DIWAKAR EDUCATION HUB

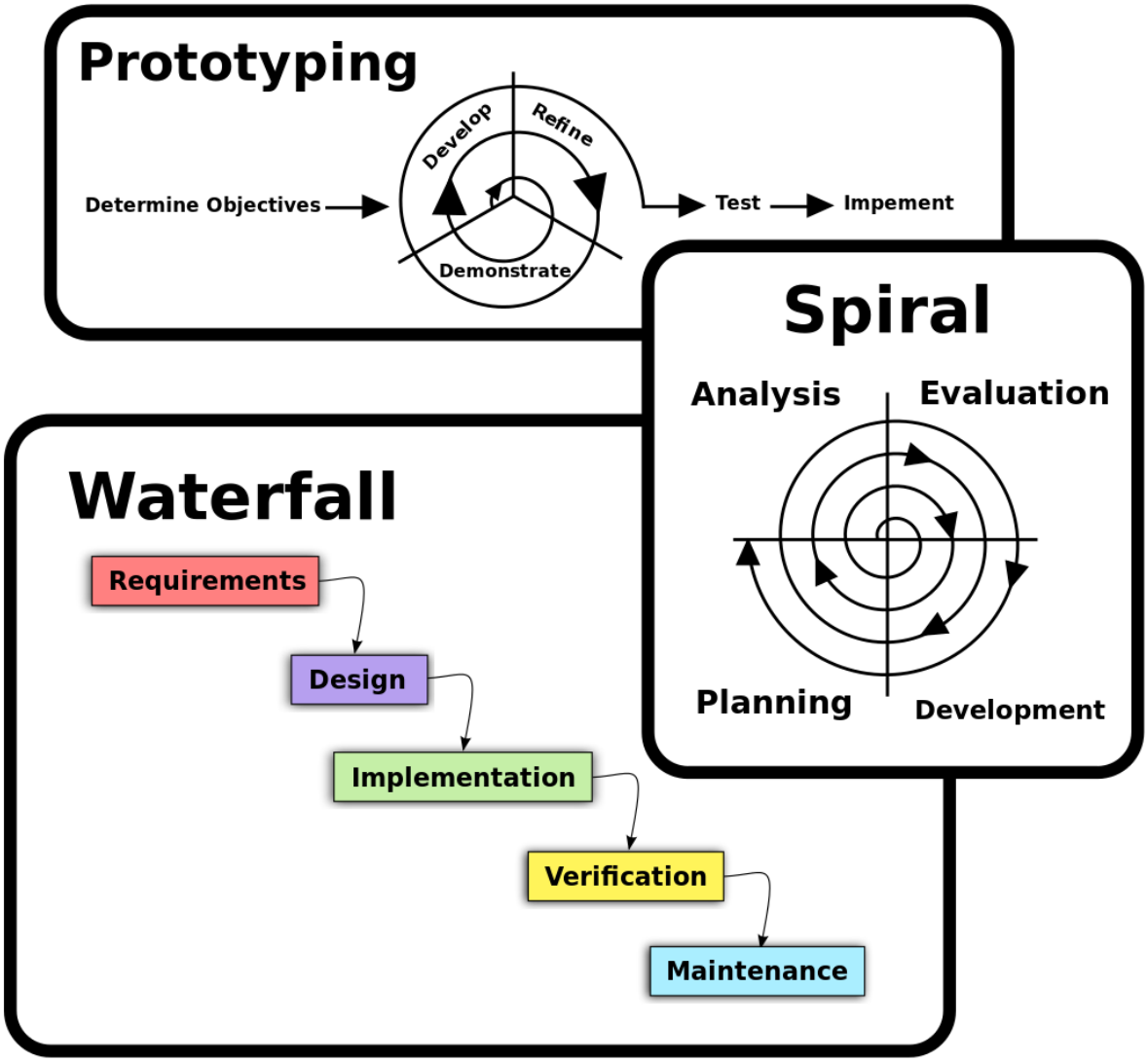
Software Engineering Unit – 6

As per updated syllabus

DIWAKAR EDUCATION HUB

2020

THE LEARN WITH EXPERTIES



Software Process

A software process (also knows as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

Any software process must include the following four activities:

1. **Software specification** (or requirements engineering): Define the main functionalities of the software and the constrains around them.

2. S**oftware design and implementation**: The software is to be designed and programmed.
3. **Software verification and validation**: The software must conforms to it's specification and meets the customer needs.
4. **Software evolution** (software maintenance): The software is being modified to meet customer and market requirements changes.

In practice, they include sub-activities such as requirements validation, architectural design, unit testing, …etc.

There are also **supporting activities** such as configuration and change management, quality assurance, project management, user experience.

Along with **other activities** aim to **improve** the above activities by introducing new techniques, tools, following the best practice, process standardization (so the diversity of software processes is reduced), etc.

When we talk about a process, we usually talk about the activities in it. However, a process also includes the process description, which includes:

1. **Products**: The outcomes of the an activity. For example, the outcome of architectural design maybe a model for the software architecture.
2. **Roles**: The responsibilities of the people involved in the process. For example, the project manager, programmer, etc.
3. **Pre and post conditions**: The conditions that must be true before and after an activity. For example, the pre condition of the architectural design is the requirements have been approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

Software process is complex, it relies on making decisions. There's no ideal process and most organizations have developed their own software process.

For example, an organization works on critical systems has a very structured process, while with business systems, with rapidly changing requirements, a less formal, flexible process is likely to be more effective.

Software Process Models

A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective.

A quick glance about very general process models.

These generic models are abstractions of the process that can be used to explain different approaches to the software development. They can be adapted and extended to create more specific processes.

Some methodologies are sometimes known as **software development life cycle** (SDLC) methodologies, though this term could also be used more generally to refer to any methodology.

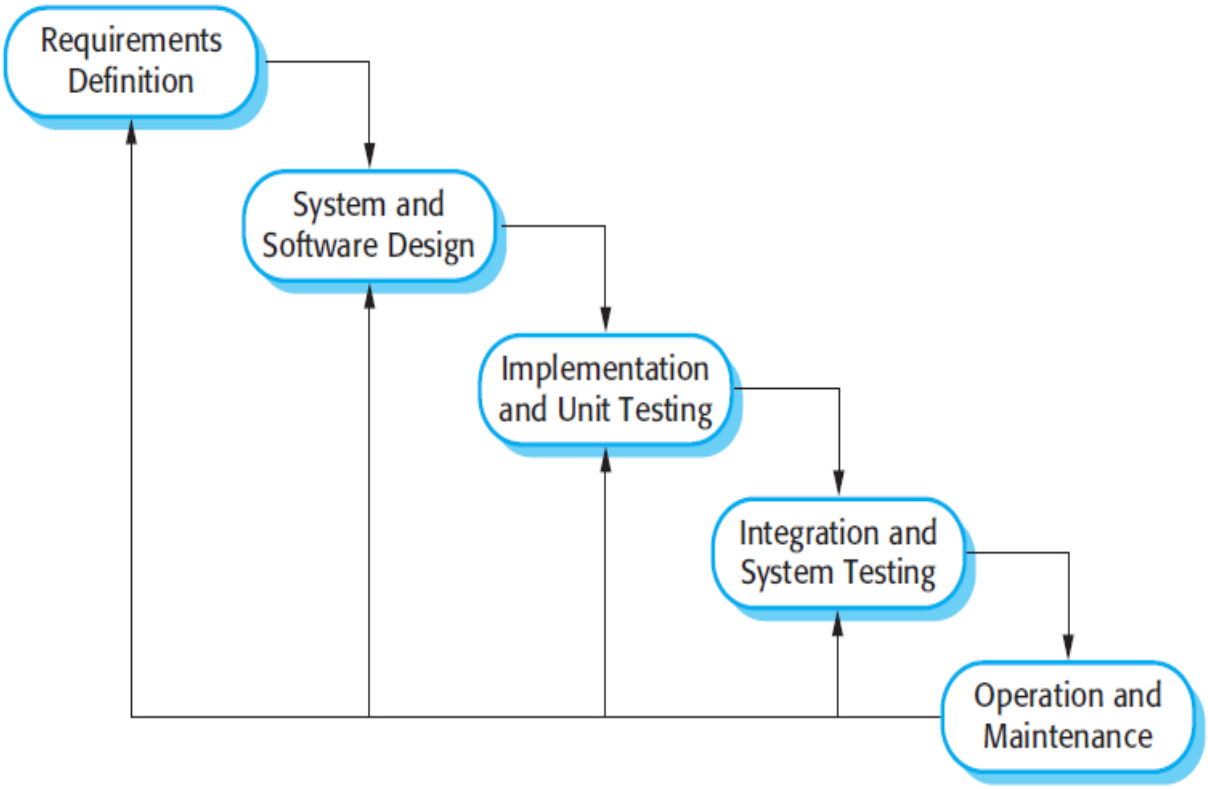
Waterfall Model

The waterfall model is a sequential approach, where each fundamental activity of a process represented as a separate phase, arranged in linear order.

In the waterfall model, you must plan and schedule all of the activities before starting working on them (plan-driven process).

Plan-driven process is a process where all the activities are planned first, and the progress is measured against the plan. While the agile process, planning is incremental and it's easier to change the process to reflect requirement changes.

The phases of the waterfall model are: **Requirements, Design, Implementation, Testing, and Maintenance.**



The Waterfall Model

The Nature of Waterfall Phases

In principle, the result of each phase is one or more documents that should be approved and the next phase shouldn't be started until the previous phase has completely been finished.

In practice, however, these phases overlap and feed information to each other. For example, during design, problems with requirements can be identified, and during coding, some of the design problems can be found, etc.

The software process therefore is not a simple linear but involves feedback from one phase to another. So, documents produced in each phase may then have to be modified to reflect the changes made.

When To Use?

In principle, the waterfall model should only be applied when requirements are well understood and unlikely to change radically during development as this model has a relatively rigid structure which makes it relatively hard to accommodate change when the process in underway.

Prototyping

A prototype is a version of a system or part of the system that's developed quickly to check the customer's requirements or feasibility of some design decisions.

So, a prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc.

In prototyping, the client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation.

While some prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.

A software prototype can be used:

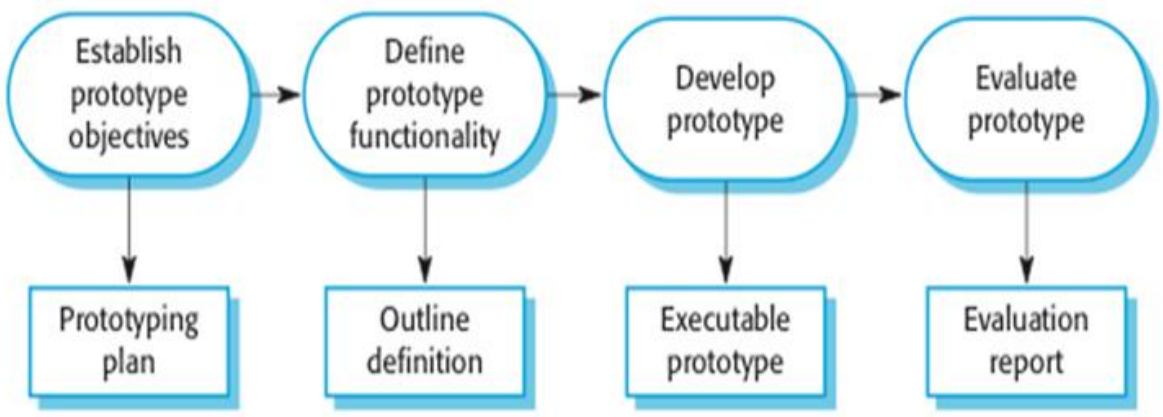
[1] In the **requirements engineering**, a prototype can help with the elicitation and validation of system requirements.

It allows the users to experiment with the system, and so, refine the requirements. They may get new ideas for requirements, and find areas of strength and weakness in the software.

Furthermore, as the prototype is developed, it may reveal errors and in the requirements. The specification maybe then modified to reflect the changes.

[2] In the **system design**, a prototype can help to carry out deign experiments to check the feasibility of a proposed design.

For example, a database design may be prototype-d and tested to check it supports efficient data access for the most common user queries.



The process of prototype development

The phases of a prototype are:

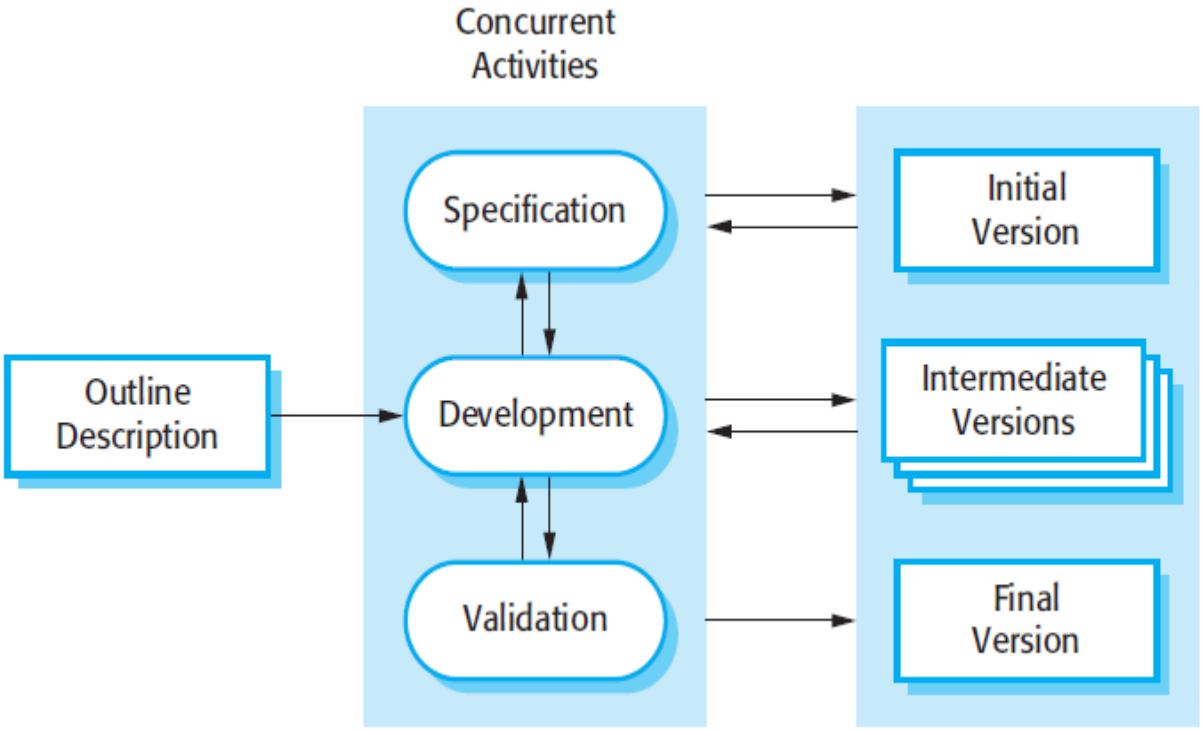
1. **Establish objectives**: The objectives of the prototype should be made explicit from the start of the process. Is it to validate system requirements, or demonstrate feasibility, etc.
2. **Define prototype functionality**: Decide what are the inputs and the expected output from a prototype. To reduce the prototyping costs and accelerate the delivery schedule, you may ignore some functionality, such as response time and memory utilization unless they are relevant to the objective of the prototype.
3. **Develop the prototype**: The initial prototype is developed that includes only user interfaces.
4. **Evaluate the prototype**: Once the users are trained to use the prototype, they then discover requirements errors. Using the feedback both the specifications and the prototype can be improved. If changes are introduced, then a repeat of steps 3 and 4 may be needed.

Prototyping is not a standalone, complete development methodology, but rather an approach to be used in the context of a full methodology (such as incremental, spiral, etc).

Incremental Development

Incremental development is based on the idea of developing an initial implementation, exposing this to user feedback, and evolving it through several versions until an acceptable system has been developed.

The activities of a process are not separated but interleaved with feedback involved across those activities.



The Incremental Development Model

Each system increment reflects a piece of the functionality that is needed by the customer. Generally, the early increments of the system should include the most important or most urgently required functionality.

This means that the customer can evaluate the system at early stage in the development to see if it delivers what's required. If not, then only the current

increment has to be changed and, possibly, new functionality defined for later increments.

Incremental Vs Waterfall Model

Incremental software development is better than a waterfall approach for most business, e-commerce, and personal systems.

By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.

