MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Autonomous Institution – UGC, Govt. of India

Department of COMPUTATIONAL INTELLIGENCE
B. TECH (CSE-AIML)

B.TECH(R-22 Regulation)
(II YEAR – I SEM)

2024-25

SOFTWARE ENGINEERING
(R22A0505)

LECTURE NOTES

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12(B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE-Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad–500100, Telangana State, India

Department of COMPUTATIONAL INTELLIGENCE

(CSE-AIML)

SOFTWARE ENGINEERING

(R22A0505)

LECTURE NOTES

Department of Computational Intelligence

CSE (Artificial Intelligence and Machine Learning)

Vision

To be a premier centre for academic excellence and research through innovative interdisciplinary collaborations and making significant contributions to the community, organizations, and society as a whole.

Mission

- To impart cutting-edge Artificial Intelligence technology in accordance with industry norms.
- To instill in students a desire to conduct research in order to tackle challenging technical problems for industry.
- To develop effective graduates who are responsible for their professional growth, leadership qualities and are committed to lifelong learning.

QUALITY POLICY

- To provide sophisticated technical infrastructure and to inspire students to reach their full potential.
- To provide students with a solid academic and research environment for a comprehensive learning experience.
- To provide research development, consulting, testing, and customized training to satisfy specific industrial demands, thereby encouraging self-employment and entrepreneurship among students.

For more information: www.mrcet.ac.in

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

L/T/P/D/C

II Year B.Tech. CSE (AI&ML) -I SEM 3 / -/-/-/ 3

(R22A0505) SOFTWARE ENGINEERING

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, Changing Nature of Software, Software myths.

A Generic view of process: Software engineering- A layered technology, a process framework, The Capability Maturity Model Integration (CMMI

Process models: The waterfall model, The spiral Model, Agile methodology.

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.

UNIT-III:

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

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,

metrics of process and products: metrics for measurement, metrics for software quality

UNIT-V:

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

Quality Management: Software Quality, Quality concepts, Software quality assurance, Software Reviews, Formal technical reviews, Statistical Software quality Assurance, Software reliability, TheISO9000 quality standards.

Case Study – ATM Management System.

TEXT BOOKS:

1. Software Engineering A practitioner's Approach, Roger S Pressman, 6thedition. McGraw Hill International Edition.
2. Software Engineering, IanSommerville, 7th edition, Pearson education.

REFERENCEBOOKS:

1. Software Engineering, A Precise Approach, Pankaj Jalote, Wiley India, 2010.
2. Software Engineering: APrimer, Waman SJawadekar, TataMcGraw-Hill,2008
3. FundamentalsofSoftwareEngineering,RajibMall,PHI,2005
4. Software Engineering, Principles and Practices,DeepakJain,OxfordUniversityPress.
5. Software Engineering: Abstraction and modelling, Diner Bjorner,Springer International edition,2006.
6. Software Engineering: Specification of systems and languages, Diner Bjorner, SpringerInternationaledition2006.
7. Software Engineering Foundations, YinguxWang, Auerbach Publications, 2008.
8. Software Engineering Principles and Practice, Hans Van Vliet, 3rd edition, John Wiley & Sons Ltd.
9. Software Engineering: Domains, Requirements, and Software Design, D.Bjorner, Springer International Edition.
10. Introduction to Software Engineering, R.J.Leach, CRC Press.

Outcomes:

1. Identify the minimum requirements for the development of application.
2. Develop, maintain, efficient, reliable and cost effective software solutions.
3. Critically thinking and evaluate assumptions and arguments.
4. Test and maintain process in an object orient software project.
5. Ensure good quality software through quality assurance.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

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 effective solution 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 MYTHS

Myths are widely held but false beliefs and views which propagate misinformation and confusion. Three types of myth are associated with software:

- Management myth
- Customer myth
- Practitioner's myth

MANAGEMENT MYTHS

- Myth(1)-The available standards and procedures for software are enough.
- Myth(2)-Each organization feel that they have state-of-art software development tools since they have latest computer.
- Myth(3)-Adding more programmers when the work is behind schedule can catch up.
- Myth(4)-Outsourcing the software project to third party, we can relax and let that party build it.

CUSTOMER MYTHS

- Myth(1)- General statement of objective is enough to begin writing programs, the details can be filled in later.
- Myth(2)-Software is easy to change because software is flexible

PRACTITIONER'S MYTH

- Myth(1)-Once the program is written, the job has been done.
- Myth(2)-Until the program is running, there is no way of assessing the quality.

- Myth(3)-The only deliverable work product is the working program
- Myth(4)-Software Engineering creates voluminous and unnecessary documentation and invariably slows down software development.

