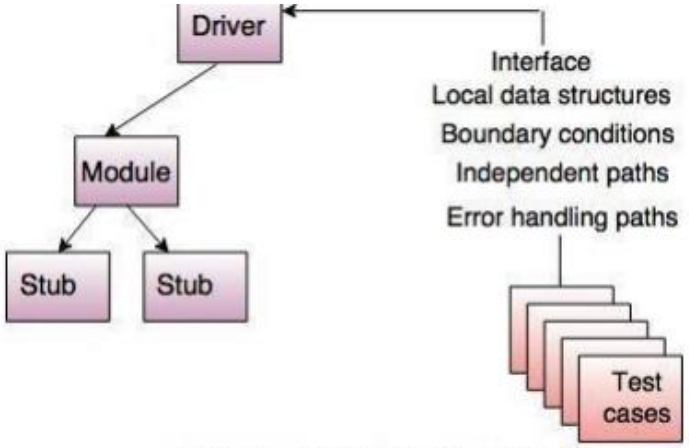


Fig. - Unit test environment

Integration testing: In this the focus is on design and construction of the software architecture. It addresses the issues associated with problems of verification and program construction by testing inputs and outputs. Though modules function independently problems may arise because of interfacing. This technique uncovers errors associated with interfacing. We can use top-down integration wherein modules are integrated by moving downward through the control hierarchy, beginning with the main control module. The other strategy is bottom –up which begins construction and testing with atomic modules which are combined into clusters as we move up the hierarchy. A combined approach called Sandwich strategy can be used i.e., topdown for higher level modules and bottom-up for lower level modules.

Validation Testing: Through Validation testing requirements are validated against s/w constructed. These are high-order tests where validation criteria must be evaluated to assure that s/w meets all functional, behavioural and performance requirements. It succeeds when the software functionsin a manner that can be reasonably expected by the customer.

1)Validation Test Criteria
2)Configuration Review
3)Alpha And Beta Testing

The validation criteria described in SRS form the basis for this testing. Here, Alpha and Beta testing is performed. Alpha testing is performed at the developers site by end users in a natural setting and with a controlled environment. Beta testing is conducted at end-user sites. It is a “live” application and environment is not controlled. End-user records all problems and reports to developer. Developer then makes modifications and releases the product.

System Testing: In system testing, s/w and other system elements are tested as a whole. This is the last high-order testing step which falls in the context of computer system engineering. Software is combined with other system elements like H/W, People, Database and the overall functioning is checked by conducting a series of tests. These tests fully exercise the computer based system.

The types of tests are:

1.Recovery testing: Systems must recover from faults and resume processing within a prespecified time. It forces the system to fail in a variety of ways and verifies that recovery is properly performed. Here the Mean Time To Repair (MTTR) is evaluated to see if it is within acceptable limits.

2.Security Testing: This verifies that protection mechanisms built into a system will protect it from improper penetrations. Tester plays the role of hacker. In reality given enough resources and time it is possible to ultimately penetrate any system. The role of system designer is to make penetration cost more than the value of the information that will be obtained.

3.Stress testing: It executes a system in a manner that demands resources in abnormal quantity, frequency or volume and tests the robustness of the system.

4.Performance Testing: This is designed to test the run-time performance of s/w within the context of an integrated system. They require both h/w and s/w instrumentation.

Testing Tactics: The goal of testing is to find errors and a good test is one that has a high probability of finding an error.

A good test is not redundant and it should be neither too simple nor too complex. Two major categories of software testing

Black box testing: It examines some fundamental aspect of a system, tests whether each function of product is fully operational.

White box testing: It examines the internal operations of a system and examines the procedural detail.

Black box testing

This is also called behavioural testing and focuses on the functional requirements of software. It fully exercises all the functional requirements for a program and finds incorrect or missing functions,interface errors, database errors etc. This is performed in the later stages in the testing process. Treatsthe system as black box whose behaviour can be determined by studying its input and related output Not concerned with the internal. The various testing methods employed here are:

1)Graph based testing method: Testing begins by creating a graph of important objects and their relationships and then devising a series of teststhat will cover the graph so that each object and relationship is exercised and errors are uncovered.

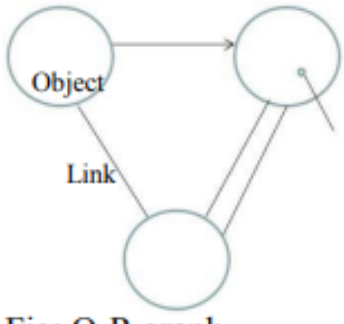


Fig: O-R graph.

2)Equivalence partitioning: This divides the input domain of a program into classes of data from which test Cases can be derived. Define test cases that uncover classes of errors so that no. of test cases are reduced.This is based on equivalence classes which represents a set of valid or invalid states for inputconditions. Reduces the cost of testing

Example

Input consists of 1 to 10 Then classes are n<1,1<=n<=10,n>10 Choose one valid class with value within the allowed range and two invalid classes where values are greater than maximum value and smaller than minimum value.

3)Boundary Value analysis

Select input from equivalence classes such that the input lies at the edge of the equivalence classes. Set of data lies on the edge or boundary of a class of input data or generates the data that lies at the boundary of a class of output data. Test cases exercise boundary values to uncover errors at the boundaries of the input domain.