Compared to the waterfall model, incremental development has three important benefits:

1. The **cost of accommodating changing** customer requirements is reduced. The amount of analysis and documentation that has to be redone is much less than that's required with waterfall model.
2. It's easier to get **customer feedback** on the work done during development than when the system is fully developed, tested, and delivered.
3. More **rapid delivery** of useful software is possible even if all the functionality hasn't been included. Customers are able to use and gain value from the software earlier than it's possible with the waterfall model.

It can be a plan-driven or agile, or both

Incremental development is one of the most common approaches. This approach can be either a plan-driven or agile, or both.

In a plan-driven approach, the system increments are identified in advance, but, in the agile approach, only the early increments are identified and the development of later increments depends on the progress and customer priorities.

It's not a problem-free

But, it's not a problem-free …

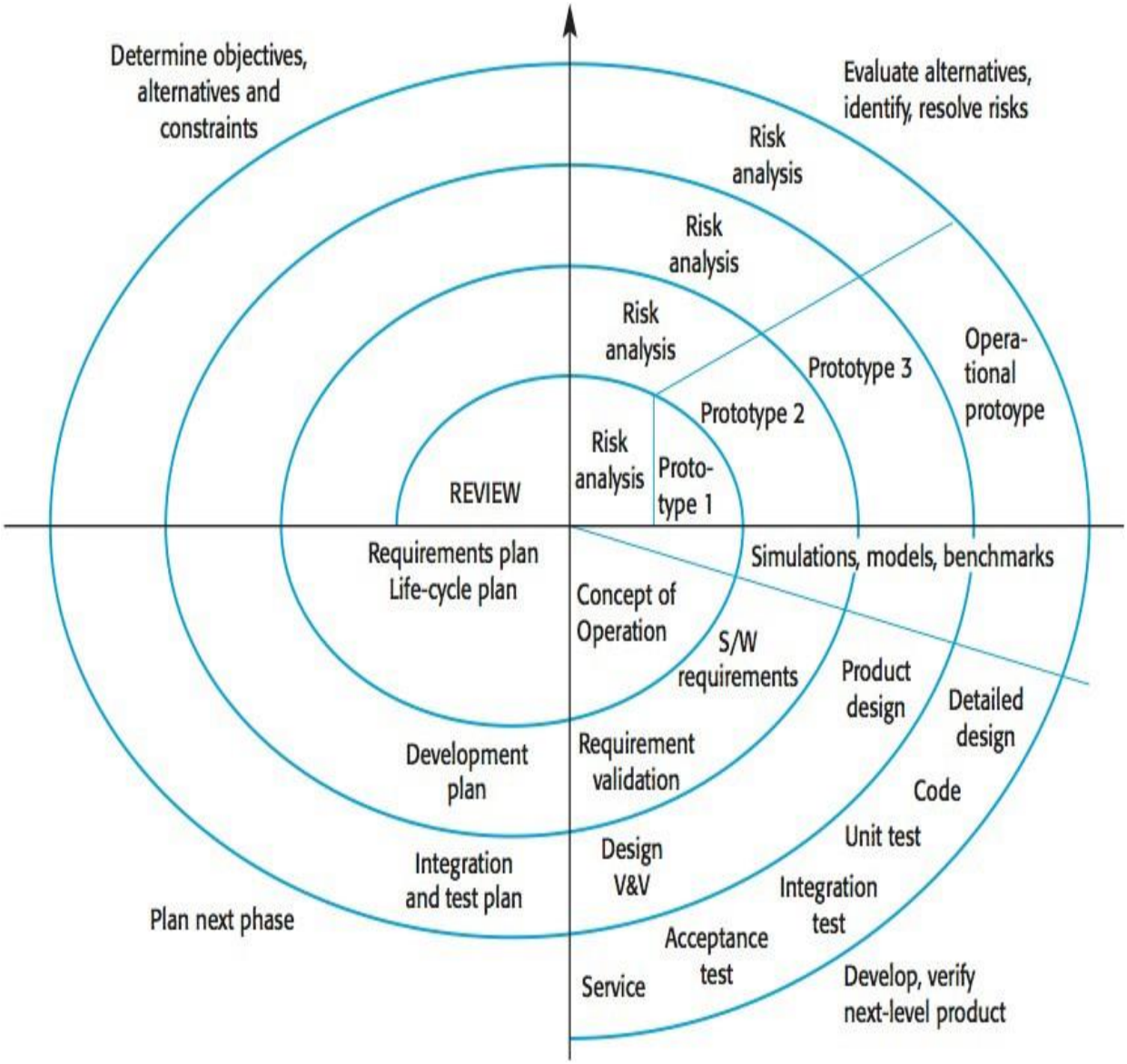
- Some organizations have procedures that have evolved over the time, and can't follow informal iterative or agile process. For example, procedures to ensure that the software properly implements external regulations.
- System structure tends to degrades as new increments are added and get corrupted as regular changes are incorporated. Even if time and money spent on refactoring to improve the software, further changes becomes more difficult and costly.

Spiral Model

The spiral model is a risk-driven where the process is represented as spiral rather than a sequence of activities.

It was designed to include the best features from the waterfall and prototyping models, and introduces a new component; risk-assessment.

Each loop (from review till service — see figure below) in the spiral represents a phase. Thus the first loop might be concerned with system feasibility, the next loop might be concerned with the requirements definition, the next loop with system design, and so on.



The spiral model

Each loop in the spiral is split into four sectors:

1. **Objective setting**: The objectives and risks for that phase of the project are defined.

2. **Risk assessment and reduction:** For each of the identified project risks, a detailed analysis is conducted, and steps are taken to reduce the risk. For example, if there's a risk that the requirements are inappropriate, a prototype may be developed.

3. **Development and validation:** After risk evaluation, a process model for the system is chosen. So if the risk is expected in the user interface then we must prototype the user interface. If the risk is in the development process itself then use the waterfall model.

4. **Planning:** The project is reviewed and a decision is made whether to continue with a further loop or not.

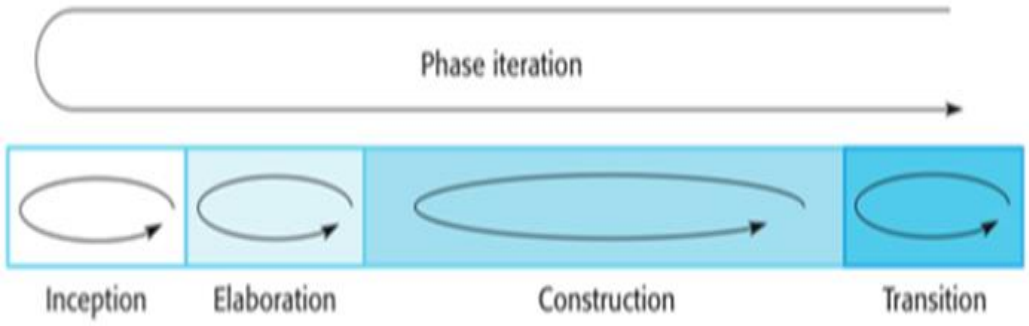
Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development. In practice, however, the model is rarely used.

Iterative Development

Iterative development model aims to develop a system through building small portions of all the features, across all components.

We build a product which meets the initial scope and release it quickly for customer feedback. An early version with limited features important to establish market and get customer feedback.

In each increment, a slice of system features is delivered, passing through the requirements till the deployment.



The phases of iterative development

The phases of iterative development are:

1. **Inception:** The goal is to establish a business case for the system. We should identify all the external entities that will interact with the system,

and define these interactions. Then, uses this information to assess the contribution that the system makes to the business. If the contribution is minor, then the project may be cancelled.

2. **Elaboration:** We develop an understanding of the problem domain and architecture framework, develop the project plan, and identify risks.
3. **Construction:** Incrementally fills-in the architecture with production-ready code produced from analysis, design, implementation, and testing of the requirements. The components of the system are dependent on each other and they're developed in parallel and integrated during this phase. On the completion of this phase, you should have a complete working software.
4. **Transition:** We deliver the system into the production operating environment.

All the phases will be done once, while the construction phase will be incrementally visited for each increment; for each slice of system features.

Agile

Agility is flexibility, it is a state of dynamic, adapted to the specific circumstances.

The agile methods refers to a group of software development models based on the incremental and iterative approach, in which the increments are small and typically, new releases of the system are created and made available to customers every few weeks.

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

The principles of agile methods

They involve customers in the development process to propose requirements changes. They minimize documentation by using informal communications rather than formal meetings with written documents.

They are best suited for application where the requirements change rapidly during the development process.

There are a number of different agile methods available such as: Scrum, Crystal, Agile Modeling (AM), Extreme Programming (XP), etc.

Increment Vs Iterative Vs Agile

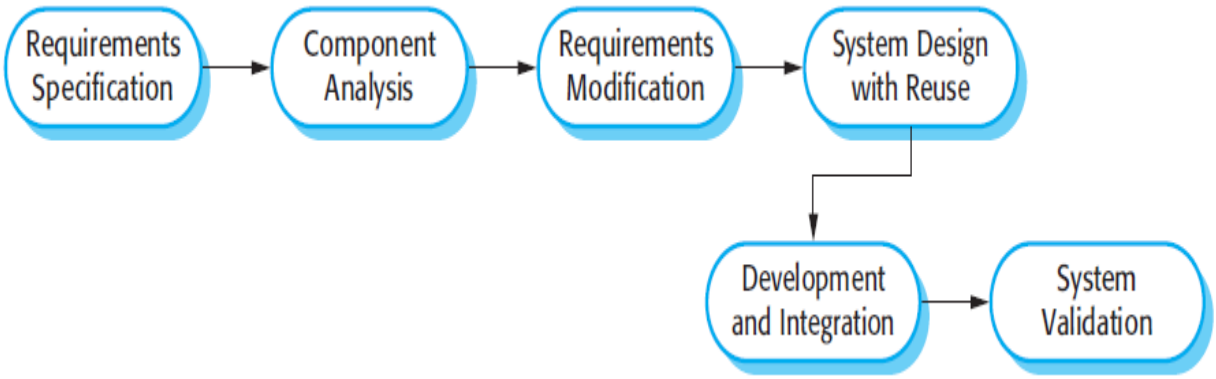
You might be asking about the difference between incremental, iterative and agile models.

Each increment in the incremental approach builds a complete feature of the software, while in iterative, it builds small portions of all the features.

An agile approach combines the incremental and iterative approach by building a small portion of each feature, one by one, and then both gradually adding features and increasing their completeness.

Reuse-oriented Software Engineering

It's attempting to reuse an existing design or code (probably also tested) that's similar to what's required. It's then modified, and incorporated to the new system.



The Reuse-oriented software engineering model

Although the initial "requirements specification" phase and the "validation " phase are comparable with other software processes, the intermediate phases in a reuse-oriented process are different. These phases are:

1. **Component analysis:** A search is made for the components to implement the given requirements specification. Usually, there's no exact match, and components may be only provide some of the functionality required.
2. **Requirements modification**: During this phase, the requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components. If the modifications are impossible, the component analysis activity may be re-entered to search for alternative solutions.

3. **System design with reuse:** During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused and they will organize the framework accordingly. Some new software has to be designed if some reusable components are not available.

4. **Development and integration:** The components are integrated to create the new system. System integration, in this model, may be part of the development process rather than a separate activity.

There are basically three types of software components that can be used in a reuse-oriented process:

1. **Web services** that are developed according to well-known service standards and which will become available for remote invocation.

2. **Collections of objects** that are developed as a **package** to be integrated with a component framework such as .NET or Java EE.

3. **Standalone software systems** that are configured for use in a particular environment.

It's has an obvious advantage, But!

Reuse-oriented software engineering has an obvious advantage of reducing the amount of software to be developed and therefore reduced cost and risks, and usually leads to faster delivery.

However, requirements compromises can't be avoided, which may lead to a system that does not meet the real needs of users.

Furthermore, some control over the system evolution might also be lost as new versions of the reusable components are not under the control of the organization using them.

Summary

Waterfall

It's useful when the requirements are clear, or following a very structured process as in critical systems which needs a detailed, precise, and accurate documents describes the system to be produced.

Not good when requirements are ambiguous, and doesn't support frequent interaction with the customers for feedback and proposing changes. It's not suitable for large projects that might take long time to be developed and delivered.

Prototype

Again, it's an early sample, or release of a product built to test a concept or to act as a thing to be replicated or learned from.

This is very useful when requirements aren't clear, and the interactions with the customer and experimenting an initial version of the software results in high satisfaction and a clearance of what to be implemented.

It's downsides are, good tools need to be acquired for quick development (like coding) in order to complete a prototype. In addition, the costs for for training the development team on prototyping may be high.

Incremental & Iterative

They're suited for large projects, less expensive to the change of requirements as they support customer interactions with each increment.

Initial versions of the software are produced early, which facilitates customer evaluation and feedback.

They don't fit into small projects, or projects that waterfall are best suited for; A structured process with a detailed, and accurate description of the system.

Spiral

It's good for high risky or large projects where the requirements are ambiguous. The risks might be due to cost, schedule, performance, user interfaces, etc.

Risk analysis requires highly specific expertise, and project's success is highly dependent on the risk analysis phase. It doesn't work well for smaller projects.

Agile

It suits small-medium size project, with rapidly changes in the requirements as customer is involved during each phase.

Very limited planning is required to get started with the project. It helps the company in saving time and money (as result of customer physical interaction in each phase). The daily meetings make it possible to measure productivity.

Difficult to scale up to large projects where documentation is essential. A highly skilled team is also needed.

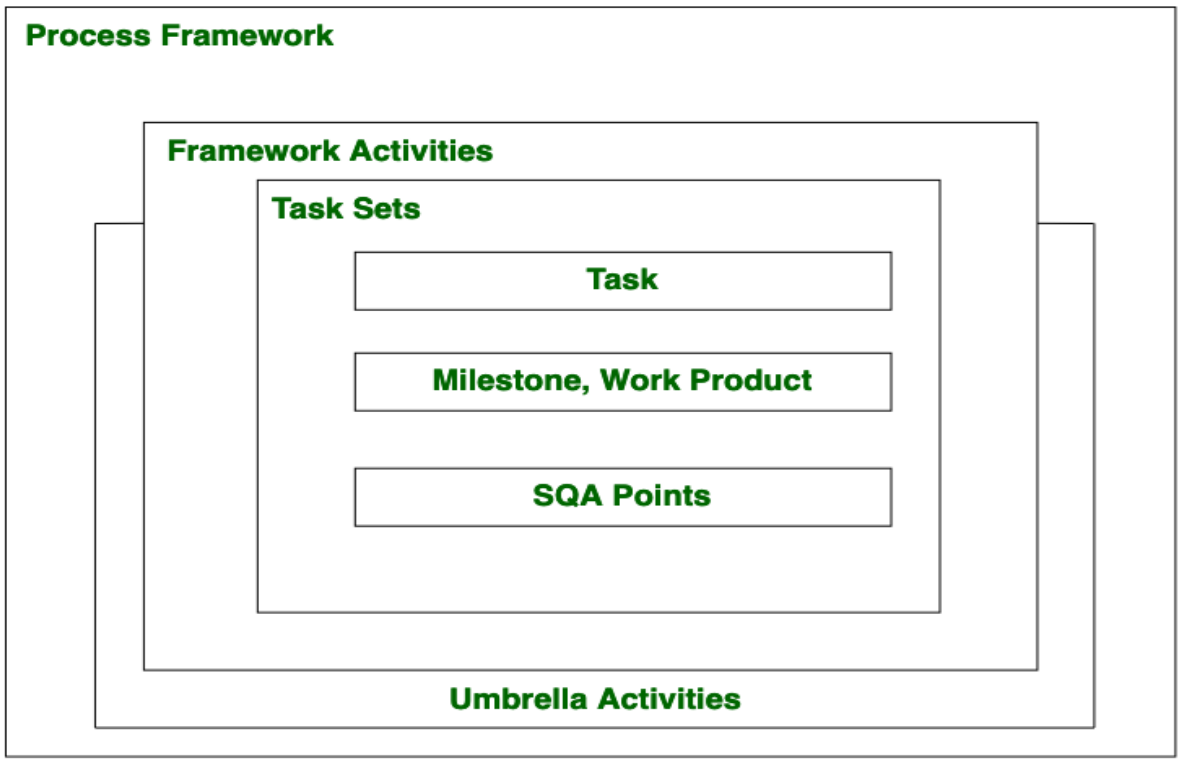
If team members aren't committed, the project will either never complete or fail. And there's always a limitation in time, like in increments, meetings, etc.