A GENERIC VIEW OF PROCESS

SOFTWARE ENGINEERING-A LAYERED TECHNOLOGY

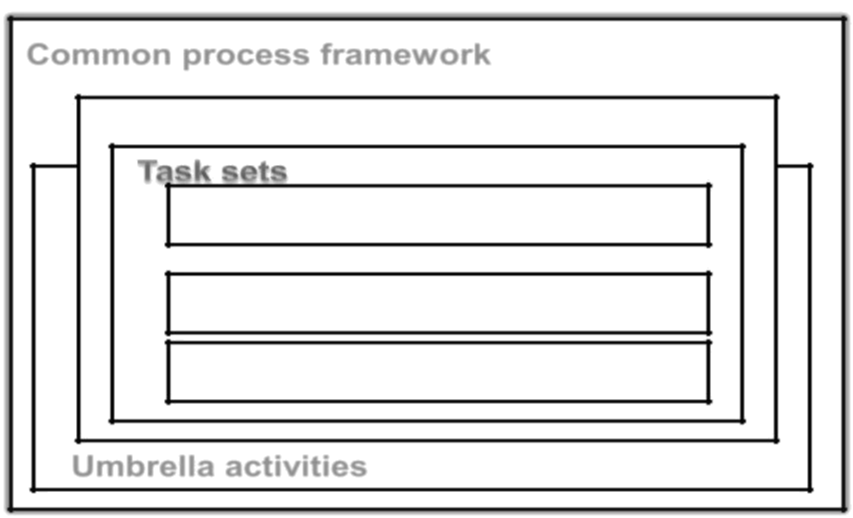
Fig: Software Engineering-A layered technology

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

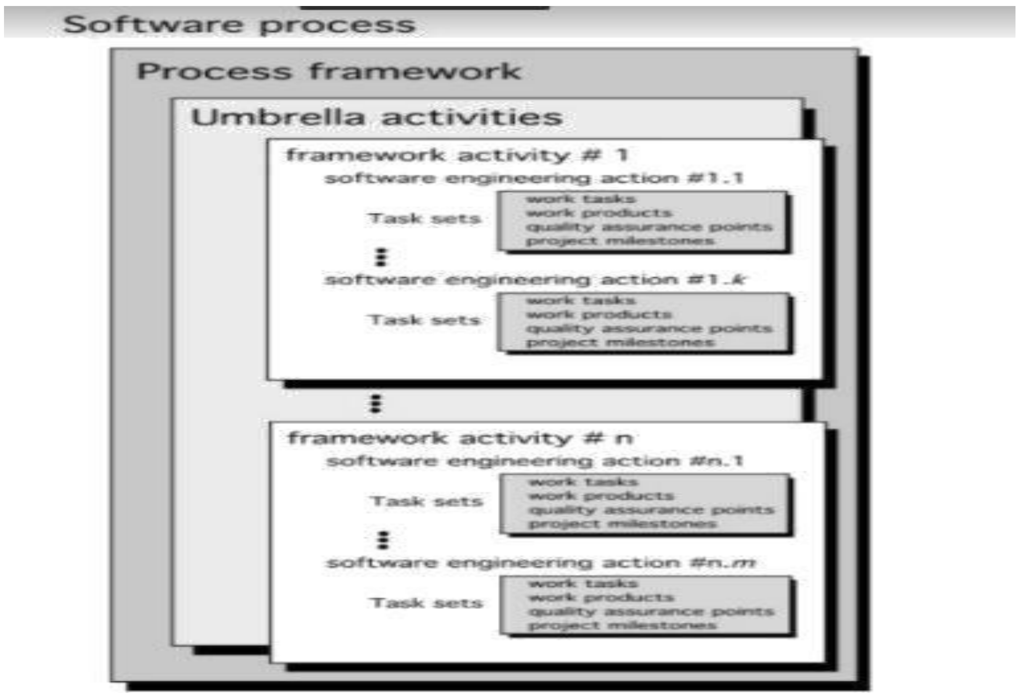
A PROCESS FRAMEWORK

- 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:

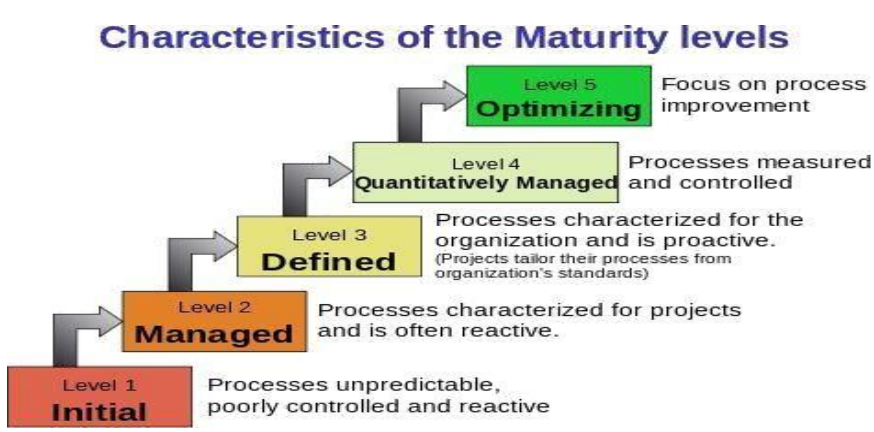
-Lets organization select specific improvement that best meet its business objectives and minimize risk-Levels are called capability levels.

-Describes a process in 2 dimensions

-Each process area is assessed against specific goals and practices and is rated according to the following capability levels.

CMMI

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



CMMI

- 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 process improvement.