Example If 0.0<=x<=1.0

Then test cases are (0.0,1.0) for valid input and (-0.1 and 1.1) for invalid input

4)Orthogonal array Testing

This method is applied to problems in which input domain is relatively small but too large for exhaustive testing

Example Three inputs A,B,C each having three values will require 27 test cases. Orthogonal testing will reduce the number of test case to 9

White Box testing

Also called glass box testing. It uses the control structure to derive test cases. It exercises all independent paths, Involves knowing the internal working of a program, Guarantees that all independent paths will be exercised at least once .Exercises all logical decisions on their true and false sides, Executes all loops,Exercises all data structures for their validity. White box testing techniques

1. Basis path testing

2.Controlstructure testing

1.Basis path testing

Proposed by Tom McCabe. Defines a basic set of execution paths based on logical complexity of a procedural design. Guarantees to execute every statement in the program at least once Steps of Basis Path Testing

1. Draw the flow graph from flow chart of the program

2.Calculate the cyclomatic complexity of the resultant flow graph

3.Prepare test cases that will force execution of each path

Two methods to compute Cyclomatic complexity number

1.V(G)=E-N+2 where E is number of edges, N is number of nodes

2.V(G)=Number of regions

The structured constructs used in the flow graph are:

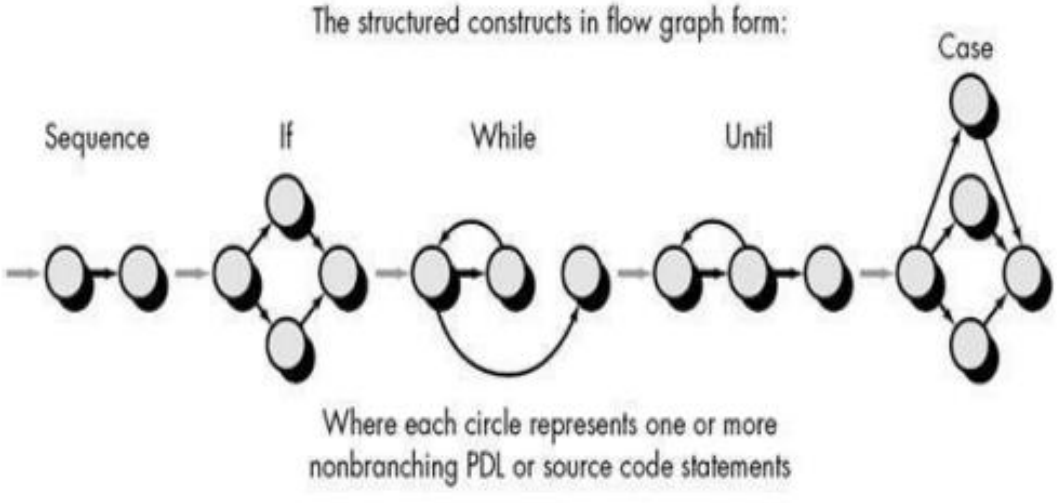


Fig: Basis path testing

Basis path testing is simple and effective It is not sufficient in itself

2.Control Structure testing

This broadens testing coverage and improves quality of testing. It uses the following methods:

a) Condition testing: Exercises the logical conditions contained in a program module. Focuses on testing each condition in the program to ensure that it does not contain errors Simple condition

E1<relation operator>E2 Compound condition

simple condition<booleon operator>simple condition

Types of errors include operator errors, variable errors, arithmetic expression errors etc.

b) Data flow Testing

This selects test paths according to the locations of definitions and use of variables in a program Aims to ensure that the definitions of variables and subsequent use is tested First construct a definition-use graph from the control flow of a program

DEF(definition):definition of a variable on the left-hand side of an assignment statement

USE: Computational use of a variable like read, write or variable on the right hand of assignment statement Every DU chain be tested at least once. c) Loop Testing

This focuses on the validity of loop constructs.