Process Activities

The four basic process activities of specification, development, validation, and evolution are organized differently in different development processes.

In the waterfall model, they are organized in sequence, while in incremental development they are interleaved. How these activities are performed might depend on the type of software, people involved in development, etc.

Framework is a Standard way to build and deploy applications. **Software Process Framework** is a foundation of complete software engineering process. Software process framework includes all set of umbrella activities. It also includes number of framework activities that are applicable to all software projects.



Software Process Framework

A generic process framework encompasses five activities which are given below one by one:

1. **Communication:**
   In this activity, heavy communication with customers and other stakeholders, requirement gathering is done.

2. **Planning:**
   In this activity, we discuss the technical related tasks, work schedule, risks, required resources etc.

3. **Modeling:**
   Modelling is about building representations of things in the 'real world'.In modelling activity, a product's model is created in order to better understanding and requirements.

4. **Construction:**
   In software engineering, construction is the application of set of procedures

that are needed to assemble the product. In this activity, we generate the code and test the product in order to make better product.

5. **Deployment:**
   In this activity, a complete or non-complete products or software are represented to the customers to evaluate and give feedback. on the basis of their feedback we modify the products for supply better product.

Umbrella activities include:

- Risk management
- Software quality assurance(SQA)
- Software configuration management(SCM)
- Measurement
- Formal technical reviws(FTR)

Task Set

A number of different process models were already discussed. These models offer different paradigms for software development. Regardless of whether a software team chooses a linear sequential paradigm, an iterative paradigm, an evolutionary paradigm, a concurrent paradigm or some permutation, the process model is populated by a set of tasks that enable a software team to define, develop, and ultimately support computer software.

No single set of tasks is appropriate for all projects. The set of tasks that would be appropriate for a large, complex system would likely be perceived as overkill for a small, relatively simple software product. Therefore, an effective software process should define a collection of task sets, each designed to meet the needs of different types of projects.

A task set is a collection of software engineering work tasks, milestones, and

deliverables that must be accomplished to complete a particular project. The task set to be chosen must provide enough discipline to achieve high software quality. But, at the same time, it must not burden the project team with unnecessary work.

Task sets are designed to accommodate different types of projects and different degrees of rigor. Although it is difficult to develop a comprehensive taxonomy of software project types, most software organizations encounter the following projects:

1. Concept development projects that are initiated to explore some new business concept or application of some new technology.
2. New application development projects that are undertaken as a consequence of a specific customer request.
3. Application enhancement projects that occur when existing software undergoes major modifications to function, performance, or interfaces that are observable by the end-user.
4. Application maintenance projects that correct, adapt, or extend existing software in ways that may not be immediately obvious to the end-user.
5. Reengineering projects that are undertaken with the intent of rebuilding an existing (legacy) system in whole or in part.

Even within a single project type, many factors influence the task set to be chosen. When taken in combination, these factors provide an indication of the degree of rigor with which the software process should be applied.

Degree of Rigor

Even for a project of a particular type, the degree of rigor with which the software process is applied may vary significantly. The degree of rigor is a function of many project characteristics. As an example, small, non-business-critical projects can generally be addressed with somewhat less rigor than large, complex business-

critical applications. It should be noted, however, that all projects must be conducted in a manner that results in timely, high-quality deliverables. Four different degrees of rigor can be defined:

Casual. All process framework activities (Chapter 2) are applied, but only a minimum task set is required. In general, umbrella tasks will be minimized and documentation requirements will be reduced. All basic principles of software engineering are still applicable.

Structured. The process framework will be applied for this project. Framework activities and related tasks appropriate to the project type will be applied and umbrella activities necessary to ensure high quality will be applied. SQA, SCM, documentation, and measurement tasks will be conducted in a streamlined manner.

Strict. The full process will be applied for this project with a degree of discipline that will ensure high quality. All umbrella activities will be applied and robust work products will be produced.

Quick reaction. The process framework will be applied for this project, but because of an emergency situation only those tasks essential to maintaining good quality will be applied. “Back-filling” (i.e., developing a complete set of documentation, conducting additional reviews) will be accomplished after the application/product is delivered to the customer.

The project manager must develop a systematic approach for selecting the degree of rigor that is appropriate for a particular project. To accomplish this, project adaptation criteria are defined and a task set selector value is computed.

Defining Adaptation Criteria

Adaptation criteria are used to determine the recommended degree of rigor with which the software process should be applied on a project. Eleven adaptation criteria are defined for software projects:

- Size of the project
- Number of potential users
- Mission criticality
- Application longevity
- Stability of requirements
- Ease of customer/developer communication
- Maturity of applicable technology
- Performance constraints
- Embedded and nonembedded characteristics
- Project staff
- Reengineering factors

Each of the adaptation criteria is assigned a grade that ranges between 1 and 5, where 1 represents a project in which a small subset of process tasks are required and overall methodological and documentation requirements are minimal, and 5 represents a project in which a complete set of process tasks should be applied and overall methodological and documentation requirements are substantial.

Computing a Task Set Selector Value

Adaptation Criteria	Grade	Weight	Entry Point Multiplier					Product
			Conc.	NDev.	Enhan.	Maint.	Reeng.	
Size of project	____	1.20	0	1	1	1	1	____
Number of users	____	1.10	0	1	1	1	1	____
Business criticality	____	1.10	0	1	1	1	1	____
Longevity	____	0.90	0	1	1	0	0	____
Stability of requirements	____	1.20	0	1	1	1	1	____
Ease of communication	____	0.90	1	1	1	1	1	____
Maturity of technology	____	0.90	1	1	0	0	1	____
Performance constraints	____	0.80	0	1	1	0	1	____
Embedded/nonembedded	____	1.20	1	1	1	0	1	____
Project staffing	____	1.00	1	1	1	1	1	____
Interoperability	____	1.10	0	1	1	1	1	____
Reengineering factors	____	1.20	0	0	0	0	1	____
Task set selector (TSS)								____

To select the appropriate task set for a project, the following steps should be conducted:

1. Review each of the adaptation criteria in Section 7.3.2 and assign the appropriate grades (1 to 5) based on the characteristics of the project. These grades should be entered into Table

2. Review the weighting factors assigned to each of the criteria. The value of a weighting factor ranges from 0.8 to 1.2 and provides an indication of the relative importance of a particular adaptation criterion to the types of software developed within the local environment. If modifications are required to better reflect local circumstances, they should be made.

3. Multiply the grade entered in Table 7.1 by the weighting factor and by the entry point multiplier for the type of project to be undertaken. The entry point multiplier takes on a value of 0 or 1 and indicates the relevance of the adaptation criterion to the project type. The result of the product grade x weighting factor x entry point multiplier is placed in the Product column of Table 7.1 for each adaptation criteria individually.

4. Compute the average of all entries in the Product column and place the result in the space marked task set selector (TSS). This value will be used to help select the task set that is most appropriate for the project.

Interpreting the TSS Value and Selecting the Task Set

Once the task set selector is computed, the following guidelines can be used to select the appropriate task set for a project:

Task set selector value	**Degree of rigor**
TSS < 1.2	casual
1.0 < TSS < 3.0	structured
TSS > 2.4	strict

The overlap in TSS values from one recommended task set to another is purposeful and is intended to illustrate that sharp boundaries are impossible to define when making task set selections. In the final analysis, the task set selector value, past experience, and common sense must all be factored into the choice of the task set for a project.

Table illustrates how TSS might be computed for a hypothetical project. The project manager selects the grades shown in the Grade column. The project type is new application development. Therefore, entry point multipliers are selected from the NDev column. The entry in the Product column is computed using Grade x Weight x NewDev entry point multiplier.

Process Pattern

The software process can be defined as a collection of patterns that define a set of activities, actions, work tasks, work products and/or related behaviors required to develop computer software.

Patterns can be defined at any level of abstraction. In some cases, a pattern might be used to describe a complete process (e.g., prototyping). In other situations, patterns can be used to describe an important framework activity (e.g., planning) or a task within framework activity (e.g., project- estimating). Ambler has proposed the following template for describing a process pattern:

Pattern Name. The pattern is given a meaningful name that describes its functions within the software process (e.g., customer-communication).

Intent. The objective of the pattern is described briefly. For example, the intent of customer-communication is "to establish a collaborative relationship with the customer in an effort to define project scope, business requirements, and other projects constraints".

Type. The pattern type is specified. Ambler suggests three types:

• Task pattern define a software engineering action or work task that is part of the process and relevant to successful software engineering practice (e.g., requirements gathering is a task pattern).

• Stage patterns represent a framework activity for the process. Since a framework activity encompasses multiple work tasks, a stage pattern incorporates multiple task patterns that are relevant to the stage. An example of a stage pattern might be communication. This pattern would incorporate the task pattern requirements gathering and others.

• Phase patterns define the sequence of framework activities is iterative in nature. An example of a phase pattern might be a spiral model or prototyping.

Initial context. The conditions under which the pattern applies are described. Prior to the initiation of the pattern, we ask (1) what organizational team created activities have already occurred (2) what is the entry state for the process? And (3) what software engineering information or project information already exists?

For example, the planning pattern (a stage pattern) requires that (1) customers and software engineers have established a collaborative communication: (2) successful completion of a number of task patterns (specified) for the customer-communication pattern has occurred: and (3) project scope, basic business requirements, and project constraints are known.

An Example Process Pattern:

The following abbreviated process pattern describes an approach that may be applicable when stakeholders have a general idea of what must be done, but are unsure of specific software requirements.

Pattern name. Prototyping.

Intent: The objective of the pattern is to build a model (a prototype) that can be assessed iteratively by stakeholders in an effort to identify or solidity software requirements.

Type: Phase pattern.

Initial context: The following conditions must be met prior to the initiation of this pattern:(1) stakeholders have been identified; (2) a mode of communication between stakeholders and the software team has been established; (3) the overriding problem to be solved has been identified by stakeholders: (4) an initial understanding of project scope, basic business requirements, and project constraints has been developed.

Problem: Requirements are hazy or nonexistent, yet there is clear recognition that there is a problem, and the problem must be addressed with a software solution. Stakeholders are unsure of what they want; that is, they cannot describe software requirements in any detail.

Solutions: A description of the prototype that identifies basic requirements (e.g., modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, (1) the prototype may evolve through a series of increments to become the production software or (2) the prototype may be discarded and the production software built using some other process pattern.

Related Patterns: The following patterns are related to this pattern: **customer-communication; iterative design; iterative development, customer assessment; requirement extraction**.

Known uses/examples: Prototyping is recommended when requirements are uncertain.

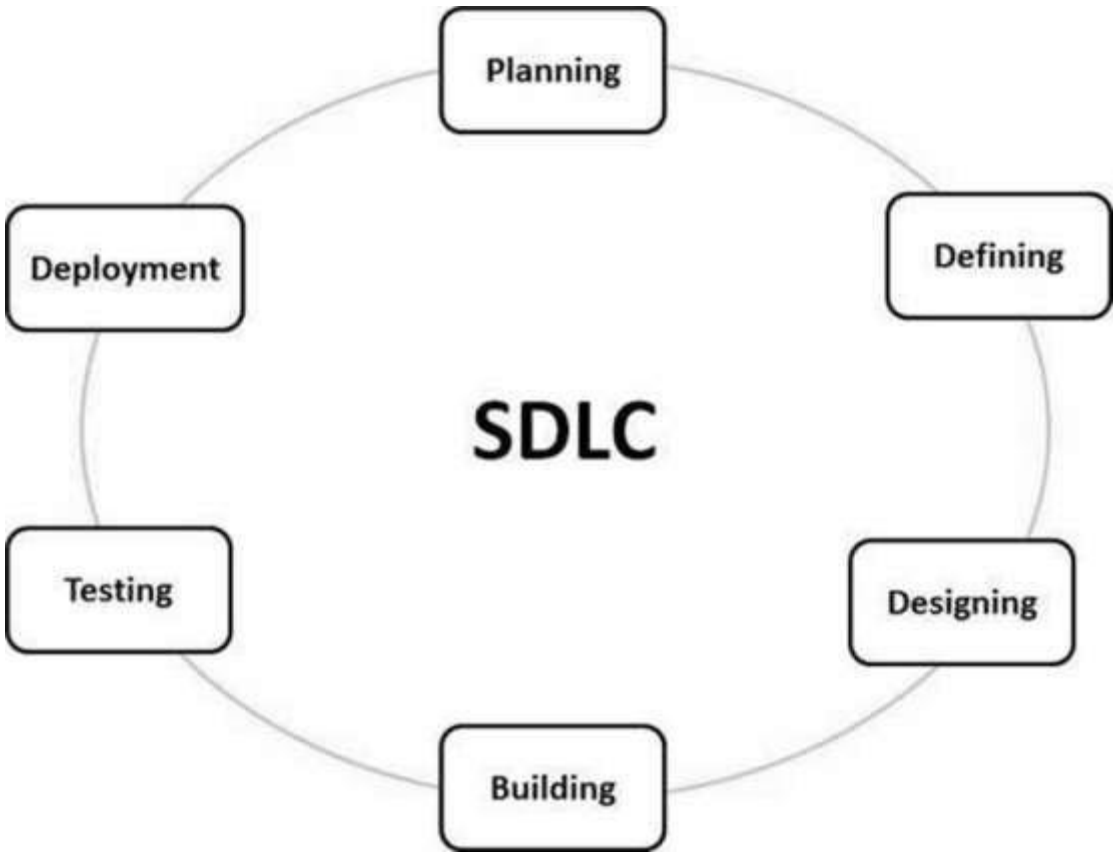
Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development Life Cycle consists of the following stages –

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model

- Big Bang Model

Other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.

The following framework activities are carried out irrespective of the process model chosen by the organization.

1. Communication
2. Planning
3. Modeling
4. Construction
5. Deployment

The name 'prescriptive' is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

There are three types of prescriptive process models. They are:

1. The Waterfall Model
2. Incremental Process model
3. RAD model

1. The Waterfall Model

- The waterfall model is also called as **'Linear sequential model'** or **'Classic life cycle model'.**
- In this model, each phase is fully completed before the beginning of the next phase.
- This model is used for the small projects.
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is complete.

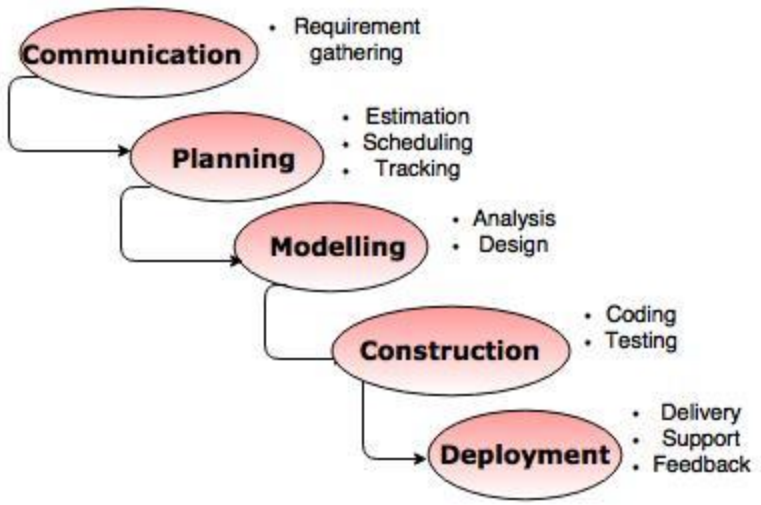


Fig. - The Waterfall model

NOTE: The description of the phases of the waterfall model is same as that of the process model.

An alternative design for 'linear sequential model' is as follows:

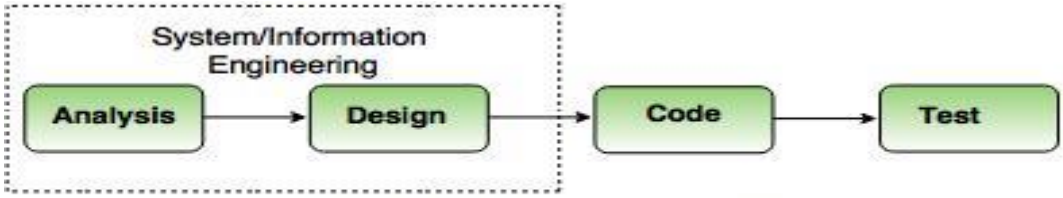


Fig. - The linear sequential model

Advantages of waterfall model

- The waterfall model is simple and easy to understand, implement, and use.
- All the requirements are known at the beginning of the project, hence it is easy to manage.
- It avoids overlapping of phases because each phase is completed at once.
- This model works for small projects because the requirements are understood very well.