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

-

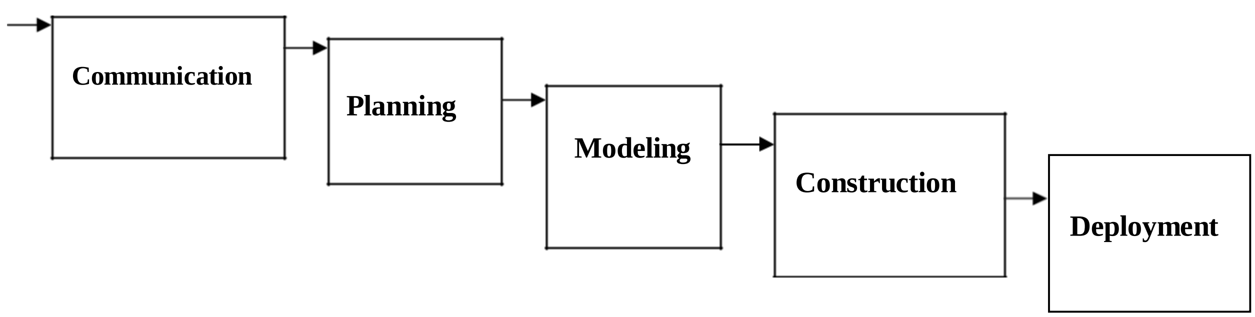
PROCESS MODELS

- 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 system level and progresses through analysis, design, code and testing

PROBLEMS IN WATERFALLMODEL

- Real projects rarely follow the sequential flow since they are always iterative

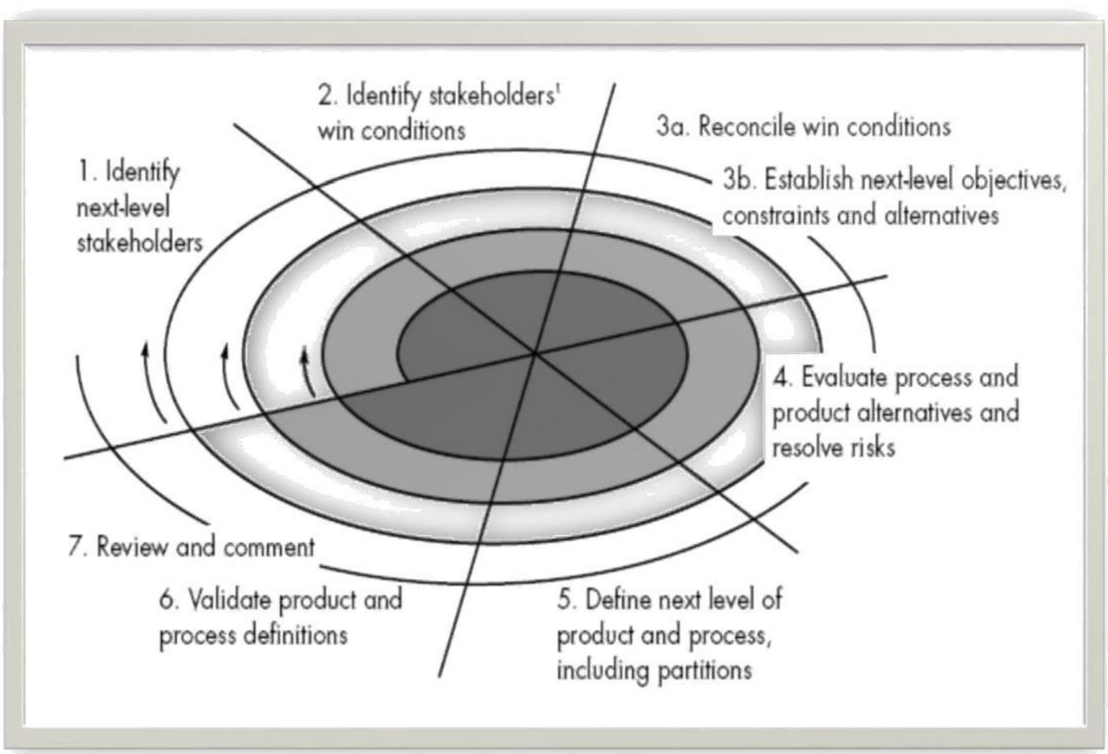
- The model requires requirements to be explicitly spelled out in the beginning, which is often difficult
- A working model is not available until late in the project time plan

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
- And sophisticated version of software



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 to provide 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 LANGUAGE SPECIFICATION

- 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. Data structures-

-Provide data passing from one sub-system to another 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 specific system.