Four categories can be defined

1.Simple loops

2.Nested loops

3.Concatenated loops

4.Unstructured loops

Testing ofsimple loops

N is the maximum number of allowable passes through the loop

1.Skip the loop entirely

2.Only one pass through the loop

3.Two passes through the loop

4.m passes through the loop where m>N

5.N-1,N,N+1 passes the loop

The Art of Debugging-

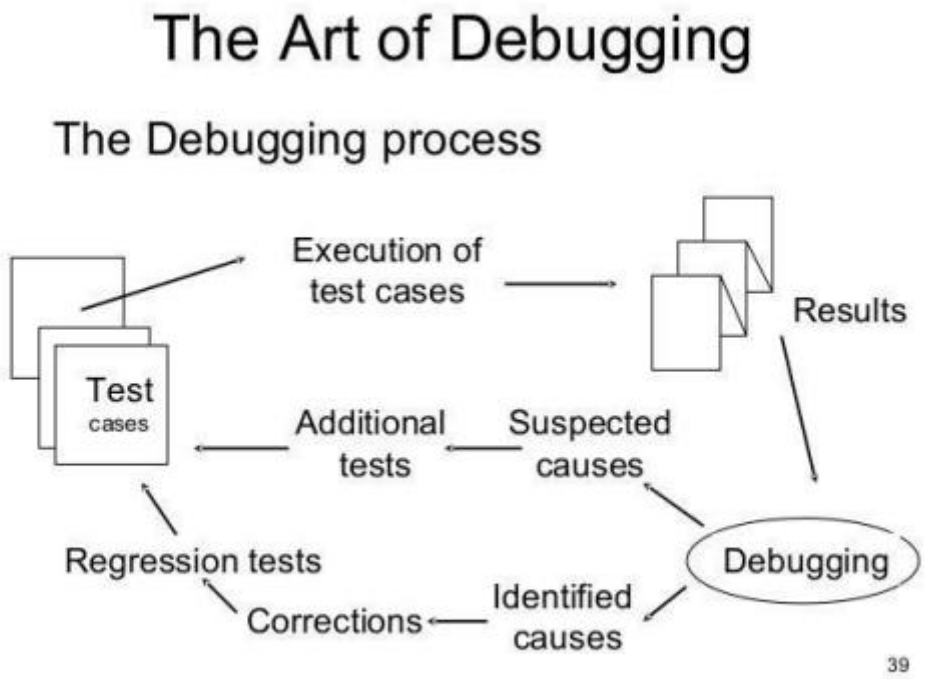


Fig: Debugging process

Debugging has two outcomes: -

cause will be found and corrected
- cause will not be found
Characteristics of bugs:
- symptom and cause can be in different locations
- Symptoms may be caused by human error or timing problems
Debugging is an innate human trait.
Some are good at it and some are not.
Debugging Strategies:
The objective of debugging is to find and correct the cause of a software error which is realized by a combination ofsystematic evaluation, intuition and luck. Three strategies are proposed:
1)Brute Force Method.
2)Back Tracking
3)Cause Elimination
Brute Force: Most common and least efficient method for isolating the cause of a s/w error. This is applied when all else fails. Memory dumps are taken, run-time traces are invoked and program is loaded with output statements. Tries to find the cause from the load of information Leads to waste of time and effort.
Back tracking: Common debugging approach. Useful for small programs Beginning at the system where the symptom has been uncovered, the source code is traced backward until the site of the cause is found. More no. of lines implies no. of paths are unmanageable.
Cause Elimination: Based on the concept of Binary partitioning. Data related to error occurenec are organized to isolate potential causes. A "cause hypothesis" is devised and data is used to prove or disprove it. A list of all possible causes is developed and tests are conducted to eliminate each Automated Debugging: This supplements the above approaches with debugging tools that provide semi-automated support like debugging compilers, dynamic debugging aids, test casegenerators, mapping tools etc.
Regression Testing: When a new module is added as part of integration testing the software changes. This may cause problems with the functions which worked properly before. This testing is the re-execution of some subset of tests that are already conducted to ensure that changes have not propagatedunintended side effects. It ensures that changes do not introduce unintended behaviour or errors. This can be done manually or automated. Software Quality Conformance to explicitly stated functional andperformance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.
Factors that affect software quality can be categorized in two broad groups:
Factors that can be directly measured (e.g. defects uncovered during testing)
Factorsthat can be measured only indirectly (e.g. usability or maintainability)
McCall's quality factors 1.Productoperation
Correctness
Reliability
Efficiency Integrity
Usability
2. Product Revision
Maintainability
Flexibility
3. Product Transition
Portability
Reusability
Interoperability

Risk Management
Risk is an undesired event or circumstance that occur while a project is underway It isnecessary for the project manager to anticipate and identify different risks that a project may besusceptible to Risk Management. It aims at reducing the impact of all kinds of risk that may effect a

projectbyidentifying,analyzingandmanagingthem.

Reactive Vs Proactive risk
Reactive : It monitors the projects likely risk and resources are set aside.
Proactive: Risk are identified, their probability and impact is accessed

Software Risk
It involve 2 characteristics
Uncertainty : Risk may or may not happen
Loss : If risk is reality unwanted loss or consequences will occur It includes
1)Project Risk 2)Technical Risk 3)Business Risk4)Known Risk5)UnpredictableRisk
6)Predictablerisk
Project risk: Threaten the project plan and affect schedule and resultant cost**Technicalrisk**: Threatenthequalityand timelinessofsoftwareto beproduced**Businessrisk:**Threatentheviabilityofsoftwaretobebuilt
Known risk: These risks can be recovered from careful evaluation**Predictablerisk:**Risksareidentifiedbypastprojectexperience
Unpredictable risk: Risksthatoccurand maybedifficulttoidentify

RiskIdentification
It concerned with identification of riskStep1: Identify all possible risksStep2:Createitemchecklist
Step3: Categorize into risk components-Performance risk, cost risk, support risk andschedule risk
Step4:Divideteriskintooneof4categoriesNegligible-0
Marginal-1Critical-2

Risk Identification: The project organizer needs to anticipate the risk in the project as early as possible so that the impact of risk can be reduced by making effective risk management planning.

A project can be of use by a large variety of risk. To identify the significant risk, this might affect a project. It is necessary to categories into the different risk of classes.

There are different types of risks which can affect a software project:

1. **Technology risks:** Risks that assume from the software or hardware technologies that are used to develop the system.
2. **People risks:** Risks that are connected with the person in the development team.
3. **Organizational risks:** Risks that assume from the organizational environment where the software is being developed.

4. **Tools risks:** Risks that assume from the software tools and other support software used to create the system.

5. **Requirement risks:** Risks that assume from the changes to the customer requirement and the process of managing the requirements change.

6. **Estimation risks:** Risks that assume from the management estimates of the resources required to build the system

RiskProjection

Alsocalledriskestimation.Itestimatestheimpactofriskontheprojectandtheproduct. EstimationisdonebyusingRiskTable.Riskprojectionaddressesriskin2ways

k	egory	bability	act	MM
eestimate ybe ificantly v		%		
ger no.of rsthan nned		%		
sreuse nplanned		%		
luser stsystem		%		

Likelihood or probability that the risk is real(L_i)Consequences(X_i)

RiskProjection

StepsinRiskprojection

1. EstimateLifor eachrisk
2. EstimatetheconsequenceXi
3. Estimatetheimpact
4. Drawtherisktable

Ignoretheriskwherethemanagementconcernislowi.e.,riskhavingimpacthighorlowwithlowprobabilityofoccurrence

Consider all risks where management concern is high i.e., high impact with high ormoderateprobabilityofoccurrenceorlowimpactwithhighprobabilityofoccurrence

Risk ProjectionProjection

TheimpactofeachriskisassessedbyImpactvaluesCatastrophic-1Critical-2Marginal-3 Negligible-4

RiskRefinement

AlsocalledRiskassessment

Refinestherisktableinreviewing theriskimpactbasedonthefollowing threefactors

a.Nature:Likelyproblemsifriskoccurs

b.Scope: Just how serious is it?c.Timing:Whenandhowlong

It is based on Risk ElaborationCalculateRiskexposureRE=P*C

WherePisprobabilityandCiscostofprojectifriskoccursRiskMitigationMonitoringAndManagement(RMMM)

Itsgoalistoassistproject teamindevelopingastrategyfor dealingwithriskThere arethreeissues ofRMMM

1) RiskAvoidance2)Risk Monitoringand3)RiskManagement

RiskMitigationMonitoringAndManagement(RMMM)

RiskMitigation

ProactiveplanningforriskavoidanceRiskMonitoring

AssessingwhetherpredictedriskoccurornotEnsuringrisk aversionstepsarebeing properlyappliedCollection of information for future risk analysisDetermine whichriskscausedwhichproblems

ActionstobetakenintheeventthatmitigationstepshavefailedandtheriskhasbecomealiveproblemDeviseRMMP(RiskMitigationMonitoring And ManagementPlan)

RMMMplan

Itdocumentsallworkperformedasapartofriskanalysis.

Each risk is documented individually by using a Risk Information Sheet.RISismaintainedbyusingadatabasesystemQualityManagement.

