SOFTWARE ENGINEERING

LECTURE NOTES (Semester – IV)

for

B**achelor of **Sc**ience in **Computer Science

Department of Computer Science and Applications

School of Arts & Science

Vinayaka Mission's Research Foundation

A V Campus, Chennai – 603104.

Lecture Notes Prepared by:

A.VIJAYA KUMAR, Asst. Professor,

SYLLABUS

UNIT-I (12)

Software Process: Introduction ,S/W Engineering Paradigm , life cycle models (water fall, incremental, spiral, evolutionary, prototyping, object oriented) , System engineering, computer based system, verification, validation, life cycle process, development process, system engineering hierarchy.

UNIT-II (12)

Software requirements: Functional and non-functional , user, system, requirement engineering process, feasibility studies, requirements, elicitation, validation and management, software prototyping, prototyping in the software process, rapid prototyping techniques, user interface prototyping, S/W document. Analysis and modeling, data, functional and behavioral models, structured analysis and data dictionary.

UNIT-III (12)

Design Concepts and Principles: Design process and concepts, modular design, design heuristic, design model and document, Architectural design, software architecture, data design, architectural design, transform and transaction mapping, user interface design, user interface design principles. Real time systems, Real time software design, system design, real time executives, data acquisition system, monitoring and control system.

UNIT-IV (12)

Software Configuration Management: The SCM process, Version control, Change control, Configuration audit, SCM standards. **Software Project Management:** Measures and measurements, S/W complexity and science measure, size measure, data and logic structure measure, information flow measure. Estimations for Software Projects, Empirical Estimation Models, Project Scheduling.

UNIT-V (12)

Testing: Taxonomy of software testing, levels, test activities, types of s/w test, black box testing testing boundary conditions, structural testing, test coverage criteria based on data flow, mechanisms, regression testing, testing in the large. S/W testing strategies, strategic approach and issues, unit testing, integration testing, validation testing, system testing and debugging.

Books Recommended:

1. Roger S.Pressman, Software engineering- A practitioner’s Approach, McGraw-Hill
2. Ian Sommerville, Software engineering, Pearson education Asia, 6th edition, 2000.
3. Pankaj Jalote- An Integrated Approach to Software Engineering, Springer Verlag, 1997.
4. James F Peters and WitoldPedryez, “Software Engineering – An Engineering Approach”, John Wiley and Sons, New Delhi, 2000.
5. Ali Behforooz and Frederick J Hudson, “Software Engineering Fundamentals”, Oxfor University Press, New Delhi, 1996.
6. Pfleeger, ”Software Engineering”, Pearson Education India, New Delhi, 1999.
7. Carlo Ghezzi, Mehdi Jazayari and Dino Mandrioli, “Fundamentals of Software Engineering”, Prentice Hall of India, New Delhi, 1991.

Unit I: Software Process

Definitions of Software Engineering

According to Firtz Bauer – "Software engineering is defined as "The establishment and use of sound engineering principles in order to obtain the software that is economical, reliable and work efficiently on real machines".

According to Boehm – "Software engineering is the application of science and mathematics by which capabilities of computer equipments are made useful to man via computer programs, procedures and related documentation".

According to IEEE – Software engineering is the systematic approach to the development, operation, maintenance and retirement of the software.

"Software engineering is the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates".

Goals of Software Engineering

The primary goals of Software Engineering is

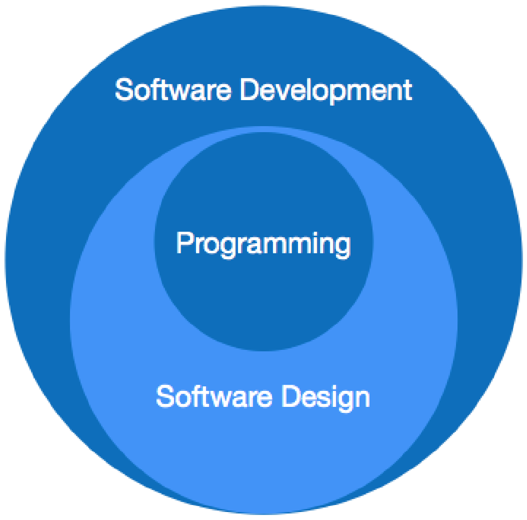
1. To improve the quality of software.
2. To increase productivity.
3. To increase the job satisfaction of software engineers.

Software engineering is based on the following disciplines

1. Computer Science.
2. Management Science.
3. Economics.
4. Communication Skills.
5. Engineering approach to problem solving.

<u>Software Paradigms</u>

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another:



Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design

- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes –

- Coding
- Testing
- Integration

Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software -** It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost-** As hardware industry has shown its skills and huge manufacturing has lower down he price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature-** The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management-** Better process of software development provides better and quality software product.

<u>Software Life Cycle Models (Software Engineering Paradigm)</u>

1**. <u>Water Fall Model</u> (Linear Sequential Model)**

1) It is a simplest model, which states that the phases are organized in a linear order.

2) The model was originally proposed by Royce.
3) The various phases in this model are

a) Feasibility Study – The main aim of the feasibility study activity is to determine whether it would be financially and technically feasible to develop the product. The feasibility study activity involves the analysis of the problem and collection of all relevant information relating to the product such as the different data items which would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system as well as various constraints on the behavior of the system.

b) Requirement Analysis – The aim of the requirement analysis is to understand the exact requirements of the customer and to document them properly. The requirement analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. During this activity, the user requirements are systematically organized into a software requirement specification (SRS) document. The important components of this document are the functional requirements, the nonfunctional requirements, and the goals of implementation. The SRS document is written in end-user terminology. This makes the SRS document understandable by the customer. After all, it is important that the SRS document be reviewed and approved by the customer.

c) Design – This phase is concerned with
1) Identifying software components like functions, data streams and data stores.
2) Specifying software structure.
3) Maintaining a record of design decisions and providing blue prints for the implementation phase
4) The are two categories of Design
a) Architectural Design – It involves
    1. Identifying the software components.
    2. Decoupling and decomposing the software components into modules and conceptual data structures.
    3. Specifying the interconnection between the various components.

b) Detailed Design – It is concerned with details of the implementation procedures to process the algorithms, data structures and interaction between the modules and data structures. The various activities that this phase includes are
    1. Adaptation of existing code.
    2. Modification of existing algorithms.

3. Design of data representation and
4. Packaging of the software product.

   This process is highly influenced by the programming language under the implementation would be carried out. This stage is different from implementation.

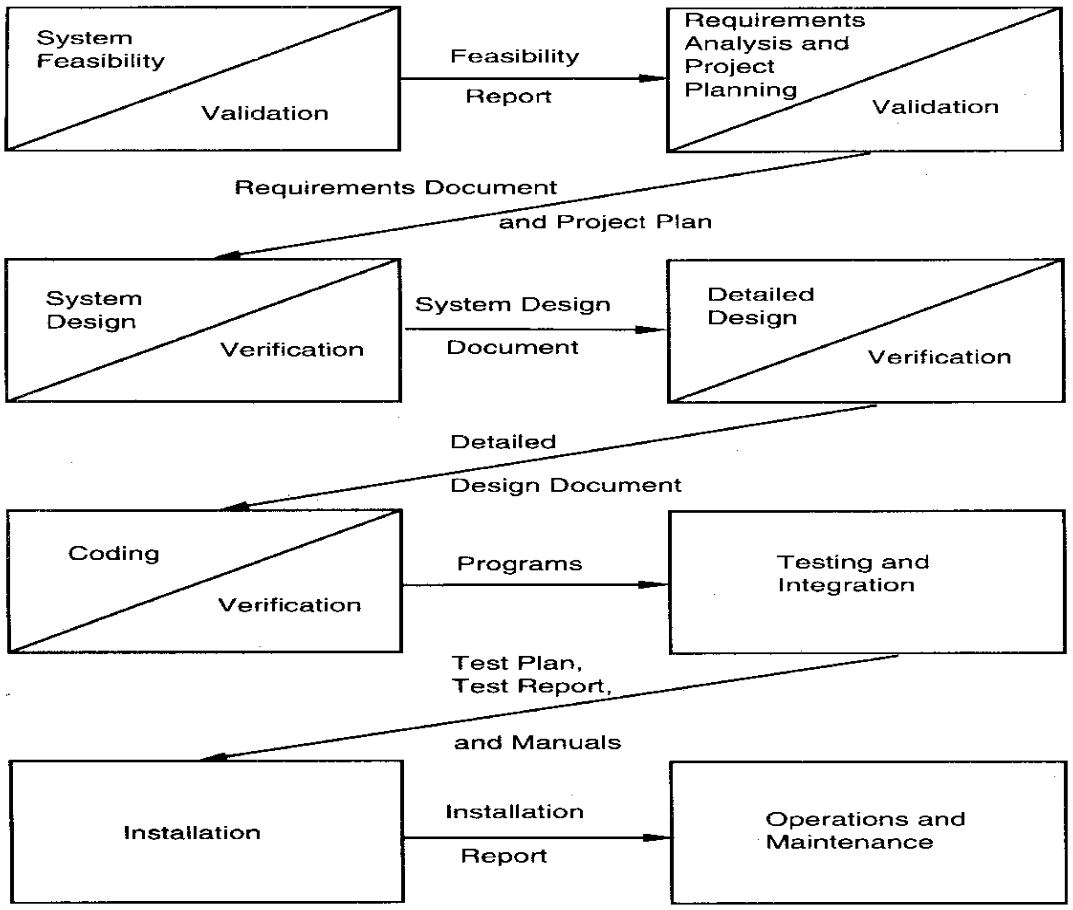
d) Implementation – It involves the translation of the design specifications into source code. It also involves activities like debugging, documentation and unit testing of the source code. In this stage various styles of programming can be followed like built-in and user defined data types, secure type checking, flexible scope rules, exception handling, concurrency control etc.

e) Testing – It involves two kinds of activities

1. Integration Testing – It involves testing of integrating various modules and testing there overall performance due to their integration.
2. Acceptance Testing – It involves planning and execution of various types of tests in order to demonstrate that the implemented software system satisfies the requirements stated in the requirements document.

f) Maintenance – In this phase the activities include

1. Corrective Maintenance – Correcting errors that were not discovered during the product development phase.
2. Perfective Maintenance – Improving the implementation of the system, and enhancing the functionalities of the system according to customer's requirements.
3. Adaptive Maintenance – Adaptation of software to new processing environments.



Advantages of Water Fall model

1. All phases are clearly defined.
2. One of the most systematic methods for software development.
3. Being oldest, this is one of the time tested models.
4. It is simple and easy to use.

Disadvantages of Water Fall Model

1. Real Projects rarely follow sequential model.
2. It is often difficult for the customer to state all requirements explicitly.
3. Poor model for complex and object oriented projects.
4. Poor model for long and ongoing projects.
5. High amount of risk and uncertainty.
6. Poor model where requirements are at a moderate to high risk of changing.
7. This model is suited to automate the existing manual system for which all requirements are known before the design starts. But for new system, having such

unchanging requirements is not possible.

2. <u>Incremental Model</u>

1. The *incremental model* combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping.
2. It is iterative in nature.
3. It focuses on the delivery of the operational product with each increment
4. The process is repeated until the complete product is produced.
5. For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment. It should be noted that the process flow for any increment can incorporate the prototyping paradigm.
6. When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered.
7. The core product is used by the customer (or undergoes detailed review). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.
8. This process is repeated following the delivery of each increment, until the complete product is produced.
9. Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project.

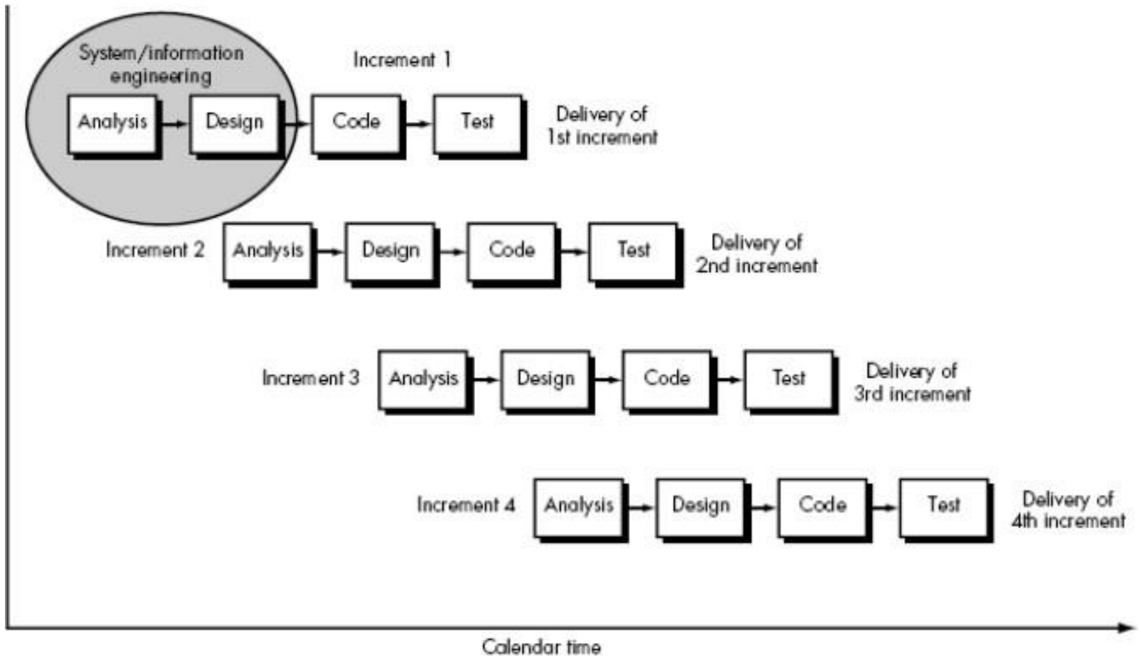
Advantages

1. It provides on the rigid nature of sequential approach.
2. This method is of great help when organization is low on staffing.
3. Generates working software quickly and early during the software life cycle.
4. More flexible – less costly to change scope and requirements.
5. Easier to test and debug during a smaller iteration.
6. Easier to manage risk because risky pieces are identified and handled during its iteration.
7. Each iteration is an easily managed milestone.

Disadvantages

1. This model could be time consuming.

2. Each phase of an iteration is rigid and do not overlap each other.
3. Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.


System/information engineering
Increment 1
Analysis
Design
Code
Test
Delivery of 1st increment
Increment 2
Delivery of 2nd increment
Increment 3
Delivery of 3rd increment
Increment 4
Delivery of 4th increment
Calendar time

3. <u>Spiral Model</u>

1. Evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.
2. It provides the potential for rapid development of incremental versions of the software.
3. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.
4. A spiral model is divided into a number of framework activities, also called *task*

regions.

5. There are six task regions.

   1. Customer communication—tasks required to establish effective communication between developer and customer.

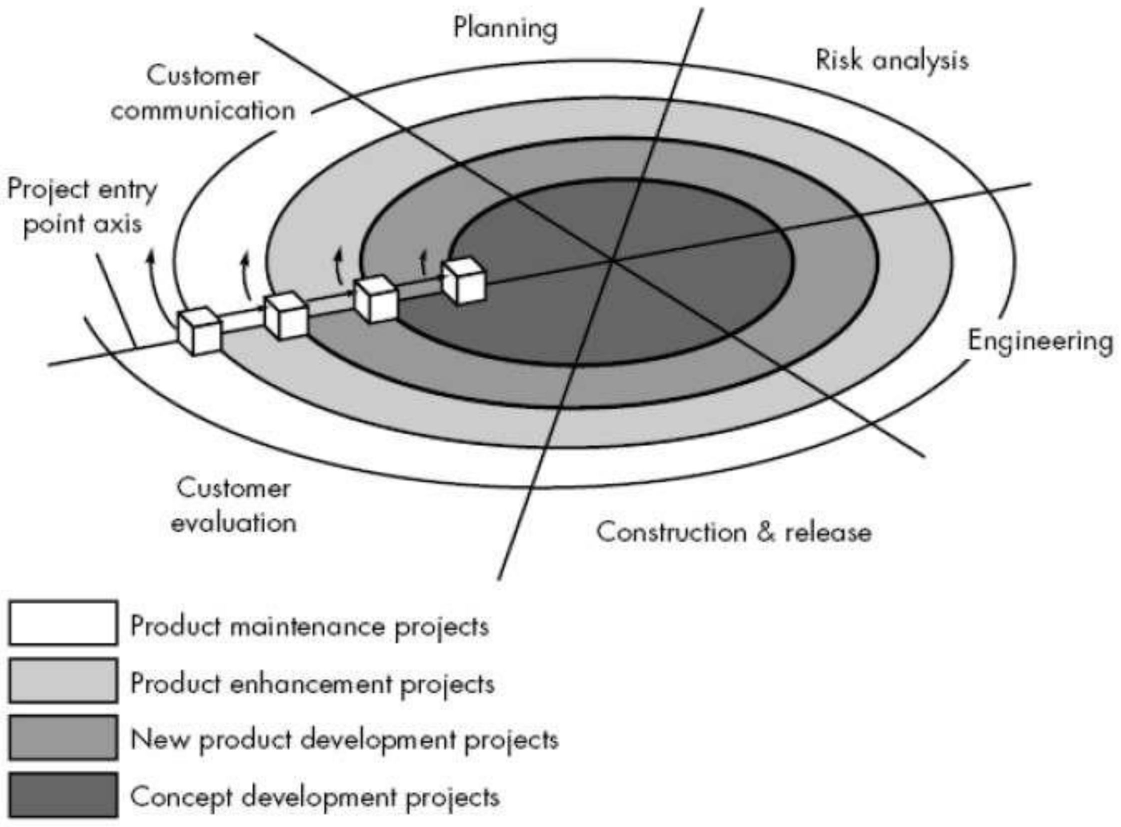
   2. Planning—tasks required to define resources, timelines, and other project related information.

   3. Risk analysis—tasks required to assess both technical and management risks.

   4. Engineering—tasks required to build one or more representations of the application.

   5. Construction and release—tasks required to construct, test, install, and provide user support (e.g., documentation and training).

   6. Customer evaluation—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.



Advantages:

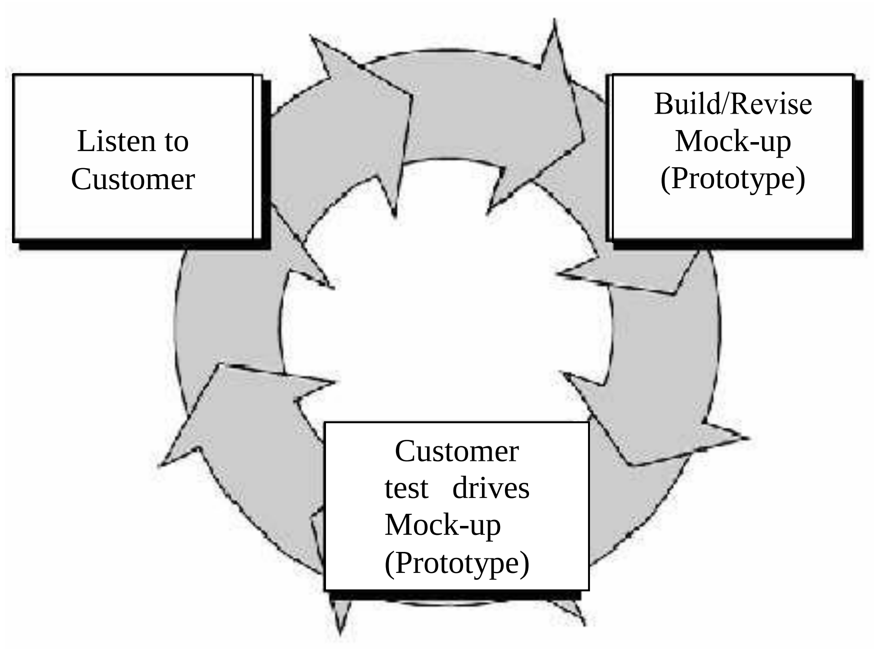
1. It is a realistic approach for development of large scale system.
2. High amount of risk analysis
3. Good for large and mission-critical projects.

4. Software is produced early in the software life cycle.

Disadvantages:

1. It is not widely used.
2. It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable.
3. It demands considerable risk assessment expertise and relies on this expertise for success. If a major risk is not uncovered and managed, problems will undoubtedly occur.
4. Can be a costly model to use.
5. Risk an alysis requires highly specific expertise.
6. Project’s success is highly dependent on the risk analysis phase.
7. Doesn’t work well for smaller projects.

4. Prototyping Model


Listen to Customer
Build/Revise Mock-up (Prototype)
Customer test drives Mock-up (Prototype)

- Often, a customer defines a set of general objectives for software, but does not identify detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these and many other situations, a prototyping paradigm may offer the best approach.
- The various phases of this model are
  - Listen to Customer: - This is the starting step, where the developer and customer together
    - Define the overall objectives for the software,
    - Identify the known requirements and
    - The analyst then outlines those factors and requirements that are not visible normally but are mandatory from development point of view.
  - Build prototype: - After the identification of the problem a quick design is done which will cause the software show the output that the customer wants. This quick design leads to the development of a prototype (a temporary working model).
  - Customer test drives the prototype: - The customer runs and checks the prototype for its perfection. The prototype is evaluated by the customer and further improvements are made to the prototype unless the customer is satisfied.

- All the stages are repeated until the customer gets satisfied. When the final prototype is fully accepted by the customer then final development processes like proper coding for attaining maintainability, proper documentation, testing

for robustness etc. are carried out. And finally the software is delivered to the customer.

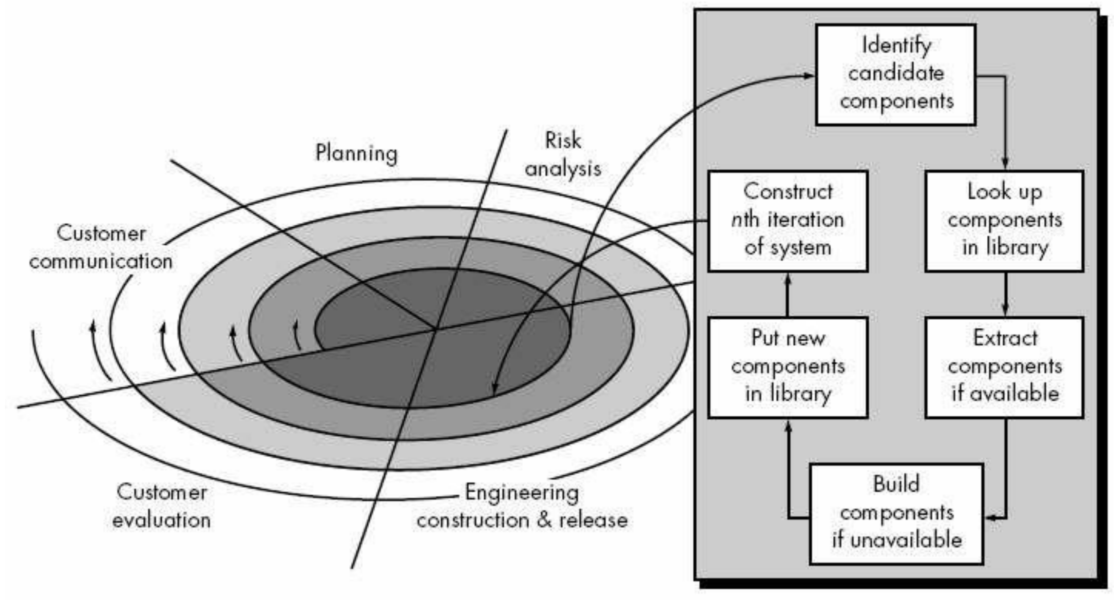
Advantages

1. It serves as the mechanism for identifying the requirements.
2. The developer use the existing program fragment means it helps to generate the software quickly.
3. Continuous developer – Consumer communication is available.

Disadvantages

1. Customer considers the prototype as an original working version of the software.
2. Developer makes implementation compromise in order to get prototype working quickly.
3. This model is time consuming.

5. <u>Object Oriented Model (Component Assembly Model)</u>


Planning
Risk analysis
Customer communication
Customer evaluation
Engineering construction & release
Identify candidate components
Construct nth iteration of system
Look up components in library
Put new components in library
Extract components if available
Build components if unavailable

1) This kind of model is used for object based (oriented) development process.

2) Here the emphasis is on the creation of reusable classes that encapsulate both data and functions, where the functions can to manipulate the data.

3) This model incorporates many of the characteristics of spiral model and iterative approach towards software development.
4) During the Engineering phase and Construction & release phase, the data is examined and accordingly algorithms applied for data manipulation is also decided. Corresponding data and algorithms are encapsulated into a Class.

5) Classes created in past are stored in a class library. Once the required classes are identified, the classes libraries are searched, if they are found then they are reused. In case where the required classes are not found in the library then they are engineered using object oriented methods.

6) When the process of recognition and usage of classes is done then the development process returns back to the spiral path and subsequently if required reenter in the component assembly process during successive iterations, unless fully acceptable product is made.

<u>Difference between WaterFall model and Spiral model</u>

No.	WaterFall Model	Spiral Model
1	It is a Systematic, sequential approach to software development that begins at system level and progress through analysis, design, coding, testing, and maintenance.	In this model software is developed in a series of incremental versions by the process of iteration through a number of framework activities like Customer communication, planning, risk analysis, engineering, construction engineers and customer evaluation.
2	It is the oldest and the most widely used paradigm in software engineering.	It is an evolutionary approach to software engineering and is currently gaining foothold.
3	It does not include provisions for risk assessment.	Risk assessment is one of the major activities of the spiral model.
4	It is not convenient to model real life large scale projects since most of them do not follow a sequential flow that the model proposes.	Spiral model incorporates the steps of the waterfall model in an interactive framework and hence can more realistically reflect the real world system.

5	This model requires the specification of all requirements at a very early stage in the development, which is often difficult to do.	The Spiral model iterates through the stages of development several times, each time providing a more improved version of the previous stage, additional requirements can be incorporated at later stages of development.
6	The effects of major risks taken can be gauges only at a very later stage of development and by then it would be too terminating the project.	Spiral model requires considerations of risks at all stages of development and if risk are too great, the process can be terminated at a much earlier stage.
7	A working version of the program is available at a much later stage of the project.	A skeleton working model of the system can be developed at earlier stage, which is the refined in successive interaction.
8	No planning is done.	Every pass through planning results in the adjustment of large scale systems of the customer.
9	Entry point is not present.	Entry points are specified.

Software Development Process

In the software development process, we have to focus on the activities directly related to production of the software, for example, design, coding, and testing. A development process model specifies some activities that, according to the model, should be performed, and the order in which they should be performed. For cost, quality, and project management reasons, development processes are generally phased.

As the development process specifies the major development and quality assurance activities that need to be performed in the project, the development process really forms the core of the software process. The management process is decided, based on the development process. Due to the importance of development process, various models have been proposed. As processes consist of a sequence of steps, let us first discuss what should be specified for a step.