- This model is preferred for those projects where the quality is more important as compared to the cost of the project.

Disadvantages of the waterfall model

- This model is not good for complex and object oriented projects.
- It is a poor model for long projects.
- The problems with this model are uncovered, until the software testing.
- The amount of risk is high.

2. Incremental Process model

- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for any suggested modifications.
- The next increment implements on the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is finished.
  For example, the word-processing software is developed using the incremental model.

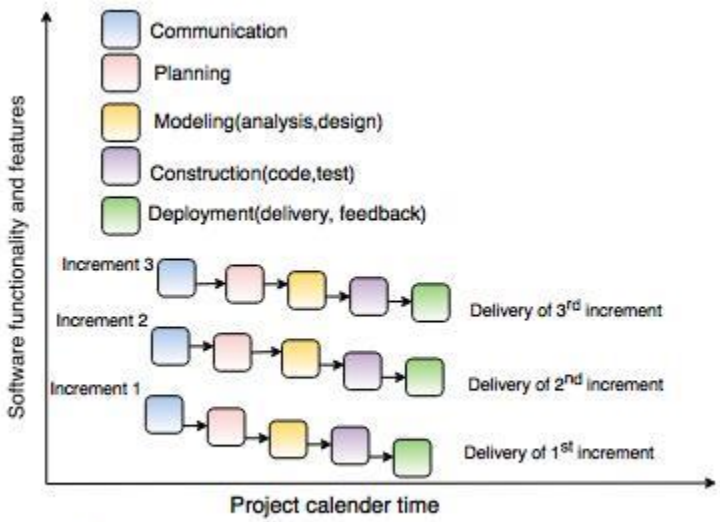


Fig. - Incremental Process Model

Advantages of incremental model

- This model is flexible because the cost of development is low and initial product delivery is faster.
- It is easier to test and debug during the smaller iteration.
- The working software generates quickly and early during the software life cycle.
- The customers can respond to its functionalities after every increment.

Disadvantages of the incremental model

- The cost of the final product may cross the cost estimated initially.
- This model requires a very clear and complete planning.
- The planning of design is required before the whole system is broken into small increments.
- The demands of customer for the additional functionalities after every increment causes problem during the system architecture.

3. RAD model

- RAD is a Rapid Application Development model.
- Using the RAD model, software product is developed in a short period of time.
- The initial activity starts with the communication between customer and developer.
- Planning depends upon the initial requirements and then the requirements are divided into groups.
- Planning is more important to work together on different modules.

The RAD model consist of following phases:

1. Business Modeling

- Business modeling consist of the flow of information between various functions in the project.
- For example what type of information is produced by every function and which are the functions to handle that information.
- A complete business analysis should be performed to get the essential business information.

2. Data modeling

- The information in the business modeling phase is refined into the set of objects and it is essential for the business.
- The attributes of each object are identified and define the relationship between objects.

3. Process modeling

- The data objects defined in the data modeling phase are changed to fulfil the information flow to implement the business model.

- The process description is created for adding, modifying, deleting or retrieving a data object.

4. Application generation

- In the application generation phase, the actual system is built.
- To construct the software the automated tools are used.

5. Testing and turnover

- The prototypes are independently tested after each iteration so that the overall testing time is reduced.
- The data flow and the interfaces between all the components are are fully tested. Hence, most of the programming components are already tested.

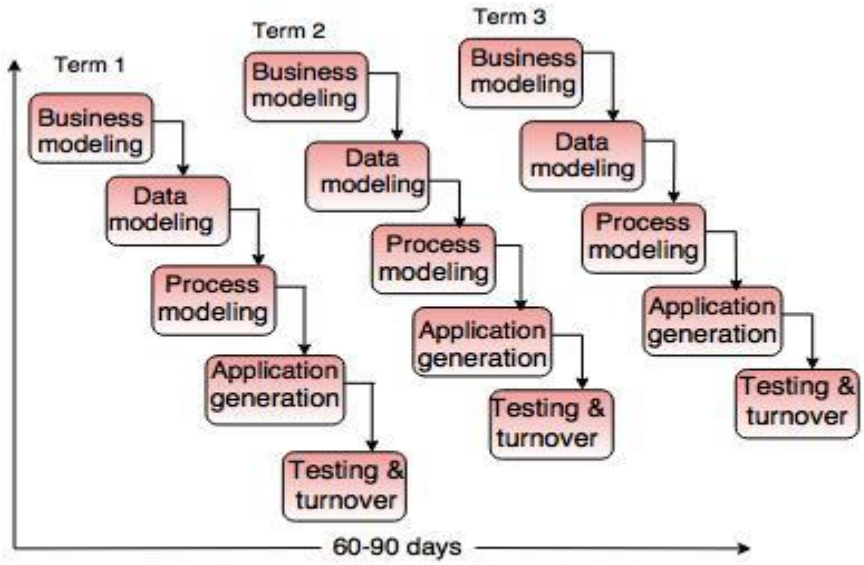


Fig. - RAD Model

Software Project Management

Software project management refers to the branch of project management dedicated to the planning, scheduling, resource allocation, execution, tracking and delivery of software and web projects.

Project management in software engineering is distinct from traditional project management in that software projects have a unique lifecycle process that requires multiple rounds of testing, updating, and customer feedback. Most IT-related projects are managed in the agile style, in order to keep up with the increasing pace of business, and iterate based on customer and stakeholder feedback.

The role and responsibility of a software project manager

Software project managers may have to do any of the following tasks:

1. Planning: This means putting together the blueprint for the entire project from ideation to fruition. It will define the scope, allocate necessary resources, propose the timeline, delineate the plan for execution, lay out a communication strategy, and indicate the steps necessary for testing and maintenance.
2. Leading: A software project manager will need to assemble and lead the project team, which likely will consist of developers, analysts, testers, graphic designers, and technical writers. This requires excellent communication, people and leadership skills.
3. Execution: The project manager will participate in and supervise the successful execution of each stage of the project. This includes monitoring progress, frequent team check-ins and creating status reports.
4. Time management: Staying on schedule is crucial to the successful completion of any project, but it's particularly challenging when it comes to managing software projects because changes to the original plan are almost certain to occur as the project evolves. Software project managers must be experts in risk management and contingency planning to ensure forward progress when roadblocks or changes occur.
5. Budget: Like traditional project managers, software project managers are tasked with creating a budget for a project, and then sticking to it as closely as possible, moderating spend and re-allocating funds when necessary.

6. Maintenance: Software project management typically encourages constant product testing in order to discover and fix bugs early, adjust the end product to the customer's needs, and keep the project on target. The software project manager is responsible for ensuring proper and consistent testing, evaluation and fixes are being made.

How to manage a software project successfully?

A recent article in Forbes suggests that there are eight ways to improve and streamline the software project management process; these eight suggestions include:

- Take non-development work off your team's plate to let them focus on developing
- Motivating your team by sharing others' success stories—like those of tech giants, which will inspire and excite your team
- Avoid altering the task once its assigned
- Try to stick to the plan (until it needs to be changed)
- Encouraging organization by being organized
- Streamline productivity through effective delegation
- Get to know your team and build a rapport
- Break down the plan and give them specific daily tasks

Component-based development (CBD)

Component-based development (CBD) is a procedure that accentuates the design and development of computer-based systems with the help of reusable software components. With CBD, the focus shifts from software programming to software system composing.

Component-based development techniques involve procedures for developing

software systems by choosing ideal off-the-shelf components and then assembling them using a well-defined software architecture. With the systematic reuse of coarse-grained components, CBD intends to deliver better quality and output.

Component-based development is also known as component-based software engineering (CBSE).

Object-oriented modeling results in a plethora of fine-grained classes, objects and relationships. It is very hard to discover reusable parts among these smaller units. The idea behind CBD is to integrate the related parts and reuse them collectively. These integrated parts are known as components.

Component-based development techniques consist of non-conventional development routines, including component evaluation, component retrieval, etc. It is important that the CBD is carried out within a middleware infrastructure that supports the process, for example, Enterprise Java Beans.

The key goals of CBD are as follows:

- Save time and money when building large and complex systems: Developing complex software systems with the help of off-the-shelf components helps reduce software development time substantially. Function points or similar techniques can be used to verify the affordability of the existing method.
- Enhance the software quality: The component quality is the key factor behind the enhancement of software quality.
- Detect defects within the systems: The CBD strategy supports fault detection by testing the components; however, finding the source of defects is challenging in CBD.

Some advantages of CBD include:

- Minimized delivery:
    - Search in component catalogs
    - Recycling of pre-fabricated components
- Improved efficiency:
    - Developers concentrate on application development
- Improved quality:
    - Component developers can permit additional time to ensure quality
- Minimized expenditures

The specific routines of CBD are:

- Component development
- Component publishing
- Component lookup as well as retrieval
- Component analysis
- Component assembly

Aspect-oriented software development (AOSD)

Aspect-oriented software development (AOSD) is a software design solution that helps address the modularity issues that are not properly resolved by other software approaches, like procedural, structured and object-oriented programming (OOP). AOSD complements, rather than replaces, these other types of software approaches.

AOSD is also known as aspect-oriented programming (AOP).

AOSD features are as follows:

- Considered a subset of post-object programming technologies

- Better software design support through isolating application business logic from supporting and secondary functions
- Provides complementary benefits and may be used with other agile processes and coding standards
- Key focus - Identification, representation and specification of concerns, which also may be cross-cutting
- Provides better modularization support of software designs, reducing software design, development and maintenance costs
- Modularization principle based on involved functionalities and processes
- Because concerns are encapsulated into different modules, localization of crosscutting concerns is better promoted and handled
- Provides tools and software coding techniques to ensure modular content support at the source code level
- Promotes reusability of code used for the modularization of cross-cutting concerns
- Smaller code size, due to tackling cross cutting concerns
- Reduced efficiency from increased overhead

Formal methods

Formal methods are system design techniques that use rigorously specified mathematical models to build software and hardware systems. In contrast to other design systems, formal methods use mathematical proof as a complement to system testing in order to ensure correct behavior. As systems become more complicated, and safety becomes a more important issue, the formal approach to system design offers another level of insurance.

Formal methods differ from other design systems through the use of formal verification schemes, the basic principles of the system must be proven correct

before they are accepted [Bowen93]. Traditional system design has used extensive testing to verify behavior, but testing is capable of only finite conclusions. Dijkstra and others have demonstrated that tests can only show the situations where a system won't fail, but cannot say anything about the behavior of the system outside of the testing scenarios [Bentley89]. In contrast, once a theorem is proven true it remains true.

It is very important to note that formal verification does not obviate the need for testing [Bowen95]. Formal verification cannot fix bad assumptions in the design, but it can help identify errors in reasoning which would otherwise be left unverified. In several cases, engineers have reported finding flaws in systems once they reviewed their designs formally [Kling95].

Roughly speaking, formal design can be seen as a three step process, following the outline given here:

1. Formal Specification: During the formal specification phase, the engineer rigorously defines a system using a modeling language. Modeling languages are fixed grammars which allow users to model complex structures out of predefined types. This process of formal specification is similar to the process of converting a word problem into algebraic notation.

In many ways, this step of the formal design process is similar to the formal software engineering technique developed by Rumbaugh, Booch and others. At the minimum, both techniques help engineers to clearly define their problems, goals and solutions. However, formal modeling languages are more rigorously defined: in a formal grammar, there is a distinction between WFFs (well-formed formulas) and non-WFFs (syntactically incorrect statements). Even at this stage, the distinction between WFF and non-WFF can help to specify the design. Several engineers who have used formal specifications say that the clarity that this stage produces is a benefit in itself [Kling95].

2. Verification: As stated above, formal methods differ from other specification systems by their heavy emphasis on provability and correctness. By building a system using a formal specification, the designer

is actually developing a set of theorems about his system. By proving these theorems correct, the formal

Verification is a difficult process, largely because even the simplest system has several dozen theorems, each of which has to be proven. Even a traditional mathematical proof is a complex affair, Wiles' proof of Fermat's Last Theorem, for example, took several years after its announcement to be completed. Given the demands of complexity and Moore's law, almost all formal systems use an automated theorem proving tool of some form. These tools can prove simple theorems, verify the semantics of theorems, and provide assistance for verifying more complicated proofs.

3. Implementation: Once the model has been specified and verified, it is implemented by converting the specification into code. As the difference between software and hardware design grows narrower, formal methods for developing embedded systems have been developed. LARCH, for example, has a VHDL implementation. Similarly, hardware systems such as the VIPER and AAMP5 processors have been developed using formal approaches.

An alternative to this approach is the lightweight approach to formal design. In a lightweight design, formal methods are applied sparingly to a system. This approach offers the benefits of formal specification, but also avoids some of the difficulties.

Formal methods are viewed with a certain degree of suspicion. While formal methods research has been progressing since 1960's, formal methods are only being slowly accepted by engineers. There are several reasons for this, but most of the problems seem to be a result of misapplication. Most formal systems are extremely descriptive and all-encompassing, modeling languages have generally been judged by their capacity to model anything. Unfortunately, these same qualities make formal methods very difficult to use, especially for engineers untrained in the type theory needed for most formal systems.[Bowen93]

Conversely, it is apparent that some form of formal specification is necessary: complex systems require formal models. In addition,the mathematics required for formal methods is becoming a more prominent fixture of engineering curricula, engineering schools in Europe are already requiring courses in VDM, Z and similar formal specifications. Ultimately, formal methods will acquire some form of acceptance, but compromises will be made in both directions: formal methods will become simpler and formal methods training will become more common.

Key Concepts

- **Provability And Automated Verification**

Formal methods are distinguished from other specification systems by their emphasis on correctness and proof, which is ultimately another measure of system integrity. Proof is a complement, not a substitute, for testing. Testing is an important part of guaranteeing any system's fitness, but it is finite. Testing cannot demonstrate that a system operates properly; it can only demonstrate that the system works for the tested cases. Because testing cannot demonstrate that the system should work outside the tested cases, formal proof is necessary.

Formally proving computer systems is not a new idea. Knuth and Dijkstra have written extensively on the topic, although their methods of proof are based on the traditional mathematical methods. In pure sciences, proofs are verified through extensive peer review before publication. Such techniques are time-intensive and less than perfect; it isn't unusual for a published proof to contain a flaw. Given the cost and time requirements of systems engineering, traditional proving techniques are not really applicable.

Because of the costs of hand verification, most formal methods use automated theorem proving systems to verify their designs. Automated theorem provers are best described as mathematical CAD tools: they can prove simple propositions and automatically and provide assistance for verifying more complex theorems.

- **Benefits Of Formal Models**

Formal methods offer additional benefits outside of provability, and these benefits do deserve some mention. However, most of these benefits are available from other systems, and usually without the steep learning curve that formal methods require.

- **Discipline**: By virtue of their rigor, formal systems require an engineer to think out his design in a more thorough fashion. In particular, a formal proof of correctness is going to require a rigorous specification of goals, not just operation. This thorough approach can help identify faulty reasoning far earlier than in traditional design.[Bowen95]

The discipline involved in formal specification has proved useful even on already existing systems. Engineers using the PVS system, for example, reported identifying several microcode errors in one of their microprocessor designs.[Miller95]

- **Precision**: Traditionally, disciplines have moved into jargons and formal notation as the weaknesses of natural language descriptions become more glaringly obvious. There is no reason that systems engineering should differ, and there are several formal methods which are used almost exclusively for notation.[Bowen93]

For engineers designing safety-critical systems, the benefits of formal methods lie in their clarity. Unlike many other design approaches, the formal verification requires very clearly defined goals and approaches. In a safety critical system, ambiguity can be extremely dangerous, and one of the primary benefits of the formal approach is the elimination of ambiguity.[Kling94].

- **Weaknesses Of Formal Methods**

: Bowen points out that formal methods are generally viewed with suspicion by the professional engineering community, and the propensity of tentative case studies and advocacy papers for the formal approach would seem to support his thesis [Bowen93]. There are several reasons why formal methods are not used as

much as they might be, most stemming from overreaching on the part of formal methods advocates.

- **Expense**:Because of the rigor involved, formal methods are always going to be more expensive than traditional approaches to engineering. However, given that software cost estimation is more of an art than a science, it is debatable exactly how much more expensive formal verification is. In general, formal methods involve a large initial cost followed by less consumption as the project progresses; this is a reverse from the normal cost model for software development.[Bowen93]
- **Limits Of Computational Models**:While not a universal problem, most formal methods introduce some form of computational model, usually hamstringing the operations allowed in order to make the notation elegant and the system provable. Unfortunately, these design limitations are usually considered intolerable from a developer's perspective.