Software Cost Estimation

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated, but how is this done? Several estimation procedures have been developed and are having the following attributes in common.

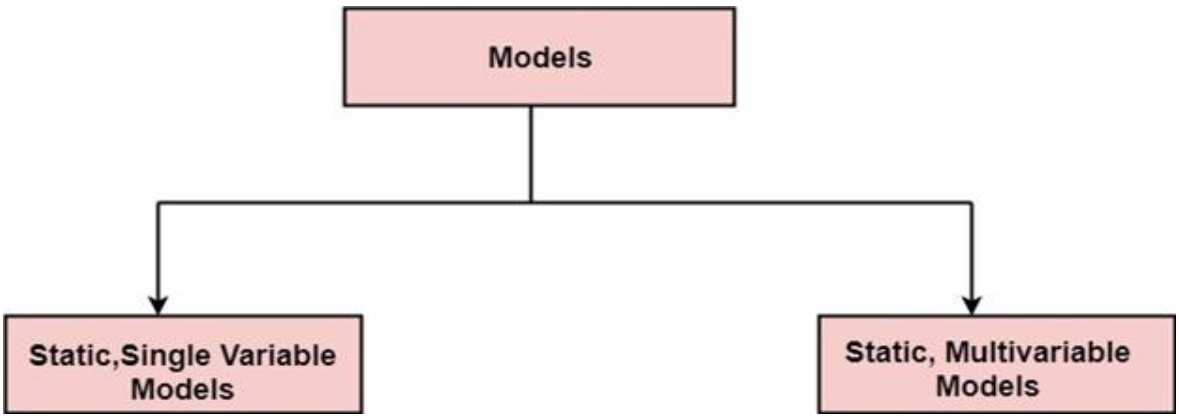
1. Project scope must be established in advanced.
2. Software metrics are used as a support from which evaluation is made.
3. The project is broken into small PCs which are estimated individually. To achieve true cost & schedule estimate, several option arise.
4. Delay estimation
5. Used symbol decomposition techniques to generate project cost and schedule estimates.
6. Acquire one or more automated estimation tools.

Uses of Cost Estimation

1. During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.
2. In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.

Cost Estimation Models

A model may be static or dynamic. In a static model, a single variable is taken as a key element for calculating cost and time. In a dynamic model, all variable are interdependent, and there is no basic variable.



Static, Single Variable Models: When a model makes use of single variables to calculate desired values such as cost, time, efforts, etc. is said to be a single variable model. The most common equation is:

$$C=aL^b$$

Where C = Costs

L= size

a and b are constants

The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.

The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.

E=1.4L0.93

DOC=30.4L0.90

D=4.6L0.26

Where E= Efforts (Person Per Month)

DOC=Documentation (Number of Pages)

D = Duration (D, in months)

L = Number of Lines per code

Static, Multivariable Models: These models are based on method (1), they depend on several variables describing various aspects of the software development environment. In some model, several variables are needed to describe the software development process, and selected equation combined these variables to give the estimate of time & cost. These models are called multivariable models.

UNIT – V

QUALITY MANAGEMENT

QualityConcepts

Variationcontrolisthe heartofqualitycontrol
Form one project to another, we want to minimize the difference between the predictedresources needed to complete a project and the actual resources used, including staff,equipment,andcalendartime
Qualityofdesign
Referstocharacteristicsthatdesignersspecifyfortheendproduct QualityManagem
ent
Qualityofconformance
Degreetowhichdesignspecificationsarefollowedinmanufacturingtheproduct Qualitycontrol
Series of inspections, reviews, and tests used to ensure conformance of a work product to itsspecifications
Qualityassurance
Consists of a set of auditing and reporting functions that assess the effectiveness andcompleteness ofqualitycontrolactivities

Cost ofQuality

Preventioncosts
Qualityplanning,formaltechnicalreviews,testequipment,training Appraisalcosts
In-processandinter-processinspection,equipmentcalibrationandmaintenance,testing Failurecosts
rework,repair,failuremodeanalysis Externalfailure
costs
Complaintresolution,productreturnandreplacement,help linesupport,warrantyworkSoftware QualityAssurance
Softwarequalityassurance(SQA)istheconcernofeverysoftwareengineerto reducecostandimproveproducttime-to-market.
A Software Quality Assurance Plan is not merely another name for a test plan, thoughtestplans are includedinanSQAplan.
SQAactivitiesareperformedoneverysoftwareproject.
Useofmetricsisanimportantpartofdevelopingastrategytoimprovethequalityofbothsoftwareprocessesand workproducts.

SoftwareQualityAssurance

DefinitionofSoftwareQualityservestoemphasize:
Conformance to software requirements is the foundation from which software quality ismeasured.

Specifiedstandardsareusedtodefinethedevelopmentcriteriathatareusedtoguidethemannerinwhichsoftwareisengineered.

Softwaremustconformtoimplicitrequirements(easeofuse,maintainability,reliability,etc.)as wellasits explicitrequirements.

SQAActivities

PrepareSQAplanfortheproject.
Participateinthedevelopmentoftheproject'ssoftwareprocessdescription.
Reviewsoftwareengineeringactivitiestoverifycompliancewiththedefinedsoftwareprocess.
Auditdesignated softwareworkproductstoverifycompliancewiththosedefinedaspartofthesoftwareprocess.

Ensure that any deviations in software or work products are documented and handledaccordingtoadocumentedprocedure.
Recordanyevidenceofnoncomplianceandreportsthemtomanagement.

SoftwareReviews

Purposeistofinderrorsbeforetheyarepassedontoanothersoftwareengineeringactivityorreleasedtothecustomer.
Softwareengineers(andothers)conductformaltechnicalreviews(FTRs)forsoftwarequalityassurance.
Using formal technical reviews (walkthroughs or inspections) is an effective means for improving software quality.

FormalTechnicalReview

AFTRisasoftwarequalitycontrolactivityperformedbysoftwareengineersandothers.Theobjectives are:
Touncovererrorsinfunction,logic orimplementationforanyrepresentationofthesoftware.
Toverifythatthesoftwareunder reviewmeetsitsrequirements.
Toensurethatthesoftwarehasbeenrepresentedaccordingtopredefinedstandards.To achieve software thatisdevelopedina uniformmannerand
Tomakeprojectsmoremanageable.