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

IEEE requirements standard

1. Introduction Purpose
   Scope
   Definitions, Acronyms and Abbreviations References
   Overview
2. General description Product perspective Product function summary User characteristics
   General constraints
   Assumptions and dependencies
3. Specific Requirements

- Functional requirements
  -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:

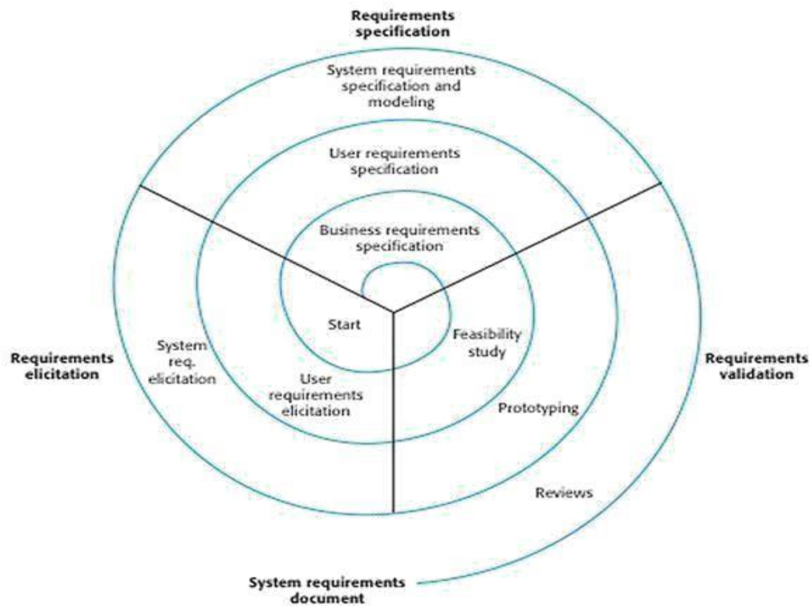
1. Feasibility study
   --Concerned with assessing whether the system is useful to the business
   2.Elicitation and analysis
   --Discovering requirements
   3.Specifications
   --Converting the requirements into a standard
   form 4.Validation
   -- Checking that the requirements actually define the 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 Feasibility report 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 of requirements in natural language is sometimes difficult to Understand.
3. Different stakeholders express requirements differently
4. Influences of political factors Change in requirements due to dynamic environments.

REQUIREMENTS ELICITATION

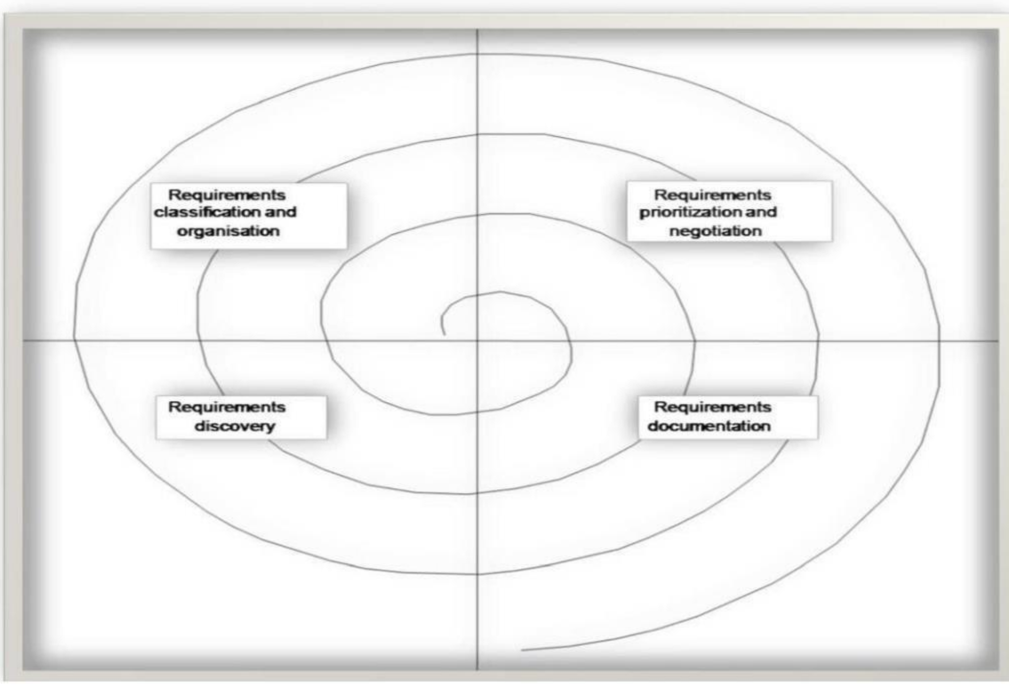
PROCESS Process activities

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

The spiral representation of Requirements Engineering



REQUIREMENTS DISCOVERY 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 by the 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 GENERATION Requirements 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. Automated tools are required for requirements storage, change management and traceability.

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 management consists 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 its implementation

UNIT III

DESIGN ENGINEERING

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

- 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 is applicable 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=cp2If 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.