An excellent example comes from SML. Statements of proof in SML depend on a purely functional programming model: all data is passed through the parameter/return mechanism of a function, no side effect alterations, modifications of global variables or the like is allowed [Paulson96]. Handling side effects and other aberrancies are a requirement for any system involving input, network operations or other systems which require interrupts, meaning that SML's model is, to some extent, broken.

- **Usability**:Traditionally, formal methods have been judged on the richness of their descriptive model. That is, 'good' formal methods have described a wide variety of systems, and 'bad' formal methods have been limited in their descriptive capacities. While an all-encompassing formal description is attractive from a theoretical perspective, it invariably involved developing an incredibly complex and nuanced description language, which returns to the difficulties of natural language. Case studies of full formal methods often

acknowledge the need for a less all-encompassing approach.[Miller95]

Arguably, many of these failures can be attributed to overreaching on the part of formal methods advocates. This reasoning has led to the lightweight approach to formal specification.

- **The Lightweight Approach**

The flaws in formal specifications have been heavily focused on in the past few years, leading to several alternate approaches. The traditional view of formal methods as all-encompassing highly abstracted schemes has led to formal methods being all-encompassing, extremely rigorous, and very expensive. While theoretically appealing, formal methods have generally been ignored by engineers in the field.

The lightweight approach to formal design recognizes that formal methods are not a panacea: there are areas where formal methods are useful, and areas where a formal specification will accomplish nothing. In a lightweight design, formal methods are used in specific locations, and different formal methods may be used in different subsystems, ideally playing to the strengths of each method [Easterbrook 98]. In such a system, Petri Nets might be used to describe the communications protocol, and a LARCH system might be used to model the data storage. For other parts of the system, formal specifications might be avoided entirely: for example, the user interface may be refined using a rapid prototyping system and client interviews.

The lightweight approach is a traditional engineering compromise, and there is a tradeoff. As formal methods become more common, engineers will have to learn type theory, modern algebra and proof techniques. Ultimately, engineers will have to think more like mathematicians.

Available tools, techniques, and metrics

- Larch:Unlike most formal systems, LARCH provides two levels of specification. A general high-level modeling language, and a collection of

implementation dialects designed to work with specific programming languages.

- SML: Standard Meta-Language is a strongly typed functional programming language originally designed for exploring ideas in type theory. SML has become the formal methods workhorse because of its strong typing and provability features.
- HOL:HOL, short for Higher Order Logic, is an automated theorem proving system. As with most automated theorem proving systems, HOL is a computer-aided proof tool: it proves simple theorems and assists in proving more complicated statements, but is still dependent on interaction with a trained operator. HOL has been extensively used for hardware verification, the VIPER chip being a good example.
- Petri Nets:Petri Nets are a good example of a very 'light' formal specification. Originally designed for modeling communications, Petri Nets are a graphically simple model for asynchronous processes.

Agile software engineering

Agile software engineering represents a reasonable compromise between to conventional software engineering for certain classes of software and certain types of software projects. Agile processes can deliver successful systems quickly. It stresses on continuous communication and collaboration among developers and customers.

Agile software engineering embraces a philosophy that encourages customer satisfaction, incremental software delivery, small project teams (composed of software engineers and stakeholders), informal methods, and minimal software engineering work products. Agile software engineering guidelines stress on-time delivery of an operational software increment over analysis and design

- An agile team is able to respond to changes during project development
- Agile development recognizes that project plans must be flexible

- Agility encourages team structures and attitudes that make communication among developers and customers more facile
- Agility eliminates the separation between customers and developers
- Agility emphasizes the importance of rapid delivery of operational software and de-emphasizes importance of intermediate work products
- Agility can be applied to any software process as long as the project team is allowed to streamline tasks and conduct planning in way that eliminate non-essential work products

Agile Processes

- Are based on three key assumptions
    1. It is difficult to predict in advance which requirements or customer priorities will change and which will not
    2. For many types of software design and construction activities are interleaved (construction is used to prove the design)
    3. Analysis, design, and testing are not as predictable from a planning perspective as one might like them to be
- Agile processes must be adapted incrementally to manage unpredictability
- Incremental adaptation requires customer feedback based on evaluation of delivered software increments (executable prototypes) over short time periods

Agility Principles

- Highest priority is to satisfy customer through early and continuous delivery of valuable software
- Welcome changing requirements even late in development, accommodating change is viewed as increasing the customer's competitive advantage

- Delivering working software frequently with a preference for shorter delivery schedules (e.g., every 2 or 3 weeks)
- Business people and developers must work together daily during the project
- Build projects around motivated individuals, given them the environment and support they need, trust them to get the job done
- Face-to-face communication is the most effective method of conveying information within the development team
- Working software is the primary measure of progress
- Agile processes support sustainable development, developers and customers should be able to continue development indefinitely
- Continuous attention to technical excellence and good design enhances agility
- Simplicity (defined as maximizing the work not done) is essential
- The best architectures, requirements, and design emerge from self-organizing teams
- At regular intervals teams reflects how to become more effective and adjusts its behavior accordingly

Human Factors

- Traits that need to exist in members of agile development teams:
    - Competence
    - Common focus
    - Collaboration
    - Decision-making ability

- Fuzzy-problem solving ability
- Mutual trust and respect
- Self-organization

Agile Process Models

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Crystal
- Feature Driven Development (FDD)
- Agile Modeling (AM)

Extreme Programming

- Relies on object-oriented approach
- Key activities
  - Planning (user stories created and ordered by customer value)
  - Design (simple designs preferred, CRC cards and design prototypes are only work products, encourages use of refactoring)
  - Coding (focuses on unit tests to exercise stories, emphasizes use of pairs programming to create story code, continuous integration and smoke testing is utilized)
  - Testing (unit tests created before coding are implemented using an automated testing framework to encourage use of regression testing, integration and validation testing done on daily basis, acceptance

tests focus on system features and functions viewable by the customer)

Adaptive Software Development

- Self-organization arises when independent agents cooperate to create a solution to a problem that is beyond the capability of any individual agent
- Emphasizes self-organizing teams, interpersonal collaboration, and both individual and team learning
- Adaptive cycle characteristics
- Phases
    - Mission-driven
    - Component-based
    - Iterative
    - Time-boxed
    - Risk driven and change-tolerant
    - Speculation (project initiated and adaptive cycle planning takes place)
    - Collaboration (requires teamwork from a jelled team, joint application development is preferred requirements gathering approach, minispecs created)
    - Learning (components implemented and testes, focus groups provide feedback, formal technical reviews, postmortems)

Dynamic Systems Development Method

- Provides a framework for building and maintaining systems which meet tight time constraints using incremental prototyping in a controlled environment

- Uses Pareto principle (80% of project can be delivered in 20% required to deliver the entire project)
- Each increment only delivers enough functionality to move to the next increment
- Uses time boxes to fix time and resources to determine how much functionality will be delivered in each increment
- Guiding principles
    - Active user involvement
    - Teams empowered to make decisions
    - Fitness foe business purpose is criterion for deliverable acceptance
    - Iterative and incremental develop needed to converge on accurate business solution
    - All changes made during development are reversible
    - Requirements are baselined at a high level
    - Testing integrates throughout life-cycle
    - Collaborative and cooperative approach between stakeholders
- Life cycle activities
    - Feasibility study (establishes requirements and constraints)
    - Business study (establishes functional and information requirements needed to provide business value)
    - Functional model iteration (produces set of incremental prototypes to demonstrate functionality to customer)

- Design and build iteration (revisits prototypes to ensure they provide business value for end users, may occur concurrently with functional model iteration)
- Implementation (latest iteration placed in operational environment)

Scrum

- Scrum principles
  - Small working teamed used to maximize communication, minimize overhead, and maximize sharing of informal knowledge
  - Process must be adaptable to both technical and business challenges to ensure bets product produced
  - Process yields frequent increments that can be inspected, adjusted, tested, documented and built on
  - Development work and people performing it are partitioned into clean, low coupling partitions
  - Testing and documentation is performed as the product is built
  - Provides the ability to declare the product done whenever required
- Process patterns defining development activities
  - Backlog (prioritized list of requirements or features the provide business value to customer, items can be added at any time)
  - Sprints (work units required to achieve one of the backlog items, must fir into a predefined time-box, affected backlog items frozen)
  - Scrum meetings (15 minute daily meetings) addressing these questions: What was done since last meeting? What obstacles were encountered? What will be done by the next meeting?
  - Demos (deliver software increment to customer for evaluation)

Crystal

- Development approach that puts a premium on maneuverability during a resource-limited game of invention and communication with the primary goal of delivering useful software and a secondary goal of setting up for the next game
- Crystal principles
    - Its always cheaper and faster to communicate face-to-face
    - As methodologies become more formal teams become weighed down and have trouble adapting to project work vagaries
    - As projects grow in size, teams become larger and methodologies become heavier
    - As projects grow in criticality some degree of formality will need to be introduced in parts of the methodology
    - As feedback and communication become more efficient the need for intermediate work products is reduced
    - Discipline, skills, and understanding counter process, formality, and documentation
    - Team members not on the critical project path can spend their excess time improving the product or helping people who are on the critical path
- Incremental development strategy used with 1 to 3 month time lines
- Reflection workshops conducted before project begins, during increment development activity, and after increment is delivered
- Crystal methodologies
    - Clear (small, low criticality projects)

- Orange (larger, moderately critical projects)
- Orange Web (typical e-business applications)

Feature Driven Development

- Practical process model for object-oriented software engineering
- Feature is a client-valued function, can be implemented in two weeks or less
- FDD Philosophy
  - Emphasizes collaboration among team members
  - Manages problem and project complexity using feature-based decomposition followed integration of software increments
  - Technical communication using verbal, graphical, and textual means
  - Software quality encouraged by using incremental development, design and code inspections, SQA audits, metric collection, and use of patterns (analysis, design, construction)
- Framework activities
  - Develop overall model (contains set of classes depicting business model of application to be built)
  - Build features list (features extracted from domain model, features are categorized and prioritized, work is broken up into two week chunks)
  - Plan by feature (features assessed based on priority, effort, technical issues, schedule dependencies)
  - Design by feature (classes relevant to feature are chosen, class and method prologs are written, preliminary design detail developed,

owner assigned to each class, owner responsible for maintaining design document for his or her own work packages)

- Build by feature (class owner translates design into source code and performs unit testing, integration performed by chief programmer)

Agile Modeling

- Practice-based methodology for effective modeling and documentation of software systems in a light-weight manner
- Modeling principles
  - Model with a purpose
  - Use multiple models
  - Travel light (only keep models with long-term value)
  - Content is more important than representation
  - Know the models and tools you use to create them
  - Adapt locally
- Requirements gathering and analysis modeling
  - Work collaboratively to find out what customer wants to do
  - Once requirements model is built collaborative analysis modeling continues with the customer
- Architectural modeling
  - Derives preliminary architecture from analysis model
  - Architectural model must be realistic for the environment and must be understandable by developers

WEB ENGINEERING:-

In today's world the web plays a vital role in a variety of complex and sophisticated industry applications. The web application continue to contribute to problems of usability, quality, reliability etc. With number of changes and advancements in the characteristics of web applications certain new methods needs to be introduced.Thus web engineering mainly focuses on the methods, and techniques that are considered as the main foundation of web application.

The World Wide Web and the Internet have drawn the general populace into the world of computing. We purchase stock and mutual funds, download music, view movies, get medical advice, book hotel rooms, sell personal items, schedule airline flights, meet people, do our banking, take college courses, buy groceries—we do just about anything and everything in the virtual world of the Web. Arguably, the Web and the Internet that empowers it are the most important developments in the history of computing. These computing technologies have drawn us all (with billions more who will eventually follow) into the information age. They have become integral to daily life in the first years of the twenty-first century.

For those of us who can remember a world without the Web, the chaotic growth of the technology harkens back to another era—the early days of software.It was a time of little discipline, but enormous enthusiasm and creativity. It was a time when programmers often hacked together systems—some good, some bad. The prevailing attitude seemed to be "Get it done fast, and get it into the field, we'll clean it up (and better understand what we really need to build) as we go." Sound familiar?

This leads us to a pivotal question: Can software engineering principles, concepts, and methods be applied to Web development? Many of them can, but their application may require a somewhat different spin.

But what if the current ad hoc approach to Web development persists? In the absence of a disciplined process for developing Web-based systems, there is increasing concern that we may face serious problems in the successful

development, deployment, and "maintenance" of these systems. In essence, the application infrastructure that we are creating today may lead to something that might be called a tangled Web as we move further into this new century. This phrase connotes a morass of poorly developed Web-based applications that have too high a probability of failure. Worse, as Web-based systems grow more complex, a failure in one can and will propagate broad-based problems across many. When this happens, confidence in the entire Internet may be shaken irreparably. Worse, it may lead to unnecessary and illconceived government regulation, leading to irreparable harm to these unique technologies.

In order to avoid a tangled Web and achieve greater success in development and application of large-scale, complex Web-based systems, there is a pressing need for disciplined Web engineering approaches and new methods and tools for development, deployment, and evaluation of Web-based systems and applications. Such approaches and techniques must take into account the special features of the new medium, the operational environments and scenarios, and the multiplicity of user profiles that pose additional challenges to Web-based application development. Web Engineering (WebE) is concerned with the establishment and use of sound scientific, engineering, and management principles and disciplined and systematic approaches to the successful development, deployment, and maintenance of highquality Web-based systems and applications.

Software Requirements

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

the process briefly -

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

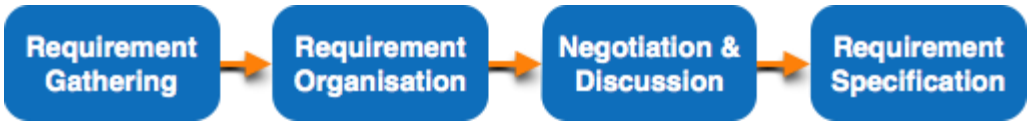
After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase

in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the folloiwng diagram:



- **Requirements gathering -** The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements -** The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion -** If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation -** All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Software Requirements Characteristics

Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct and well-defined.

A complete Software Requirement Specifications must be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirements are expected from the software system.

Broadly software requirements should be categorized in two categories:

Functional Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

Examples -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders and negation, whereas 'could have' and 'wish list' can be kept for software updates.

User Interface requirements

UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is -

- easy to operate
- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system can not be used in convenient way. A system is said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive

- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategical use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Software System Analyst

System analyst in an IT organization is a person, who analyzes the requirement of proposed system and ensures that requirements are conceived and documented properly & correctly. Role of an analyst starts during Software Analysis Phase of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- Analyzing and understanding requirements of intended software
- Understanding how the project will contribute in the organization objectives
- Identify sources of requirement
- Validation of requirement
- Develop and implement requirement management plan
- Documentation of business, technical, process and product requirements
- Coordination with clients to prioritize requirements and remove and ambiguity

- Finalizing acceptance criteria with client and other stakeholders

Software Metrics and Measures

Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software.

Software Metrics provide measures for various aspects of software process and software product.

Software measures are fundamental requirement of software engineering. They not only help to control the software development process but also aid to keep quality of ultimate product excellent.

According to Tom DeMarco, a (Software Engineer), "You cannot control what you cannot measure." By his saying, it is very clear how important software measures are.

some software metrics:

- **Size Metrics -** LOC (Lines of Code), mostly calculated in thousands of delivered source code lines, denoted as KLOC.

Function Point Count is measure of the functionality provided by the software. Function Point count defines the size of functional aspect of software.

- **Complexity Metrics -** McCabe's Cyclomatic complexity quantifies the upper bound of the number of independent paths in a program, which is perceived as complexity of the program or its modules. It is represented in terms of graph theory concepts by using control flow graph.
- **Quality Metrics -** Defects, their types and causes, consequence, intensity of severity and their implications define the quality of product.

The number of defects found in development process and number of defects reported by the client after the product is installed or delivered at client-end, define quality of product.

- **Process Metrics -** In various phases of SDLC, the methods and tools used, the company standards and the performance of development are software process metrics.
- **Resource Metrics -** Effort, time and various resources used, represents metrics for resource measurement.

Use case

After the requirements are clear, then the use cases can be designed/decided which will clear the overall flow of the system to developers first and then they can design the system accordingly. Once the flow is clear to them, then it becomes somehow simple to test the system by QA.