Review meeting in FTR

TheReviewmeetinginaFTRshouldabidetothefollowingconstraintsReviewmeetingmembersshouldbebetweenthreeandfive.
Everypersonshouldprepareforthemeeting and should notrequiremorethantwohoursofworkforeachperson.
Thedurationofthereviewmeetingshouldbelessthantwohours.
The focus of FTR is on a work product that is requirement specification, a detailedcomponentdesign,a sourcecode listingfora component.
Theindividualwhohasdevelopedtheworkproducti.e,theproducerinformstheprojectleaderthatthe workproductiscomplete andthatareviewisrequired.
The project leader contacts a review leader, who evaluates the product for readiness,generatescopyofproductmaterialand distributesthemtotwoorthreereviewmembersforadvance preparation.

Each reviewer is expected to spend between one and two hours reviewing the product, making notes
The review leader also reviews the product and establish an agenda for the review meeting The review meeting is attended by review leader, all reviewers and the producer.
One of the reviewer act as a recorder,who notes down all important points discussed in the meeting.
The meeting(FTR) is started by introducing the agenda of meeting and then the producer introduces his product. Then the producer "walkthrough" the product, the reviewers raise issues which they have prepared in advance.
If errors are found the recorder notes down

ReviewreportingandRecordkeeping
During the FTR, a reviewer(recorder) records all issues that have been raised A review summary report answers three questions
What was reviewed? Who reviewed it?
What were the findings and conclusions?
Review summary report is a single page form with possible attachments

The review issues list serves two purposes To identify problem areas in the product
To serve as an action item checklist that guides the producer as corrections are made

Review Guidelines
Review the product, not the producer Set an agenda and maintain it
Limit debate and rebuttal
Enunciate problem areas, but don't attempt to solve every problem noted Take return notes
Limit the number of participants and insist upon advance preparation. Develop a checklist for each product i.e likely to be reviewed Allocate resources and schedule time for FTRS
Conduct meaningful training for all reviewer Review your early reviews
Software Defects
Industry studies suggest that design activities introduce 50-65% of all defects or errors during the software process
Review techniques have been shown to be up to 75% effective in uncovering design flaws which ultimately reduces the cost of subsequent activities in the software process

Statistical Quality Assurance
Information about software defects is collected and categorized. Each defect is traced back to its cause
Using the Pareto principle (80% of the defects can be traced to 20% of the causes) isolate the "vital few" defect causes.

Move to correct the problems that caused the defects in the "vital few”

Six Sigma for Software Engineering The most widely used strategy for statistical quality assurance Three core steps:
Define customer requirements, deliverables, and project goals via well-defined methods of customer communication.
Measure each existing process and its output to determine current quality performance (e.g., compute defect metrics)
Analyze defect metrics and determine vital few causes.

For an existing process that needs improvement
Improve process by eliminating the root causes for defects
Control future work to ensure that future work does not reintroduce causes of defects
If new processes are being developed
Design each new process to avoid root causes of defects and to meet customer requirements
Verify that the process model will avoid defects and meet customer requirements
Software Reliability Defined as the probability of failure free operation of a computer program in a specified environment for a specified time period
Can be measured directly and estimated using historical and developmental data
Software reliability problems can usually be traced back to errors in design or implementation.
Measures of Reliability
Mean time between failure (MTBF) = MTTF + MTTR MTTF = mean time to failure
MTTR = mean time to repair
Availability = [MTTF / (MTTF + MTTR)] x 100%

ISO 9000 Quality Standards
ISO (International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization. ISO declared its 9000 series of standards in 1987. It serves as a reference for the contract between independent parties. The ISO 9000 standard determines the guidelines for maintaining a quality system. The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc. ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself.

TypesofISO9000QualityStandards
The ISO 9000 series of standards is based on the assumption that if a proper stage isfollowed for production, then good quality products are bound to follow automatically. Thetypesofindustries towhichthe various ISO standardsapplyare asfollows.

1. **ISO9001:**Thisstandardappliestotheorganizationsengagedindesign,development,producti on, and servicing of goods. This is the standard that applies to most softwaredevelopmentorganizations.

2. **ISO 9002:** This standard applies to those organizations which do not design products butareonlyinvolvedintheproduction.Examplesofthesecategoryindustriescontainsteelandcar manufacturing industries that buy the product and plants designs from external sourcesandareengagedinonlymanufacturingthoseproducts.Therefore,ISO9002doesnotapplytosoftwaredev elopmentorganizations.

3. **ISO9003:**Thisstandardappliestoorganizationsthatareinvolvedonlyintheinstallationandtesting ofthe products.Forexample,Gascompanies.

AnorganizationdeterminestoobtainISO9000certificationappliestoISOregistrarofficeforregistration.Theproce ssconsistsofthefollowingstages:

1. **Application:**OnceanorganizationdecidedtogoforISOcertification,itappliestotheregistrarfo rregistration.

2. **Pre-Assessment:**Duringthisstage,theregistrarmakesaroughassessmentoftheorganization.

3. **DocumentreviewandAdequacyofAudit:**Duringthisstage,theregistrarreviewsthe documentsubmittedbytheorganizationandsuggestanimprovement.

4. **Compliance Audit:** During this stage, the registrar checks whether the organization has compiled thesuggestionmadebyitduringthereviewornot.