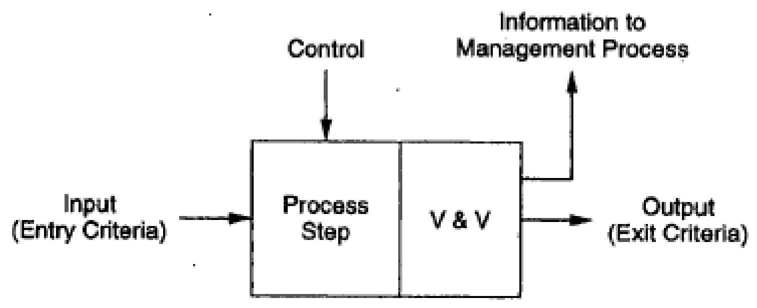
A Process Step Specification

A production process is a sequence of steps. Each step performs a well-defined activity leading towards the satisfaction of the project goals, with the output of one step forming the input of the next one. Most process models specify the steps that need to be performed and the order in which they need to be performed. However, when implementing a process model, there are some practical issues, like when to initiate a step and when to terminate a step, that need to be addressed. Here we discuss some of these issues.

A process should aim to detect defects in the phase in which they are introduced. This requires that there be some verification and validation (V & V) at the end of each step. (In verification, consistency with the inputs of the phase is checked, while in validation the consistency with the needs of user is checked.) This implies that there is a clearly defined output of a phase, which can be verified by some means and can form input to the next phase (which may be performed by other people). In other words, it is not acceptable to say that the output of a phase is an idea or a thought in the mind of someone; the output must be a formal and tangible entity. Such outputs of a development process, which are not the final output, are frequently called the work products. In software, a work product can be the requirements document, design document, code, prototype, etc. This restriction that the output of each step should be some work product, that can be verified, suggests that the process should have a small number of steps. Having too many steps results in too many work products or documents, each requiring V & V, and can be very expensive. Due to this, at the top level, a development process typically consists of a few steps, each satisfying a clear objective and producing a document used for V & V. How to perform the activity of the particular step or phase is generally not an issue of the development process.

As a development process, typically, contains a sequence of steps, the next issue that comes is when a phase should be initiated and terminated. This is frequently done by specifying the entry criteria and exit criteria for a phase. The entry criteria of a phase specify the conditions that the input to the phase should satisfy in order to initiate the activities of that phase. The output criteria specify the conditions that the work product of this phase should satisfy in order to terminate the activities of the phase. The entry and exit criteria specify constraints of when to start and stop an activity. It should be clear that the entry criteria of a phase should be consistent with the exit criteria of the previous phase.

The entry and exit criteria for a phase in a process depend largely on the implementation of the process. For example, for the same process, one organization may have the entry criteria for the design phase as "requirements document signed by the client" and another may have "no more than X errors detected per page in the requirement review." As each phase ends with some V & V activity, a common exit criteria for a phase is "V & V of the phase completed satisfactorily," where satisfactorily is defined by the organization based on the objectives of the project and its experience in using the process. The specification of a step with its input, output, and entry exit criteria is shown in Figure below.



A step in the development process

Verification & Validation (V&V)

The goals of verification and validation activities are to asses and improve the quality of the work products generated during development and modification of software. The quality attributes include – a) Correctness, b) Completeness, c) Consistency, d) Reliability, e) Usefulness, f) Conformance to standards and g) overall cost effectiveness.

Verification: - It's the process of determining whether the product is built in a right manner or not. There are two categories of verification: -
a) Life-cycle verification and
b) Formal verification.

a) Life-Cycle verification: - It's the process of determining the degree to which the work products of a given phase of the development cycle fulfill the specifications established during prior phases.
b) Formal verification: - It's a rigorous mathematical demonstration that whether the source code conforms to the specification.

Validation: - It's the process of evaluation of the software at the end of the software development process to determine compliance with the requirements. In other words it's the process of determining whether the correct product is built or not.

Verification and validation involve the assessment of work products to determine conformance to the specifications. Specifications include –

a) Requirements specification
b) The design documentation
c) Various stylistic guidelines
d) Implementation language standards
e) Project standards
f) Organizational standards
g) User expectations.

Verification

1. Verification is done to ensure that the work product of a given phase of the development cycle fulfill the specifications established during prior phase.
2. According to Boehm :

   Verification: Are we producing the product right?

3. Verification ensures the product is designed to deliver all functionality to the customer; it typically involves reviews and meetings to evaluate documents, plans, code, requirements and specifications; this can be done with checklists, issues lists, and walkthroughs and inspection meetings.
4. In verification uncovering of defects will be done in primary ways. Here no code will be executed. Before building actual system this checking will be done. This process will be called Quality Assurance.
5. Verification is nothing but the Static Testing.
6. Inputs to the verification are check list, issue list, walkthroughs and inspection meetings, reviews and meetings.
7. The output of the verification is a nearly a perfect set of documents, plans, specifications and requirements document.
8. According to the CMM, Validation - The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

Validation

1. Validation is the process of evaluating software at the end of the software development process to determine compliance with the requirements.
2. According to Boehm:

   Validation: Are we producing the right product?

3. Validation ensures that functionality, as defined in requirements, is the intended behavior of the product; validation typically involves actual testing and takes place after verifications are completed.
4. Validation concern, checking will be done by executing code for errors (defects.). This can be called as Quality Control.
5. Validation is nothing but the Dynamic Testing.
6. The input on the validation on the other hand is the actual testing of an actual product.
7. The output of the validation on the other is a nearly perfect, actual product.

8. According to CMM, Verification- The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

<u>Difference between Verification and Validation</u>

No.	Verification	Validation
1	Verification is done to ensure that the work product of a given phase of the development cycle fulfill the specifications established during prior phase.	Validation is the process of evaluating software at the end of the software development process to determine compliance with the requirements.
2	Verification: Are we producing the product right?	Validation: Are we producing the right product?
3	Verification ensures the product is designed to deliver all functionality to the customer; it typically involves reviews and meetings to evaluate documents, plans, code, requirements and specifications; this can be done with checklists, issues lists, and walkthroughs and inspection meetings.	Validation ensures that functionality, as defined in requirements, is the intended behavior of the product; validation typically involves actual testing and takes place after verifications are completed.
4	In verification uncovering of defects will be done in primary ways. Here no code will be executed. Before building actual system this checking will be done. This process will be called Quality Assurance.	Validation concern, checking will be done by executing code for errors (defects). This also can be called as Quality Control.
5	Verification is nothing but the Static Testing.	Validation is nothing but the Dynamic Testing.
6	Verification is done before validation.	Validation cannot be done before verification.
7	Inputs to the verification are check list, issue list, walkthroughs and inspection meetings, reviews and meetings.	The input on the validation on the other hand is the actual testing of an actual product.
8	The output of the verification is a nearly a perfect set of documents, plans, specifications and requirements document.	The output of the validation on the other is a nearly perfect, actual product.

System

1. A set or arrangement of things so related as to form a unity or organic whole.
2. A set of facts, principles, rules, etc., classified and arranged in an orderly form so as to show a logical plan linking the various parts.
3. A method or plan of classification or arrangement.
4. An established way of doing something, method, procedure….

Computer Based System

"A set of arrangements of elements that are organized to accomplish some predefined goal by processing information."

The goal may be to support some business function or to develop a product that can be said to generate business revenue. To accomplish the goal, a computer based system makes use of a variety of system elements:

<u>Software</u> – Computer programs, data structures, and related work products that serve to affect the logical method, procedure, or control that is required.

<u>Hardware</u> – Electronic devices that provide computing capability, the interconnectivity devices (e.g., network switches, telecommunications devices) that enable the flow of data, and electromechanical devices (e.g., sensors, motors, pumps) that provide external world function.

<u>People</u> – Users and operators of hardware and software.

<u>Database</u> – A large, organized collection of information that is accessed via software and persists over time.

<u>Documentation</u> – Descriptive information (e.g., models, specifications, hardcopy manuals, online help files, web sites) that portrays the use and/or operation of the system.

<u>Procedures</u> – The steps that define the specific use of each system element or the procedural context in which the system resides.

These elements combine in a variety of ways to transform information. For example, a marketing department transforms raw sales data into a profile of the typical purchaser of a product; a robot transforms a command file containing specific instructions into a set of control signals that cause some specific physical action. Creating an information system to assist the marketing department and control software to support the robot both require system engineering.

System Engineering

“System engineering is an activity of specifying, designing, implementing, validating, deploying and maintaining technical system.”

System engineers are not just concerned with software but also with hardware and the system’s interactions with users and its environment. They must think about the services that the system provides the constraints under which the system must be built and operated and the ways in which the system is used to fulfill its purpose.

There are important distinctions between the system engineering process and software development process

1. Limited scope of rework during system development – Once some system engineering decisions have been made, they are very expensive to change. Reworking the system design to solve these problems is rarely possible. One reason software has become so important in systems is that it allows changes to be made during system development, in response to new requirements.

2. Interdisciplinary involvement – Many engineering disciplines may be involved in system engineering. There is a lot of scope for misunderstanding because different engineers terminology and conventions.

System engineering hierarchy

System engineering encompasses a collection of top-down and bottom-up methods to navigate the hierarchy illustrated in fig (a). The system engineering process usually begins with a “world view”. That is, the entire business or product domain is examined to ensure that the proper business or technology context can be established. The world view is refined to focus more fully on a specific domain of interest. Within a specific domain, the need for targeted system elements (e.g., data, software, hardware, and people) is analyzed. Finally, the analysis, design, and construction of a targeted system element is initiated. At the top of the hierarchy, very broad contexts are established and, at the bottom, detailed technical activities, performed by the relevant engineering discipline (e.g., hardware or software engineering), are conducted.

Stated in a slightly more formal manner, the world view (WV) is composed of a set of domain (D_j), which can each be a system or system of systems in its own right.

WV = {D1, D2, D3… Dn)

Each domain is composed of specific elements (Ej) each of which serves some role in accomplishing the objective and goals of the domain or component:

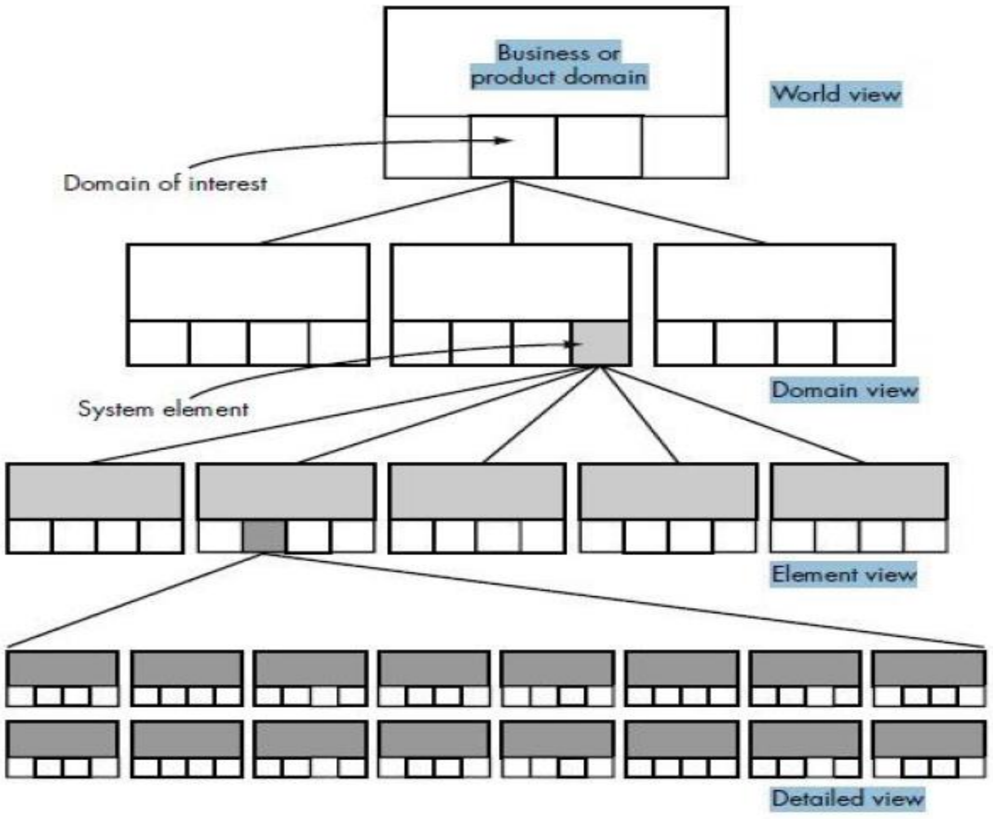
Di = {E1, E2, E3… En}

Finally, each element is implemented by specifying the technical components (Ck) that achieve the necessary function for an element:

Ej = {C1, C2, C3… Cn}

In the software context, a component could be a computer program, a reusable program component, a module, a class or object, or even a programming language statement.

It is important to note that the system engineer narrows the focus of work as she moves downward in the hierarchy. However, the world view portrays a clear definition of overall functionality that will enable the engineer to understand the domain, and ultimately the system or product, in the proper context.



Unit II: Software Requirements

The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed
Requirements may be functional or non-functional
Functional requirements describe system services or functions
Non-functional requirements is a constraint on the system or on the development process

Types of requirements
User requirements
Statements in natural language (NL) plus diagrams of the services the system provides and its operational constraints. Written for customers
System requirements
A structured document setting out detailed descriptions of the system services. Written as a contract between client and contractor
Software specification
A detailed software description which can serve as a basis for a design or implementation. Written for developers

Functional and Non-Functional

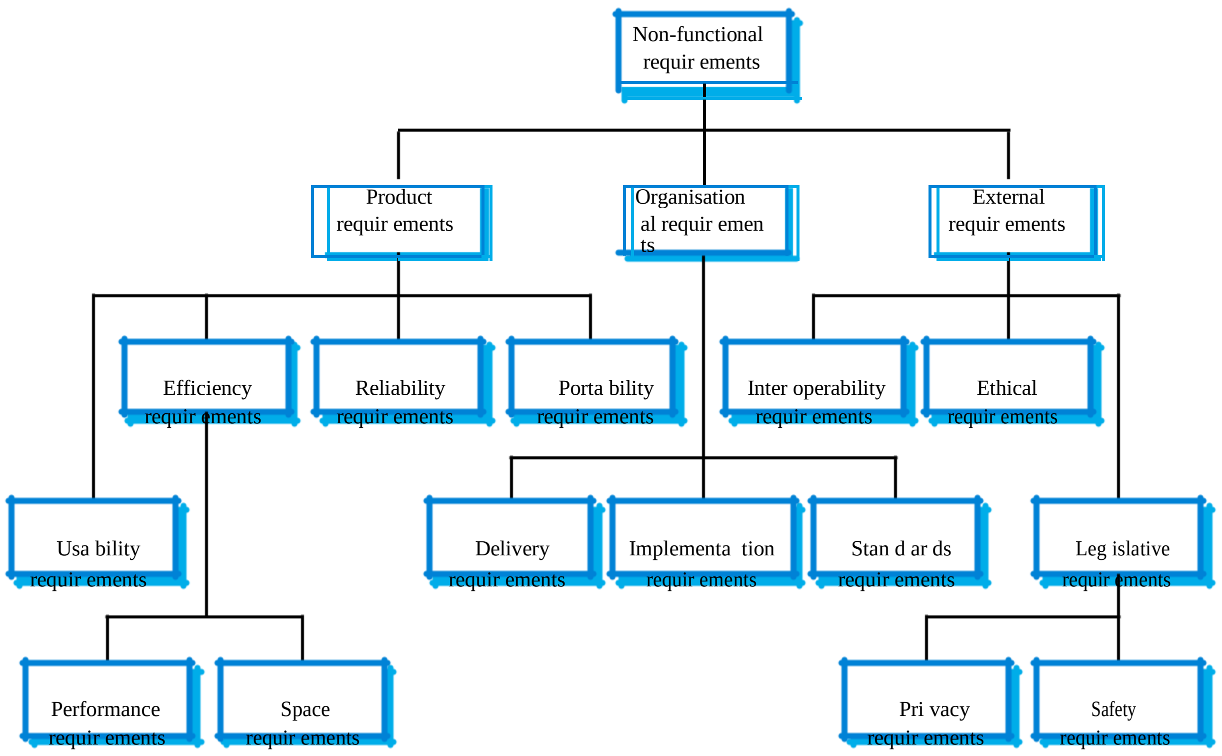
Functional requirements
Functionality or services that the system is expected to provide.
Functional requirements may also explicitly state what the system shouldn‘t do.
Functional requirements specification should be:
Complete: All services required by the user should be defined
Consistent: should not have contradictory definition (also avoid ambiguity don‘t leave room for different interpretations)

Examples of functional requirements
The LIBSYS system
A library system that provides a single interface to a number of databases of articles in different libraries.
Users can search for, download and print these articles for personal study.
The user shall be able to search either all of the initial set of databases or select a subset from it.
The system shall provide appropriate viewers for the user to read documents in the document store.
Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account‘s permanent storage area.

Non-Functional requirements

Requirements that are not directly concerned with the specific functions delivered by the system
Typically relate to the system as a whole rather than the individual system features
Often could be deciding factor on the survival of the system (e.g. reliability, cost, response time)

Non-Functional requirements classifications:


Non-functional requir ements
Product requir ements
Organisation al requir emen ts
External requir ements
Efficiency requir ements
Reliability requir ements
Porta bility requir ements
Inter operability requir ements
Ethical requir ements
Usa bility requir ements
Delivery requir ements
Implementa tion requir ements
Stan d ar ds requir ements
Leg islative requir ements
Performance requir ements
Space requir ements
Pri vacy requir ements
Safety requir ements

Domain requirements

Domain requirements are derived from the application domain of the system rather than from the specific needs of the system users.
May be new functional requirements, constrain existing requirements or set out how particular computation must take place.
Example: tolerance level of landing gear on an aircraft (different on dirt, asphalt, water), or what happens to fiber optics line in case of sever weather during winter Olympics (Only domain-area experts know)

Product requirements

Specify the desired characteristics that a system or subsystem must possess.
Most NFRs are concerned with specifying constraints on the behaviour of the executing system.

Specifying product requirements

Some product requirements can be formulated precisely, and thus easily quantified
Performance
Capacity

Others are more difficult to quantify and, consequently, are often stated informally
Usability

Process requirements

Process requirements are constraints placed upon the development process of the system
Process requirements include:
Requirements on development standards and methods which must be followed
CASE tools which should be used
The management reports which must be provided

Examples of process requirements

The development process to be used must be explicitly defined and must be conformant with ISO 9000 standards
The system must be developed using the XYZ suite of CASE tools
Management reports setting out the effort expended on each identified system component must be produced every two weeks
A disaster recovery plan for the system development must be specified

External requirements

May be placed on both the product and the process
Derived from the environment in which the system is developed
External requirements are based on:
application domain information
organisational considerations
the need for the system to work with other systems
health and safety or data protection regulations
or even basic natural laws such as the laws of physics

Examples of external requirements

Medical data system The organisation‘s data protection officer must certify that all data is maintained according to data protection legislation before the system is put into operation.
Train protection system The time required to bring the train to a complete halt is computed using the following function:
The deceleration of the train shall be taken as:

where: $g_{train} = g_{control} + g_{gradient}$

$g_{gradient}$ = 9.81 ms^{-2} * compensated gradient / alpha and where the values of *9.81 ms*$^{-2/}$ *alpha* are known for the different types of train.

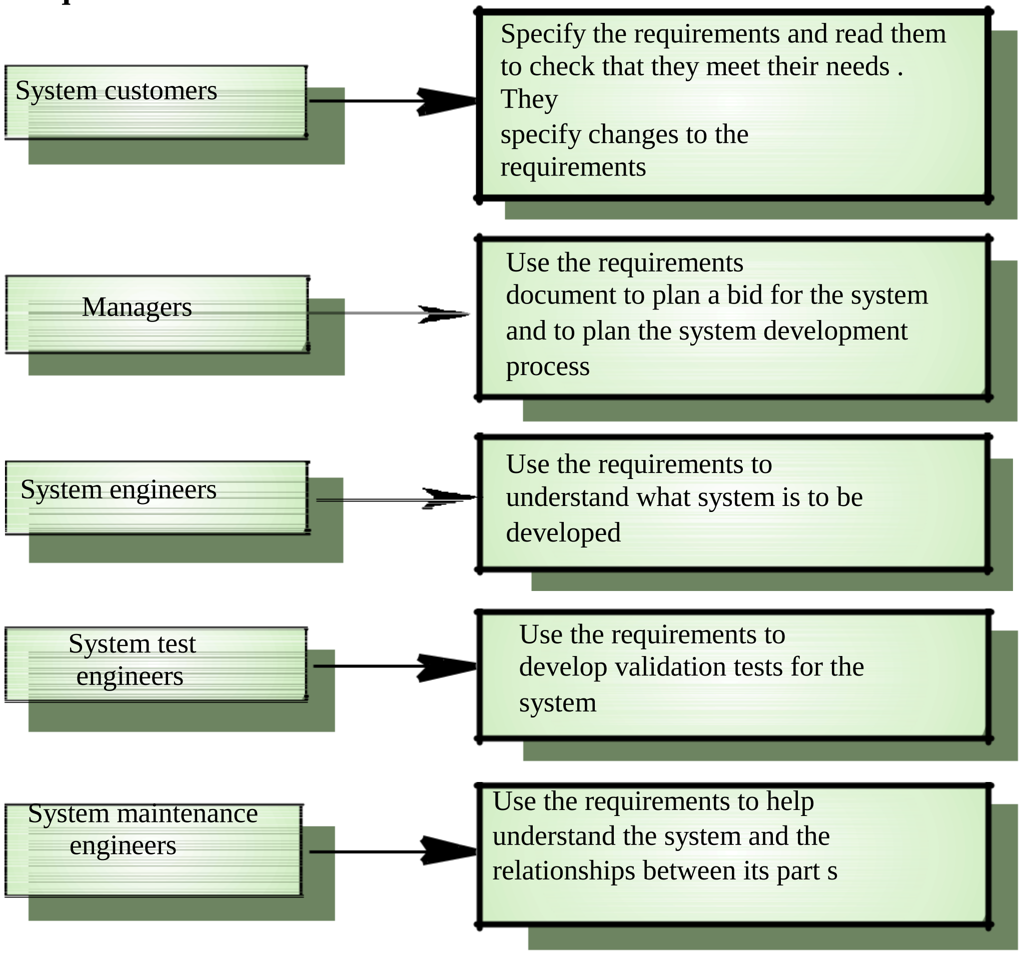
$g_{control}$ is initialised at 0.8 ms^{-2} - this value being parameterised in order to remain adjustable. The illustrates an example of the train‘s deceleration by using the parabolas derived from the above formula where there is a change in gradient before the (predicted) stopping point of the train.

Software Document

Should provide for communication among team members

Should act as an information repository to be used by maintenance engineers
Should provide enough information to management to allow them to perform all program management related activities
Should describe to users how to operate and administer the system
Specify external system behaviour
Specify implementation constraints
Easy to change
Serve as reference tool for maintenance
Record forethought about the life cycle of the system i.e. predict changes
Characterise responses to unexpected events

Users of a requirements document


System customers
Specify the requirements and read them to check that they meet their needs . They specify changes to the requirements
Managers
Use the requirements document to plan a bid for the system and to plan the system development process
System engineers
Use the requirements to understand what system is to be developed
System test engineers
Use the requirements to develop validation tests for the system
System maintenance engineers
Use the requirements to help understand the system and the relationships between its part s

Process Documentation

Used to record and track the development process
- Planning documentation
- Cost, Schedule, Funding tracking
- Schedules
- Standards

- This documentation is created to allow for successful management of a software product
- Has a relatively short lifespan
    - Only important to internal development process
    - Except in cases where the customer requires a view into this data
- Some items, such as papers that describe design decisions should be extracted and moved into the *product* documentation category when they become implemented
    - Product Documentation
- Describes the delivered product
- Must evolve with the development of the software product
- Two main categories:
    - System Documentation
    - User Documentation

Product Documentation

- System Documentation
    - Describes how the system works, but not how to operate it
- Examples:
    - Requirements Spec
    - Architectural Design
    - Detailed Design
    - Commented Source Code
        - Including output such as JavaDoc
    - Test Plans
        - Including test cases
    - V&V plan and results
    - List of Known Bugs
- User Documentation has two main types
    - End User
    - System Administrator
        - In some cases these are the same people
    - The target audience must be well understood!
- There are five important areas that should be documented for a formal release of a software application
    - These do not necessarily each have to have their own document, but the topics should be covered thoroughly
- Functional Description of the Software
- Installation Instructions
- Introductory Manual
- Reference Manual
- System Administrator‘s Guide

Document Quality

- Providing thorough and professional documentation is important for any size product development team

The problem is that many software professionals lack the writing skills to create professional level documents

Document Structure

All documents for a given product should have a similar structure

A good reason for product standards

The IEEE Standard for User Documentation lists such a structure

It is a superset of what most documents need

The authors ―best practices‖ are:

Put a cover page on all documents

Divide documents into chapters with sections and subsections

Add an index if there is lots of reference information

Add a glossary to define ambiguous terms

Standards

Standards play an important role in the development, maintenance and usefulness of documentation

Standards can act as a basis for quality documentation

But are not good enough on their own

Usually define high level content and organization

There are three types of documentation standards

1.Process Standards

Define the approach that is to be used when creating the documentation

Don‗t actually define any of the content of the documents

2. Product Standards

Goal is to have all documents created for a specific product attain a consistent structure and appearance

Can be based on organizational or contractually required standards

Four main types:

Documentation Identification Standards

Document Structure Standards

Document Presentation Standards

Document Update Standards

One caveat:

Documentation that will be viewed by end users should be created in a way that is best consumed and is most attractive to them

Internal development documentation generally does not meet this need

3. Interchange Standards

Deals with the creation of documents in a format that allows others to effectively use

PDF may be good for end users who don‗t need to edit

Word may be good for text editing

Specialized CASE tools need to be considered

This is usually not a problem within a single organization, but when sharing data between organizations it can occur

This same problem is faced all the time during software integration

Other Standards

IEEE

Has a published standard for user documentation

Provides a structure and superset of content areas

Many organizations probably won‘t create documents that completely match the standard

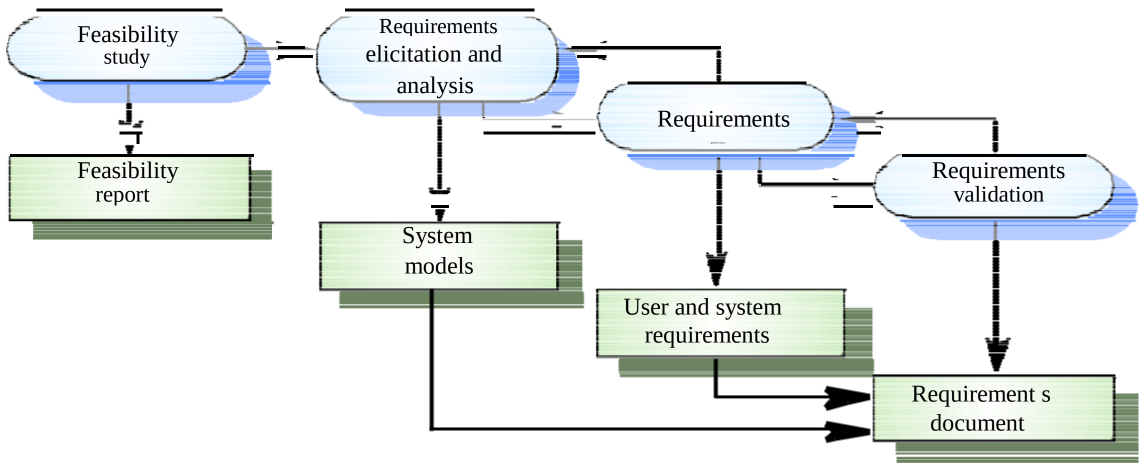
Writing Style

Ten ―best practices‖ when writing are provided

Author proposes that group edits of important documents should occur in a similar fashion to software walkthroughs

Requirement Engineering Process

The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification and requirements management



Feasibility Studies

A feasibility study decides whether or not the proposed system is worthwhile

A short focused study that checks

If the system contributes to organisational objectives

If the system can be engineered using current technology and within budget

If the system can be integrated with other systems that are used

Based on information assessment (what is required), information collection and report writing

Questions for people in the organisation

What if the system wasn‘t implemented?