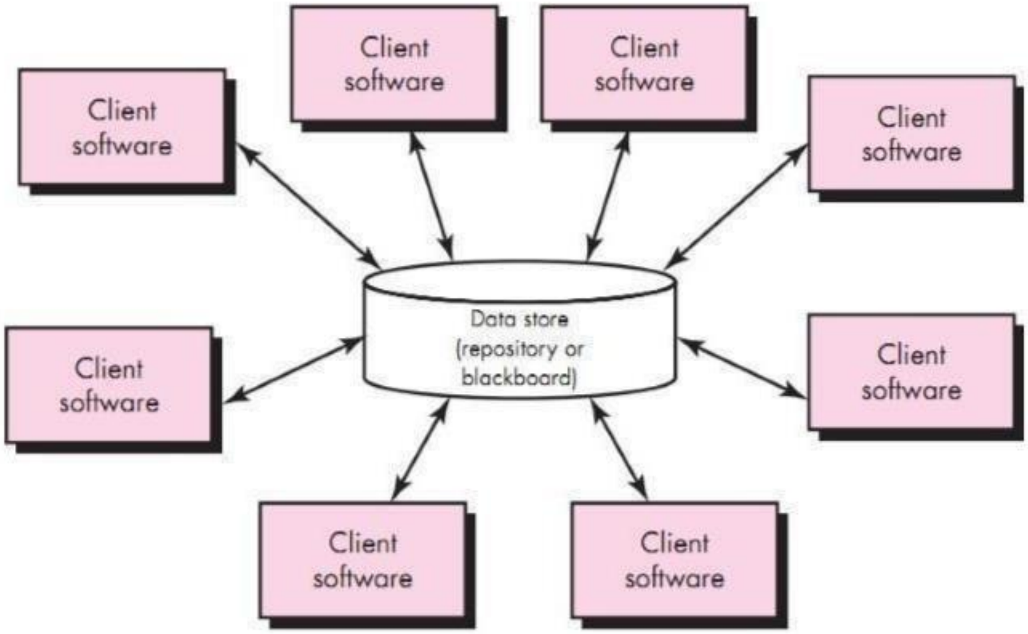
DATA DESIGN AT COMPONENT
LEVEL Principles for data specification:

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. 3.Mechanismfor 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) C*onstraints* that define how components can be integrated to form the system
(4) S*emantic 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)

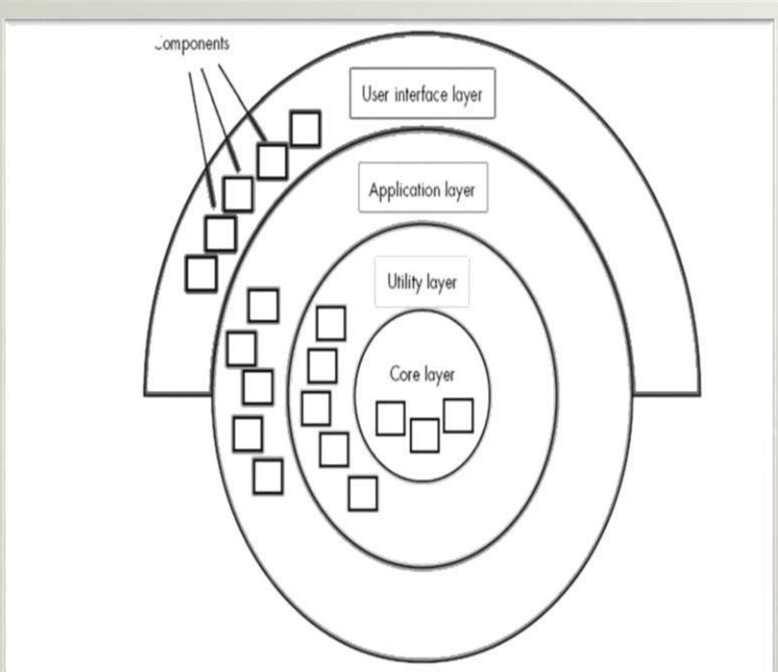


FIG: Layered

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
(c) 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 models5)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.

UNIT IV

TESTING STRATEGIES

Software is tested to uncover errors introduced during design and construction. Testing often accounts for ore 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 tothe Customer 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

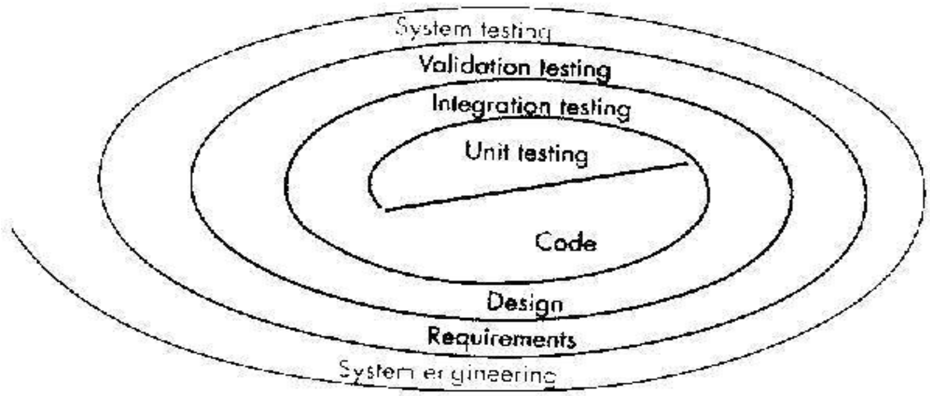
Low level tests verifies small code segments. High level tests validate major system functions against customer requirements

<u>Test Strategies for Conventional Software:</u>

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

1) Unit Testing 2) Integration Testing 3)Validation Testing and
4) System Testing

<u>Spiral Representation of Testing for Conventional Software</u>



<u>Unit Testing</u> begins at the vortex of the spiral and concentrates on each unit of software insource code.