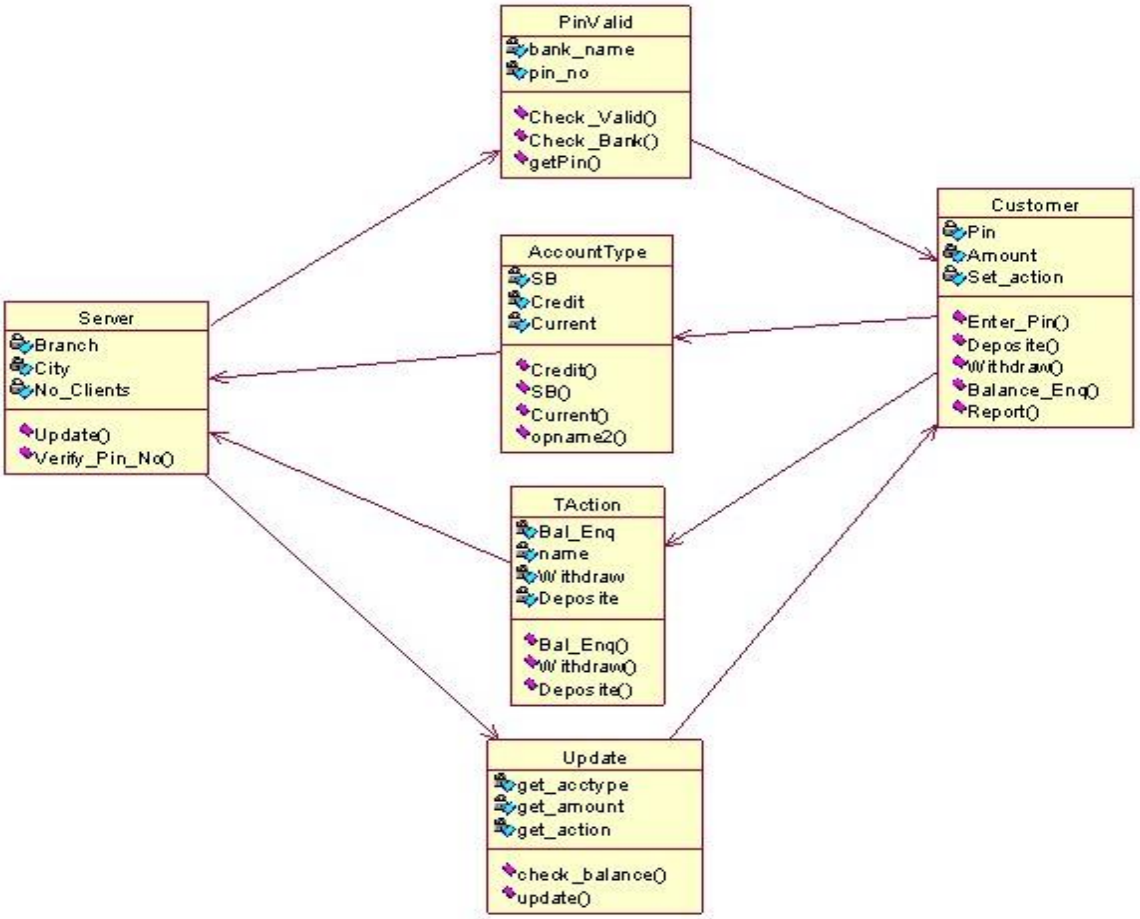
5. **Registration:**TheRegistrarawardstheISOcertificationafterthesuccessfulcompletionofallt

hephases.

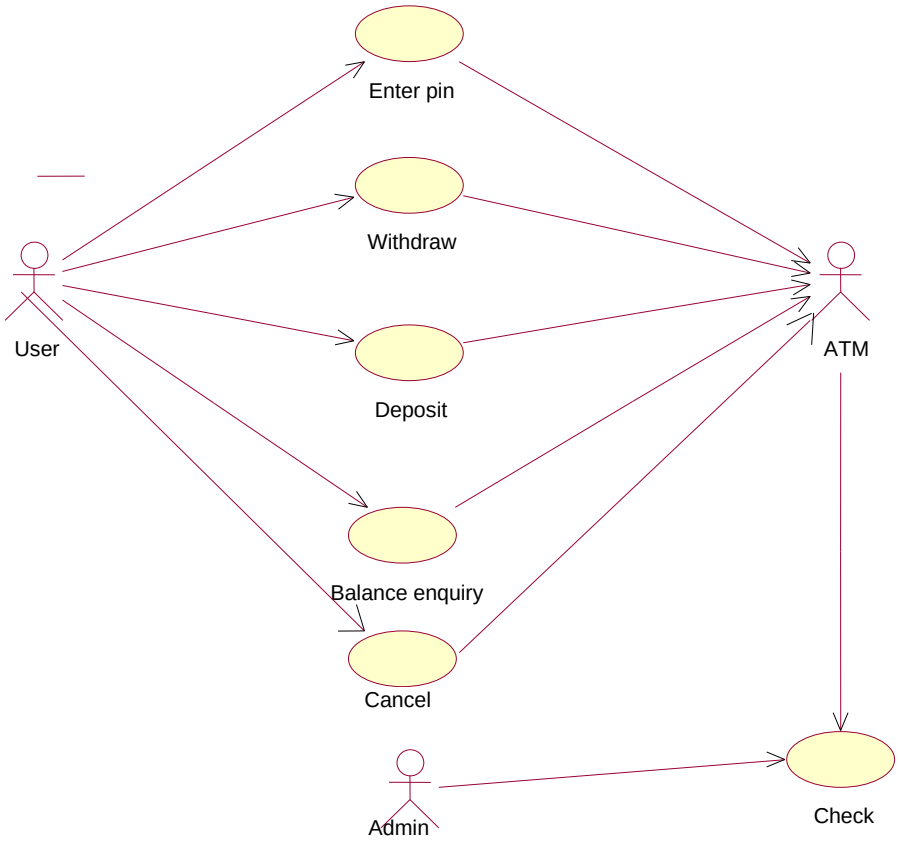
6. **ContinuedInspection:** Theregistrar continuedtomonitor the organizationtimebytime.