What are current process problems?

How will the proposed system help?

What will be the integration problems?
Is new technology needed? What skills?
What facilities must be supported by the proposed system?

Elicitation and analysis

Sometimes called requirements elicitation or requirements discovery
Involves technical staff working with customers to find out about
- the application domain
- the services that the system should provide
- the system's operational constraints

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

Problems of requirements analysis

Stakeholders don't know what they really want
Stakeholders express requirements in their own terms
Different stakeholders may have conflicting requirements
Organisational and political factors may influence the system requirements
The requirements change during the analysis process
- New stakeholders may emerge and the business environment change

System models

Different models may be produced during the requirements analysis activity
Requirements analysis may involve three structuring activities which result in these different models
- Partitioning – Identifies the structural (part-of) relationships between entities
- Abstraction – Identifies generalities among entities
- Projection – Identifies different ways of looking at a problem

System models will be covered on January 30

Scenarios

Scenarios are descriptions of how a system is used in practice
They are helpful in requirements elicitation as people can relate to these more readily than abstract statement of what they require from a system
Scenarios are particularly useful for adding detail to an outline requirements description

Ethnography

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

Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error
- Requirements checking
  - Validity
  - Consistency
  - Completeness
  - Realism
  - Verifiability

Requirements validation techniques

- Reviews
  - Systematic manual analysis of the requirements
- Prototyping
  - Using an executable model of the system to check requirements.
- Test-case generation
  - Developing tests for requirements to check testability
- Automated consistency analysis
  - Checking the consistency of a structured requirements description

Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed
  - Different viewpoints have different requirements and these are often contradictory

Software prototyping

Incomplete versions of the software program being developed. Prototyping can also be used by end users to describe and prove requirements that developers have not considered

Benefits:

The software designer and implementer can obtain feedback from the users early in the project. The client and the contractor can compare if the software made matches the software specification, according to which the software program is built.

It also allows the software engineer some insight into the accuracy of initial project estimates and whether the deadlines and milestones proposed can be successfully met.

Process of prototyping

1. Identify basic requirements

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

2. Develop Initial Prototype

The initial prototype is developed that includes only user interfaces. (See Horizontal Prototype, below)

3. Review

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

4. Revise and Enhance the Prototype

Using the feedback both the specifications and the prototype can be improved. Negotiation about what is within the scope of the contract/product may be necessary. If changes are introduced then a repeat of steps #3 and #4 may be needed.

<u>Dimensions of prototypes</u>

1. Horizontal Prototype

It provides a broad view of an entire system or subsystem, focusing on user interaction more than low-level system functionality, such as database access. Horizontal prototypes are useful for:

Confirmation of user interface requirements and system scope

Develop preliminary estimates of development time, cost and effort.

2 Vertical Prototypes

A vertical prototype is a more complete elaboration of a single subsystem or function. It is useful for obtaining detailed requirements for a given function, with the following benefits:

Refinement database design

Obtain information on data volumes and system interface needs, for network sizing and performance engineering

<u>Types of prototyping</u>

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

1. Throwaway prototyping

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

The most obvious reason for using Throwaway Prototyping is that it can be done quickly. If the users can get quick feedback on their requirements, they may be able to refine them early in the development of the software. Making changes early in the development lifecycle is extremely cost effective since there is nothing at that point to redo. If a project is changed after a considerable work has been done then small changes could require large efforts to implement since software systems have many dependencies. Speed is crucial in implementing a throwaway prototype, since with a limited budget of time and money little can be expended on a prototype that will be discarded.

Strength of Throwaway Prototyping is its ability to construct interfaces that the users can test. The user interface is what the user sees as the system, and by seeing it in front of them, it is much easier to grasp how the system will work.

2. Evolutionary prototyping

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

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

In Evolutionary Prototyping, developers can focus themselves to develop parts of the system that they understand instead of working on developing a whole system. To minimize risk, the developer does not implement poorly understood features. The partial system is sent to customer sites. As users work with the system, they detect opportunities for new features and give requests for these features to developers. Developers then take these enhancement requests along with their own and use sound configuration-management practices to change the software-requirements specification, update the design, recode and retest.

3. Incremental prototyping

The final product is built as separate prototypes. At the end the separate prototypes are merged in an overall design.

4. Extreme prototyping

Extreme Prototyping as a development process is used especially for developing web applications. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase the services are implemented. The process is called Extreme Prototyping to draw attention to the second phase of the process, where a fully-functional UI is developed with very little regard to the services other than their contract.

Advantages of prototyping

Reduced time and costs: Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of what the user really wants can result in faster and less expensive software.

Improved and increased user involvement: Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications. The presence of the prototype being examined by the user prevents many misunderstandings and miscommunications that occur when each side believe the other understands what they said. Since users know the problem domain better than anyone on the development team does, increased interaction can result in final product that has greater tangible and intangible quality. The final product is more likely to satisfy the users‘ desire for look, feel and performance.

Disadvantages of prototyping

Insufficient analysis: The focus on a limited prototype can distract developers from properly analyzing the complete project. This can lead to overlooking better solutions, preparation of incomplete specifications or the conversion of limited prototypes into poorly engineered final projects that are hard to maintain. Further, since a prototype is limited in functionality it may not scale well if the prototype is used as the basis of a final deliverable, which may not be noticed if developers are too focused on building a prototype as a model.

User confusion of prototype and finished system: Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished. (They are, for example, often unaware of the effort needed to add error -checking and security features which a prototype may not have.) This can lead them to expect the prototype to accurately model the performance of the final system when this is not the intent of the developers. Users can also become attached to features that were included in a prototype for consideration and then removed from the specification for a final system. If users are able to require all proposed features be included in the final system this can lead to conflict.

Developer misunderstanding of user objectives: Developers may assume that users share their objectives (e.g. to deliver core functionality on time and within budget), without understanding wider commercial issues. For example, user representatives attending Enterprise software (e.g. PeopleSoft) events may have seen demonstrations of "transaction auditing" (where changes are logged and displayed in a difference grid view) without being told that this feature demands additional coding and often requires more hardware to handle extra database accesses. Users might believe they can demand auditing on every field, whereas developers might think this is feature creep because they have made assumptions about the extent of user requirements. If the developer has committed delivery before the user requirements were reviewed, developers are between a rock and a hard place, particularly if user management derives some advantage from their failure to implement requirements.

Developer attachment to prototype: Developers can also become attached to prototypes they have spent a great deal of effort producing; this can lead to problems like attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture. (This may suggest that throwaway prototyping, rather than evolutionary prototyping, should be used.)

Excessive development time of the prototype: A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex. When the prototype is thrown away the precisely developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. Users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product.

Expense of implementing prototyping: the start up costs for building a development team focused on prototyping may be high. Many companies have development methodologies in place, and changing them can mean retraining, retooling, or both. Many companies tend to just jump into the prototyping without bothering to retrain their workers as much as they should.

A common problem with adopting prototyping technology is high expectations for productivity with insufficient effort behind the learning curve. In addition to training for the use of a prototyping technique, there is an often overlooked need for developing corporate and project specific underlying structure to support the technology. When this underlying structure is omitted, lower productivity can often result.

Best projects to use prototyping

It has been found that prototyping is very effective in the analysis and design of on-line systems, especially for transaction processing, where the use of screen dialogs is much more in evidence. The greater the interaction between the computer and the user, the greater the benefit is that can be obtained from building a quick system and letting the user play with it.

Systems with little user interaction, such as batch processing or systems that mostly do calculations, benefit little from prototyping. Sometimes, the coding needed to perform the system functions may be too intensive and the potential gains that prototyping could provide are too small.

Prototyping is especially good for designing good human-computer interfaces. "One of the most productive uses of rapid prototyping to date has been as a tool for iterative user requirements engineering and human-computer interface design.

Methods

There are few formal prototyping methodologies even though most Agile Methods rely heavily upon prototyping techniques.

1. Dynamic systems development method

Dynamic Systems Development Method (DSDM) is a framework for delivering business solutions that relies heavily upon prototyping as a core technique, and is itself ISO 9001 approved. It expands upon most understood definitions of a prototype. According to DSDM the prototype may be a diagram, a business process, or even a system placed into production. DSDM prototypes are intended to be incremental, evolving from simple forms into more comprehensive ones.
DSDM prototypes may be throwaway or evolutionary. Evolutionary prototypes may be evolved horizontally (breadth then depth) or vertically (each section is built in detail with additional iterations detailing subsequent sections). Evolutionary prototypes can eventually evolve into final systems.

The four categories of prototypes as recommended by DSDM are:

Business prototypes – used to design and demonstrate the business processes being automated.
Usability prototypes – used to define, refine, and demonstrate user interface design usability, accessibility, look and feel.
Performance and capacity prototypes - used to define, demonstrate, and predict how systems will perform under peak loads as well as to demonstrate and evaluate other non-functional aspects of the system (transaction rates, data storage volume, response time)
Capability/technique prototypes – used to develop, demonstrate, and evaluate a design approach or concept.

Identify prototype
Agree to a plan
Create the prototype
Review the prototype

2. Operational prototyping

Operational Prototyping was proposed by Alan Davis as a way to integrate throwaway and evolutionary prototyping with conventional system development. "[It] offers the best of both the quick-and-dirty and conventional-development worlds in a sensible manner. Designers develop only well-understood features in building the evolutionary baseline, while using throwaway prototyping to experiment with the poorly understood features."

Davis' belief is that to try to "retrofit quality onto a rapid prototype" is not the correct approach when trying to combine the two approaches. His idea is to engage in an evolutionary prototyping methodology and rapidly prototype the features of the system after each evolution.

The specific methodology follows these steps:

An evolutionary prototype is constructed and made into a baseline using conventional development strategies, specifying and implementing only the requirements that are well understood.

Copies of the baseline are sent to multiple customer sites along with a trained prototyper.

At each site, the prototyper watches the user at the system.

Whenever the user encounters a problem or thinks of a new feature or requirement, the prototyper logs it. This frees the user from having to record the problem, and allows them to continue working.

After the user session is over, the prototyper constructs a throwaway prototype on top of the baseline system.

The user now uses the new system and evaluates. If the new changes aren't effective, the prototyper removes them.

If the user likes the changes, the prototyper writes feature-enhancement requests and forwards them to the development team.

The development team, with the change requests in hand from all the sites, then produce a new evolutionary prototype using conventional methods.

Obviously, a key to this method is to have well trained prototypers available to go to the user sites. The Operational Prototyping methodology has many benefits in systems that are complex and have few known requirements in advance.

3. Evolutionary systems development

Evolutionary Systems Development is a class of methodologies that attempt to formally implement Evolutionary Prototyping. One particular type, called Systems craft is described by John Crinnion in his book: Evolutionary Systems Development.

Systemscraft was designed as a 'prototype' methodology that should be modified and adapted to fit the specific environment in which it was implemented.

Systemscraft was not designed as a rigid 'cookbook' approach to the development process. It is now generally recognised[sic] that a good methodology should be flexible enough to be adjustable to suit all kinds of environment and situation…

The basis of Systemscraft, not unlike Evolutionary Prototyping, is to create a working system from the initial requirements and build upon it in a series of revisions. Systemscraft places heavy emphasis on traditional analysis being used throughout the development of the system.

4. Evolutionary rapid development

Evolutionary Rapid Development (ERD) was developed by the Software Productivity Consortium, a technology development and integration agent for the Information Technology Office of the Defense Advanced Research Projects Agency (DARPA).

Fundamental to ERD is the concept of composing software systems based on the reuse of components, the use of software templates and on an architectural template. Continuous evolution of system capabilities in rapid response to changing user needs and technology is highlighted by the evolvable architecture, representing a class of solutions. The process focuses on the use of small artisan-based teams integrating software and systems engineering disciplines working multiple, often parallel short-duration timeboxes with frequent customer interaction.

Key to the success of the ERD-based projects is parallel exploratory analysis and development of features, infrastructures, and components with and adoption of leading edge technologies enabling the quick reaction to changes in technologies, the marketplace, or customer requirements.

To elicit customer/user input, frequent scheduled and ad hoc/impromptu meetings with the stakeholders are held. Demonstrations of system capabilities are held to solicit feedback before design/implementation decisions are solidified. Frequent releases (e.g., betas) are made availa ble for use to provide insight into how the system could better support user and customer needs. This assures that the system evolves to satisfy existing user needs.

The design framework for the system is based on using existing published or de facto standards. The system is organized to allow for evolving a set of capabilities that includes considerations for performance, capacities, and functionality. The architecture is defined in terms of abstract interfaces that encapsulate the services and their implementation (e.g., COTS applications). The architecture serves as a template to be used for guiding development of more than a single instance of the system. It allows for multiple application components to be used to implement the services. A core set of functionality not likely to change is also identified and established.

The ERD process is structured to use demonstrated functionality rather than paper products as a way for stakeholders to communicate their needs and expectations. Central to this goal of rapid delivery is the use of the "time box" method. Timeboxes are fixed periods of time in which specific tasks (e.g., developing a set of functionality) must be performed. Rather than allowing time to expand to satisfy some vague set of goals, the time is fixed (both in terms of calendar weeks and person-hours) and a set of goals is defined that realistically can be achieved within these constraints. To keep development from degenerating into a "random walk," long-range plans are defined to guide the iterations. These plans provide a vision for the overall system and set boundaries (e.g., constraints) for the project. Each iteration within the process is conducted in the context of these long-range plans.

Once architecture is established, software is integrated and tested on a daily basis. This allows the team to assess progress objectively and identify potential problems quickly. Since small amounts of the system are integrated at one time, diagnosing and removing the defect is rapid. User demonstrations can be held at short notice since the system is generally ready to exercise at all times.

5. Scrum

Scrum is an agile method for project management. The approach was first described by Takeuchi and Nonaka in "The New New Product Development Game" (Harvard Business Review, Jan-Feb 1986).

Tools

Efficiently using prototyping requires that an organization have proper tools and a staff trained to use those tools. Tools used in prototyping can vary from individual tools like 4th generation programming languages used for rapid prototyping to complex integrated CASE tools. 4th generation programming languages like Visual Basic and ColdFusion are frequently used since they are cheap, well known and relatively easy and fast to use. CASE tools are often developed or selected by the military or large organizations. Users may prototype elements of an application themselves in a spreadsheet.

1. Screen generators, design tools & Software Factories

Commonly used screen generating programs that enable prototypers to show users systems that don't function, but show what the screens may look like. Developing Human Computer Interfaces can sometimes be the critical part of the development effort, since to the users the interface essentially is the system.

Software Factories are Code Generators that allow you to model the domain model and then drag and drop the UI. Also they enable you to run the prototype and use basic database functionality. This approach allows you to explore the domain model and make sure it is in sync with the GUI prototype.

2. Application definition or simulation software

It enables users to rapidly build lightweight, animated simulations of another computer program, without writing code. Application simulation software allows both technical and non-technical users to experience, test, collaborate and validate the simulated program, and provides reports such as annotations, screenshot and schematics. To simulate applications one can also use software which simulate real-world software programs for computer based training, demonstration, and customer support, such as screen casting software as those areas are closely related.

3. Sketchflow

Sketch Flow, a feature of Microsoft Expression Studio Ultimate, gives the ability to quickly and effectively map out and iterate the flow of an application UI, the layout of individual screens and transition from one application state to another.

Interactive Visual Tool
Easy to learn
Dynamic
Provides enviroment to collect feedback

Visual Basic

One of the most popular tools for Rapid Prototyping is Visual Basic (VB). Microsoft Access, which includes a Visual Basic extensibility module, is also a widely accepted prototyping tool that is used by many non-technical business analysts. Although VB is a programming language it has many features that facilitate using it to create prototypes, including:

An interactive/visual user interface design tool.
Easy connection of user interface components to underlying functional behavior.
Modifications to the resulting software are easy to perform.

5. Requirements Engineering Environment

It provides an integrated toolset for rapidly representing, building, and executing models of critical aspects of complex systems.

It is currently used by the Air Force to develop systems. It is: an integrated set of tools that allows systems analysts to rapidly build functional, user interface, and performance prototype models of system components. These modeling activities are performed to gain a greater understanding of complex systems and lessen the impact that inaccurate requirement specifications have on cost and scheduling during the system development process.

REE is composed of three parts. The first, called proto is a CASE tool specifically designed to support rapid prototyping. The second part is called the Rapid Interface Prototyping System or RIP, which is a collection of tools that facilitate the creation of user interfaces. The third part of REE is a user interface to RIP and proto that is graphical and intended to be easy to use.

Rome Laboratory, the developer of REE, intended that to support their internal requirements gathering methodology. Their method has three main parts:

Elicitation from various sources which means u loose (users, interfaces to other systems), specification, and consistency checking

Analysis that the needs of diverse users taken together do not conflict and are technically and economically feasible

Validation that requirements so derived are an accurate reflection of user needs.

LYMB

LYMB is an object-oriented development environment aimed at developing applications that require combining graphics-based user interfaces, visualization, and rapid prototyping.

7. Non-relational environments

Non-relational definition of data (e.g. using Cache or associative models can help make end-user prototyping more productive by delaying or avoiding the need to normalize data at every iteration of a simulation. This may yield earlier/greater clarity of business requirements, though it does not specifically confirm that requirements are technically and economically feasible in the target production system.

8. PSDL

PSDL is a prototype description language to describe real-time software.

Prototyping in the Software Process

System prototyping

Prototyping is the rapid development of a system

In the past, the developed system was normally thought of as inferior in some way to the required system so further development was required

Now, the boundary between prototyping and normal system development is blurred and many systems are developed using an evolutionary approach

Uses of system prototypes

The principal use is to help customers and developers understand the requirements for the system

Requirements elicitation. Users can experiment with a prototype to see how the system supports their work

Requirements validation. The prototype can reveal errors and omissions in the requirements

Prototyping can be considered as a risk reduction activity which reduces requirements risks

Prototyping benefits

Misunderstandings between software users and developers are exposed

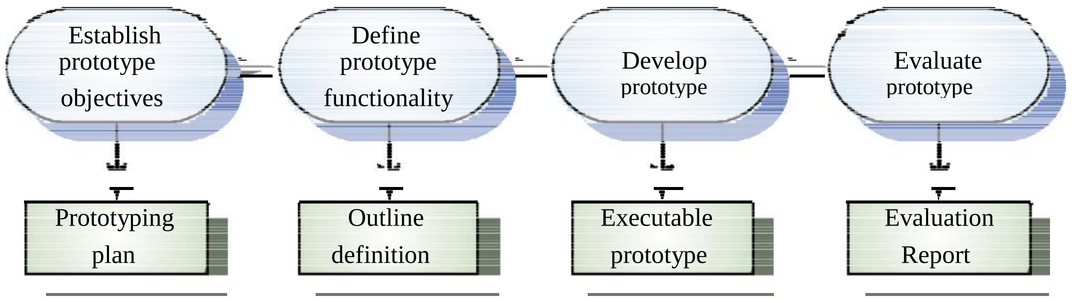
Missing services may be detected and confusing services may be identified

A working system is available early in the process

The prototype may serve as a basis for deriving a system specification

The system can support user training and system testing

Prototyping process



Prototyping in the software process

Evolutionary prototyping

An approach to system development where an initial prototype is produced and refined through a number of stages to the final system

Throw-away prototyping

A prototype which is usually a practical implementation of the system is produced to help discover requirements problems and then discarded. The system is then developed using some other development process

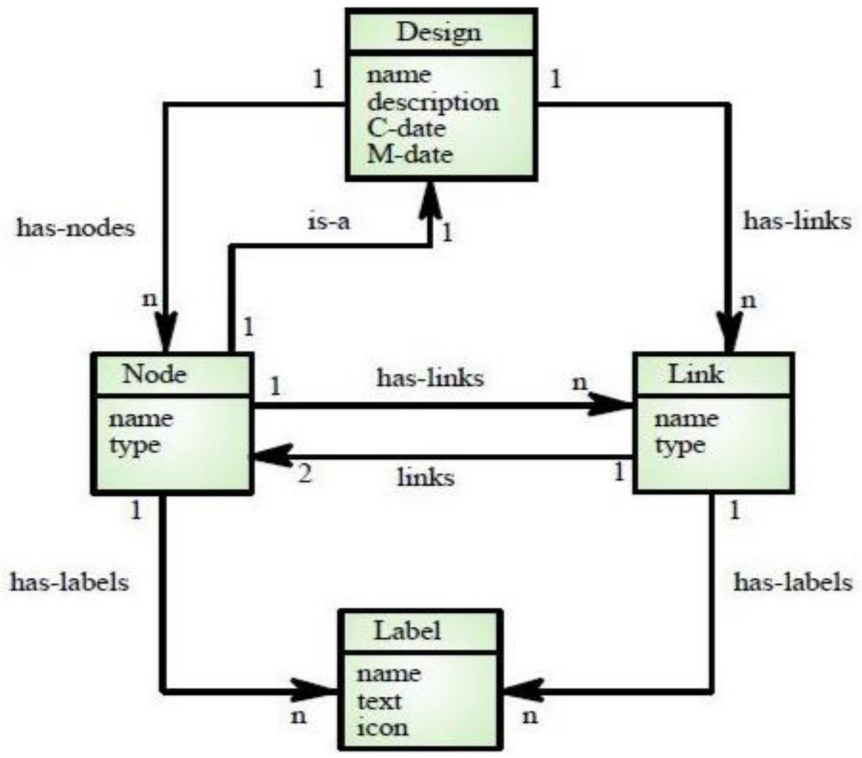
Data Model

Used to describe the logical structure of data processed by the system

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



Behavioural Model

Behavioural models are used to describe the overall behaviour of a system
Two types of behavioural model are shown here
Data processing models that show how data is processed as it moves through the system
State machine models that show the systems response to events
Both of these models are required for a description of the system‘s behaviour

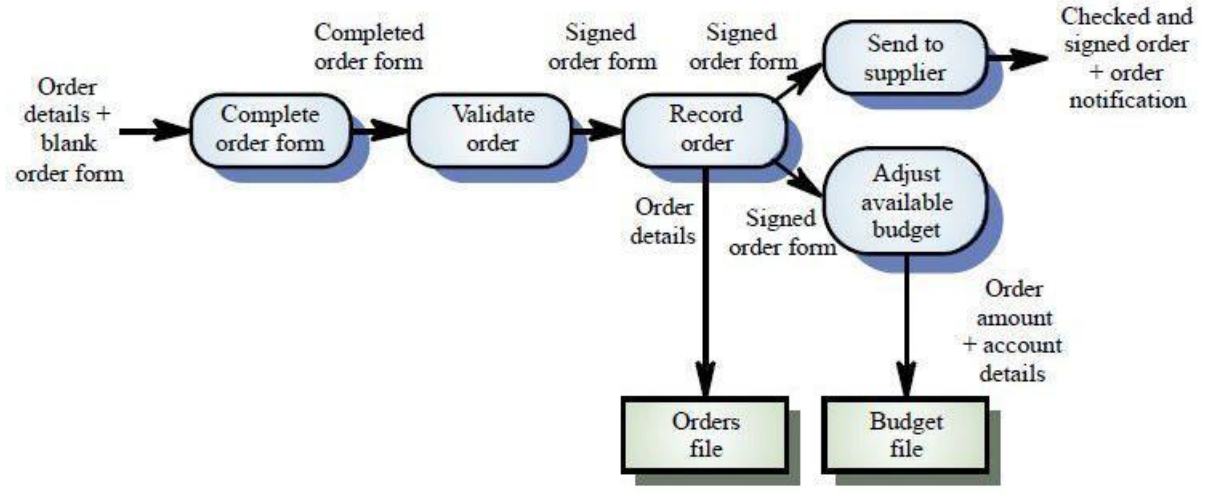
Data-processing models

Data flow diagrams are used to model the system‘s data processing
These show the processing steps as data flows through a system
Intrinsic part of many analysis methods
Simple and intuitive notation that customers can understand
Show end-to-end processing of data

Data flow diagrams

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

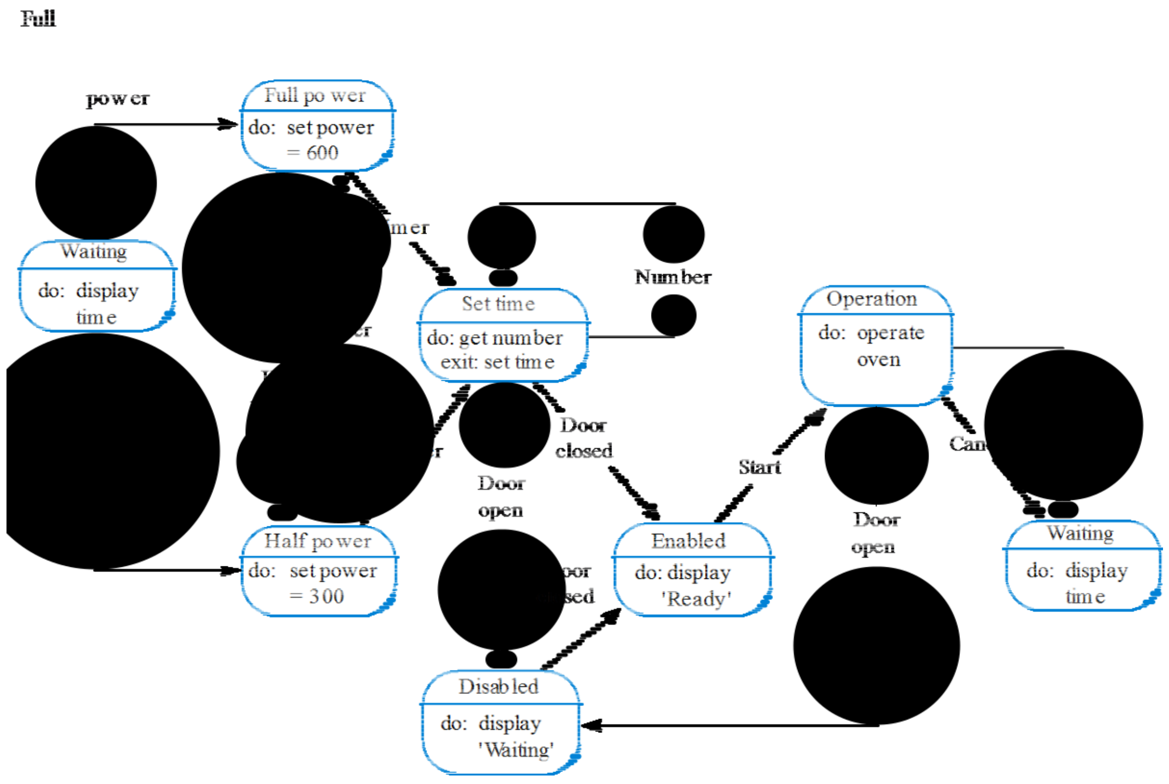
Order processing DFD



2. State machine models

These model the behaviour of the system in response to external and internal events
They show the system‘s responses to stimuli so are often used for modelling real-time systems
State machine models show system states as nodes and events as arcs between these nodes.
When an event occurs, the system moves from one state to another
Statecharts are an integral part of the UML

Microwave oven model



Statecharts

Allow the decomposition of a model into submodels

A brief description of the actions is included following the ‗do‘ in each state

Can be complemented by tables describing the states and the stimuli

Structured Analysis

The data-flow approach is typified by the Structured Analysis method (SA)

Two major strategies dominate structured analysis

‗Old‘ method popularised by DeMarco

‗Modern‘ approach by Yourdon

DeMarco

A top-down approach

The analyst maps the current physical system onto the current logical data-flow model

The approach can be summarised in four steps:

Analysis of current physical system

Derivation of logical model

Derivation of proposed logical model

Implementation of new physical system

Modern structured analysis

Distinguishes between user‘s real needs and those requirements that represent the external behaviour satisfying those needs

Includes real-time extensions

Other structured analysis approaches include:

Structured Analysis and Design Technique (SADT)

Structured Systems Analysis and Design Methodology (SSADM)

Method weaknesses

They do not model non-functional system requirements.