Use cases attempt to bridge the problem of requirements not being tied to user interaction. A use case is written as a series of interactions between the user and the system, similar to a call and response where the focus is on how the user will use the system. In many ways, use cases are better than a traditional requirement because they emphasize user-oriented context. The value of the use case to the user can be divined, and tests based on the system response can be figured out based on the interactions. Use cases usually have two main components: Use case diagrams, which graphically describe actors and their use cases, and the text of the use case itself.

Use cases are sometimes used in heavyweight, control-oriented processes much like traditional requirements. The system is specified to a high level of completion via the use cases and then locked down with change control on the assumption that the use cases capture everything.

Both use cases and traditional requirements can be used in agile software development, but they may encourage leaning heavily on documented specification of the system rather than collaboration. Since there is no built-in focus on collaboration, it can be tempting to delve into a detailed specification where the use case becomes the source of record rather than a mechanism for conversations.

Use Case Example

The placement specialist selects a candidate search.

The system retrieves a list of candidate specialties and populates the candidacy list.

The system retrieves a predefined list of candidate specialties and populates the specialty list.

The system retrieves a predefined list of geographic regions.

The system displays the candidate search screen.

The placement specialist selects a candidate specialty.

The placement specialist selects a candidate role.

The placement specialist selects a geographic region.

The system identifies the region and populates the subregion.

The placement specialist selects the geographic subregion.

The placement specialist selects the search initiation button or mechanism.

The system retrieves a list of candidates who match the candidate specialty search. [Alt 1]

The system displays a list of candidates who match the search. [Alt 2]

End use case

Alt 1: If there are no matches, the system displays a message indicating that no matches were found. End use case.

Alt 2: If there are more matches than the user can view, the system will provide the capability to display multiple pages. End use case.

User Stories

User stories are more narrative than either traditional requirements or use cases.

They are intended to describe what the user wants to be able to do. Additionally, user stories focus on the value that comes from using the system rather than a detailed specification of what the system should do. User stories are often called a metaphor for the work being done by the system. The purpose is to capture enough that the agile team can estimate the user story and understand it enough to draft a rough design. The story's details are collaboratively worked during conversations so the user story itself becomes a promise to hold a conversation as the project progresses. User stories are intended as a means to foster collaboration.

Although user stories aren't agile in and of themselves, they do emphasize collaboration, which makes user stories difficult to use in non-agile projects. I've seen teams that had trouble practicing agile end up with user stories that seemed like highly detailed traditional requirements. Over specification can lead to a series of iterations that become mini waterfalls with each phase (analysis, design, code, and, test) becoming an exercise in writing out an inordinate amount of detail. This is counter to the Agile Manifesto's principle that "the most efficient and effective method of conveying information to and within a development team is face-to-face conversation."

To sum up the differences: Traditional requirements focus on system operations with a tendency toward detailed system specification; use cases focus on interactions between the user and the system with a similar tendency of detailed specification; and user stories focus on customer value with a built-in imprecision meant to encourage communication.

Requirement Modeling

Following are the requirement modeling strategies:

1. Flow Oriented Modeling
2. Class-based Modeling

1. Flow Oriented Modeling

It shows how data objects are transformed by processing the function.

The Flow oriented elements are:

i. Data flow model

- It is a graphical technique. It is used to represent information flow.
- The data objects are flowing within the software and transformed by processing the elements.
- The data objects are represented by labeled arrows. Transformation are represented by circles called as bubbles.
- DFD shown in a hierarchical fashion. The DFD is split into different levels. It also called as 'context level diagram'.

ii. Control flow model

- Large class applications require a control flow modeling.
- The application creates control information instated of reports or displays.
- The applications process the information in specified time.
- An event is implemented as a boolean value.
  For example, the boolean values are true or false, on or off, 1 or 0.

iii. Control Specification

- A short term for control specification is CSPEC.
- It represents the behaviour of the system.
- The state diagram in CSPEC is a sequential specification of the behaviour.
- The state diagram includes states, transitions, events and activities.
- State diagram shows the transition from one state to another state if a particular event has occurred.

iv. Process Specification

- A short term for process specification is PSPEC.
- The process specification is used to describe all flow model processes.
- The content of process specification consists narrative text, Program Design Language(PDL) of the process algorithm, mathematical equations, tables or UML activity diagram.

2. Class-based Modeling

- Class based modeling represents the object. The system manipulates the operations.
- The elements of the class based model consist of classes and object, attributes, operations, class – responsibility - collaborator (CRS) models.

Classes

Classes are determined using underlining each noun or noun clause and enter it into the simple table.

Classes are found in following forms:

- **External entities:** The system, people or the device generates the information that is used by the computer based system.
- **Things:** The reports, displays, letter, signal are the part of the information domain or the problem.
- **Occurrences or events:** A property transfer or the completion of a series or robot movements occurs in the context of the system operation.
- **Roles:** The people like manager, engineer, salesperson are interacting with the system.
- **Organizational units:** The division, group, team are suitable for an application.

- **Places:** The manufacturing floor or loading dock from the context of the problem and the overall function of the system.
- **Structures:** The sensors, computers are defined a class of objects or related classes of objects.

Attributes

Attributes are the set of data objects that are defining a complete class within the context of the problem.

For example, 'employee' is a class and it consists of name, Id, department, designation and salary of the employee are the attributes.

Operations

The operations define the behaviour of an object.

The operations are characterized into following types:

- The operations manipulate the data like adding, modifying, deleting and displaying etc.
- The operations perform a computation.
- The operation monitors the objects for the occurrence of controlling an event.

CRS Modeling

- The CRS stands for Class-Responsibility-Collaborator.
- It provides a simple method for identifying and organizing the classes that are applicable to the system or product requirement.
- Class is an object-oriented class name. It consists of information about sub classes and super class.
- Responsibilities are the attributes and operations that are related to the class.

- Collaborations are identified and determined when a class can achieve each responsibility of it. If the class cannot identify itself, then it needs to interact with another class.

Behavioral patterns for requirement modeling

Behavioral model shows the response of software to an external event.

Steps for creating behavioral patterns for requirement modeling as follows:

- Evaluate all the use cases to completely understand the sequence, interaction within the system.
- Identify the event and understand the relation between the specific event.
- Generate a sequence for each use case.
- Construct a state diagram for the system.
- To verify the accuracy and consistency review the behavioral model.

Software Requirement Specification (SRS)

- The requirements are specified in specific format known as SRS.
- This document is created before starting the development work.
- The software requirement specification is an official document.
- It shows the detail about the performance of expected system.
- SRS indicates to a developer and a customer what is implemented in the software.
- SRS is useful if the software system is developed by the outside contractor.
- SRS must include an interface, functional capabilities, quality, reliability, privacy etc.

Characteristics of SRS

- The SRS should be complete and consistence.
- The modification like logical and hierarchical must be allowed in SRS.
- The requirement should be easy to implement.
- Each requirement should be uniquely identified.
- The statement in SRS must be unambiguous means it should have only one meaning.
- All the requirement must be valid for the specified project.

Requirements Review

The requirements review is an opportunity to check the requirements document for errors and omissions, and confirm that the overall quality of the requirements is acceptable before the project proceeds. It is also one final opportunity to communicate and confirm project requirements with all stakeholders.

While stakeholders may complain that they are too busy to attend, requirements reviews are crucial. Simply sending the document out for review will likely lead to rubber stamping by busy people, although a formal signoff can still happen following the requirements review meeting. Since some important stakeholders may not even really know what they are looking at, they need guidance to effectively understand the requirements.

There are many benefits of a face-to-face review meeting.

- The discussion among the broader team to validate the clarity and accuracy of requirements is likely to be higher quality than individuals independently reviewing the requirements document.
- It ties the web of requirements together in everyone's minds as they discuss the complete requirements in context.

- Stakeholders align around the scope and confirm the requirements are clear enough that they all interpret them the same way. They also gain an understanding of how the requirements affect them.
- Stakeholders who may not be knowledgeable about requirements gathering, authoring, and reviewing will be educated in the process. This may not be important for the project at hand, but it facilitates long-term knowledge transfer and retention.

Some benefits can be enhanced both before and during the review by tools like QVscribe:

- Improving the quality of requirements
- Finding redundancies and similarities between requirements
- Improving the consistency of the requirements
- Generate configurable, shareable reports

Steps in the review process

1. Decide how to conduct the review

There are several things to consider as you decide how the review will be conducted.

Will the review be formal or informal?

Depending on project complexity, stakeholder maturity, and organizational culture, the review process can be conducted either formally or informally. In a formal review, the project team steps through each requirement with the stakeholders and explains the implications one-by-one. An informal review is more relaxed in its depth and is generally better suited to stakeholders who are very familiar with the process.

Will the review be broad or focused?

In a broad review, the entire team reviews the entire document. A broad review is usually best for smaller documents. In a focused review, different parts of the document are distributed among different teams so the review is a less overwhelming task. This approach can also be useful if different stakeholders are affected by different parts of the requirements document. Of course, it is easier to miss the big picture and potentially conflicting requirements when the review is divided. You will likely still want a broad review of the entire document following focused reviews, but it will go faster knowing that the details have already been reviewed. A QVscribe report can give you an added level of confidence in the cohesiveness of large requirements documents.

Will the review be onsite or offsite?

Consider holding the review meeting offsite. While this likely involves the cost of renting meeting space and can be less convenient, it offers the opportunity for participants to be more focused. They will not have the ability to wander off to their desks or to other meetings or to be caught in the hallway by other employees with questions.

Will the review be face-to-face or involve remote meetings?

Without a doubt, face-to-face meetings are always better. But there are almost certainly stakeholders distributed around the country – or even around the world. If remote meetings are the only option, check that necessary communication and screen sharing technology is available and functions well. One option to consider for dispersed participants is to meet face-to-face in an airport hub city. Although this means everyone must travel, it also means all participants can get a non-stop flight. Many airports even have meeting space available for rental right at the airport.

2. Identify the participants and schedule the meeting

It is essential to identify the right participants for the review. Take the time to do this well. Look back at your stakeholder register. Assuming you have kept this current throughout the project, it will make it easy to generate your participant list and ensure the proper cross-section of stakeholders.

Consider how best to get representation from each department or function, from the shop floor users to management, and one personality type to another. Be sure there are people with different skills, backgrounds, and knowledge in the review. Wide representation is best so as many stakeholders as possible are involved and no viewpoint or input is missed.

In most organizations, it is difficult to find a time when all participants are available for a lengthy review. Schedule the meeting as far in advance as possible to allow all key stakeholders to attend.

Software Design And Implementation

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

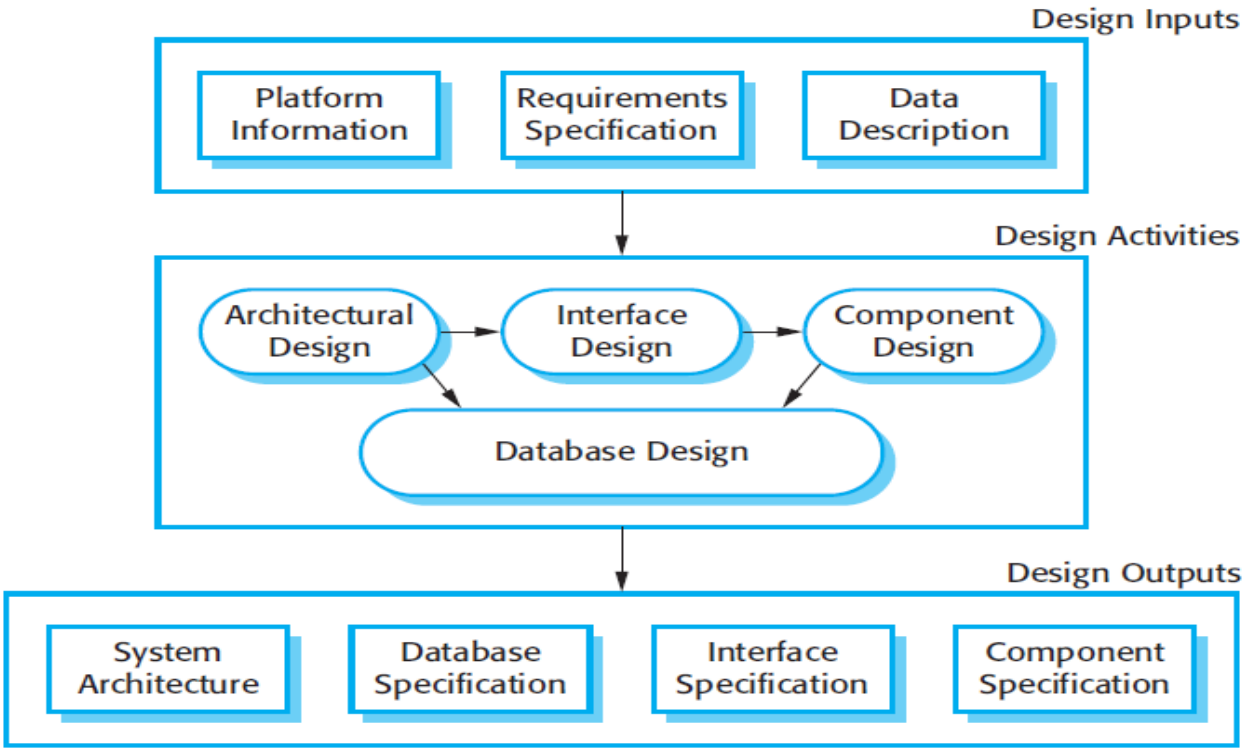
Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

The implementation phase is the process of converting a system specification into an executable system. If an incremental approach is used, it may also involve refinement of the software specification.

A software design is a description of the structure of the software to be implemented, data models, interfaces between system components, and maybe the algorithms used.

The software designers develop the software design iteratively; they add formality and detail and correct the design as they develop their design.

Here's an abstract model of the design process showing the inputs, activities, and the documents to be produced as output.



The software design process

The diagram suggests that the stages of the design process are sequential. In fact, they are interleaved. A feedback from one stage to another and rework can't be avoided in any design process.

These activities can vary depending on the type of the system needs to be developed. We've showed four main activities that may be part of the design process for information systems, and they are:

1. **Architectural design:** It defines the overall structure of the system, the main components, their relationships.
2. **Interface design:** It defines the interfaces between these components. The interface specification must be clear. Therefore, a component can be used without having to know it's implemented. Once the interface specification are agreed, the components can be designed and developed concurrently.

3. **Component design:** Take each component and design how it will operate, with the specific design left to the programmer, or a list of changes to be made to a reusable component.

4. **Database design:** The system data structures are designed and their representation in a database is defined. This depends on whether an existing database is to be reused or a new database to be created.

These activities lead to a set of design outputs. The detail and representation vary based on the system being developed.

For example, in critical systems, detailed design documents giving a precise and accurate description of the system must be produced.

These outputs may be graphical models of the system, and in many cases, automatically generating code from these models.

Software Design Levels

Software design yields three levels of results:

- **Architectural Design -** The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

- **High-level Design-** The high-level design breaks the 'single entity-multiple component' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.

- **Detailed Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines

logical structure of each module and their interfaces to communicate with other modules.

Modularization

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of 'divide and conquer' problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

Advantage of modularization:

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- Concurrent execution can be made possible
- Desired from security aspect

Concurrency

Back in time, all software are meant to be executed sequentially. By sequential execution we mean that the coded instruction will be executed one after another implying only one portion of program being activated at any given time. Say, a software has multiple modules, then only one of all the modules can be found active at any time of execution.

In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in

parallel. In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.

It is necessary for the programmers and designers to recognize those modules, which can be made parallel execution.

Example

The spell check feature in word processor is a module of software, which runs along side the word processor itself.

Coupling and Cohesion

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

- **Co-incidental cohesion -** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
- **Logical cohesion -** When logically categorized elements are put together into a module, it is called logical cohesion.
- **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.

- **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion -** When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **Functional cohesion -** It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Coupling

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

There are five levels of coupling, namely -

- **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.
- **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.

- **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- **Data coupling-** Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

Design Verification

The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional or non-functional requirements.