CASE STUDY--ATM MANAGEMENT SYSTEMS

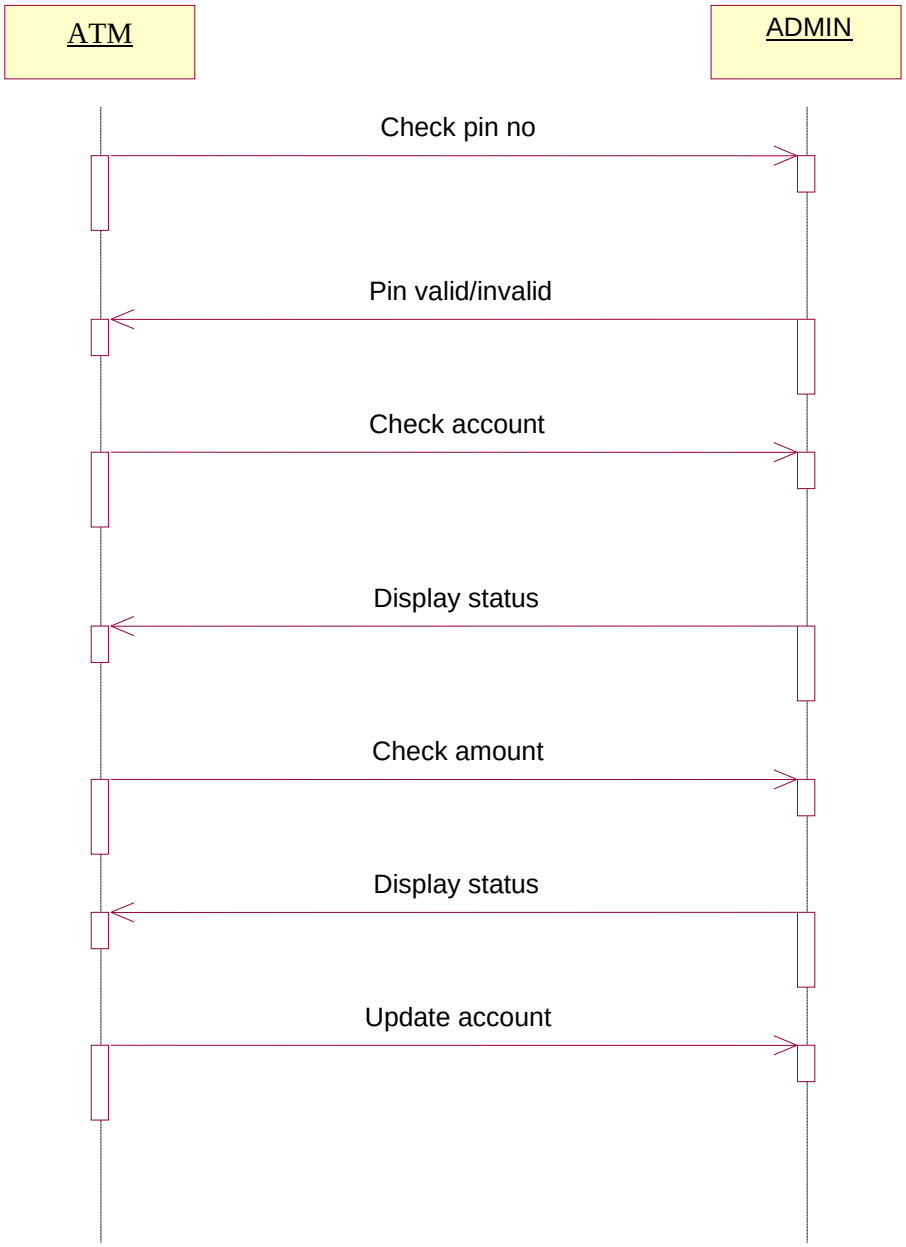
CLASS DIAGRAM FOR ATM



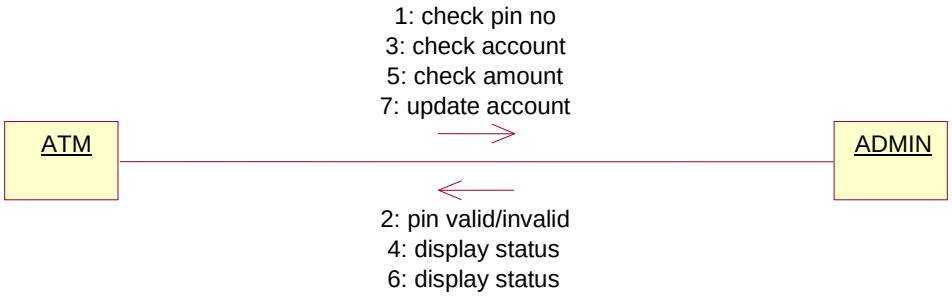
Use case diagram for ATM



Sequence diagram for ATM

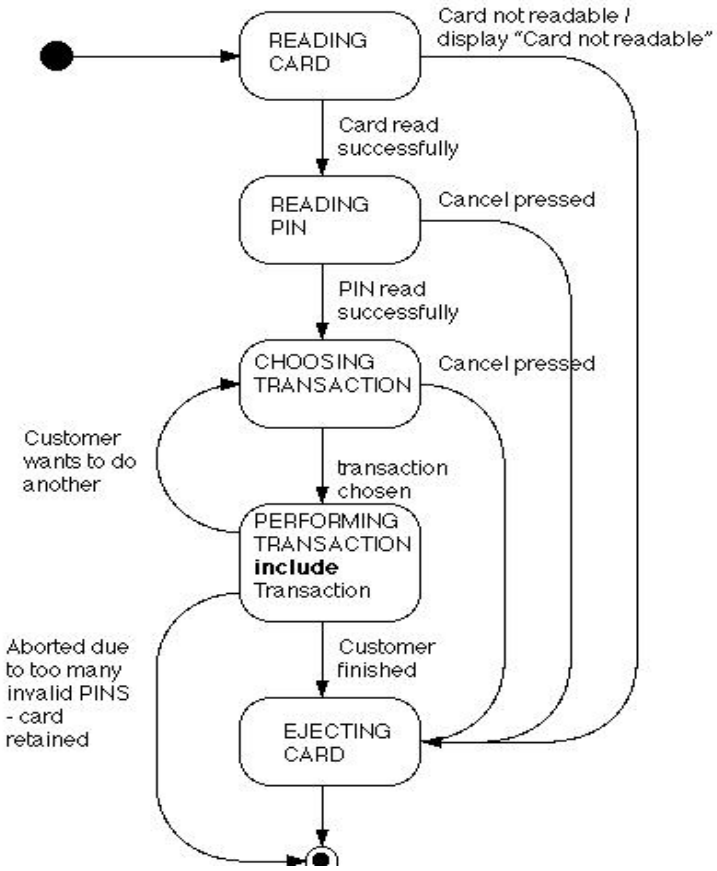