They do not usually include information about whether a method is appropriate for a given problem.

The may produce too much documentation.

The system models are sometimes too detailed and difficult for users to understand.

CASE workbenches

A coherent set of tools that is designed to support related software process activities such as analysis, design or testing.

Analysis and design workbenches support system modelling during both requirements engineering and system design.

These workbenches may support a specific design method or may provide support for a creating several different types of system model.

An analysis and design workbench

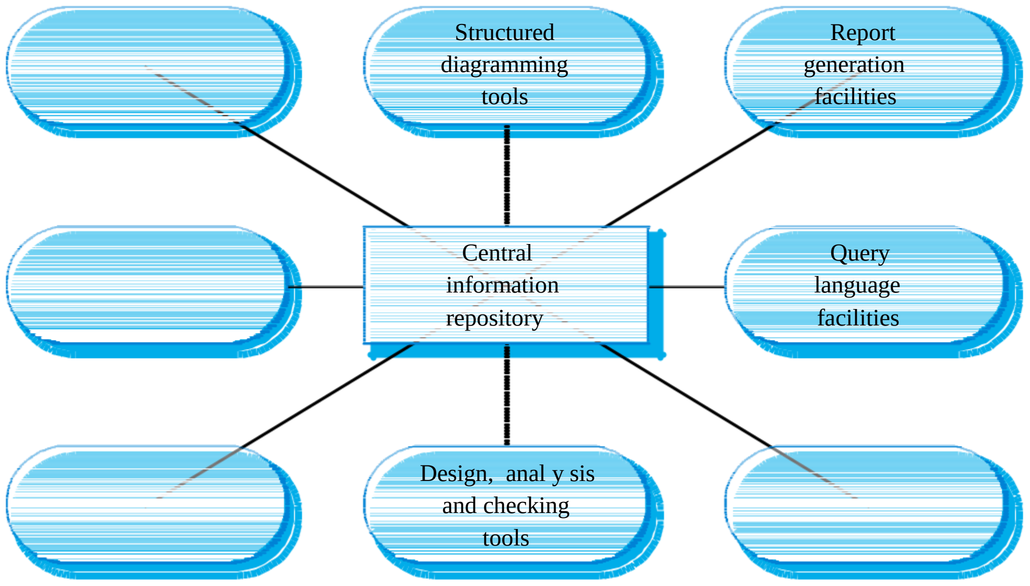
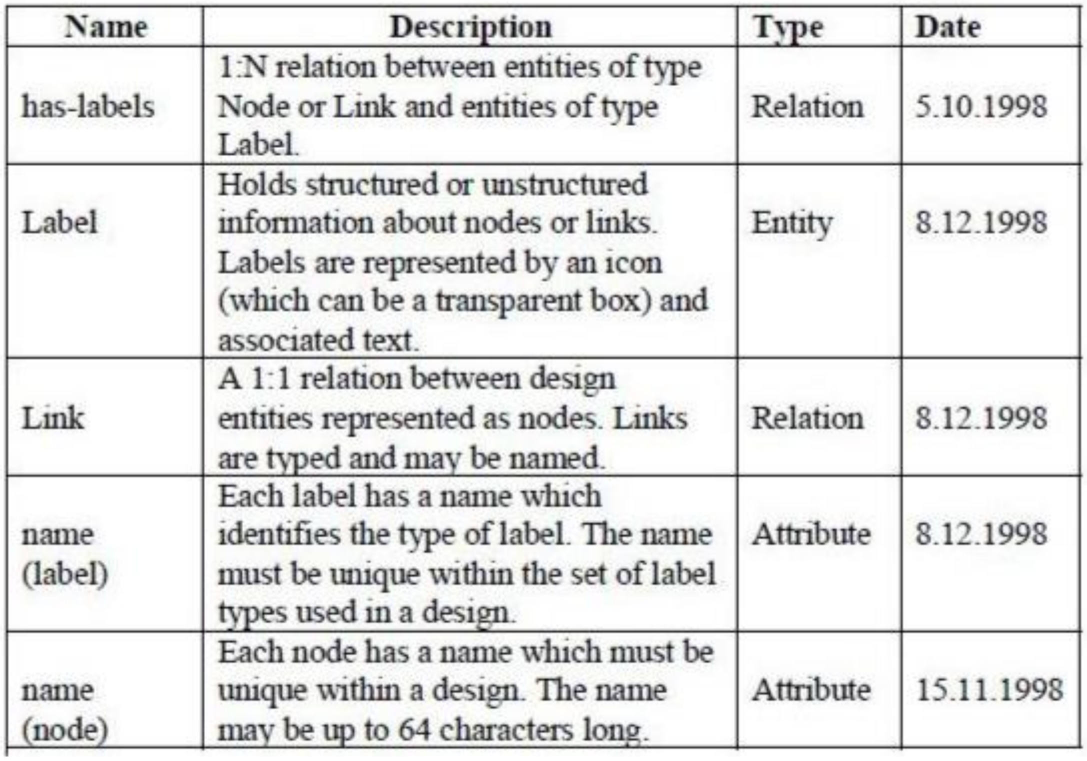

Structured diagramming tools
Report generation facilities
Central information repository
Query language facilities
Design, anal y sis and checking tools

Diagram editors
Model analysis and checking tools
Repository and associated query language
Data dictionary
Report definition and generation tools
Forms definition tools
Import/export translators
Code generation tools

Data Dictionary

Data dictionaries are lists of all of the names used in the system models. Descriptions of the entities, relationships and attributes are also included
Advantages
Support name management and avoid duplication
Store of organisational knowledge linking analysis, design and implementation
Many CASE workbenches support data dictionaries

Data dictionary entries



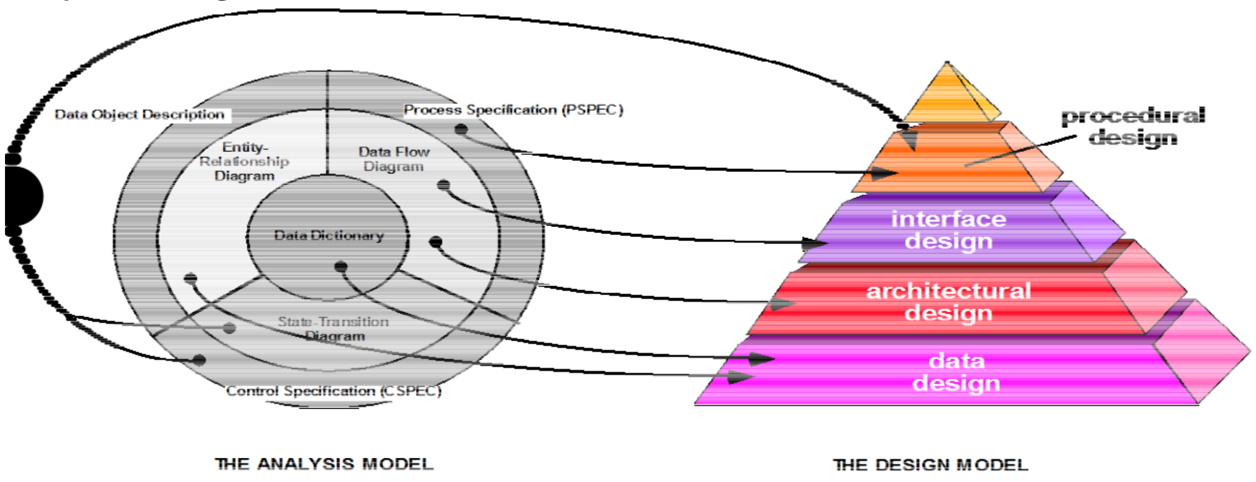
Name	Description	Type	Date
has-labels	1:N relation between entities of type Node or Link and entities of type Label.	Relation	5.10.1998
Label	Holds structured or unstructured information about nodes or links. Labels are represented by an icon (which can be a transparent box) and associated text.	Entity	8.12.1998
Link	A 1:1 relation between design entities represented as nodes. Links are typed and may be named.	Relation	8.12.1998
name (label)	Each label has a name which identifies the type of label. The name must be unique within the set of label types used in a design.	Attribute	8.12.1998
name (node)	Each node has a name which must be unique within a design. The name may be up to 64 characters long.	Attribute	15.11.1998

Unit III : Design Concepts and Principles

Design Concepts and Principles:

Map the information from the analysis model to the design representations - data design, architectural design, interface design, procedural design

Analysis to Design:



Design Models – 1:

Data Design

– created by transforming the data dictionary and ERD into implementation data structures

– requires as much attention as algorithm design

Architectural Design

– derived from the analysis model and the subsystem interactions defined in the DFD

Interface Design

– derived from DFD and CFD

– describes software elements communication with

other software elements
other systems
human users

Procedure-level design

– created by transforming the structural elements defined by the software architecture into procedural descriptions of software components

– Derived from information in the PSPEC, CSPEC, and STD

Process should not suffer from tunnel vision – consider alternative approaches
Design should be traceable to analysis model
Do not try to reinvent the wheel
use design patterns ie reusable components
Design should exhibit both uniformity and integration

Should be structured to accommodate changes

Design is not coding and coding is not design
Should be structured to degrade gently, when bad data, events, or operating conditions are encountered
Needs to be assessed for quality as it is being created
Needs to be reviewed to minimize conceptual (semantic) errors

Abstraction
– allows designers to focus on solving a problem without being concerned about irrelevant lower level details

Procedural abstraction is a named sequence of instructions that has a specific and limited function
e.g open a door
Open implies a long sequence of procedural steps
data abstraction is collection of data that describes a data object
e.g door type, opening mech, weight,dimen

Design Patterns
– description of a design structure that solves a particular design problem within a specific context and its impact when applied

Design Concepts -2 :

Software Architecture
– overall structure of the software components and the ways in which that structure
– provides conceptual integrity for a system

Information Hiding
– information (data and procedure) contained within a module is inaccessible to modules that have no need for such information

Functional Independence
– achieved by developing modules with single-minded purpose and an aversion to excessive interaction with other models

Fowler [FOW99] defines refactoring in the following manner:
– "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure.‖

When software is refectories, the existing design is examined for
– redundancy
– unused design elements
– inefficient or unnecessary algorithms
– poorly constructed or inappropriate data structures
– or any other design failure that can be corrected to yield a better design.

Objects
– encapsulate both data and data manipulation procedures needed to describe the content and behavior of a real world entity

Class

– generalized description (template or pattern) that describes a collection of similar objects

Inheritance

– provides a means for allowing subclasses to reuse existing superclass data and procedures; also provides mechanism for propagating changes

Messages

– the means by which objects exchange information with one another

Polymorphism

– a mechanism that allows several objects in an class hierarchy to have different methods with the same name

– instances of each subclass will be free to respond to messages by calling their own version of the method

Modular Design Methodology Evaluation – 1:

Modularity

– the degree to which software can be understood by examining its components independently of one another

Modular decomposability

– provides systematic means for breaking problem into sub problems

Modular compos ability

– supports reuse of existing modules in new systems

Modular understandability

– module can be understood as a stand-alone unit

Modular Design Methodology Evaluation – 2:

Modular continuity

– module change side-effects minimized

Modular protection

– processing error side-effects minimized

Functional independence

– modules have high cohesion and low coupling

Cohesion

– qualitative indication of the degree to which a module focuses on just one thing

Coupling

– qualitative indication of the degree to which a module is connected to other modules and to the outside world

The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:

analyze the effectiveness of the design in meeting its stated requirements,

consider architectural alternatives at a stage when making design changes is still relatively easy, and

reduce the risks associated with the construction of the software.

Software architecture representations enable communications among stakeholders

Architecture highlights early design decisions that will have a profound impact on the ultimate success of the system as an operational entity
The architecture constitutes an intellectually graspable model of how the system is structured and how its components work together

Data centered
– file or database lies at the center of this architecture and is accessed frequently by other components that modify data

Data flow
– input data is transformed by a series of computational components into output data

– Pipe and filter pattern has a set of components called filters, connected by pipes that transmit data from one component to the next.
– If the data flow degenerates into a single line of transforms, it is termed batch sequential

Object-oriented
– components of system encapsulate data and operations, communication between components is by message passing

Layered
– several layers are defined
– each layer performs operations that become closer to the machine instruction set in the lower layers

– program structure decomposes function into control hierarchy with main program invoking several subprograms

Software to be developed must be put into context
– model external entities and define interfaces

Identify architectural archetypes
– collection of abstractions that must be modeled if the system is to be constructed

Object oriented Architecture :

The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication and coordination between components is accomplished via message passing

Specify structure of the system
– define and refine the software components needed to implement each archet ype

Continue the process iteratively until a complete architectural structure has been derived

Layered Architecture:

Number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set
At the outer layer –components service user interface operations.
At the inner layer – components perform operating system interfacing.
Intermediate layers provide utility services and application software function

Architecture Tradeoff Analysis – 1:

1. Collect scenarios
2. Elicit requirements, constraints, and environmental description
3. Describe architectural styles/patterns chosen to address scenarios and requirements

    module view
    process view
    data flow view

Architecture Tradeoff Analysis – 2:

Evaluate quality attributes independently (e.g. reliability, performance, security, maintainability, flexibility, testability, portability, reusability, interoperability)

Identify sensitivity points for architecture

any attributes significantly affected by changing in the architecture

Refining Architectural Design:

- Processing narrative developed for each module
- Interface description provided for each module
- Local and global data structures are defined
- Design restrictions/limitations noted
- Design reviews conducted

Refinement considered if required and justified

Architectural Design

An early stage of the system design process.

Represents the link between specification and design processes.

Often carried out in parallel with some specification activities.

It involves identifying major system components and their communications.

Advantages of explicit architecture

Stakeholder communication

Architecture may be used as a focus of discussion by system stakeholders.

System analysis

Means that analysis of whether the system can meet its non-functional requirements is possible.

Large-scale reuse

The architecture may be reusable across a range of systems.

Architecture and system characteristics

Performance

Localise critical operations and minimise communications. Use large rather than fine-grain components.

Security

Use a layered architecture with critical assets in the inner layers.

Safety

Localise safety-critical features in a small number of sub-systems.

Availability

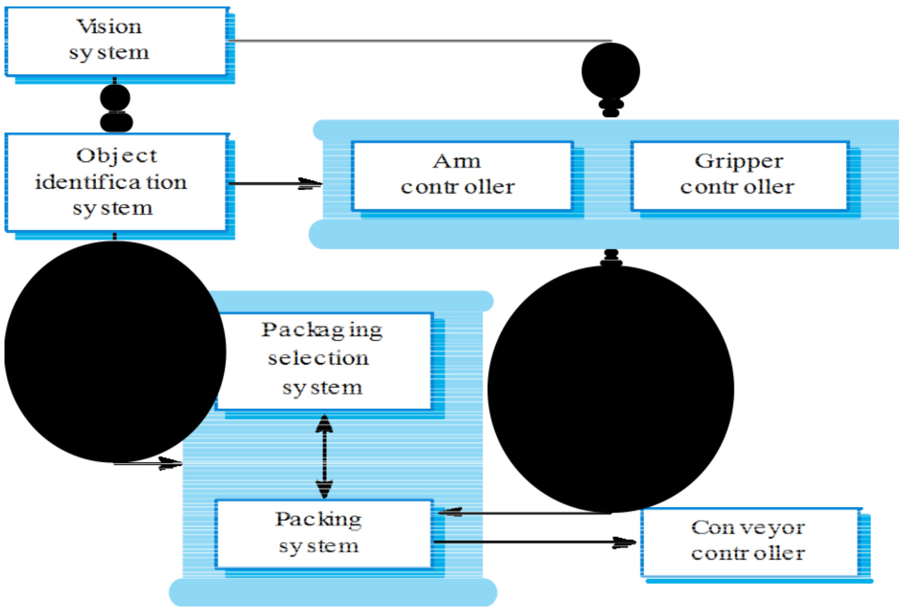
Include redundant components and mechanisms for fault tolerance.

Maintainability
Use fine-grain, replaceable components.

Using large-grain components improves performance but reduces maintainability.
Introducing redundant data improves availability but makes security more difficult.
Localising safety-related features usually means more communication so degraded performance.

Concerned with decomposing the system into interacting sub-systems.
The architectural design is normally expressed as a block diagram presenting an overview of the system structure.
More specific models showing how sub-systems share data, are distributed and interface with each other may also be developed.

Packing robot control system



Box and line diagrams
Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
However, useful for communication with stakeholders and for project planning.

Architectural design decisions
Architectural design is a creative process so the process differs depending on the type of system being developed.
However, a number of common decisions span all design processes.
Is there a generic application architecture that can be used?
How will the system be distributed?
What architectural styles are appropriate?
What approach will be used to structure the system?

How will the system be decomposed into modules?
What control strategy should be used?
How will the architectural design be evaluated?
How should the architecture be documented?

Systems in the same domain often have similar architectures that reflect domain concepts. Application product lines are built around a core architecture with variants that satisfy particular customer requirements.

The architectural model of a system may conform to a generic architectural model or style. An awareness of these styles can simplify the problem of defining system architectures. However, most large systems are heterogeneous and do not follow a single architectural style.

Used to document an architectural design.

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

System organisation

Reflects the basic strategy that is used to structure a system.
Three organisational styles are widely used:

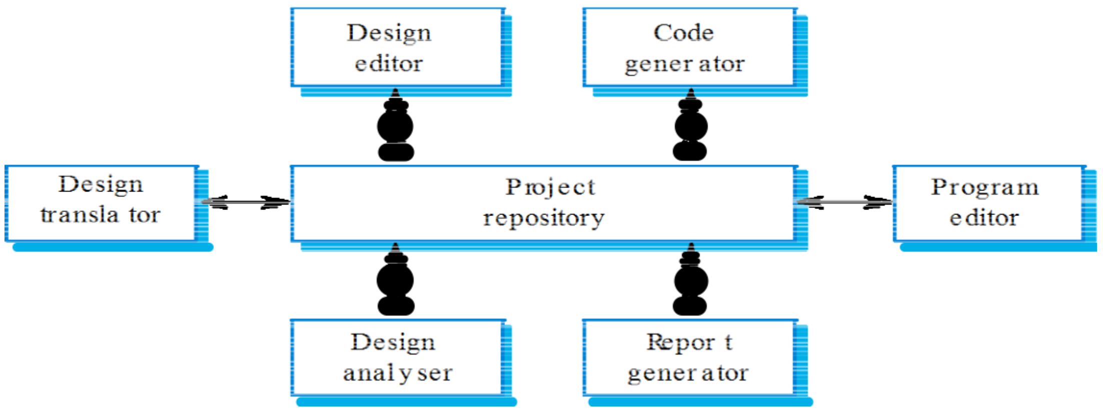
A shared data repository style;
A shared services and servers style;
An abstract machine or layered style.

Sub-systems must exchange data. This may be done in two ways:

Shared data is held in a central database or repository and may be accessed by all sub-systems;
Each sub-system maintains its own database and passes data explicitly to other sub-systems.

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

CASE toolset architecture



Repository model characteristics

Advantages

Efficient way to share large amounts of data;

Sub-systems need not be concerned with how data is produced Centralised management e.g. backup, security, etc.

Sharing model is published as the repository schema.

Disadvantages

Sub-systems must agree on a repository data model. Inevitably a compromise;

Data evolution is difficult and expensive;

No scope for specific management policies;

Difficult to distribute efficiently.

Distributed system model which shows how data and processing is distributed across a range of components.

Set of stand-alone servers which provide specific services such as printing, data management, etc.

Set of clients which call on these services.

Network which allows clients to access servers.

Distribution of data is straightforward;

Makes effective use of networked systems. May require cheaper hardware;

Easy to add new servers or upgrade existing servers.

No shared data model so sub-systems use different data organisation. Data interchange may be inefficient;

Redundant management in each server;

No central register of names and services - it may be hard to find out what servers and services are available.

Used to model the interfacing of sub-systems.

Organises the system into a set of layers (or abstract machines) each of which provide a set of services.

Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.

However, often artificial to structure systems in this way.

Styles of decomposing sub-systems into modules.

No rigid distinction between system organisation and modular decomposition.

Sub-systems and modules

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

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

Modular decomposition

Another structural level where sub-systems are decomposed into modules.

Two modular decomposition models covered

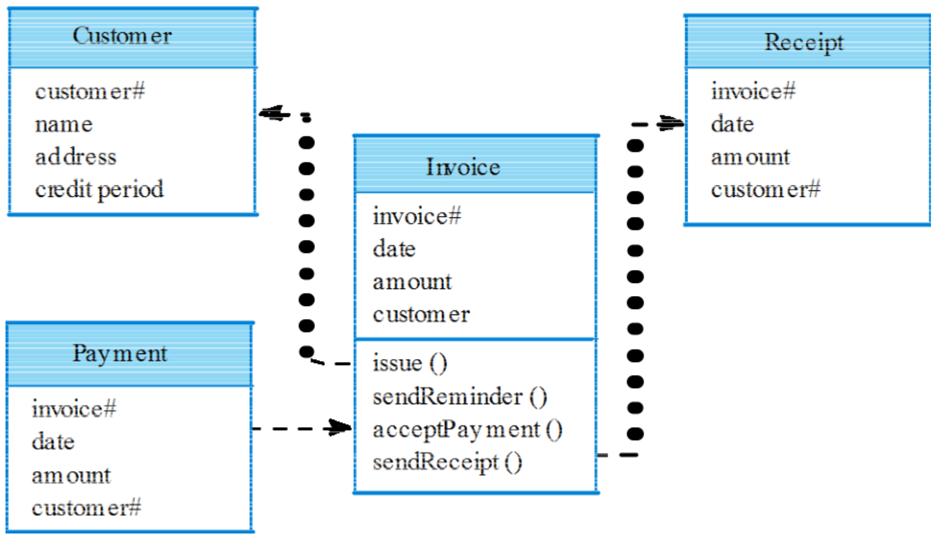
An object model where the system is decomposed into interacting object;
A pipeline or data-flow model where the system is decomposed into functional modules which transform inputs to outputs.

If possible, decisions about concurrency should be delayed until modules are implemented.

Object models

Structure the system into a set of loosely coupled objects with well-defined interfaces.
Object-oriented decomposition is concerned with identifying object classes, their attributes and operations.
When implemented, objects are created from these classes and some control model used to coordinate object operations.

Invoice processing system



Object model advantages

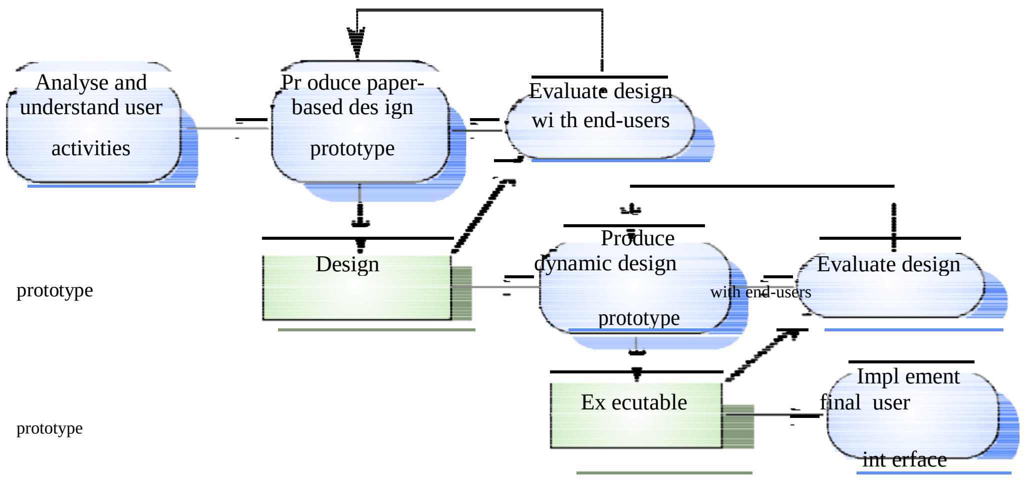
Objects are loosely coupled so their implementation can be modified without affecting other objects.
The objects may reflect real-world entities.
OO implementation languages are widely used.
However, object interface changes may cause problems and complex entities may be hard to represent as objects.

Functional transformations process their inputs to produce outputs.
May be referred to as a pipe and filter model (as in UNIX shell).
Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems.
Not really suitable for interactive systems.

Designing effective interfaces for software systems
System users often judge a system by its interface rather than its functionality
A poorly designed interface can cause a user to make catastrophic errors

Poor user interface design is the reason why so many software systems are never used
Most users of business systems interact with these systems through graphical user interfaces (GUIs)
In some cases, legacy text-based interfaces are still used

User interface design process


Analyse and understand user activities
Pr oduce paper-based des ign prototype
Evaluate design wi th end-users
Design prototype
Produce dynamic design prototype
Evaluate design with end-users
Ex ecutable prototype
Impl ement final user int erface

UI design principles

User familiarity

The interface should be based on user-oriented terms and concepts rather than computer concepts
E.g., an office system should use concepts such as letters, documents, folders etc. rather than directories, file identifiers, etc.