It uses testing techniques that exercise specific paths in a component and its control structure to ensure complete coverage and maximum error detection. It focuses on the internal processing logic and data structures. Test cases should uncover errors.

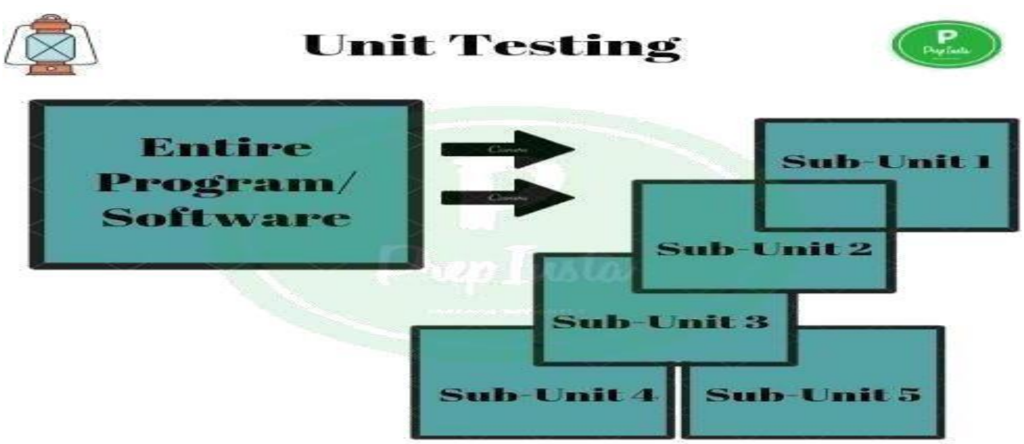


Fig: Unit Testing

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.

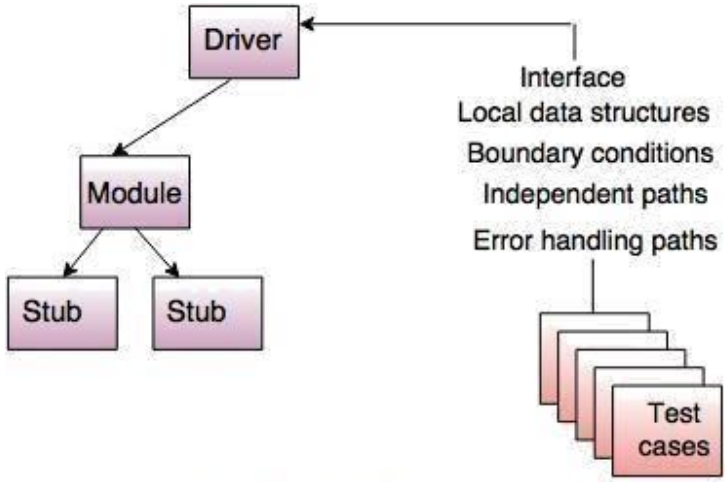


Fig. - Unit test environment

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., top- down 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 functions in a manner that can be reasonably expected by the customer.

1) Validation Test Criteria2)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. Treats the 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 tests that will cover the graph so that each object and relationship is exercised and errors are uncovered.

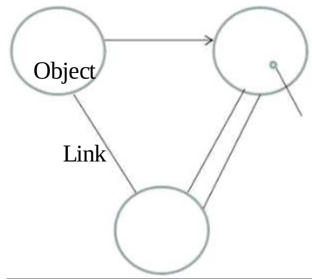


Fig: O-R graph.

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 input conditions. Reduces the cost of testing

<u>Example</u>

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.

<u>Example</u>

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

<u>Example</u>

Three inputs A,B,C each having three values will require 27 test cases**. O**rthogonal testing will reduce the number of test case to 9 as shown below

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.Control structure testing1.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 graph3.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 nodes2.V(G)=Number of regions

The structured constructs used in the flow graph are:

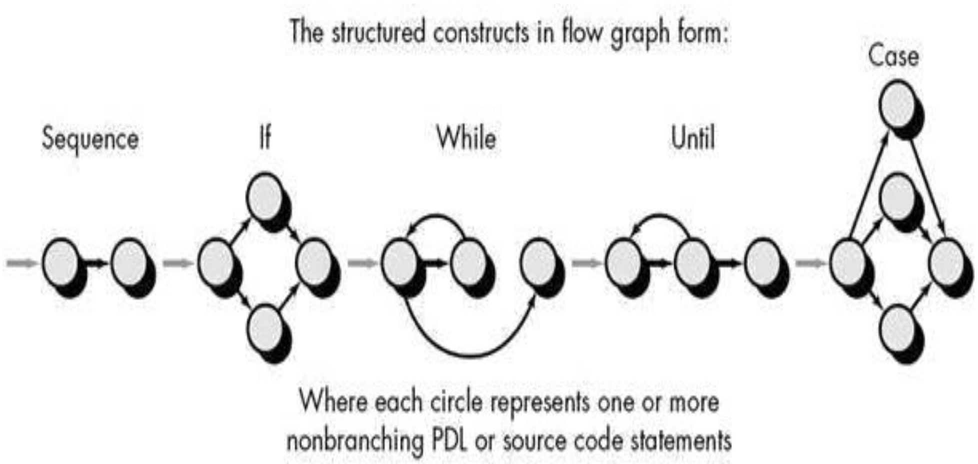


Fig: Basis path testing

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<Boolean 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 loops2.Nested loops

3.Concatenated loops4.Unstructured loops

Testing of simple loops

N is the maximum number of allowable passes through the loop1.Skip the loop entirely 2.Only one pass through the loop3.Two passes through the loop

4.m passes through the loop where m>N5.N-1,N,N+1 passes the loop

The Art of Debugging

Debugging occurs as a consequence of successful testing. It is an action that results in the removal of errors.