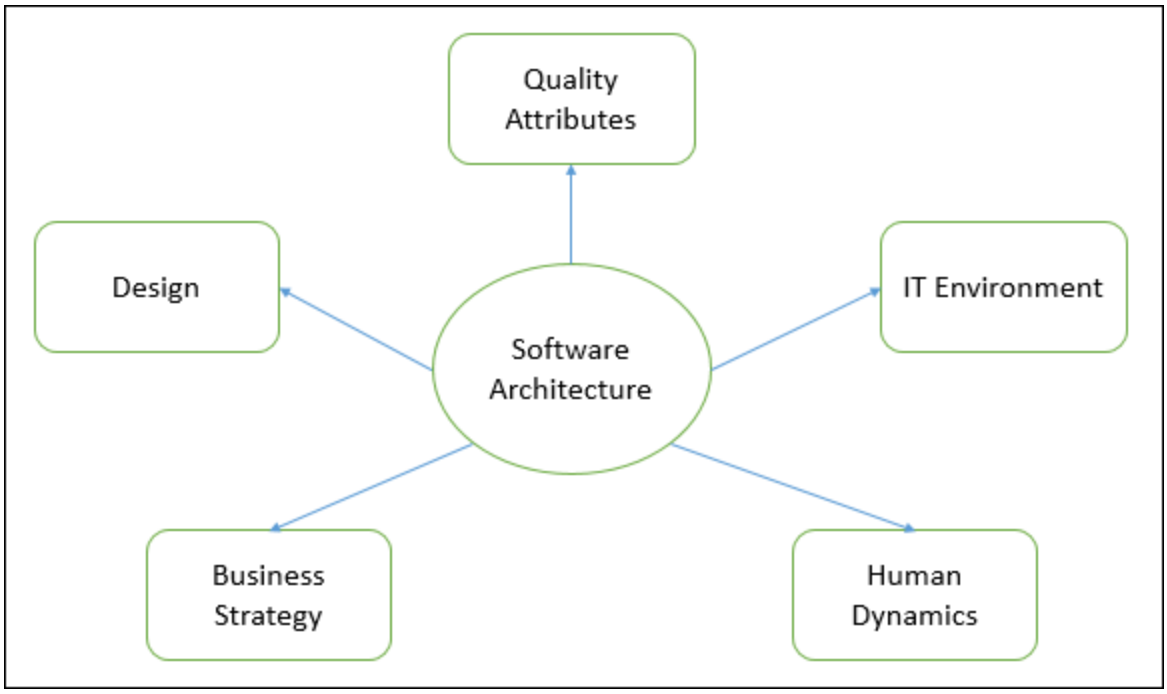
The next phase, which is the implementation of software, depends on all outputs mentioned above.

It is then becomes necessary to verify the output before proceeding to the next phase. The early any mistake is detected, the better it is or it might not be detected until testing of the product. If the outputs of design phase are in formal notation form, then their associated tools for verification should be used otherwise a thorough design review can be used for verification and validation.

By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions. A good design review is important for good software design, accuracy and quality.

Software Architecture

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

Software Architecture

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of –

- Selection of structural elements and their interfaces by which the system is composed.
- Behavior as specified in collaborations among those elements.
- Composition of these structural and behavioral elements into large subsystem.
- Architectural decisions align with business objectives.
- Architectural styles guide the organization.

Goals of Architecture

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

Some of the other goals are as follows –

- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization's market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

Limitations

Software architecture is still an emerging discipline within software engineering. It has the following limitations –

- Lack of tools and standardized ways to represent architecture.
- Lack of analysis methods to predict whether architecture will result in an implementation that meets the requirements.
- Lack of awareness of the importance of architectural design to software development.
- Lack of understanding of the role of software architect and poor communication among stakeholders.
- Lack of understanding of the design process, design experience and evaluation of design.

Role of Software Architect

A Software Architect provides a solution that the technical team can create and design for the entire application. A software architect should have expertise in the following areas –

Design Expertise

- Expert in software design, including diverse methods and approaches such as object-oriented design, event-driven design, etc.
- Lead the development team and coordinate the development efforts for the integrity of the design.
- Should be able to review design proposals and tradeoff among themselves.

Domain Expertise

- Expert on the system being developed and plan for software evolution.
- Assist in the requirement investigation process, assuring completeness and consistency.
- Coordinate the definition of domain model for the system being developed.

Technology Expertise

- Expert on available technologies that helps in the implementation of the system.
- Coordinate the selection of programming language, framework, platforms, databases, etc.

Methodological Expertise

- Expert on software development methodologies that may be adopted during SDLC (Software Development Life Cycle).
- Choose the appropriate approaches for development that helps the entire team.

Hidden Role of Software Architect

- Facilitates the technical work among team members and reinforcing the trust relationship in the team.
- Information specialist who shares knowledge and has vast experience.
- Protect the team members from external forces that would distract them and bring less value to the project.

Deliverables of the Architect

- A clear, complete, consistent, and achievable set of functional goals
- A functional description of the system, with at least two layers of decomposition
- A concept for the system
- A design in the form of the system, with at least two layers of decomposition
- A notion of the timing, operator attributes, and the implementation and operation plans

- A document or process which ensures functional decomposition is followed, and the form of interfaces is controlled

Information hiding

Information hiding for programmers is executed to prevent system design change. If design decisions are hidden, certain program code cannot be modified or changed. Information hiding is usually done for internally changeable code, which is sometimes especially designed not to be exposed. Such stored and derived data is not expounded upon, most generally. Change resilience of classes and ease of use by client objects are two byproducts of hidden data.

In 1972, David Parnas introduced the idea of information hiding. He believed that critical design systems should be hidden from clients and front end users. He defined information hiding as a way in which clients could be shielded from internal program workings.

For example, a calculation producing a given result may be hidden. It follows a model of functionality that can be described as a type of information hiding.

One advantage of information hiding is yielding flexibility, such as allowing a programmer to more readily modify a program. This also may be done by placing source code within modules for easy access in the future, as the program develops and evolves.

Functional independence

Functional independence in software engineering means that when a module focuses on a single task, it should be able to accomplish it with very little interaction with other modules.

In software engineering, if a module is functionally independent of other module then it means it has high cohesion and low coupling.

Functional independence is essential for good software design.

Advantage of functional independence

Advantages of functional independence are given below:

Error isolation

When a module is functionally independent then it performs most of its task independently without interacting with other modules much. This reduces the chances of error getting propagated to other modules. This helps in easily isolating and tracing the error.

Module reusability

A functionally independent module performs some well defined and specific task. So it becomes easy to reuse such modules in different program requiring same functionality.

Understandability

A functionally independent module is less complex so easy to understand. Since such modules are less interaction with other modules so can be understood in isolation.

Object-Oriented Design

After the analysis phase, the conceptual model is developed further into an object-oriented model using object-oriented design (OOD). In OOD, the technology-independent concepts in the analysis model are mapped onto implementing classes, constraints are identified, and interfaces are designed, resulting in a model for the solution domain. In a nutshell, a detailed description is constructed specifying how the system is to be built on concrete technologies

The stages for object–oriented design can be identified as −

- Definition of the context of the system
- Designing system architecture
- Identification of the objects in the system

- Construction of design models
- Specification of object interfaces

System Design

Object-oriented system design involves defining the context of a system followed by designing the architecture of the system.

- **Context** − The context of a system has a static and a dynamic part. The static context of the system is designed using a simple block diagram of the whole system which is expanded into a hierarchy of subsystems. The subsystem model is represented by UML packages. The dynamic context describes how the system interacts with its environment. It is modelled using **use case diagrams**.
- **System Architecture** − The system architecture is designed on the basis of the context of the system in accordance with the principles of architectural design as well as domain knowledge. Typically, a system is partitioned into layers and each layer is decomposed to form the subsystems.

Object-Oriented Decomposition

Decomposition means dividing a large complex system into a hierarchy of smaller components with lesser complexities, on the principles of divide–and–conquer. Each major component of the system is called a subsystem. Object-oriented decomposition identifies individual autonomous objects in a system and the communication among these objects.

The advantages of decomposition are −

- The individual components are of lesser complexity, and so more understandable and manageable.
- It enables division of workforce having specialized skills.
- It allows subsystems to be replaced or modified without affecting other subsystems.

Identifying Concurrency

Concurrency allows more than one objects to receive events at the same time and more than one activity to be executed simultaneously. Concurrency is identified and represented in the dynamic model.

To enable concurrency, each concurrent element is assigned a separate thread of control. If the concurrency is at object level, then two concurrent objects are assigned two different threads of control. If two operations of a single object are concurrent in nature, then that object is split among different threads.

Concurrency is associated with the problems of data integrity, deadlock, and starvation. So a clear strategy needs to be made whenever concurrency is required. Besides, concurrency requires to be identified at the design stage itself, and cannot be left for implementation stage.

Identifying Patterns

While designing applications, some commonly accepted solutions are adopted for some categories of problems. These are the patterns of design. A pattern can be defined as a documented set of building blocks that can be used in certain types of application development problems.

Some commonly used design patterns are −

- Façade pattern
- Model view separation pattern
- Observer pattern
- Model view controller pattern
- Publish subscribe pattern
- Proxy pattern

Controlling Events

During system design, the events that may occur in the objects of the system need to be identified and appropriately dealt with.

An event is a specification of a significant occurrence that has a location in time and space.

There are four types of events that can be modelled, namely –

- **Signal Event** – A named object thrown by one object and caught by another object.
- **Call Event** – A synchronous event representing dispatch of an operation.
- **Time Event** – An event representing passage of time.
- **Change Event** – An event representing change in state.

Handling Boundary Conditions

The system design phase needs to address the initialization and the termination of the system as a whole as well as each subsystem. The different aspects that are documented are as follows –

- The start–up of the system, i.e., the transition of the system from non-initialized state to steady state.
- The termination of the system, i.e., the closing of all running threads, cleaning up of resources, and the messages to be sent.
- The initial configuration of the system and the reconfiguration of the system when needed.
- Foreseeing failures or undesired termination of the system.

Boundary conditions are modelled using boundary use cases.

Object Design

After the hierarchy of subsystems has been developed, the objects in the system are identified and their details are designed. Here, the designer details out the

strategy chosen during the system design. The emphasis shifts from application domain concepts toward computer concepts. The objects identified during analysis are etched out for implementation with an aim to minimize execution time, memory consumption, and overall cost.

Object design includes the following phases –

- Object identification
- Object representation, i.e., construction of design models
- Classification of operations
- Algorithm design
- Design of relationships
- Implementation of control for external interactions
- Package classes and associations into modules

Object Identification

The first step of object design is object identification. The objects identified in the object–oriented analysis phases are grouped into classes and refined so that they are suitable for actual implementation.

The functions of this stage are −

- Identifying and refining the classes in each subsystem or package
- Defining the links and associations between the classes
- Designing the hierarchical associations among the classes, i.e., the generalization/specialization and inheritances
- Designing aggregations

Object Representation

Once the classes are identified, they need to be represented using object modelling techniques. This stage essentially involves constructing UML diagrams.

There are two types of design models that need to be produced −

- **Static Models** − To describe the static structure of a system using class diagrams and object diagrams.
- **Dynamic Models** − To describe the dynamic structure of a system and show the interaction between classes using interaction diagrams and state–chart diagrams.

Classification of Operations

In this step, the operation to be performed on objects are defined by combining the three models developed in the OOA phase, namely, object model, dynamic model, and functional model. An operation specifies what is to be done and not how it should be done.

The following tasks are performed regarding operations −

- The state transition diagram of each object in the system is developed.
- Operations are defined for the events received by the objects.
- Cases in which one event triggers other events in same or different objects are identified.
- The sub–operations within the actions are identified.
- The main actions are expanded to data flow diagrams.

Algorithm Design

The operations in the objects are defined using algorithms. An algorithm is a stepwise procedure that solves the problem laid down in an operation. Algorithms focus on how it is to be done.

There may be more than one algorithm corresponding to a given operation. Once the alternative algorithms are identified, the optimal algorithm is selected for the given problem domain. The metrics for choosing the optimal algorithm are –

- **Computational Complexity** – Complexity determines the efficiency of an algorithm in terms of computation time and memory requirements.
- **Flexibility** – Flexibility determines whether the chosen algorithm can be implemented suitably, without loss of appropriateness in various environments.
- **Understandability** – This determines whether the chosen algorithm is easy to understand and implement.

Design of Relationships

The strategy to implement the relationships needs to be chalked out during the object design phase. The main relationships that are addressed comprise of associations, aggregations, and inheritances.

The designer should do the following regarding associations –

- Identify whether an association is unidirectional or bidirectional.
- Analyze the path of associations and update them if necessary.
- Implement the associations as a distinct object, in case of many–to-many relationships; or as a link to other object in case of one–to-one or one–to-many relationships.

Regarding inheritances, the designer should do the following –

- Adjust the classes and their associations.
- Identify abstract classes.
- Make provisions so that behaviors are shared when needed.

Implementation of Control

The object designer may incorporate refinements in the strategy of the state–chart model. In system design, a basic strategy for realizing the dynamic model is made. During object design, this strategy is aptly embellished for appropriate implementation.

The approaches for implementation of the dynamic model are −

- **Represent State as a Location within a Program** − This is the traditional procedure-driven approach whereby the location of control defines the program state. A finite state machine can be implemented as a program. A transition forms an input statement, the main control path forms the sequence of instructions, the branches form the conditions, and the backward paths form the loops or iterations.
- **State Machine Engine** − This approach directly represents a state machine through a state machine engine class. This class executes the state machine through a set of transitions and actions provided by the application.
- **Control as Concurrent Tasks** − In this approach, an object is implemented as a task in the programming language or the operating system. Here, an event is implemented as an inter-task call. It preserves inherent concurrency of real objects.

Packaging Classes

In any large project, meticulous partitioning of an implementation into modules or packages is important. During object design, classes and objects are grouped into packages to enable multiple groups to work cooperatively on a project.

The different aspects of packaging are –

- **Hiding Internal Information from Outside View** − It allows a class to be viewed as a “black box” and permits class implementation to be changed without requiring any clients of the class to modify code.

- **Coherence of Elements** − An element, such as a class, an operation, or a module, is coherent if it is organized on a consistent plan and all its parts are intrinsically related so that they serve a common goal.
- **Construction of Physical Modules** − The following guidelines help while constructing physical modules −
    - Classes in a module should represent similar things or components in the same composite object.
    - Closely connected classes should be in the same module.
    - Unconnected or weakly connected classes should be placed in separate modules.
    - Modules should have good cohesion, i.e., high cooperation among its components.
    - A module should have low coupling with other modules, i.e., interaction or interdependence between modules should be minimum.

Design Optimization

The analysis model captures the logical information about the system, while the design model adds details to support efficient information access. Before a design is implemented, it should be optimized so as to make the implementation more efficient. The aim of optimization is to minimize the cost in terms of time, space, and other metrics.

However, design optimization should not be excess, as ease of implementation, maintainability, and extensibility are also important concerns. It is often seen that a perfectly optimized design is more efficient but less readable and reusable. So the designer must strike a balance between the two.

The various things that may be done for design optimization are −

- Add redundant associations

- Omit non-usable associations
- Optimization of algorithms
- Save derived attributes to avoid re-computation of complex expressions

Addition of Redundant Associations

During design optimization, it is checked if deriving new associations can reduce access costs. Though these redundant associations may not add any information, they may increase the efficiency of the overall model.

Omission of Non-Usable Associations

Presence of too many associations may render a system indecipherable and hence reduce the overall efficiency of the system. So, during optimization, all non-usable associations are removed.

Optimization of Algorithms

In object-oriented systems, optimization of data structure and algorithms are done in a collaborative manner. Once the class design is in place, the operations and the algorithms need to be optimized.

Optimization of algorithms is obtained by –

- Rearrangement of the order of computational tasks
- Reversal of execution order of loops from that laid down in the functional model
- Removal of dead paths within the algorithm

Saving and Storing of Derived Attributes

Derived attributes are those attributes whose values are computed as a function of other attributes (base attributes). Re-computation of the values of derived attributes every time they are needed is a time–consuming procedure. To avoid this, the values can be computed and stored in their computed forms.

However, this may pose update anomalies, i.e., a change in the values of base attributes with no corresponding change in the values of the derived attributes. To avoid this, the following steps are taken –

- With each update of the base attribute value, the derived attribute is also re-computed.
- All the derived attributes are re-computed and updated periodically in a group rather than after each update.

Design Documentation

Documentation is an essential part of any software development process that records the procedure of making the software. The design decisions need to be documented for any non–trivial software system for transmitting the design to others.

Usage Areas

Though a secondary product, a good documentation is indispensable, particularly in the following areas –

- In designing software that is being developed by a number of developers
- In iterative software development strategies
- In developing subsequent versions of a software project
- For evaluating a software
- For finding conditions and areas of testing
- For maintenance of the software.