Consistency

The system should display an appropriate level of consistency
Commands and menus should have the same format, command punctuation should be similar, etc.

Minimal surprise

If a command operates in a known way, the user should be able to predict the operation of comparable commands

Recoverability

The system should provide some interface to user errors and allow the user to recover from errors

User guidance

Some user guidance such as help systems, on-line manuals, etc. should be supplied

User diversity

Interaction facilities for different types of user should be supported
E.g., some users have seeing difficulties and so larger text should be available

User-system interaction

Two problems must be addressed in interactive systems design

How should information from the user be provided to the computer system?

How should information from the computer system be presented to the user?

Interaction styles

Direct manipulation
- Easiest to grasp with immediate feedback
- Difficult to program

Menu selection
- User effort and errors minimized
- Large numbers and combinations of choices a problem

Form fill-in
- Ease of use, simple data entry
- Tedious, takes a lot of screen space

Natural language
- Great for casual users
- Tedious for expert users

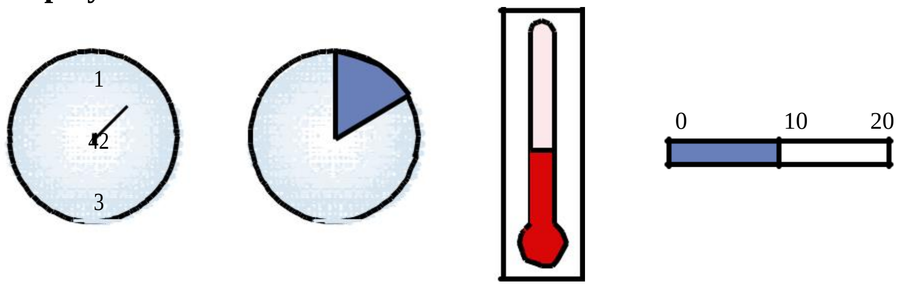
Information presentation

Information presentation is concerned with presenting system information to system users

The information may be presented directly or may be transformed in some way for presentation

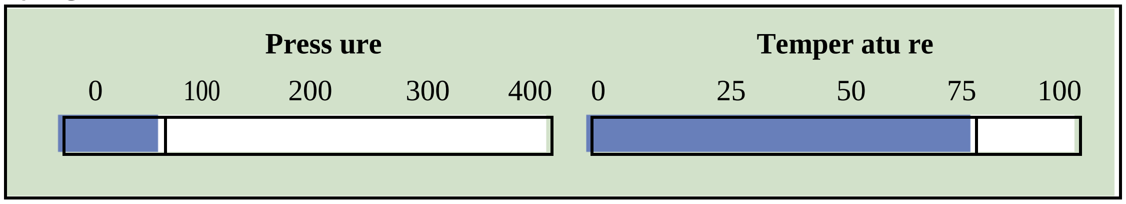
The Model-View-Controller approach is a way of supporting multiple presentations of data

Information display



Dial with needle Pie chart Thermometer Horizontal bar

Displaying relative values



Data visualisation

Concerned with techniques for displaying large amounts of information

Visualisation can reveal relationships between entities and trends in the data

Possible data visualisations are:

- Weather information
- State of a telephone network
- Chemical plant pressures and temperatures
- A model of a molecule

Colour adds an extra dimension to an interface and can help the user understand complex information structures

Can be used to highlight exceptional events

- The use of colour to communicate meaning

Error message design is critically important. Poor error messages can mean that a user rejects rather than accepts a system

Messages should be polite, concise, consistent and constructive

The background and experience of users should be the determining factor in message design

Some evaluation of a user interface design should be carried out to assess its suitability

Full scale evaluation is very expensive and impractical for most systems

Ideally, an interface should be evaluated against req

However, it is rare for such specifications to be produced

Real Time Software Design

Systems which monitor and control their environment

Inevitably associated with hardware devices

- Sensors: Collect data from the system environment
- Actuators: Change (in some way) the system's environment

Time is critical. Real-time systems MUST respond within specified times

A real-time system is a software system where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced

A ‗soft' real-time system is a system whose operation is degraded if results are not produced according to the specified timing requirements

A ‗hard' real-time system is a system whose operation is incorrect if results are not produced according to the timing specification

Given a stimulus, the system must produce a response within a specified time

2 classes

Periodic stimuli. Stimuli which occur at predictable time intervals

- For example, a temperature sensor may be polled 10 times per second

Aperiodic stimuli. Stimuli which occur at unpredictable times

- For example, a system power failure may trigger an interrupt which must be processed by the system

Architectural considerations

Because of the need to respond to timing demands made by different stimuli / responses, the system architecture must allow for fast switching between stimulus handlers
Timing demands of different stimuli are different so a simple sequential loop is not usually adequate

Designing embedded software systems whose behaviour is subject to timing constraints
To explain the concept of a real-time system and why these systems are usually implemented as concurrent processes
To describe a design process for real-time systems
To explain the role of a real-time executive
To introduce generic architectures for monitoring and control and data acquisition systems

Real-time systems:

Systems which monitor and control their environment
Inevitably associated with hardware devices

– Sensors: Collect data from the system environment
– Actuators: Change (in some way) the system's environment

Time is critical. Real-time systems MUST respond within specified times

Definition:

A real-time system is a software system where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced
A ‗soft‘ real-time system is a system whose operation is degraded if results are not produced according to the specified timing requirements
A ‗hard‘ real-time system is a system whose operation is incorrect if results are not produced according to the timing specification

Given a stimulus, the system must produce a esponse within a specified time
Periodic stimuli. Stimuli which occur at predictable time intervals

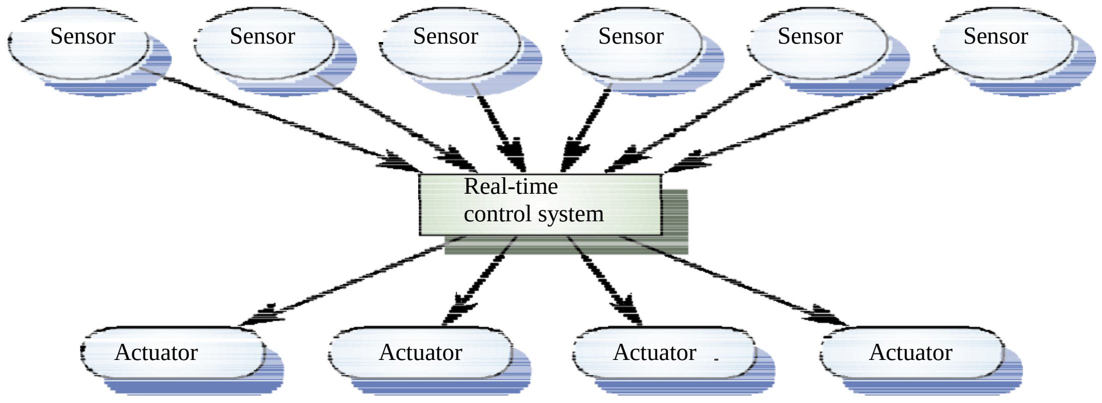
– For example, a temperature sensor may be polled 10 times per second

Aperiodic stimuli. Stimuli which occur at unpredictable times

– For example, a system power failure may trigger an interrupt which must be processed by the system

Because of the need to respond to timing demands made by different stimuli/responses, the system architecture must allow for fast switching between stimulus handlers
Timing demands of different stimuli are different so a simple sequential loop is not usually adequate
Real-time systems are usually designed as cooperating processes with a real-time executive controlling these processes

A real-time system model:



System elements:

Sensors control processes
- Collect information from sensors. May buffer information collected in response to a sensor stimulus

Data processor
- Carries out processing of collected information and computes the system response

Actuator control
- Generates control signals for the actuator

Identify the stimuli to be processed and the required responses to these stimuli
For each stimulus and response, identify the timing constraints
Aggregate the stimulus and response processing into concurrent processes. A process may be associated with each class of stimulus and response
Design algorithms to process each class of stimulus and response. These must meet the given timing requirements
Design a scheduling system which will ensure that processes are started in time to meet their deadlines
Integrate using a real-time executive or operating system

Timing constraints:

May require extensive simulation and experiment to ensure that these are met by the system
May mean that certain design strategies such as object-oriented design cannot be used because of the additional overhead involved
May mean that low-level programming language features have to be used for performance reasons

Hard-real time systems may have to programmed in assembly language to ensure that deadlines are met
Languages such as C allow efficient programs to be written but do not have constructs to support concurrency or shared resource management
Ada as a language designed to support real-time systems design so includes a general purpose concurrency mechanism

Non-stop system components:

Configuration manager
- Responsible for the dynamic reconfiguration of the system software and hardware. Hardware modules may be replaced and software upgraded without stopping the systems

Fault manager
- Responsible for detecting software and hardware faults and taking appropriate actions (e.g. switching to backup disks) to ensure that the system continues in operation

Burglar alarm system e.g

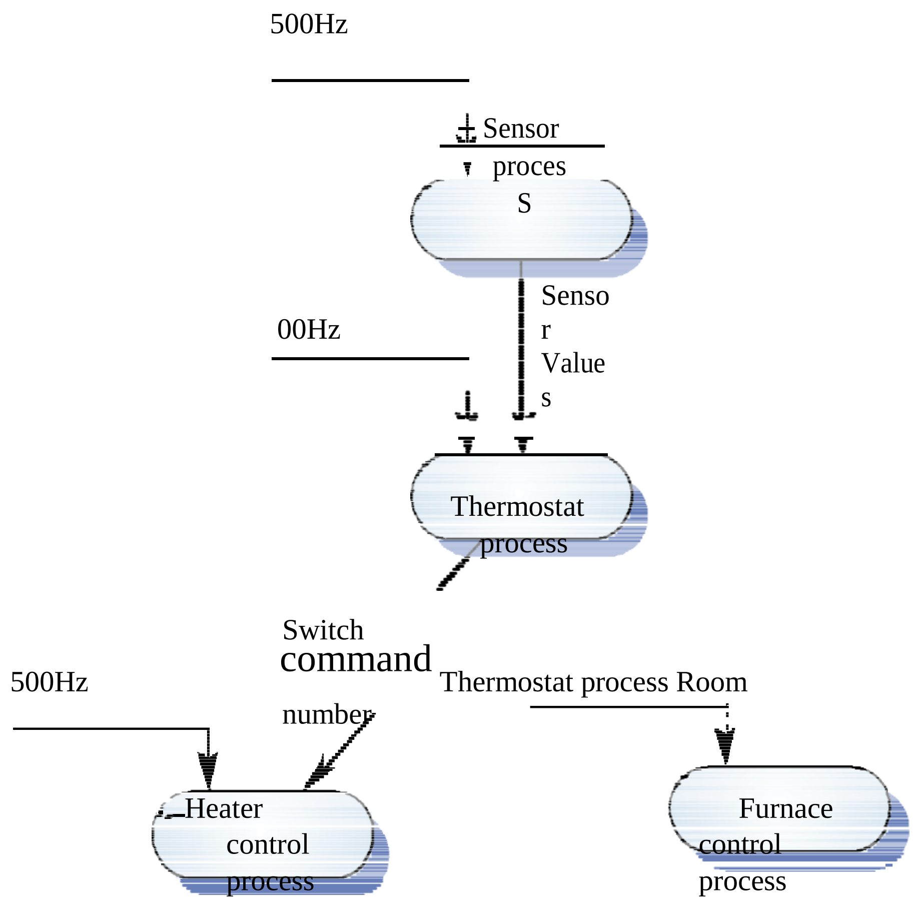
A system is required to monitor sensors on doors and windows to detect the presence of intruders in a building
When a sensor indicates a break-in, the system switches on lights around the area and calls police automatically
The system should include provision for operation without a mains power supply
Sensors
Movement detectors, window sensors, door sensors.
50 window sensors, 30 door sensors and 200 movement detectors
Voltage drop sensor
Actions
When an intruder is detected, police are called automatically.
Lights are switched on in rooms with active sensors.
An audible alarm is switched on.
The system switches automatically to backup power when a voltage drop is detected.

Identify stimuli and associated responses
Define the timing constraints associated with each stimulus and response
Allocate system functions to concurrent processes
Design algorithms for stimulus processing and response generation
Design a scheduling system which ensures that processes will always be scheduled to meet their deadlines

A burglar alarm system is primarily a monitoring system. It collects data from sensors but no real-time actuator control
Control systems are similar but, in response to sensor values, the system sends control signals to actuators
An example of a monitoring and control system is a system which monitors temperature and switches heaters on and off

Collect data from sensors for subsequent processing and analysis.
Data collection processes and processing processes may have different periods and deadlines.
Data collection may be faster than processing e.g. collecting information about an explosion.
Circular or ring buffers are a mechanism for smoothing speed differences.

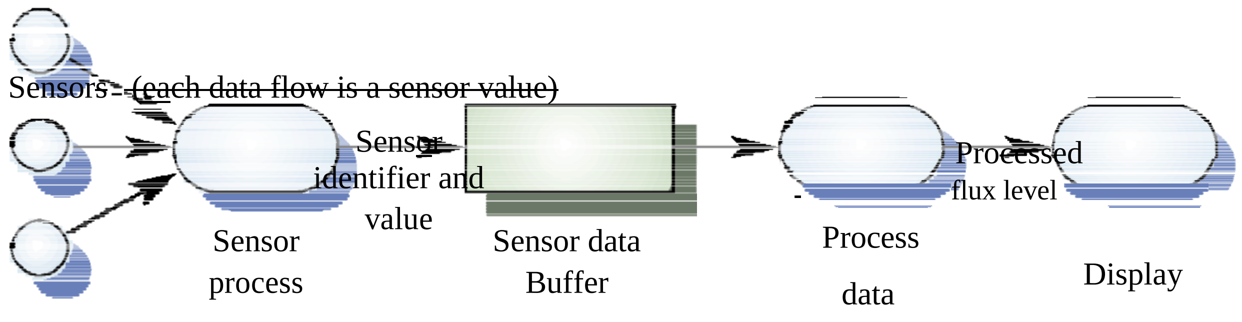
A temperature control system:



Reactor data collection:

A system collects data from a set of sensors monitoring the neutron flux from a nuclear reactor.
Flux data is placed in a ring buffer for later processing.
The ring buffer is itself implemented as a concurrent process so that the collection and processing processes may be synchronized.

Reactor flux monitoring:


Sensors ~~(each data flow is a sensor value)~~
Sensor process
Sensor identifier and value
Sensor data Buffer
Process data
Processed flux level
Display

Mutual exclusion:

Producer processes collect data and add it to the buffer. Consumer processes take data from the buffer and make elements available

Producer and consumer processes must be mutually excluded from accessing the same element.

The buffer must stop producer processes adding information to a full buffer and consumer processes trying to take information from an empty buffer

System Design

Design both the hardware and the software associated with system. Partition functions to either hardware or software

Design decisions should be made on the basis on non-functional system requirements

Hardware delivers better performance but potentially longer development and less scope for change

System elements

Sensors control processes

Collect information from sensors. May buffer information collected in response t o a sensor stimulus

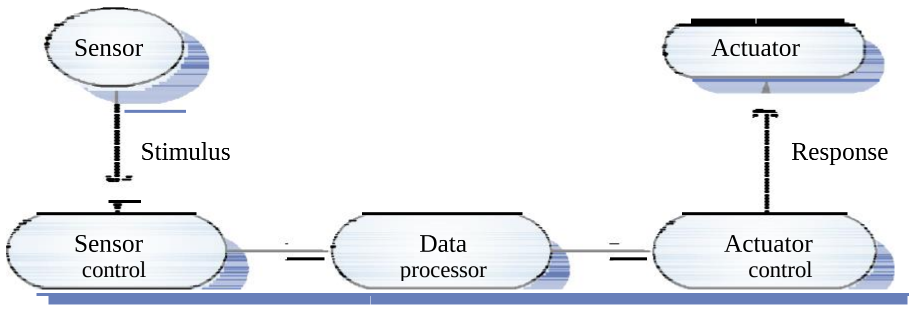
Data processor

Carries out processing of collected information and computes the system response

Actuator control

Generates control signals for the actuator

Sensor/actuator processes



R-T systems design process

Identify the stimuli to be processed and the required responses to these stimuli
For each stimulus and response, identify the timing constraints
Aggregate the stimulus and response processing into concurrent processes. A process may be associated with each class of stimulus and response
Design algorithms to process each class of stimulus and response. These must meet the given timing requirements
Design a scheduling system which will ensure that processes are started in time to meet their deadlines
Integrate using a real-time executive or operating system

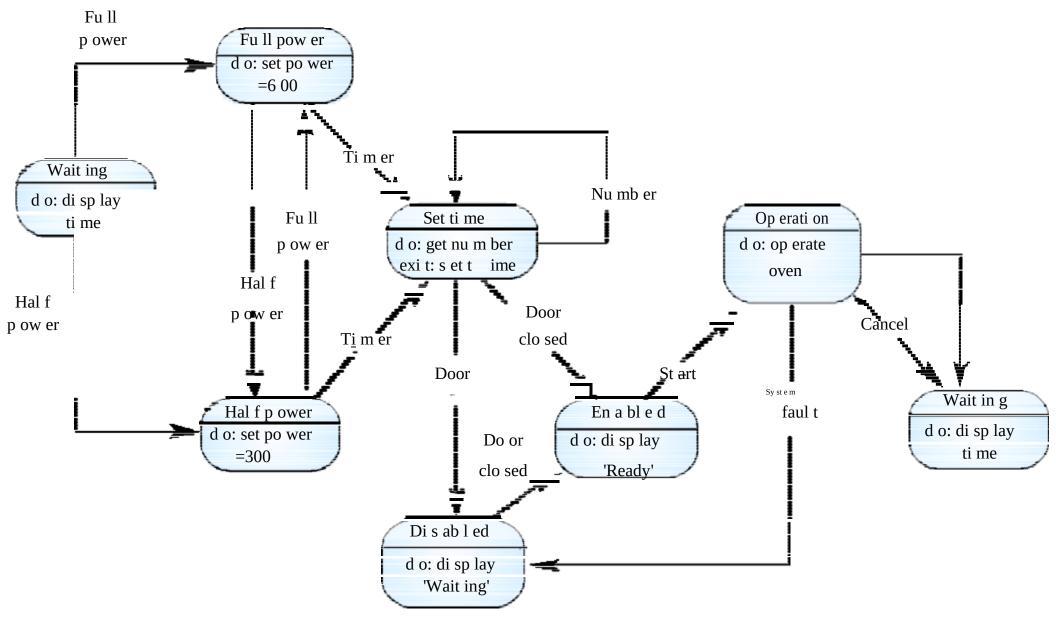
Timing constraints

For aperiodic stimuli, designers make assumptions about probability of occurrence of stimuli.
May mean that certain design strategies such as object-oriented design cannot be used because of the additional overhead involved

State machine modelling

The effect of a stimulus in a real-time system may trigger a transition from one state to another.
Finite state machines can be used for modelling real-time systems.
However, FSM models lack structure. Even simple systems can have a complex model.
The UML includes notations for defining state machine models

Microwave oven state machine



Real-time programming

Hard-real time systems may have to programmed in assembly language to ensure that deadlines are met

Languages such as C allow efficient programs to be written but do not have constructs to support concurrency or shared resource management

Ada as a language designed to support real-time systems design so includes a general purpose concurrency mechanism

Java as a real-time language

Java supports lightweight concurrency (threads and synchonized methods) and can be used for some soft real-time systems

Java 2.0 is not suitable for hard RT programming or programming where precise control of timing is required

- Not possible to specify thread execution time
- Uncontrollable garbage collection
- Not possible to discover queue sizes for shared resources
- Variable virtual machine implementation
- Not possible to do space or timing analysis

Real Time Executives

Real-time executives are specialised operating systems which manage processes in the RTS

Responsible for process management and resource (processor and memory) allocation

Storage management, fault management.

Components depend on complexity of system

Executive components

Real-time clock
- Provides information for process scheduling.

Interrupt handler
- Manages aperiodic requests for service.

Scheduler
- Chooses the next process to be run.

Resource manager
- Allocates memory and processor resources.

Dispatchers
- Starts process execution.

Non-stop system components

Configuration manager
- Responsible for the dynamic reconfiguration of the system software and hardware. Hardware modules may be replaced and software upgraded without stopping the systems

Fault manager
- Responsible for detecting software and hardware faults and taking appropriate actions (e.g. switching to backup disks) to ensure that the system continues in operation

Process priority

The processing of some types of stimuli must sometimes take priority
Interrupt level priority. Highest priority which is allocated to processes requiring a very fast response
Clock level priority. Allocated to periodic processes
Within these, further levels of priority may be assigned

Interrupt servicing

Control is transferred automatically to a pre-determined memory location
This location contains an instruction to jump to an interrupt service routine
Further interrupts are disabled, the interrupt serviced and control returned to the interrupted process
Interrupt service routines MUST be short, simple and fast

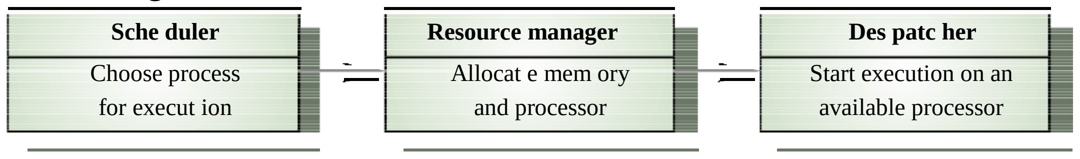
Periodic process servicing

In most real-time systems, there will be several classes of periodic process, each with different periods (the time between executions), execution times and deadlines (the time by which processing must be completed)
The real-time clock ticks periodically and each tick causes an interrupt which schedules the process manager for periodic processes
The process manager selects a process which is ready for execution

Process management

Concerned with managing the set of concurrent processes
Periodic processes are executed at pre-specified time intervals
The executive uses the real-time clock to determine when to execute a process
Process period - time between executions
Process deadline - the time by which processing must be complete

RTE process management


Sche duler
Choose process for execut ion
Resource manager
Allocat e mem ory and processor
Des patc her
Start execution on an available processor

Process switching

The scheduler chooses the next process to be executed by the processor. This depends on a scheduling strategy which may take the process priority into account
The resource manager allocates memory and a processor for the process to be executed
The despatcher takes the process from ready list, loads it onto a processor and starts execution

Scheduling strategies

Non pre-emptive scheduling
Once a process has been scheduled for execution, it runs to completion or until it is blocked for some reason (e.g. waiting for I/O)
Pre-emptive scheduling
The execution of an executing processes may be stopped if a higher priority process requires service

Scheduling algorithms
Round-robin
Shortest deadline first

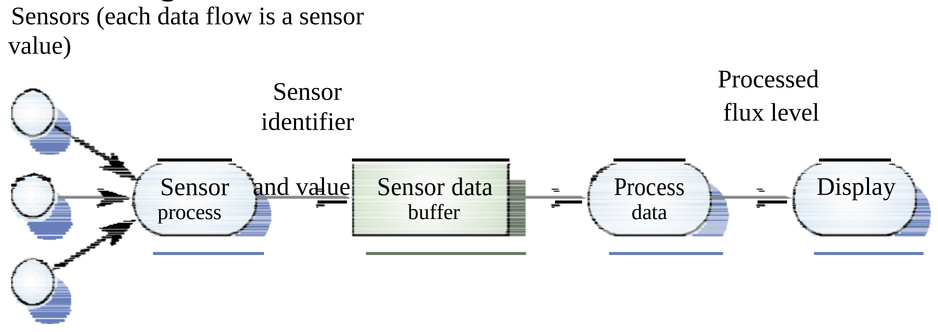
Data Acquisition System

Collect data from sensors for subsequent processing and analysis.
Data collection processes and processing processes may have different periods and deadlines.
Data collection may be faster than processing
e.g. collecting information about an explosion, scientific experiments
Circular or ring buffers are a mechanism for smoothing speed differences.

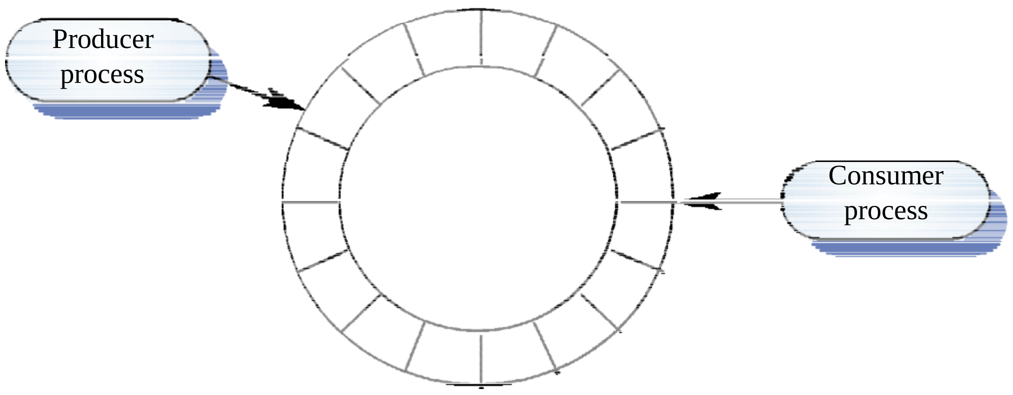
Reactor data collection

A system collects data from a set of sensors monitoring the neutron flux from a nuclear reactor.
Flux data is placed in a ring buffer for later processing.
The ring buffer is itself implemented as a concurrent process so that the collection and processing processes may be synchronized.

Reactor flux monitoring



A ring buffer



Mutual exclusion

Producer processes collect data and add it to the buffer. Consumer processes take data from the buffer and make elements available.
Producer and consumer processes must be mutually excluded from accessing the same element.
The buffer must stop producer processes adding information to a full buffer and consumer processes trying to take information from an empty buffer.

Java implementation of a ring buffer

```
class CircularBuffer
{
        int bufsize ;
        SensorRecord [] store ;
        int numberOfEntries = 0 ;
        int front = 0, back = 0 ;

        CircularBuffer (int n) {
                bufsize = n ;
                store = new SensorRecord [bufsize] ;
        } // CircularBuffer

        synchronized void put (SensorRecord rec ) throws
        InterruptedException {
                if ( numberOfEntries == bufsize)
                        wait () ;
                store [back] = new SensorRecord (rec.sensorId, rec.sensorVal)
                ; back = back + 1 ;
                if (back == bufsize)
                        back = 0 ;
                numberOfEntries = numberOfEntries + 1 ;
                notify () ;
        } // put

        synchronized SensorRecord get () throws InterruptedException
        {
                SensorRecord result = new SensorRecord (-1, -1) ;
                if (numberOfEntries == 0)
                                wait () ;
                result = store [front] ;
                front = front + 1 ;
                if (front == bufsize)
                        front = 0 ;
                numberOfEntries = numberOfEntries - 1 ;
                notify () ;
                return result ;
        } // get
} // CircularBuffer
```

Monitoring and Control System

Important class of real-time systems
Continuously check sensors and take actions depending on sensor values
Monitoring systems examine sensors and report their results
Control systems take sensor values and control hardware actuators
Burglar alarm system e.g
A system is required to monitor sensors on doors and windows to detect the presence of intruders in a building

When a sensor indicates a break-in, the system switches on lights around the area and calls police automatically

The system should include provision for operation without a mains power supply

Burglar alarm system

Sensors

- Movement detectors, window sensors, door sensors.
- 50 window sensors, 30 door sensors and 200 movement detectors
- Voltage drop sensor

Actions

- When an intruder is detected, police are called automatically.
- Lights are switched on in rooms with active sensors.
- An audible alarm is switched on.
- The system switches automatically to backup power when a voltage drop is detected.