It is very much an art.

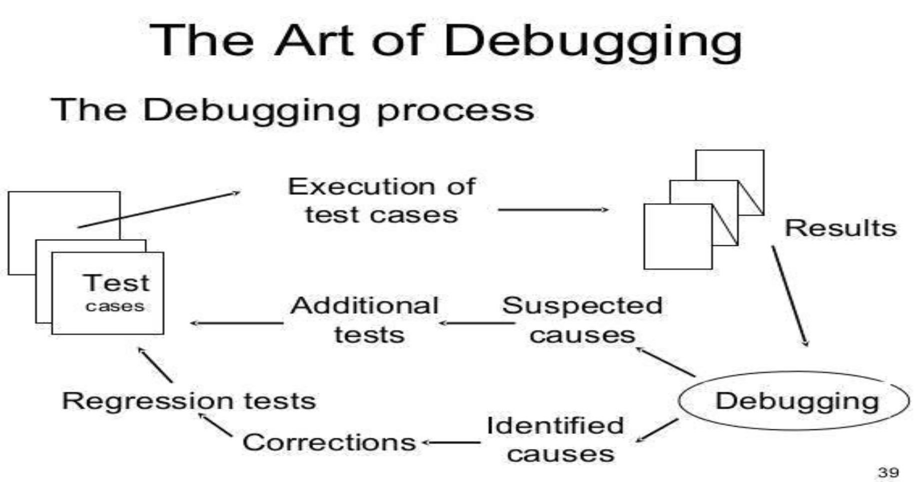


Fig: Debugging process

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 of systematic 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 occurrence 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 case generators, 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 there-execution of some subset of tests that are already conducted to ensure that changes have not propagated unintended side effects. It ensures that changes do not introduce unintended behaviour or errors. This can be done manually or automated.

UNIT – V

Risk Management

Risk is an undesired event or circumstance that occur while a project is underway It is necessary for the project manager to anticipate and identify different risks that a project may be susceptible to Risk Management. It aims at reducing the impact of all kinds of risk that may effect a project by identifying, analyzing and managing them

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 Risk 4)Known Risk 5)Unpredictable Risk

6) Predictable risk

Project risk: Threaten the project plan and affect schedule and resultant cost **Technical risk**: Threaten the quality and timeliness of software to be produced **Business risk:** Threaten the viability of software to be built

Known risk: These risks can be recovered from careful evaluation **Predictable risk:** Risks are identified by past project experience **Unpredictable risk:** Risks that occur and may be difficult to identify

Risk Identification

It concerned with identification of riskStep1: Identify all possible risks Step2: Create item check list

Step3: Categorize into risk components-Performance risk, cost risk, support risk and schedule risk

Step4: Divide the risk into one of 4 categoriesNegligible-0

Marginal-1Critical-2

Risk Identification

Risk Identification includes Product size

Business impact Development environment Process definition Customer characteristics Technology to be built Staff size and experience

Risk Projection

Also called risk estimation. It estimates the impact of risk on the project and the product.

Estimation is done by using Risk Table. Risk projection addresses risk in 2 ways

Risk	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger no. of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End user resist system	BU	40%	3	

Likelihood or probability that the risk is real(Li)

Consequences (Xi)

Risk Projection

Steps in Risk projection

1. Estimate Li for each risk
2. Estimate the consequence Xi
3. Estimate the impact
4. Draw the risk table

Ignore the risk where the management concern is low i.e., risk having impact high or lowwith low probability of occurrence

Consider all risks where management concern is high i.e., high impact with high or moderate probability of occurrence or low impact with high probability of occurrence

Risk Projection Projection

The impact of each risk is assessed by Impact valuesCatastrophic-1 Critical-2 Marginal-3 Negligible-4

Risk Refinement

Also called Risk assessment

Refines the risk table in reviewing the risk impact based on the following three factorsa. Nature:

Likely problems if risk occurs

b. Scope: Just how serious is it?c. Timing: When and how long

It is based on Risk Elaboration Calculate Risk exposure RE=P*C

Where P is probability and C is cost of project if risk occurs Risk Mitigation Monitoring And Management (RMMM)

Its goal is to assist project team in developing a strategy for dealing with risk There are three issues of RMMM

1) Risk Avoidance 2)Risk Monitoring and3)Risk Management

Risk Mitigation Monitoring And Management (RMMM)

Risk Mitigation

Proactive planning for risk avoidance Risk Monitoring

Assessing whether predicted risk occur or not Ensuring risk aversion steps are being properly applied Collection of information for future risk analysis Determine which risks caused which problems