Collaboration diagram for ATM

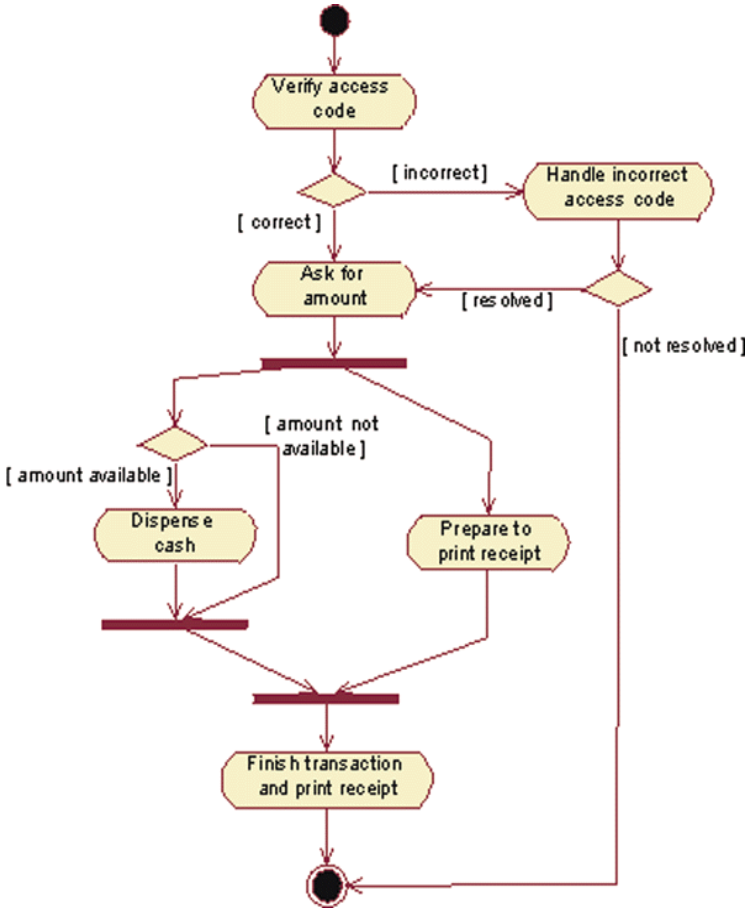


State chart diagram for ATM

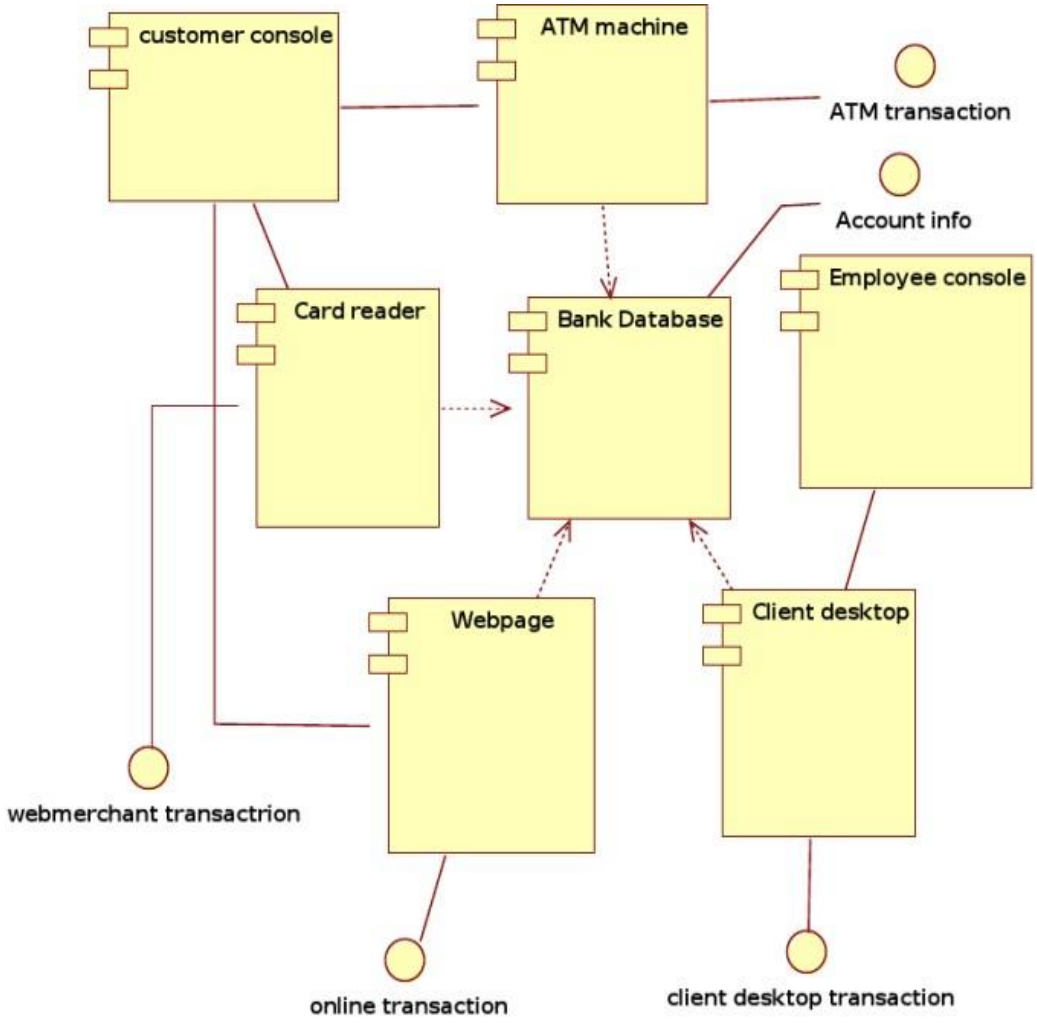
State-Chart for One Session



Activity diagram for ATM



Component diagram



Deployment diagram for ATM