The R-T system design process

Identify stimuli and associated responses
Define the timing constraints associated with each stimulus and response
Allocate system functions to concurrent processes
Design algorithms for stimulus processing and response generation
Design a scheduling system which ensures that processes will always be scheduled to meet their deadlines
Stimuli to be processed
Power failure

- Generated by a circuit monitor. When received, the system must switch to backup power within 50 ms

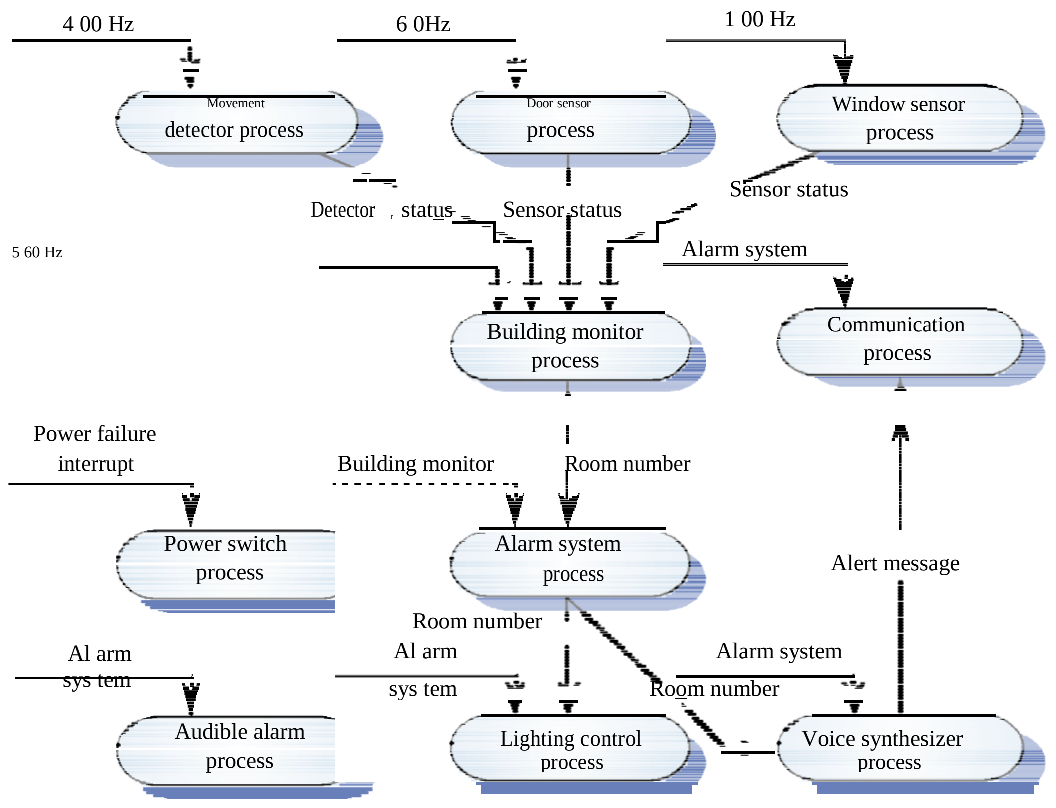
Intruder alarm

- Stimulus generated by system sensors. Response is to call the police, switch on building lights and the audible alarm

Timing requirements

Stimulus/Response	**Timing requirements**
Power fail interrupt	The switch to backup power must be completed within a deadline of 50 ms.
Door alarm	Each door alarm should be polled twice per second.
Window alarm	Each window alarm should be polled twice per second.
Movement detector	Each movement detector should be polled twice per second.
Audible alarm	The audible alarm should be switched on within 1/2 second of an alarm being raised by a sensor.
Lights switch	The lights should be switched on within 1/2 second of an alarm being raised by a sensor.
Communications	The call to the police should be started within 2 seconds of an alarm being raised by a sensor.
Voice synthesiser	A synthesised message should be available within 4 seconds of an alarm being raised by a sensor.

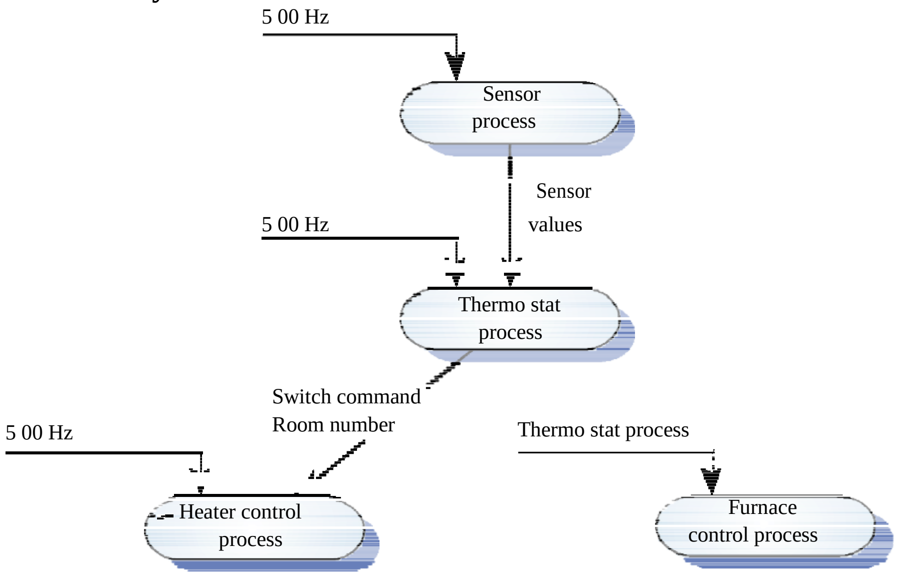
Process architecture



Building monitor process

```
class BuildingMonitor extends Thread {

        BuildingSensor win, door, move ;

        Siren   siren = new Siren () ;
        Lights lights = new Lights () ;
        Synthesizer synthesizer = new Synthesizer () ;
        DoorSensors doors = new DoorSensors (30) ; WindowSensors
                                windows = new WindowSensors (50) ;
        MovementSensors movements = new MovementSensors (200)
        ; PowerMonitor pm = new PowerMonitor () ;

        BuildingMonitor()
        {
                  initialise all the sensors and start the
                processes siren.start () ; lights.start () ;
                synthesizer.start () ; windows.start () ;
                doors.start () ; movements.start () ; pm.start () ;
        }
        public void run ()
        {
                int room = 0 ;
                while (true)
                {
                           poll the movement sensors at least twice per second (400
                        Hz) move = movements.getVal () ;
                           poll the window sensors at least twice/second (100 Hz)
                        win = windows.getVal () ;
                           poll the door sensors at least twice per second (60
                        Hz) door = doors.getVal () ;
                        if (move.sensorVal == 1 | door.sensorVal == 1 | win.sensorVal == 1)
                                {
                                           a sensor has indicated an intruder
                                        if (move.sensorVal == 1)      room = move.room ;
                                        if (door.sensorVal == 1)      room = door.room ;
                                        if (win.sensorVal == 1 )              room = win.room ;

                                        lights.on (room) ; siren.on () ; synthesizer.on (room)
                                        ; break ;
                                }
                }
                lights.shutdown () ; siren.shutdown () ; synthesizer.shutdown () ;
                windows.shutdown () ; doors.shutdown () ; movements.shutdown () ;

        } // run
} //BuildingMonitor
```

A temperature control system



Control systems

A burglar alarm system is primarily a monitoring system. It collects data from sensors but no real-time actuator control

- Control systems are similar but, in response to sensor values, the system sends control signals to actuators

An example of a monitoring and control system is a system which monitors temperature and switches heaters on and off

Unit – IV: SCM and SPM

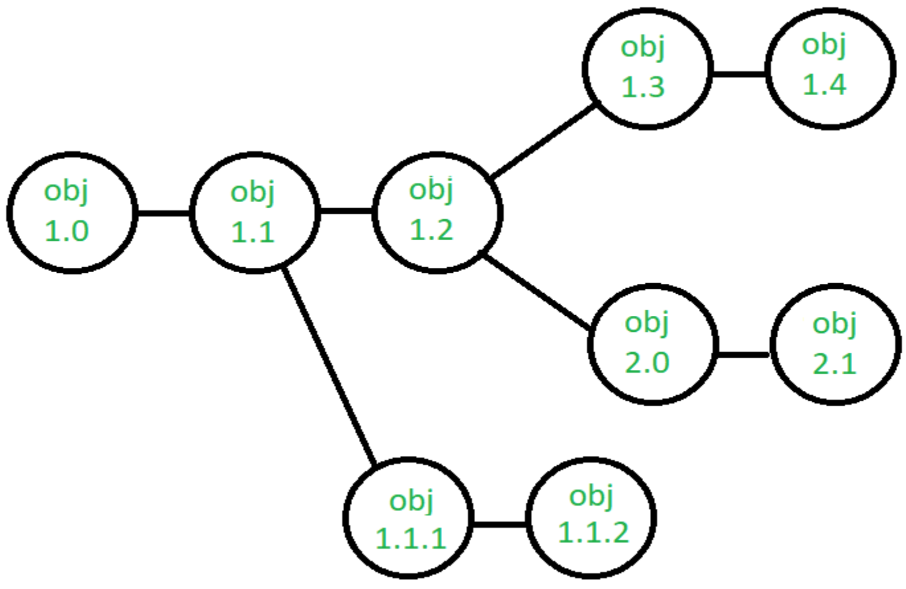
System Configuration Management (SCM)

System Configuration Management (SCM) is an arrangement of exercises which controls change by recognizing the items for change, setting up connections between those things, making/characterizing instruments for overseeing diverse variants, controlling the changes being executed in the current framework, inspecting and revealing/reporting on the changes made. It is essential to control the changes in light of the fact that if the changes are not checked legitimately then they may wind up undermining a well-run programming. In this way, SCM is a fundamental piece of all project management activities.

Processes involved in SCM

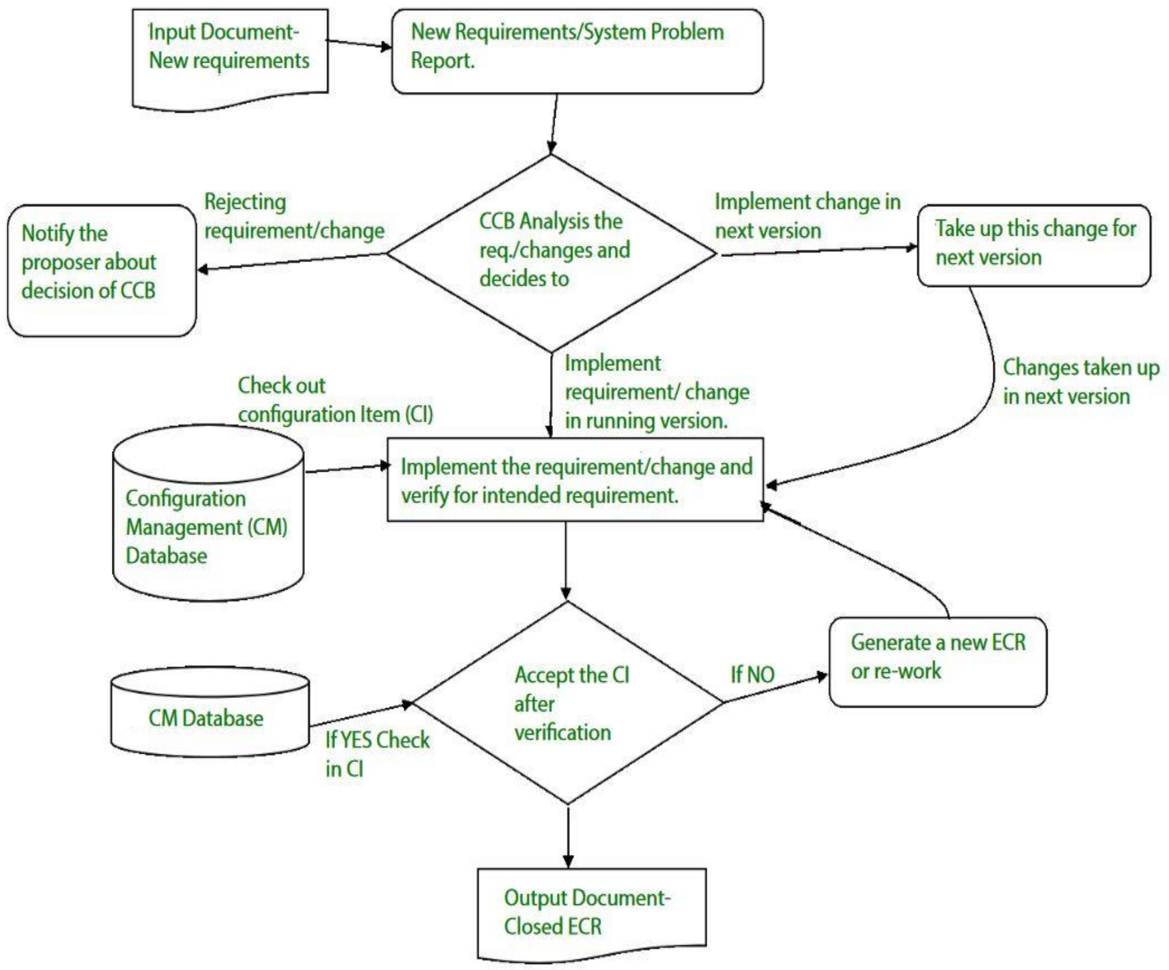
Configuration management provides a disciplined environment for smooth control of work products. It involves the following activities:

1. **Identification and Establishment –** Identifying the configuration items from products that compose baselines at given points in time (a baseline is a set of mutually consistent Configuration Items, which has been formally reviewed and agreed upon, and serves as the basis of further development). Establishing relationship among items, creating a mechanism to manage multiple level of control and procedure for change management system.
2. **Version control –** Creating versions/specifications of the existing product to build new products from the help of SCM system. A description of version is given below:



Suppose after some changes, the version of configuration object changes from 1.0 to 1.1. Minor corrections and changes result in versions 1.1.1 and 1.1.2, which is followed by a major update that is object 1.2. The development of object 1.0 continues through 1.3 and 1.4, but finally, a noteworthy change to the object results in a new evolutionary path, version 2.0. Both versions are currently supported.

3. **Change control** – Controlling changes to Configuration items (CI). The change control process is explained in Figure below:


Input Document-New requirements
New Requirements/System Problem Report.
CCB Analysis the req./changes and decides to
Rejecting requirement/change
Notify the proposer about decision of CCB
Implement change in next version
Take up this change for next version
Changes taken up in next version
Implement requirement/ change in running version.
Check out configuration Item (CI)
Configuration Management (CM) Database
Implement the requirement/change and verify for intended requirement.
Accept the CI after verification
If NO
Generate a new ECR or re-work
CM Database
If YES Check in CI
Output Document-Closed ECR

A change request (CR) is submitted and evaluated to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change. The results of the evaluation are presented as a change report, which is used by a change control board (CCB) —a person or group who makes a final decision on the status and priority of the change. An engineering change Request (ECR) is generated for each approved change.

Also CCB notifies the developer in case the change is rejected with proper reason. The ECR describes the change to be made, the constraints that must be respected, and the criteria for review and audit. The object to be changed is "checked out" of the project database, the change is made, and then the object is tested again. The object is then "checked in" to the database and appropriate version control mechanisms are used to create the next version of the software.

4. **Configuration auditing** – A software configuration audit complements the formal technical review of the process and product. It focuses on the technical correctness of the configuration

object that has been modified. The audit confirms the completeness, correctness and consistency of items in the SCM system and track action items from the audit to closure.

5. **Reporting –** Providing accurate status and current configuration data to developers, tester, end users, customers and stakeholders through admin guides, user guides, FAQs, Release notes, Memos, Installation Guide, Configuration guide etc .

SCM Tools

Different tools are available in market for SCM like: CFEngine, Bcfg2 server, Vagrant, SmartFrog, CLEAR CASETOOL (CC), SaltStack, CLEAR QUEST TOOL, Puppet, SVN- Subversion, Perforce, TortoiseSVN, IBM Rational team concert, IBM Configuration management version management, Razor, Ansible, etc.

SCM Standards

While there is no single definition of CM, there are three widely disseminated views from three different sources: the Institute of Electrical and Electronics Engineers (IEEE), The International Organisation for Standardisation (ISO), and the Software Engineering Institute (SEI) at Carnegie Mellon University.

❖ **<u>The IEEE perspective on CM</u>**

A most widely understood description of the practices associated with configuration management is found in the IEEE Standard 828-1990, *Software Configuration Management Plans.*

[Numbers in brackets are added]

"SCM activities are traditionally grouped into four functions: [1] configuration identification, [2] configuration control, [3] status accounting, and [4] configuration audits and reviews."

IEEE Standard 828-1990 goes on to list specific activities associated with each of the four functions (the number of the paragraph containing the reference appears in parentheses):

- **Identification:** identify, name, and describe the documented physical and functional characteristics of the code, specifications, design, and data elements to be controlled for the project. (Paragraph 2.3.1)
- **Control:** request, evaluate, approve or disapprove, and implement changes (Paragraph 2.3.2)
- **Status accounting:** record and report the status of project configuration items [initial approved version. status of requested changes, implementation status of approved changes] (Paragraph 2.3.3)
- **Audits and reviews:** determine to what extent the actual configuration item reflects the required physical and functional characteristics (Paragraph 2.3.4)

This list is similar to the set of activities noted by Pressman:

"Software configuration management is an umbrella activity ... developed to (1) identify change, (2) control change, (3) ensure that change is being properly implemented, and (4) report change to others who may have an interest."

❖ **<u>The ISO perspective on CM</u>**

In the guideline document, ISO 9000-3:1991 *Guidelines for the application of ISO 9001 to the development, supply and maintenance of software*, the International Organisation for Standardisation identifies a similar set of practices as CM:

"Configuration management provides a mechanism for identifying, controlling and tracking the versions of each software item. In many cases earlier versions still in use must also be maintained and controlled.

"The [CM] system should

"a) identify uniquely the versions of each software item;

"b) identify the versions of each software item which together constitute a specific version of a complete product;

"c) identity the build status of software products in development or delivered and installed;

"d) control simultaneous updating of a given software item by more than one person;

"e) provide coordination for the updating of multiple products in one or more locations as required;

"f) identify and track all actions and changes resulting from a change request, from initiation ... to release."

❖ **<u>The SEI perspective on CM</u>**

Based on a review of currently available tools and an evolving understanding of the organizational role of CM, the SEI advocates a broader definition of CM in SEI-92-TR-8:

"The standard definition for CM taken from IEEE standard 729-1983 [updated as IEEE Std 610.12-1990] includes:

"Identification: identifying the structure of the product, its components and their type, and making them unique and accessible in some form

"Control: controlling the release of product and changes to it throughout the life cycle �

"Status Accounting: recording and reporting the status of components and change requests, and

gathering vital statistics about components in the product

"*Audit and review*: validating the completeness of a product and maintaining consistency among the components �

"[The IEEE] definition of CM � needs to be broadened to encompass � :

"*Manufacturing*: managing the construction and building of the product

"*Process management*: ensuring the correct execution of the organization's procedures, policies, and life-cycle model

"*Team work*: controlling the work and interactions between multiple developers on a product."

Software Project Management (SPM)

Measurement and Metrics

Software Measurement: A measurement is an manifestation of the size, quantity, amount or dimension of a particular attributes of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process. It is an authority within software engineering. Software measurement process is defined and governed by ISO Standard.

Need of Software Measurement:

Software is measured to:

1. Create the quality of the current product or process.
2. Anticipate future qualities of the product or process.
3. Enhance the quality of a product or process.
4. Regulate the state of the project in relation to budget and schedule.

Classification of Software Measurement:

There are 2 types of software measurement:

1. **Direct Measurement:**
   In direct measurement the product, process or thing is measured directly using standard scale.
2. **Indirect Measurement:**
   In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference.

Metrics:

A metrics is a measurement of the level that any impute belongs to a system product or process. There are 4 functions related to software metrics:

1. Planning
2. Organizing
3. Controlling
4. Improving

Characteristics of software Metrics:

1. **Quantitative:**
   Metrics must possess quantitative nature.It means metrics can be expressed in values.
2. **Understandable:**
   Metric computation should be easily understood ,the method of computing metric should be clearly defined.
3. **Applicability:**
   Metrics should be applicable in the initial phases of development of the software.
4. **Repeatable:**
   The metric values should be same when measured repeatedly and consistent in nature.
5. **Economical:**
   Computation of metric should be economical.
6. **Language Independent:**
   Metrics should not depend on any programming language.

Classification of Software Metrics:

There are 2 types of software metrics:

1. **Product Metrics:**
   Product metrics are used to evaluate the state of the product, tracing risks and undercovering prospective problem areas. The ability of team to control quality is evaluated.

2. **Process Metrics:**
   Process metrics pay particular attention on enhancing the long term process of the team or organisation.
3. **Project Metrics:**
   Project matrix is describes the project characteristic and execution process.
   - Number of software developer
   - Staffing pattern over the life cycle of software
   - Cost and schedule
   - Productivity

Project Estimation

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation**

  Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

- **Effort estimation**

  The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.

- **Time estimation**

  Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months.

  The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

- **Cost estimation**

  This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

  - Size of software
  - Software quality
  - Hardware
  - Additional software or tools, licenses etc.
  - Skilled personnel with task-specific skills
  - Travel involved
  - Communication
  - Training and support

Project Estimation Techniques

We discussed various parameters involving project estimation such as size, effort, time and cost.

Project manager can estimate the listed factors using two broadly recognized techniques –

- **Decomposition Technique**

This technique assumes the software as a product of various compositions.

There are two main models -

- **Line of Code** Estimation is done on behalf of number of line of codes in the software product.
- **Function Points** Estimation is done on behalf of number of function points in the software product.

- **Empirical Estimation Technique**

This technique uses empirically derived formulae to make estimation.These formulae are based on LOC or FPs.

- **Putnam Model**

  This model is made by Lawrence H. Putnam, which is based on Norden's frequency distribution (Rayleigh curve). Putnam model maps time and efforts required with software size.

- **COCOMO**

  COCOMO stands for COnstructive COst MOdel, developed by Barry W. Boehm. It divides the software product into three categories of software: organic, semi-detached and embedded.

Empirical Estimation Technique : COCOMO Model

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality. It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.
The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.

- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

Boehm's definition of organic, semidetached, and embedded systems:

1. **Organic** – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
2. **Semi-detached** – A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type.
3. **Embedded** – A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

All the above system types utilize different values of the constants used in Effort Calculations.

Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

The first level, **Basic COCOMO** can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations. **Intermediate COCOMO** takes these Cost Drivers into account and **Detailed COCOMO** additionally accounts for the influence of individual project phases, i.e in case of Detailed it accounts for both these cost drivers and also calculations are performed phase wise henceforth producing a more accurate result. These two models are further discussed below.

Estimation of Effort: Calculations –

Basic Model –

The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a and b for the Basic Model for the different categories of system:

SOFTWARE PROJECTS	A	B
Organic	2.4	1.05
Semi Detached	3.0	1.12
Embedded	3.6	1.20

The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.

Intermediate Model –

The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system. However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their attributes:

(i) Product attributes –

- Required software reliability extent
- Size of the application database
- The complexity of the product

(ii) Hardware attributes –

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

(iii) Personnel attributes –

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

(iv) Project attributes –

- Use of software tools
- Application of software engineering methods
- Required development schedule

;

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Product Attributes					
Required Software Reliability	0.75	0.88	1.00	1.15	1.40
Size of Application Database		0.94	1.00	1.08	1.16
Complexity of The Product	0.70	0.85	1.00	1.15	1.30
Hardware Attributes					
Runtime Performance Constraints			1.00	1.11	1.30
Memory Constraints			1.00	1.06	1.21
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30
Required turnabout time		0.94	1.00	1.07	1.15
Personnel attributes					
Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications experience	1.29	1.13	1.00	0.91	0.82
Software engineer capability	1.42	1.17	1.00	0.86	0.70
Virtual machine experience	1.21	1.10	1.00	0.90	
Programming language experience	1.14	1.07	1.00	0.95	
Project Attributes					
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82
Use of software tools	1.24	1.10	1.00	0.91	0.83
Required development schedule	1.23	1.08	1.00	1.04	1.10

The project manager is to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, appropriate cost driver values are taken from the above table. These 15 values are then multiplied to calculate the EAF (Effort Adjustment Factor). The Intermediate COCOMO formula now takes the form:

The values of a and b in case of the intermediate model are as follows:

SOFTWARE PROJECTS	A	B
Organic	3.2	1.05
Semi Detached	3.0	1.12
Embeddedc	2.8	1.20

Detailed Model

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.
The Six phases of detailed COCOMO are:

16. Planning and requirements
17. System design
18. Detailed design
19. Module code and test
20. Integration and test
21. Cost Constructive model

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.

Project Scheduling

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks, and project milestones and arrange them keeping various factors in mind. They look for tasks lie in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. Arrangement of tasks which lies out of critical path are less likely to impact over all schedule of the project.

For scheduling a project, it is necessary to -

- Break down the project tasks into smaller, manageable form
- Find out various tasks and correlate them
- Estimate time frame required for each task
- Divide time into work-units
- Assign adequate number of work-units for each task
- Calculate total time required for the project from start to finish

Resource management

All elements used to develop a software product may be assumed as resource for that project. This may include human resource, productive tools and software libraries.

The resources are available in limited quantity and stay in the organization as a pool of assets. The shortage of resources hampers the development of project and it can lag behind the schedule. Allocating extra resources increases development cost in the end. It is therefore necessary to estimate and allocate adequate resources for the project.

Resource management includes -

- Defining proper organization project by creating a project team and allocating responsibilities to each team member
- Determining resources required at a particular stage and their availability
- Manage Resources by generating resource request when they are required and de-allocating them when they are no more needed.

Project Risk Management

Risk management involves all activities pertaining to identification, analyzing and making provision for predictable and non-predictable risks in the project. Risk may include the following:

- Experienced staff leaving the project and new staff coming in.
- Change in organizational management.
- Requirement change or misinterpreting requirement.

- Under-estimation of required time and resources.
- Technological changes, environmental changes, business competition.

Risk Management Process

There are following activities involved in risk management process:

- **Identification -** Make note of all possible risks, which may occur in the project.
- **Categorize -** Categorize known risks into high, medium and low risk intensity as per their possible impact on the project.
- **Manage -** Analyze the probability of occurrence of risks at various phases. Make plan to avoid or face risks. Attempt to minimize their side-effects.
- **Monitor -** Closely monitor the potential risks and their early symptoms. Also monitor the effects of steps taken to mitigate or avoid them.