Risk Mitigation Monitoring And Management (RMMM)Risk Management Contingency planning Actions to be taken in the event that mitigation steps have failed and the risk has become a live

problem Devise RMMP(Risk Mitigation Monitoring And Management Plan)

QUALITY CONCEPTS

Variation control is the heart of quality control

Form one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment, and calendar time

Quality of design

Refers to characteristics that designers specify for the end product Quality Management

Quality of conformance

Degree to which design specifications are followed in manufacturing the product Quality control

Series of inspections, reviews, and tests used to ensure conformance of a work product to itsspecifications

Quality assurance

Consists of a set of auditing and reporting functions that assess the effectiveness and completeness of quality control activities

COST OF QUALITY

Prevention costs

Quality planning, formal technical reviews, test equipment, training Appraisal costs
In-process and inter-process inspection, equipment calibration and maintenance, testing Failure costs
rework, repair, failure mode analysis External failure costs
Complaint resolution, product return and replacement, help line support, warranty work Software Quality Assurance

Software quality assurance (SQA) is the concern of every software engineer to reduce cost and improve product time-to-market.

A Software Quality Assurance Plan is not merely another name for a test plan, though test plans are
included in an SQA plan.
SQA activities are performed on every software project.

Use of metrics is an important part of developing a strategy to improve the quality of both software processes and work products.

SOFTWARE QUALITY ASSURANCE Definition
of Software Quality serves to emphasize:
Conformance to software requirements is the foundation from which software quality is measured.

Specified standards are used to define the development criteria that are used to guide the manner in which software is engineered.
Software must conform to implicit requirements (ease of use, maintainability,

reliability,etc.) as well as its explicit requirements.

SQA Activities

Prepare SQA plan for the project.

Participate in the development of the project's software process description.

Review software engineering activities to verify compliance with the defined software process.

Audit designated software work products to verify compliance with those defined as part of the software process.

Ensure that any deviations in software or work products are documented and handled according to a documented procedure.

Record any evidence of noncompliance and reports them to management.

SOFTWARE REVIEWS

Purpose is to find errors before they are passed on to another software engineering activity or released to the customer.

Software engineers (and others) conduct formal technical reviews (FTRs) for software quality assurance.

Using formal technical reviews (walkthroughs or inspections) is an effective means for improving software quality.

FORMAL TECHNICAL REVIEW

A FTR is a software quality control activity performed by software engineers and others. The objectives are:

To uncover errors in function, logic or implementation for any representation of the software.

To verify that the software under review meets its requirements.

To ensure that the software has been represented according to predefined standards. To achieve software that is developed in a uniform manner and To make projects more manageable.

Review meeting in FTR

The Review meeting in a FTR should abide to the following constraints Review meeting members should be between three and five.

Every person should prepare for the meeting and should not require more than two hours of work for each person.

The duration of the review meeting should be less than two hours.

The focus of FTR is on a work product that is requirement specification, a detailed component design, a source code listing for a component.

The individual who has developed the work product i.e, the producer informs the project leader that the work product is complete and that a review is required.

The project leader contacts a review leader, who evaluates the product for readiness, generates copy of product material and distributes them to two or three review members for advance 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

Review reporting and Record keeping
During the FTR, a reviewer(recorder) records all issues that have been raisedA 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 upto 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:

1. Define customer requirements, deliverables, and project goals via well-defined methods of customer communication.
2. Measure each existing process and its output to determine current quality performance (e.g., compute defect metrics)
3. Analyze defect metrics and determine vital few causes.

For an existing process that needs improvement

1. Improve process by eliminating the root causes for defects
2. Control future work to ensure that future work does not reintroduce causes of defects

1. Design each new process to avoid root causes of defects and to meet customer requirements
2. 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 + MTTRMTTF = 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.

Types of ISO 9000 Quality Standards

The ISO 9000 series of standards is based on the assumption that if a proper stage is followed for production, then good quality products are bound to follow automatically. The types of industries to which the various ISO standards apply are as follows.

1. **ISO 9001:** This standard applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that applies to most software development organizations.

2. **ISO 9002:** This standard applies to those organizations which do not design products but are only involved in the production. Examples of these category industries contain steel and car manufacturing industries that buy the product and plants designs from external sources and are engaged in only manufacturing those products. Therefore, ISO 9002 does not apply to software development organizations.

3. **ISO 9003:** This standard applies to organizations that are involved only in the installation and testing of the products. For example, Gas companies.

An organization determines to obtain ISO 9000 certification applies to ISO registrar office for registration. The process consists of the following stages:

1. **Application:** Once an organization decided to go for ISO certification, it applies to the registrar for registration.

2. **Pre-Assessment:** During this stage, the registrar makes a rough assessment of the organization.

3. **Document review and Adequacy of Audit:** During this stage, the registrar reviews the document submitted by the organization and suggest an improvement.

4. **Compliance Audit:** During this stage, the registrar checks whether the organization has compiled the suggestion made by it during the review or not.

5. **Registration:** The Registrar awards the ISO certification after the successful completion of all the phases.
6. **Continued Inspection:** The registrar continued to monitor the organization time by time.