Project Execution & Monitoring

In this phase, the tasks described in project plans are executed according to their schedules.

Execution needs monitoring in order to check whether everything is going according to the plan. Monitoring is observing to check the probability of risk and taking measures to address the risk or report the status of various tasks.

These measures include -

- **Activity Monitoring -** All activities scheduled within some task can be monitored on day-to-day basis. When all activities in a task are completed, it is considered as complete.
- **Status Reports -** The reports contain status of activities and tasks completed within a given time frame, generally a week. Status can be marked as finished, pending or work-in-progress etc.
- **Milestones Checklist -** Every project is divided into multiple phases where major tasks are performed (milestones) based on the phases of SDLC. This milestone checklist is prepared once every few weeks and reports the status of milestones.

Project Communication Management

Effective communication plays vital role in the success of a project. It bridges gaps between client and the organization, among the team members as well as other stake holders in the project such as hardware suppliers.

Communication can be oral or written. Communication management process may have the following steps:

- **Planning** - This step includes the identifications of all the stakeholders in the project and the mode of communication among them. It also considers if any additional communication facilities are required.

- **Sharing** - After determining various aspects of planning, manager focuses on sharing correct information with the correct person on correct time. This keeps every one involved the project up to date with project progress and its status.
- **Feedback** - Project managers use various measures and feedback mechanism and create status and performance reports. This mechanism ensures that input from various stakeholders is coming to the project manager as their feedback.
- **Closure** - At the end of each major event, end of a phase of SDLC or end of the project itself, administrative closure is formally announced to update every stakeholder by sending email, by distributing a hardcopy of document or by other mean of effective communication.

Configuration Management

Configuration management is a process of tracking and controlling the changes in software in terms of the requirements, design, functions and development of the product.

IEEE defines it as "the process of identifying and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items".

Generally, once the SRS is finalized there is less chance of requirement of changes from user. If they occur, the changes are addressed only with prior approval of higher management, as there is a possibility of cost and time overrun.

Baseline

A phase of SDLC is assumed over if it baselined, i.e. baseline is a measurement that defines completeness of a phase. A phase is baselined when all activities pertaining to it are finished and well documented. If it was not the final phase, its output would be used in next immediate phase.

Configuration management is a discipline of organization administration, which takes care of occurrence of any change (process, requirement, technological, strategical etc.) after a phase is baselined. CM keeps check on any changes done in software.

Change Control

Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

A change in the configuration of product goes through following steps -

- **Identification** - A change request arrives from either internal or external source. When change request is identified formally, it is properly documented.
- **Validation** - Validity of the change request is checked and its handling procedure is confirmed.
- **Analysis -** The impact of change request is analyzed in terms of schedule, cost and required efforts. Overall impact of the prospective change on system is analyzed.
- **Control** - If the prospective change either impacts too many entities in the system or it is unavoidable, it is mandatory to take approval of high authorities before change is incorporated into the system. It is decided if the change is worth incorporation or not. If it is not, change request is refused formally.
- **Execution** - If the previous phase determines to execute the change request, this phase take appropriate actions to execute the change, does a thorough revision if necessary.
- **Close request** - The change is verified for correct implementation and merging with the rest of the system. This newly incorporated change in the software is documented properly and the request is formally is closed.

Unit V : Software Testing

Software Testing Overview

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

Software Validation

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

- Validation ensures the product under development is as per the user requirements.
- Validation answers the question – "Are we developing the product which attempts all that user needs from this software ?".
- Validation emphasizes on user requirements.

Software Verification

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verification answers the question– "Are we developing this product by firmly following all design specifications ?"
- Verifications concentrates on the design and system specifications.

Target of the test are -

- **Errors -** These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
- **Fault -** When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
- **Failure -** failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

Manual Vs Automated Testing

Testing can either be done manually or using an automated testing tool:

- **Manual -** This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.

  Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.

- **Automated** This testing is a testing procedure done with aid of automated testing tools. The limitations with manual testing can be overcome using automated test tools.

A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually.

There are software and hardware tools which helps tester in conducting load testing, stress testing, regression testing.

Testing Approaches

Tests can be conducted based on two approaches –

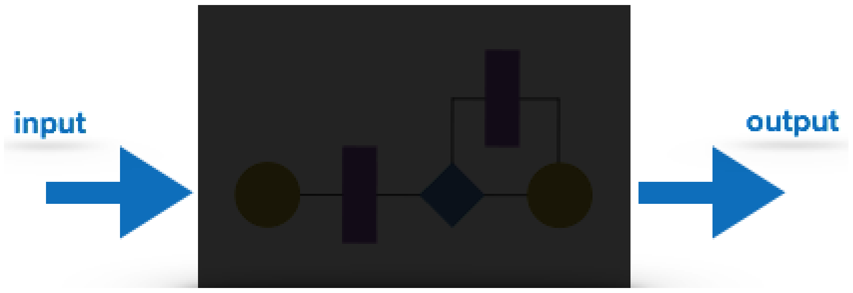
- Functionality testing
- Implementation testing

When functionality is being tested without taking the actual implementation in concern it is known as black-box testing. The other side is known as white-box testing where not only functionality is tested but the way it is implemented is also analyzed.

Exhaustive tests are the best-desired method for a perfect testing. Every single possible value in the range of the input and output values is tested. It is not possible to test each and every value in real world scenario if the range of values is large.

Black-box testing

It is carried out to test functionality of the program. It is also called 'Behavioral' testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested 'ok', and problematic otherwise.



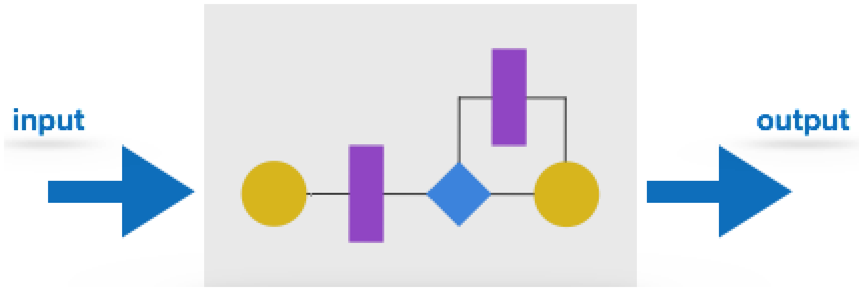
In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

Black-box testing techniques:

- **Equivalence class -** The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.
- **Boundary values -** The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.
- **Cause-effect graphing -** In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.
- **Pair-wise Testing -** The behavior of software depends on multiple parameters. In pairwise testing, the multiple parameters are tested pair-wise for their different values.
- **State-based testing -** The system changes state on provision of input. These systems are tested based on their states and input.

White-box testing

It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as 'Structural' testing.



In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

The below are some White-box testing techniques:

- **Control-flow testing -** The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.
- **Data-flow testing -** This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

<u>Testing Documentation</u>

Testing documents are prepared at different stages -

Before Testing

Testing starts with test cases generation. Following documents are needed for reference –

- **SRS document -** Functional Requirements document
- **Test Policy document -** This describes how far testing should take place before releasing the product.
- **Test Strategy document -** This mentions detail aspects of test team, responsibility matrix and rights/responsibility of test manager and test engineer.
- **Traceability Matrix document -** This is SDLC document, which is related to requirement gathering process. As new requirements come, they are added to this matrix. These matrices help testers know the source of requirement. They can be traced forward and backward.

While Being Tested

The following documents may be required while testing is started and is being done:

- **Test Case document -** This document contains list of tests required to be conducted. It includes Unit test plan, Integration test plan, System test plan and Acceptance test plan.
- **Test description -** This document is a detailed description of all test cases and procedures to execute them.
- **Test case report -** This document contains test case report as a result of the test.
- **Test logs -** This document contains test logs for every test case report.

After Testing

The following documents may be generated after testing :

- **Test summary -** This test summary is collective analysis of all test reports and logs. It summarizes and concludes if the software is ready to be launched. The software is released under version control system if it is ready to launch.

Testing vs. Quality Control, Quality Assurance and Audit

We need to understand that software testing is different from software quality assurance, software quality control and software auditing.

- **Software quality assurance -** These are software development process monitoring means, by which it is assured that all the measures are taken as per the standards of organization. This monitoring is done to make sure that proper software development methods were followed.
- **Software quality control -** This is a system to maintain the quality of software product. It may include functional and non-functional aspects of software product, which enhance the goodwill of the organization. This system makes sure that the customer is receiving quality product for their requirement and the product certified as ‘fit for use’.
- **Software audit -** This is a review of procedure used by the organization to develop the software. A team of auditors, independent of development team examines the software process, procedure, requirements and other aspects of SDLC. The purpose of software audit is to check that software and its development process, both conform standards, rules and regulations.

Strategy of testing

A strategy of software testing is shown in the context of spiral.

Following figure shows the testing strategy:

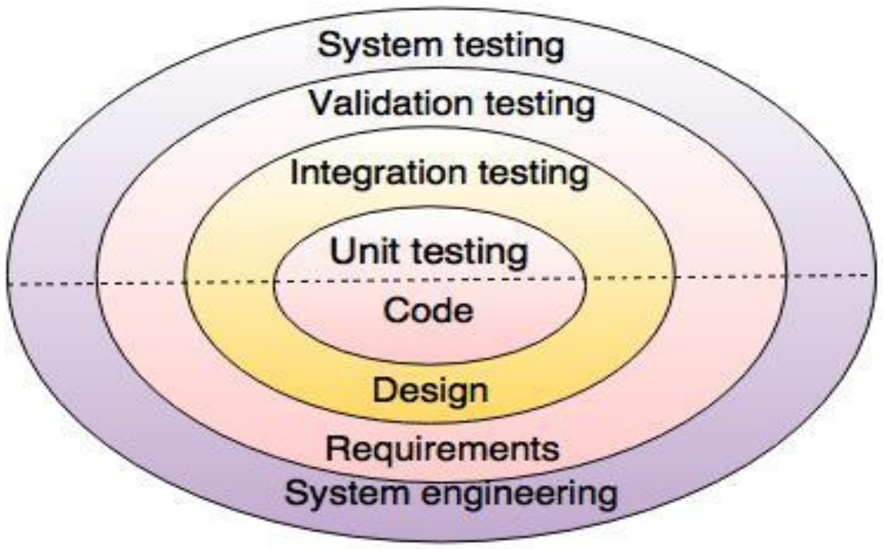


Fig. - Testing Strategy

Unit testing

Unit testing starts at the centre and each unit is implemented in source code.

Integration testing

An integration testing focuses on the construction and design of the software.

Validation testing

Check all the requirements like functional, behavioral and performance requirement are validate against the construction software.

System testing

System testing confirms all system elements and performance are tested entirely.

Testing strategy for procedural point of view

As per the procedural point of view the testing includes following steps.

1) Unit testing
2) Integration testing
3) High-order tests
4) Validation testing

These steps are shown in following figure:

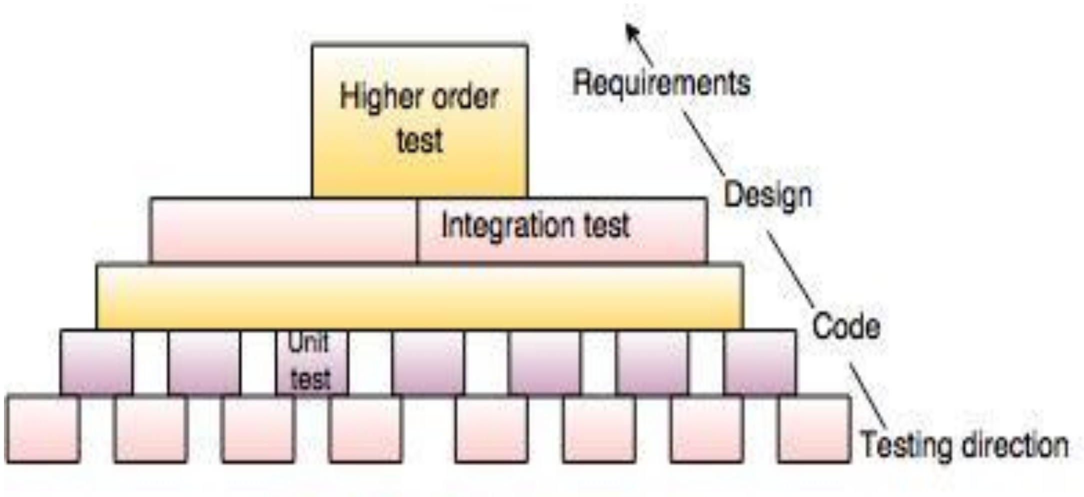


Fig.- Steps of software testing

Software Testing Levels

There are different levels during the process of testing. In this chapter, a brief description is provided about these levels.

Levels of testing include different methodologies that can be used while conducting software testing. The main levels of software testing are −

- Functional Testing
- Non-functional Testing

Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved while testing an application for functionality.

Steps	Description
I	The determination of the functionality that the intended application is meant to perform.
II	The creation of test data based on the specifications of the application.
III	The output based on the test data and the specifications of the application.
IV	The writing of test scenarios and the execution of test cases.
V	The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

Unit Testing

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Limitations of Unit Testing

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Integration Testing

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

Sr.No.	Integration Testing Method
1	**Bottom-up integration** This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2	**Top-down integration** In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

System Testing

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons −

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons −

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.

Acceptance Testing

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client’s requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application.

By performing acceptance tests on an application, the testing team will reduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

Alpha Testing

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application −

- Spelling Mistakes
- Broken Links
- Cloudy Directions
- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

Beta Testing

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as **pre-release testing**. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following −

- Users will install, run the application and send their feedback to the project team.
- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.

- Having a higher-quality application when you release it to the general public will increase customer satisfaction.

Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are discussed below.

Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. There are different causes that contribute in lowering the performance of a software −

- Network delay
- Client-side processing
- Database transaction processing
- Load balancing between servers
- Data rendering

Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects −

- Speed (i.e. Response Time, data rendering and accessing)
- Capacity
- Stability
- Scalability

Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as **Load testing** and **Stress testing**.

Load Testing

It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time.

Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software. The number of users can be increased or decreased concurrently or incrementally based upon the requirements.

Stress Testing

Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as −

- Shutdown or restart of network ports randomly
- Turning the database on or off
- Running different processes that consume resources such as CPU, memory, server, etc.

Usability Testing

Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation.

According to Nielsen, usability can be defined in terms of five factors, i.e. efficiency of use, learn-ability, memory-ability, errors/safety, and satisfaction. According to him, the usability of a product will be good and the system is usable if it possesses the above factors.

Nigel Bevan and Macleod considered that usability is the quality requirement that can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled and the end-user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

Molich in 2000 stated that a user-friendly system should fulfill the following five goals, i.e., easy to Learn, easy to remember, efficient to use, satisfactory to use, and easy to understand.

In addition to the different definitions of usability, there are some standards and quality models and methods that define usability in the form of attributes and sub-attributes such as ISO-9126, ISO-9241-11, ISO-13407, and IEEE std.610.12, etc.

UI vs Usability Testing

UI testing involves testing the Graphical User Interface of the Software. UI testing ensures that the GUI functions according to the requirements and tested in terms of color, alignment, size, and other properties.

On the other hand, usability testing ensures a good and user-friendly GUI that can be easily handled. UI testing can be considered as a sub-part of usability testing.

Security Testing

Security testing involves testing a software in order to identify any flaws and gaps from security and vulnerability point of view. Listed below are the main aspects that security testing should ensure −

- Confidentiality
- Integrity
- Authentication
- Availability
- Authorization
- Non-repudiation
- Software is secure against known and unknown vulnerabilities
- Software data is secure
- Software is according to all security regulations
- Input checking and validation

- SQL insertion attacks
- Injection flaws
- Session management issues
- Cross-site scripting attacks
- Buffer overflows vulnerabilities
- Directory traversal attacks

Portability Testing

Portability testing includes testing a software with the aim to ensure its reusability and that it can be moved from another software as well. Following are the strategies that can be used for portability testing −

- Transferring an installed software from one computer to another.
- Building executable (.exe) to run the software on different platforms.

Portability testing can be considered as one of the sub-parts of system testing, as this testing type includes overall testing of a software with respect to its usage over different environments. Computer hardware, operating systems, and browsers are the major focus of portability testing. Some of the pre-conditions for portability testing are as follows −

- Software should be designed and coded, keeping in mind the portability requirements.
- Unit testing has been performed on the associated components.
- Integration testing has been performed.
- Test environment has been established.

Differences Between Black Box Testing and White Box Testing

#	Black Box Testing	White Box Testing
1	Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.
2	This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.
3	Implementation Knowledge is not required to carry out Black Box Testing.	Implementation Knowledge is required to carry out White Box Testing.
4	Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.
5	Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
6	Black box testing means functional test or external testing.	White box testing means structural test or interior testing.
7	In Black Box testing is primarily concentrate on the functionality of the system under test.	In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.
8	The main aim of this testing to check on what functionality is performing by the system under test.	The main aim of White Box testing to check on how System is performing.
9	Black Box testing can be started based on Requirement Specifications documents.	White Box testing can be started based on Detail Design documents.
10	The Functional testing, Behavior testing, Close box testing is carried out under Black Box testing, so there is no required of the programming	The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know

knowledge.	about programming knowledge.

Gray Box Testing

Gray Box Testing is a software testing technique which is a combination of Black Box Testing technique and White Box Testing technique. In Black Box Testing technique, tester is unknown to the internal structure of the item being tested and in White Box Testing the internal structure is known to tester. The internal structure is partially known in Gray Box Testing. This includes access to internal data structures and algorithms for purpose of designing the test cases.
Gray Box Testing is named so because the software program is like a semitransparent or grey box inside which tester can partially see. It commonly focuses on context-specific errors related to web systems.

Black Box Testing
+
White Box Testing
=
Gray Box Testing

Objective of Gray Box Testing:

The objective of Gray Box Testing is:

1. To provide combined advantages of both black box testing and white box testing.
2. To combine the input of developers as well as testers.
3. To improve overall product quality.
4. To reduce the overhead of long process of functional and non-functional testings.
5. To provide enough free time to developers to fix defects.
6. To test from the user point of view rather than a designer point of view.

Gray Box Testing Techniques:

- **Matrix Testing:**
  In matrix testing technique, business and technical risks which are defined by the developers in software programs are examined. Developers define all the variables that exist in the program. Each of the variables has an inherent technical and business risk and can be used

with varied frequencies during its life cycle.

- **Pattern Testing:**
  To perform the testing, previous defects are analyzed. It determines the cause of the failure by looking into the code. Analysis template includes reasons for the defect. This helps test cases designed as they are proactive in finding other failures before hitting production.
- **Orthogonal Array Testing:**
  It is mainly a black box testing technique. In orthogonal array testing, test data have n numbers of permutations and combinations. Orthogonal array testing is preferred when maximum coverage is required when there are very few test cases and test data is large. This is very helpful in testing complex applications.
- **Regression Testing:**
  Regression testing is testing the software after every change in the software to make sure that the changes or the new functionalities are not affecting the existing functioning of the system. Regression testing is also carried out to ensure that fixing any defect has not affected other functionality of the software.

Advantages of Gray Box Testing:

- Users and developers have clear goals while doing testing.
- Gray box testing is mostly done by the user perspective.
- Testers are not required to have high programming skills for this testing.
- Gray box testing is non-intrusive.
- Overall quality of the product is improved.
- In gray box testing, developers have more time for defect fixing.
- By doing gray box testing, benefits of both black box and white box testing is obtained.
- Gray box testing is unbiased. It avoids conflicts between a tester and a developer.
- Gray box testing is much more effective in integration testing.

Disadvantages of gray box testing:

- Defect association is difficult when gray testing is performed for distributed systems.
- Limited access to internal structure leads to limited access for code path traversal.
- Because source code cannot be accessed, doing complete white box testing is not possible.
- Gray box testing is not suitable for algorithm testing.
- Most of the test cases are difficult to design.

Black Box Testing

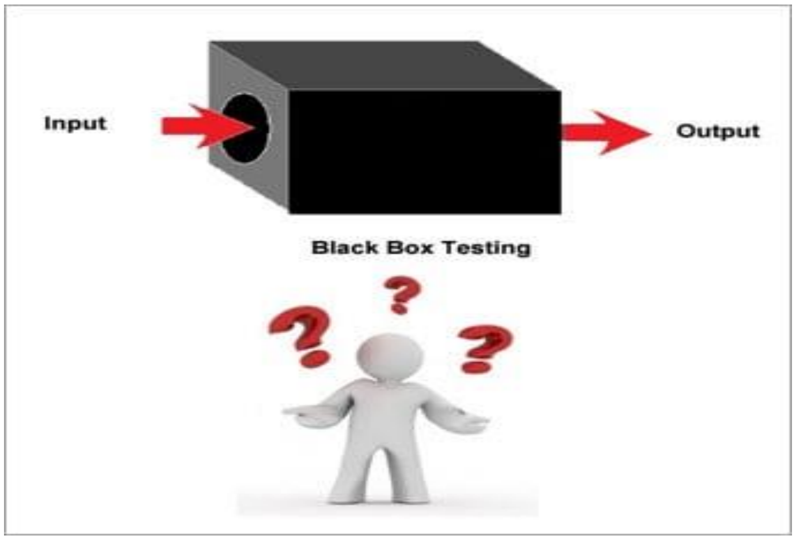
Black Box Testing is also known as behavioral, opaque-box, closed-box, specification-based or eye-to-eye testing.

It is a Software Testing method that analyses the functionality of a software/application without knowing much about the internal structure/design of the item that is being tested and compares the input value with the output value.

The main focus in Black Box Testing is on the functionality of the system as a whole. The term **'Behavioral Testing'** is also used for Black Box Testing. Behavioral test design is slightly different from the black-box test design because the use of internal knowledge isn't strictly forbidden, but it's still discouraged.
Each testing method has its own advantages and disadvantages. There are some bugs that cannot be found using the only black box or only white box technique.

Majority of the applications are tested by Black Box method. We need to cover the majority of test cases so that most of the bugs will get discovered by a Black-Box method.
This testing occurs throughout the software development and Testing Life Cycle i.e in Unit, Integration, System, Acceptance, and Regression Testing stages.

This can be both Functional or Non-Functional.



Types Of Black Box Testing

Practically, there are several types of Black Box Testing that are possible but if we consider the major variant of it then below mentioned are the two fundamental ones.

1) **Functional Testing**

This type deals with the functional requirements or specifications of an application. Here, different actions or functions of the system are being tested by providing the input and comparing the actual output with the expected output.

For Example, when we test a Dropdown list, we click on it and verify that it expands and all the expected values are showing in the list.

Few major types of Functional Testing are:

- Smoke Testing
- Sanity Testing
- Integration Testing
- System Testing
- Regression Testing
- User Acceptance Testing

2) Non-Functional Testing

Apart from the functionalities of the requirements, there are several non-functional aspects as well that are required to be tested to improve the quality and performance of the application.

Few major types of Non-Functional Testing include:

- Usability Testing
- Load Testing
- Performance Testing
- Compatibility Testing
- Stress Testing
- Scalability Testing

<u>Black Box Testing Tools</u>

Black Box Testing tools are mainly record and playback tools. These tools are used for Regression Testing to check whether new build has created any bug in previous working application functionality.

These record and playback tools record test cases in the form of some scripts like TSL, VB script, Javascript, Perl, etc.

Here is One Recommended BBT Tool

#1) Ranorex Studio

Ranorex Studio supports Black Box Testing for desktop, web, and mobile applications. Record user interactions, add validations, specify conditions for execution, and use data from an Excel file or SQL database, all without coding.
Selenium WebDriver is built-in for easy cross-browser testing. Ranorex Studio integrates with your CI/CD/DevOps tools including Jira, Bugzilla, Jenkins, Bamboo, and many more.

<u>Black Box Testing Techniques</u>

In order to systematically test a set of functions, it is necessary to design test cases. Testers can create test cases from the requirement specification document using the following Black Box Testing techniques.

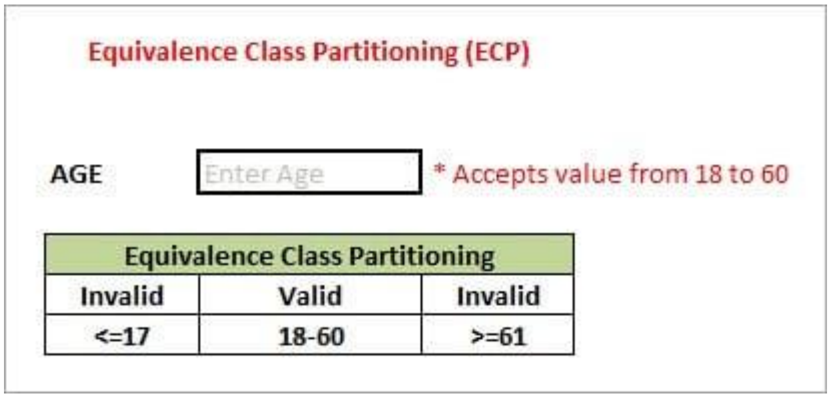
- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table Testing
- State Transition Testing
- Error Guessing
- Graph-Based Testing Methods
- Comparison Testing

❖ Equivalence Partitioning

This technique is also known as Equivalence Class Partitioning (ECP). In this technique, input values to the system or application are divided into different classes or groups based on its similarity in the outcome.

Hence, instead of using each and every input value we can now use any one value from the group/class to test the outcome. In this way, we can maintain the test coverage while we can reduce a lot of rework and most importantly the time spent.

For Example:


Equivalence Class Partitioning (ECP)
AGE
Enter Age
* Accepts value from 18 to 60

Equivalence Class Partitioning		
Invalid	Valid	Invalid
<=17	18-60	>=61

As present in the above image, an "AGE" text field accepts only the numbers from 18 to 60. There will be three sets of classes or groups.

Two invalid classes will be:

a) Less than or equal to 17.

b) Greater than or equal to 61.

One valid class will be anything between 18 to 60.

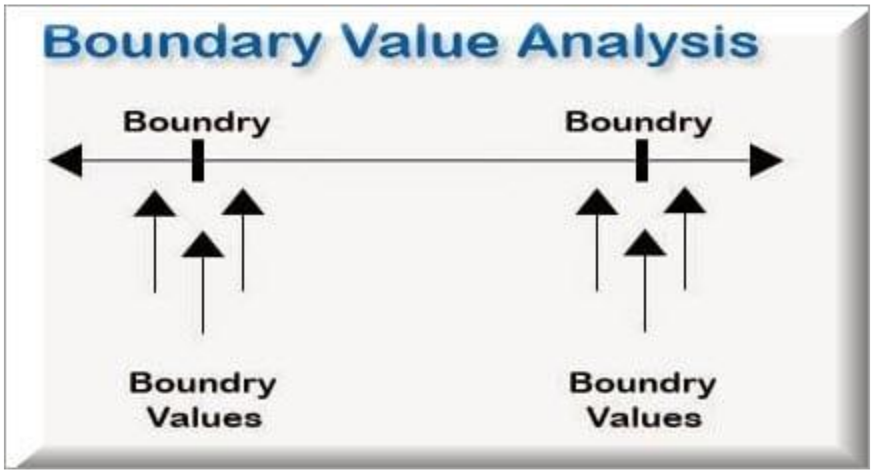
We have thus reduced the test cases to only 3 test cases based on the formed classes thereby covering all the possibilities. So, testing with anyone value from each set of the class is sufficient to test the above scenario.

❖ Boundary Value Analysis

From the name itself, we can understand that in this technique we focus on the values at boundaries as it is found that many applications have a high amount of issues on the boundaries.

Boundary means the values near the limit where the behavior of the system changes. In boundary value analysis both the valid inputs and invalid inputs are being tested to verify the issues.

For Example:



If we want to test a field where values from 1 to 100 should be accepted then we choose the boundary values: 1-1, 1, 1+1, 100-1, 100, and 100+1. Instead of using all the values from 1 to 100, we just use 0, 1, 2, 99, 100, and 101.

❖ Decision Table Testing

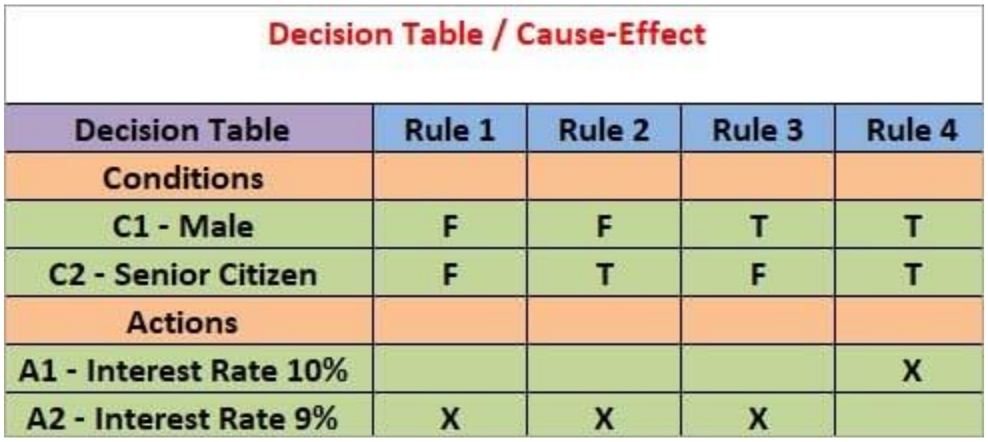
As the name itself suggests that, wherever there are logical relationships like:

If
{
(Condition = True)
then action1 ;
}
else action2; /(condition = False)*/*

Then a tester will identify two outputs (action1 and action2) for two conditions (True and False). So based on the probable scenarios a Decision table is carved to prepare a set of test cases.

For Example:

Take an example of XYZ bank that provides interest rate for the Male senior citizen as 10% and for the rest of the people 9%.



Decision Table / Cause-Effect

Decision Table	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
C1 - Male	F	F	T	T
C2 - Senior Citizen	F	T	F	T
Actions				
A1 - Interest Rate 10%				X
A2 - Interest Rate 9%	X	X	X	

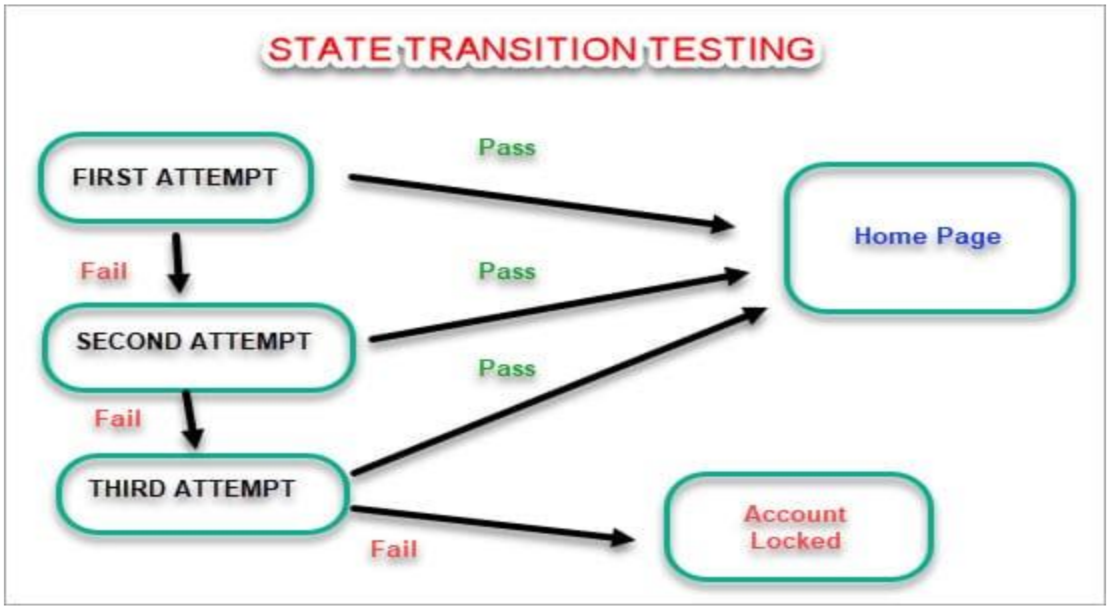
In this example condition, C1 has two values as true and false, condition C2 also has two values as true and false. The number of total possible combinations would then be four. This way we can derive test cases using a decision table.

❖ **State Transition Testing**

State Transition Testing is a technique that is used to test the different states of the system under test. The state of the system changes depending upon the conditions or events. The events trigger states which become scenarios and a tester needs to test them.

A systematic state transition diagram gives a clear view of the state changes but it is effective for simpler applications. More complex projects may lead to more complex transition diagrams thus making it less effective.

For Example:



❖ Error Guessing

This is a classic example of Experience-Based Testing.

In this technique, the tester can use his/her experience about the application behavior and functionalities to guess the error-prone areas. Many defects can be found using error guessing where most of the developers usually make mistakes.

Few common mistakes that developers usually forget to handle:

- Divide by zero.
- Handling null values in text fields.
- Accepting the Submit button without any value.
- File upload without attachment.
- File upload with less than or more than the limit size.

❖ Graph-Based Testing Methods

Each and every application is a build-up of some objects. All such objects are identified and the graph is prepared. From this object graph, each object relationship is identified and test cases are written accordingly to discover the errors.

❖ Comparison Testing

Different independent versions of the same software are used to compare to each other for testing in this method.

Advantages

- The tester need not have a technical background. It is important to test by being in the user's shoes and think from the user's point of view.
- Testing can be started once the development of the project/application is done. Both the testers and developers work independently without interfering in each other's space.
- It is more effective for large and complex applications.
- Defects and inconsistencies can be identified at the early stage of testing.

Disadvantages

- Without any technical or programming knowledge, there are chances of ignoring possible conditions of the scenario to be tested.

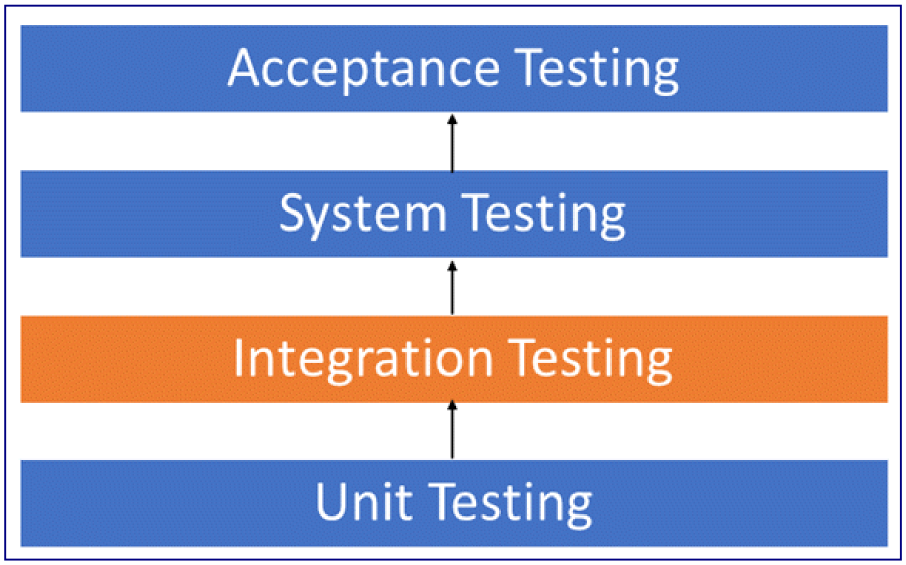
- In a stipulated time there are possibilities of testing less and skipping all possible inputs and their output testing.
- A Complete Test Coverage is not possible for large and complex projects.

Integration Testing

INTEGRATION TESTING is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated

Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as **'I & T'** (Integration and Testing), **'String Testing'** and sometimes **'Thread Testing'**.

Why do Integration Testing?



Although each software module is unit tested, defects still exist for various reasons like

A Module, in general, is designed by an individual software developer whose understanding and programming logic may differ from other programmers. Integration Testing becomes necessary **to verify the software modules work in unity**

At the time of module development, there are wide chances of change in requirements by the clients. These **new requirements may not be unit tested** and **hence system integration Testing becomes necessary**.

Interfaces of the software modules with the database could be erroneous

External Hardware interfaces, if any, could be erroneous

Inadequate exception handling could cause issues.

Example of Integration Test Case

Integration Test Case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.

Similarly Mail Box: Check its integration to the Delete Mails Module.

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Selected email should appear in the Deleted/Trash folder

Approaches, Strategies, Methodologies of Integration Testing

Software Engineering defines variety of strategies to execute Integration testing, viz.

Big Bang Approach :

Incremental Approach: which is further divided into the following

Top Down Approach

Bottom Up Approach

Sandwich Approach - Combination of Top Down and Bottom Up

Big Bang Approach:

Here all component are integrated together at **once** and then tested.

Advantages:

Convenient for small systems.

Disadvantages:

Fault Localization is difficult.

Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.

Since the Integration testing can commence only after "all" the modules are designed, the testing team will have less time for execution in the testing phase.

Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Incremental Approach

In this approach, testing is done by joining two or more modules that are *logically related*. Then the other related modules are added and tested for the proper functioning. The process continues until all of the modules are joined and tested successfully.

Incremental Approach, in turn, is carried out by two different Methods:

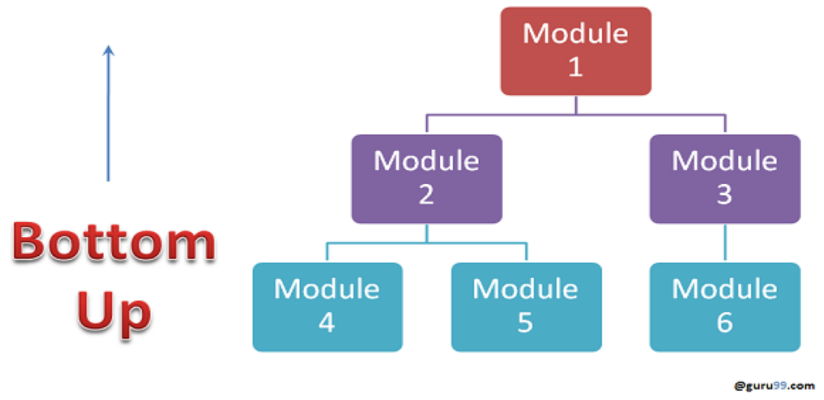
- Bottom Up

Top Down

Bottom-up Integration

In the bottom-up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing

Diagrammatic Representation:



Advantages:

Fault localization is easier.

No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

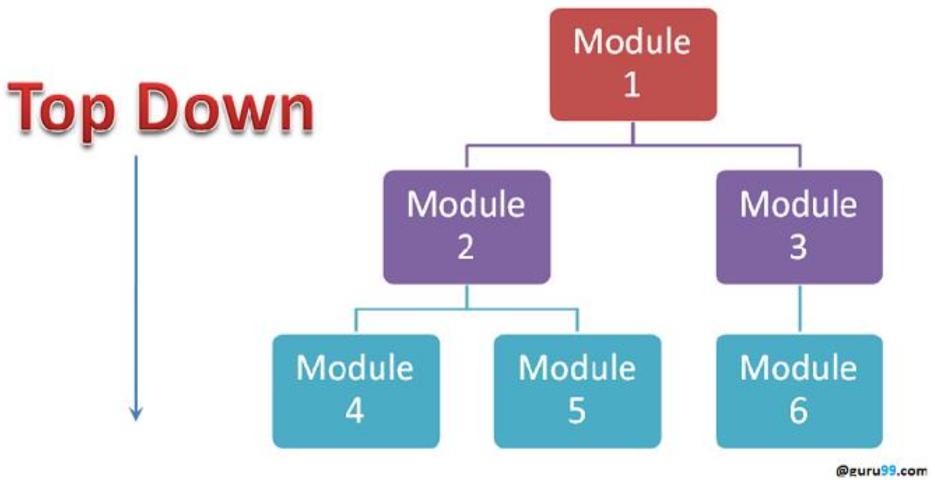
Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.

An early prototype is not possible

Top-down Integration:

In Top to down approach, testing takes place from top to down following the control flow of the software system.

Diagrammatic Representation:



Advantages:

Fault Localization is easier.

Possibility to obtain an early prototype.

Critical Modules are tested on priority; major design flaws could be found and fixed first.

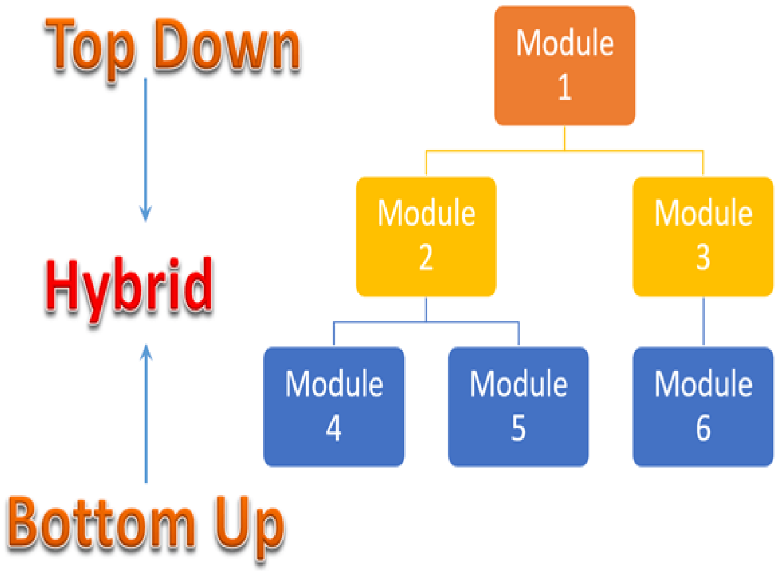
Disadvantages:

Needs many Stubs.

Modules at a lower level are tested inadequately.

Hybrid/ Sandwich Integration

In the sandwich/hybrid strategy is a combination of Top Down and Bottom up approaches. Here, top modules are tested with lower modules at the same time lower modules are integrated with top modules and tested. This strategy makes use of stubs as well as drivers.



How to do Integration Testing?

The Integration test procedure irrespective of the Software testing strategies (discussed above):

1. Prepare the Integration Tests Plan
2. Design the Test Scenarios, Cases, and Scripts.
3. Executing the test Cases followed by reporting the defects.
4. Tracking & re-testing the defects.
5. Steps 3 and 4 are repeated until the completion of Integration is successful.

Brief Description of Integration Test Plans:

It includes the following attributes:

Methods/Approaches to testing (as discussed above).

Scopes and Out of Scopes Items of Integration Testing.

Roles and Responsibilities.

Pre-requisites for Integration testing.

Testing environment.

Risk and Mitigation Plans.

Entry and Exit Criteria of Integration Testing

Entry and Exit Criteria to Integration testing phase in any software development model

Entry Criteria:

Unit Tested Components/Modules

All High prioritized bugs fixed and closed

All Modules to be code completed and in

tegrated successfully.

Integration tests Plan, test case, scenarios to be signed off and documented.

Required Test Environment to be set up for Integration testing

Exit Criteria:

Successful Testing of Integrated Application.

Executed Test Cases are documented

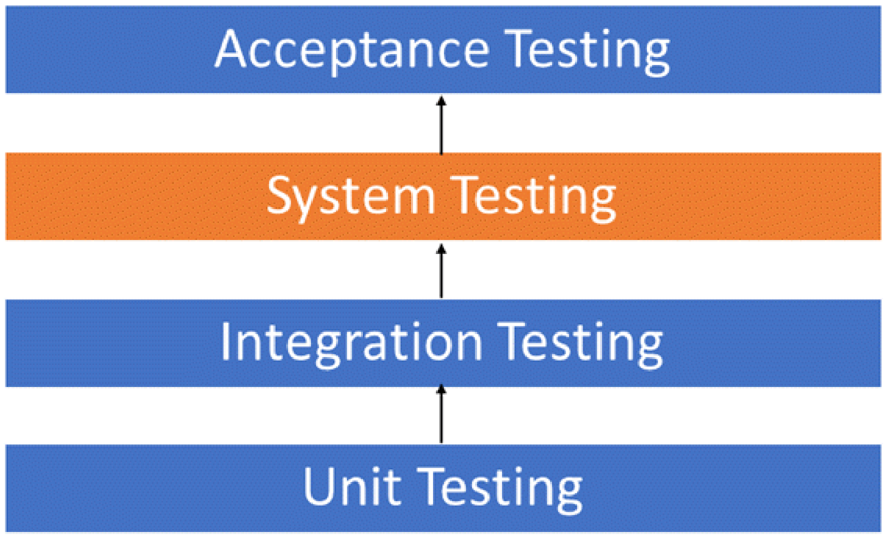
All High prioritized bugs fixed and closed

Technical documents to be submitted followed by release Notes.

System Testing

SYSTEM TESTING is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

Software Testing Hierarchy



As with almost any software engineering process, software testing has a prescribed order in which things should be done. The following is a list of software testing categories arranged in chronological order. These are the steps taken to fully test new software in preparation for marketing it:

- Unit testing performed on each module or block of code during development. Unit Testing is normally done by the programmer who writes the code.
- Integration testing done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.
- System testing done by a professional testing agent on the completed software product before it is introduced to the market.
- Acceptance testing - beta testing of the product done by the actual end users.

Different Types of System Testing

There are more than 50 types of System Testing. Below we have listed types of system testing a large software development company would typically use

1. **Usability Testing-** mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives
2. **Load Testing-** is necessary to know that a software solution will perform under real-life loads.
3. **Regression Testing-** involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.
4. **Recovery testing** - is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.
5. **Migration testing-** is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.
6. **Functional Testing** - Also known as functional completeness testing, Functional Testing involves trying to think of any possible missing functions. Testers might make a list of additional functionalities that a product could have to improve it during functional testing.
7. **Hardware/Software Testing** - IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

What Types of System Testing Should Testers Use?

There are over 50 different types of system testing. The specific types used by a tester depend on several variables. Those variables include:

- Who the tester works for - This is a major factor in determining the types of system testing a tester will use. Methods used by large companies are different than that used by medium and small companies.
- Time available for testing - Ultimately, all 50 testing types could be used. Time is often what limits us to using only the types that are most relevant for the software project.
- Resources available to the tester - Of course some testers will not have the necessary resources to conduct a testing type. For example, if you are a tester working for a large software development firm, you are likely to have expensive automated testing software not available to others.
- Software Tester's Education- There is a certain learning curve for each type of software testing available. To use some of the software involved, a tester has to learn how to use it.
- Testing Budget - Money becomes a factor not just for smaller companies and individual software developers but large companies as well.

QUESTION BANK

PROGRAM : BSc COMPUTER SCIENCE BATCH (2018-21)
REGULATION : 2017
SEM/YEAR : IV SEM / II YEAR
COURSE TITLE : SOFTWARE ENGINEERING

UNIT I

Part A (2 Marks)

1. What is the prime objective of software engineering?
2. Define software engineering paradigm.
3. What do you mean by spiral model?
4. Write a brief note on waterfall model.
5. Distinguish between process and methods.
6. Give the importance of software engineering
7. Define software process. State the important features of a process.
8. Write any two characteristics of software as a product.
9. Distinguish clearly between verification & validation.
10. State the System Engineering Hierarchy.
11. Define Computer Based System
12. Write a note on Software Process.
13. What do you mean by Optimality in Software Process?
14. Define Scalability.
15. List out Projects Outputs in Waterfall Model.

Part B (5 marks)

1. Explain Spiral model with a neat diagram.
2. Write in detail about System Engineering.
3. Write a short note on Software Prototype Model.
4. Draw a neat sketch of System Engineering Hierarchy.
5. Explain the Elements of Computer Based System.
6. What are the steps involved in Software Development Process.
7. List several software process paradigms.
8. State and explain the phases of Management Process.
9. Write a Short note on Verification.
10. Explain Validation Process.

Part C (10 Marks)

1. List out the phases involved in Software Development Lifecycle Process(SDLC).
2. Explain in detail about Waterfall Model with a neat diagram.
3. Write in detail about Spiral Model.
4. Discuss in detail about Evolutionary and Incremental Model
5. Explain system engineering process with Air Traffic Control Example.

UNIT II

Part A (2 marks)

1. Mention any two non-functional requirements on software to be developed
2. What is known as SRS review? How is it conducted?
3. Distinguish between expected requirements and excited requirements
4. What is meant by software prototyping?
5. What are the non-functional requirements of software?
6. What is data dictionary? How is it used in software engineering?
7. Write the distinct steps in requirements engineering process?
8. Compare evolutionary and throw away prototyping?
9. What is the role of data dictionary?
10. Define DFD.
11. What is meant by Data dictionary?
12. Define Process Specification.
13. What does data dictionary contains?
14. Write a note on Feasibility study.
15. What is meant by Throw away Prototyping?

Part B (5 Marks)

1. Compare functional models with behavioral models.
2. With a suitable diagram explain the elements of the analysis model
3. With an example explain about DFD
4. Discuss in detail the data modeling activity.
5. Explain in detail about Requirements Engineering Process.
6. Write Short note on View Points.
7. Briefly discuss about Validation Checks.
8. Write a note on User and System Requirements.
9. Explain Requirements Management.
10. Write a short note on Requirements Change Management.

Part C (10 marks)

1. Explain the ways and means for collecting the software requirements and how are they organized and represented?
2. Describe various prototyping techniques and discuss on analysis sand modeling
3. Narrate the importance of software specification of requirements. Explain a typical SRS structure and its parts.
4. Discuss in detail the FAST method of
   a. Requirement elicitation with an example.
   b. What is software specification?
5. Discuss in detail the basic structure of analysis model.

UNIT III

Part A (2 marks)

1. What do you mean by horizontal and vertical partitioning?
2. How do you evaluate user interface?
3. Why software architecture is important in a software process?
4. How reliability is related to quality assurance?
5. What is the software architecture?
6. Compare data flow oriented design and data structured oriented design
7. What is the role of verification during a software exercise?
8. Distinguish between hard and soft real time systems.
9. Distinguish between product and process metrics.
10. What is the work product of software design process and who does this?
11. Enumerate different data flow architectures
12. How do you describe software interface?
13. What is transaction mapping? How it is used in software design?
14. What are the various models produce by the software design process?
15. What is the quality parameters considered for effective modular design?

Part B (5 marks)

1. Write short notes on user interface design process?
2. Explain data architectural and procedural design for a software
3. Describe the design procedure for data acquisition system
4. discuss in detail about the design process in software development process
5. Justify “Design is not coding and coding is not design”.
6. Describe the golden rules for interface design.
7. What are the various model of abstraction? Discuss any two in detail?
8. What are the various model of abstraction?
9. Describe the concept of information hiding
10. What is data flow oriented design

Part C (10 Marks)

1. What is transform mapping? Explain the process with an illustration. What is its strength and weakness?
2. Explain the importance of user interface design in sale of software.
3. What are the characteristics of a good design? Describe different types of coupling and cohesion. How design evaluation is performed?
4. Explain the set of principles for software engineering design?
5. Draw the basic structure of analysis model and explain each entity in detail

Unit IV

Part A (2 marks)

1. List out the importance of cost estimation in software development.
2. Mention the advantages of CASE tools.
3. How do you estimate time required for a software development project?
4. Draw the structure of CASE REPOSITORY and explain.
5. What is meant by software change?
6. Write short notes on empirical estimation models.
7. Why the software needs maintenance?
8. Define software re-engineering.
9. List any 4 categories of CASE tools.
10. What is Delphi cost estimation technique?
11. Define Software Configuration Management.
12. What is CASE?
13. What are the standards used in SCM?
14. What are Change Control impacts?
15. Define Project Scheduling.

Part B (5 marks)

1. Briefly explain the concepts of SCM Process with its standards.
2. Explain in detail about the maintenance process.
3. Discuss in detail about software evolution.
4. Explain about function point metric in detail.
5. Write briefly on CASE.
6. Write briefly on Software complexity measure.
7. Discuss the concept of software maintenance process
8. Write short notes on COCOMO estimation criteria.
9. Discuss the concept of software maintenance process.
10. Write short notes on Task scheduling with an example.

Part C (10 marks)

1. Write in detail about Software Configuration Management Process and explain with its Standards.
2. Explain the need for software measures and describe various metrics.
3. Describe two metrics which are used to measure the software in detail. Discuss clearly the advantages and disadvantages of these metrics
4. Explain various cost estimation models and compare
5. Explain the maintenance activities and maintenance problems. How the cost of maintenance is estimated?

UNIT V

Part A (2 marks)

1. What is stress testing?
2. State the objectives and guidelines for debugging.
3. Distinguish between verification and validation
4. What are the roles of testing tools?
5. What do you mean by test case management?
6. Distinguish between alpha and beta testing?
7. What are the approaches of debugging?
8. What are the roles of cyclomatic complexity value in software resting?
9. Distinguish between black and white box testing
10. What is white box testing and what is the difficulty while exercising it?
11. Why testing is important with respect to software?
12. What is static and dynamic testing?
13. How regression and stress tests are performed?
14. Write short notes on equivalence partitioning?
15. Write the types of system tests?

Part B (5 marks)

1. Explain boundary value analysis.
2. Justify the importance of testing process
3. Discuss in detail about alpha and beta testing.
4. Write short notes on Data flow testing
5. Write short notes on Integration Testing.
6. What are the types of Testing? Explain in short note.
7. Explain about the Software testing strategies.
8. Differentiate between Unit testing and Integration testing.
9. Write a short note on Debugging and System testing
10. Discuss about the advantages of Software Testing.

Part C (10 Marks)

1. Discuss the differences between black box and white box testing models. Discuss how these testing models may be used together to test a program schedule
2. What do you mean by integration testing? Explain their outcomes
3. What is black box testing? Is it necessary to perform this? Explain various test activities
4. Explain black box testing methods and its advantages and disadvantages.
5. Write short notes on
   a) Data flow testing.
   b) Integration testing.