Contents

A beneficial documentation should essentially include the following contents –

- **High–level system architecture** – Process diagrams and module diagrams

- **Key abstractions and mechanisms** – Class diagrams and object diagrams.
- **Scenarios that illustrate the behavior of the main aspects** – Behavioural diagrams

Features

The features of a good documentation are –

- Concise and at the same time, unambiguous, consistent, and complete
- Traceable to the system's requirement specifications
- Well-structured
- Diagrammatic instead of descriptive

Data Design

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

a) Data Design at the Architectural Level

The challenge in data design is to extract useful information from this data environment, particularly when the information desired is cross-functional.
To solve this challenge, the business IT community has developed data mining techniques, also called knowledge discovery in database (KDD) , that navigate through existing databases in an attempt to extract appropriate business-level information. An alternative solution, called a data warehouse, adds an additional layer to the data architecture.

A data warehouse is a separate data environment that is not directly integrated with day –to-day application but encompasses all data used by a business.

b) Data Design at the Component Level

Data design at the component level focuses on the representation of the data structures that are directly accessed by one or more software components. We consider the following set of principles (adapted from for data specification):

1. The systematic analysis principles applied to function and behavior should also be applied to data.
2. All data structure and the operations to be performed on each should be identified.
3. A mechanism for defining the content of each data object should be established and used to define both data and the operation applied it.
4. Low-level design decision should be known only to those modules that must make direct use of the data contained within the structure.
5. The representation of a data structure should be known only to those modules that must make direct use of the data contained within the structure.
6. A library of useful data structures and the operations that may be applied to them should be developed.
7. A software design and programming language should support the specification and realization of abstract data types.

Architectural Styles

The software that is built for computer- based systems also exhibits one of many architectural styles. Each styles describes a system category that encompasses (1) a set of components that performs a function required by a system; (2) a set of connectors that enables “communication, coordination and cooperation “between components; (3) constraints that define how components can be integrated to form the system, and (4) semantic models that enable a designer to understand the overall properties of a system by analyzing the unknown properties of its constituent parts.

a) Data-centered architecture. A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store. Figure 4 .1 illustrates a typical data- centered style. Client software accesses a central

repository.

b) Data-flow architecture. The architecture is applied when input data are to be transformed through a series of computational or manipulate components into output data. A pipe and filter structure in figure 4.2 has a set of components, called filters, connected by pips that transmit data from one component to accept data input of a certain form , and produces data output (to the next filters) of a specified form. However, the filter does not require knowledge of the workings of its neighboring filters.

c) Call and return architecture. This architectural style enables a software designer (system architect) to achieve a program structure that is relatively easy to modify and scale. Two sub styles within category:

• Main programs/ subprograms architecture. This classic program structure decomposes functions into a control hierarchy where a “main” program invokes a number of program components, which in turn may invoke still other components.
• Remote procedure called architecture. The components of a main program/sub-program architecture are distributed across multiple computers on a network.

d) 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.

e) Architectural Design. As architectural design begins, the software to be developed must be put into context that is, the design should define the external entities that the software interacts with and the nature of the interactions. Once context is modeled and all external software interfaces have been described, the designer specifies the structure of the system by defining and refining software components that implement the architecture. This process continues iteratively until complete architectural structure has been derived.

f) Defining Archetypes. An archetypes is a class or pattern that represents a core abstraction that is critical to the design of architecture for the target system.

Archetypes can be derived by examining the analysis model. Continuing our discussion of the safehome home security function, we might define the following archetypes.

• Node. Represents a cohesive collection of input and output elements of the home security function. For example a node might be comprised of (1) various sensors and (2) a variety of alarm (output) indications.
• Detector. An abstraction that represents all mechanism s (e.g., alarm siren. Flashing lights, bell) for indicating that an alarm condition is occurring.
• Controller. An abstraction that depicts the mechanism that allows the aiming or disarming of a node. If controllers reside on a network, they have the ability to communicate with one another.

g) Refining the Architecture into components. As the software architecture is refined in to components, the structure of the system begins to emerge. These analysis classes represents entities within the application domain is one source for the derivation and refinement of components. Another source is the infrastructure domain. The architecture must accommodate many infrastructure components that enable application components.

Continuing the safehome home security function example, we might define the set of top-level components that address the following functionally:

• External communication management—coordinates communication of the security function with external entities, for example, Internal based system, external alarm notification.
• Control panel processing- manages all control panel functionally.
• Detector management- coordinates access to all detectors attached to the system.
• Alarm processing-verifies and acts on all alarm conditions.

The design details of all attributes and operations would not be specified until components- level design, is generated.

Data Design at the Component Level

Data design at the component level focuses on the representation of data structures that are directly accessed by one or more software components. Wasserman has proposed a set of principles that may be used to specify and design such data structures. In actuality, the design of data begins during the creation of the analysis model. Recalling that requirements analysis and design often overlap, we consider the following set of principles for data specification:

1. The systematic analysis principles applied to function and behavior should also be applied to data. We spend much time and effort deriving, reviewing, and specifying functional requirements and preliminary design. Representations of data flow and content should also be developed and reviewed, data objects should be identified, alternative data organizations should be considered, and the impact of data modeling on software design should be evaluated. For example, specification of a multiringed linked list may nicely satisfy data requirements but lead to an unwieldy software design. An alternative data organization may lead to better results.

2. All data structures and the operations to be performed on each should be identified. The design of an efficient data structure must take the operations to be performed on the data structure into account . For example, consider a data structure made up of a set of diverse data elements. The data structure is to be manipulated in a number of major software functions. Upon evaluation of the operations performed on the data structure, an abstract data type is defined for use in subsequent software design. Specification of the abstract data type may simplify software design considerably.

3. A data dictionary should be established and used to define both data and

program design. The concept of a data dictionary has been introduced in . A data dictionary explicitly represents the relationships among data objects and the constraints on the elements of a data structure. Algorithms that must take advantage of specific relationships can be more easily defined if a dictionarylike data specification exists.

4. Low-level data design decisions should be deferred until late in the design process. A process of stepwise refinement may be used for the design of data. That is, overall data organization may be defined during requirements analysis, refined during data design work, and specified in detail during componentlevel design. The top-down approach to data design provides benefits that are analogous to a top-down approach to software design—major structural attributes are designed and evaluated first so that the architecture of the data may be established.

5. The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure. The concept of information hiding and the related concept of coupling provide important insight into the quality of a software design. This principle alludes to the importance of these concepts as well as "the importance of separating the logical view of a data object from its physical view".

6. A library of useful data structures and the operations that may be applied to them should be developed. Data structures and operations should be viewed as a resource for software design. Data structures can be designed for reusability. A library of data structure templates (abstract data types) can reduce both specification and design effort for data.

7. A software design and programming language should support the specification and realization of abstract data types. The implementation of a sophisticated data

structure can be made exceedingly difficult if no means for direct specification of the structure exists in the programming language chosen for implementation.

These principles form a basis for a component-level data design approach that can be integrated into both the analysis and design activities.

Architectural Design

Objective: Architectural design tools model the overall software structure by representing component interfaces, dependencies and relationships, and interactions.

Mechanics: Tools mechanics vary. In most cases, architectural design capability is part of the functionality provided by automated tools for analysis and design modeling.

Representative Tools:
Adalon, developed by Synthis Corp. (www.synthis.com), is a specialized design tool for the design and construction of specific Web-based component architectures.

ObjectiF, developed mico Tools GmbH (www.microtool.com), is a UML-based design tool that leads to architectures (e.g., Coldfusion, j2EE, Fusebox) amenable to component- based software engineering.

Rational Rose, developed by Rational (www.ra is a UML- based design tool that support all aspects of architectural design).

Pattern–Based Software Design:
The best designers in any field have an uncanny ability to see patterns that characterize a problem and corresponding patterns that can be combined to create a solution. Throughout the design process, a software engineer should look for every opportunity to reuse existing design patterns (when they meet the

needs of the design) rather than creating new ones.

Describing a Design Pattern

Mature engineering disciplines make use of thousands of design patterns, For example, a mechanical engineer uses a two- step, keyed shaft as a design pattern. Inherent in the pattern are attributes (the diameters of the shaft, the dimensions of the keyway, etc.) and operations (e.g., shaft rotation, shaft connection). An electrical engineer uses an integrated circuit (an extremely complex design pattern) to solve a specific element of a new problem. Design patterns may be described using the template shown in the sidebar.

Design Pattern Template:

Pattern name describes the essence of the pattern in a short but expressive name.

Intent - describes the pattern and what it does.

Also - known - as - lists any synonyms for the pattern.

Motivation - provides an example of the problem.

Applicability – notes specific design situations in which the pattern is applicable.

Structure - describes the responsibilities of the classes that are required to implement the pattern.

Participants - describes the responsibilities of the classes that are required to implement the pattern.

Collaborations - describes the responsibilities of the classes that are required to implement the pattern.

Collaborations – describes how the participants collaborate to carry out their responsibilities.

Consequences - describes the "design forces" that affect the pattern and the potential trade- offs that must be considered when the pattern is implemented.

Related Patterns - cross- references related design patterns.

A description of the design pattern may also consider a set of design forces. Design forces describe non- functional requirements (e.g., ease of maintainability, portability) associated with the software for which the pattern is to applied. In addition forces define the constraints that may restrict the manner in which the design is to be implemented. In essence, design forces describe the environment and conditions that must exist to make the design pattern applicable. The pattern characteristics (classes, responsibilities, and collaborations) indicate the attributes of the design that may be adjusted to enable the pattern to accommodate a variety of problems. These attributes represent characteristics of the design that can be searched (e.g., via a database) so that an appropriate pattern can be found. Finally, guidance associated with the use of a design pattern provides an indication of the ramifications of design decisions.

Martin Fowler "patterns are half- baked—meaning you always have to finish them yourself and adapt them to your own environment."

The names of design patterns should be chosen with care. One of the key technical problems in software reuse is the inability to find existing reusable patterns when hundreds or thousands of candidate pattern exist. The search for the "right" pattern is aided immeasurably by a meaningful pattern name.

Using Patterns in design

Design patterns can be used throughout software design. Once the analysis model has been developed, the designer can examine a detailed representation of the problem to be solved and the constraints that are imposed by the problem. The

problem description is examined at various levels of abstraction to determine if it is amenable to one or more of the following types of design patterns:

Architectural patterns: These patterns define the overall structure of the software and indicate the relationship among subsystems and software components, and ages, components, subsystems) of the architecture.

Design patterns: These patterns address a specific element of the design such as an aggregation of components to solve some design problem, among components, or the mechanisms of effecting components-to-components communications.

Idioms: Sometimes called coding patterns, these language – specific patterns generally implement an algorithmic element of a component, a specific protocol, or a mechanism for communication among components.

Each of these pattern types differs in the level of abstraction with which it is represented and the degree to which it provides direct guidance for the construction activity (in this case, coding) of the software process.

Frameworks:

In some cases it may be necessary to provide an implementation- specific skeletal infrastructure, called a framework, for design work. That is, the designer may select a “reusable mini- architecture that provides the generic structure and behavior for a family of software abstractions, along with a context… which specifies their collaboration and use within domain.”

A framework is not an architectural pattern, but rather a skeleton with a collection with a collection of “plug points” (also called hooks and slots) that enables it to be adapted to specific classes or functionality within the skeleton. In an object- oriented context, a framework is a collection of cooperating classes.

In essence, the designer of a framework will argue that one reusable mini – architecture is applicable to all software to be developed within a limited domain of application. To be most effective, frameworks are applied with no changes. Additional design elements may be added, but only via the plug points that allow the designer to flush out the framework skeleton.

Design engineering commences, as the first iteration of requirements engineering comes to a conclusion. The intent of software design is to apply a set of principles, concepts, and practices that lead to the development of a high- quality system or product. The goal design is to create a model of software that will implement all customer requirements correctly and bring delight to those who use it. Design engineers must shift through many design alternatives and coverage on a solution that best suits the needs of project stakeholders.

The design process moves from a “big picture” view of software to a more narrow view that defines the details required to implement a system. The process begins by focusing on architecture. Subsystems are defined; communication mechanisms among subsystems are established; components are identified; and a detailed description of each component is developed. In addition, external, internal, and user interfaces are designed.

Design concepts have evolved over the first half-century of software engineering work. They describe attributes of computer software that should be present regardless of the software engineering process that is chosen, the design methods that are applied, or the programming languages that are used.

The design model encompasses four different elements. As each of these elements is developed, a more complete view of the design evolves. The architectural elements used information derived from the application domain, the analysis model, and available catalogs for patterns and styles to derive a complete structural representation of the software, its subsystems and components. Interface design elements model external and internal interfaces and the user interface. Component- level elements define each of the modules (components)

that populates the architecture. Finally, deployment- level design elements allocate the architecture, its components, and the interfaces to the physical configuration that will house the software.

Pattern- based design is a technique that ruses design elements that have proven successful in the past. Each architectural pattern, design pattern, or idiom is cataloged, thoroughly documented, and carefully considered as it assessed for inclusion in a specific application. Frameworks, an extension of patterns, provide an architectural skeleton for the design of complete subsystems within a specific application domain.

Component-Level Design

(i) Class Base Component

In the context of object- oriented software engineering, a component contains a set of collaborating classes. Each class within a component has been fully elaborated to include all attributes and operations that are relevant to its implementation. As part of the design elaboration. As part of the design elaboration, all interfaces (messages) that enable the classes to communicate and collaborate with other design classes must also be defined. To accomplish this, the designer begins with analysis model and elaborate analysis classes (for components that relate to the problem domain) and infrastructure classes (for components that provide support services for the problem domain).

(ii) Component – Level Design Guidelines

The guidelines are:

Components

Naming conventions should be established for components that are specified as part of the architectural model and then refined and elaborated as part of the component- level model. Architectural component names should be drawn from the problem domain and should have meaning to all stakeholders who view the architectural model.

Interfaces

Interfaces provide important information about communication and collaboration. Ambler recommends that (1) lollipop representation of an interface should be used in lieu of the more formal UML box and dashed arrow approach, when diagram grows complex; (2) for consistency, interfaces should flow from the left-hand side of the component box; (3) only those interfaces that are relevant to the component under consideration should be shown, even if other interfaces are available. These recommendations are intended to simplify the visual nature of UML component diagram.

Dependencies and inheritance.

For improved readability, it is a good idea to model dependencies from left to right and inheritance from bottom (derived classes) to top (base classes). In addition, component- to component dependency.

(iii) Conducting Component-Level Design

The designer must transform information from the analysis and architectural models into a design representation that provides sufficient detail to guide thee construction (coding and testing) activity. The following steps represent a typical task set for component-level design, when it is applied for an object –oriented system.

Step 1. Identity all design classes that correspond to the problem domain.
Using the analysis and architectural models, each analysis class and architectural component are elaborated.

Step 2. Identify all design classes that correspond to the infrastructure domain.

Step 3. Elaborate all design classes that are not acquired as reusable components. Elaboration requires that all interfaces, attributes, and operations necessary to implement the class be described in detail.

a. Specify message details when classes or components collaborate.
The analysis model makes use of a collaboration diagram to show how analysis classes collaborate with one another. As component- level design proceeds, it is sometimes useful to show the details of these collaborations by specifying the structure of message that are passed between objects within a system. Although this design is optional, it can be used as a precursor to thee specification of interfaces that show how components within the system communicate and collaborate.

b. Identify appropriate interfaces for each component.
Within the context of component- level design, a UML interface is "a group of externally visible operations."

c. Elaborate attributes and define data types and data structures required to implement them.
Data structures and types used to describe attributes are defined within the context of the programming language that is to be used.

d. Describe proceeding flow within each operation in detail.
This may be accomplished using a programming language- based pseudo code or with a UML activity diagram each software component is elaborated through a number of iterations that apply the stepwise refinement concept.

Step 4. Develop and elaborate behavioral representations for a class or component.
UML state diagrams were used as part of the analysis model to resent the externally observable behavior of the system and more localized behavior of individual analysis classes. During component- level- design, it is sometimes necessary to model the behavior of a design class.

The dynamic behavior of an object is affected by events that are external to it and the current state of the object. To understand the dynamic behavior of an object,

the designer must examine all use cases that are relevant to the design class throughout its life. These use – cases provide information that helps the designer to delineate the event that affects the object and the stats in which the object resides as time passes and events occur.

Step 6. Elaborate deployment diagrams to provide additional detail.
During component—level design, deployment diagram can be elaborated to represent the location of key packages of components. However, components generally are not represented individually within a component diagram. The reason for this is to avoid diagrammatic complexity.

Step 7. Factor every component- level design representation and always consider alternatives.

User interface Design

User interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, ships etc.

User interface is part of software and is designed such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction.

UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time

- Clear to understand
- Consistent on all interfacing screens

UI is broadly divided into two categories:

- Command Line Interface
- Graphical User Interface

Command Line Interface (CLI)

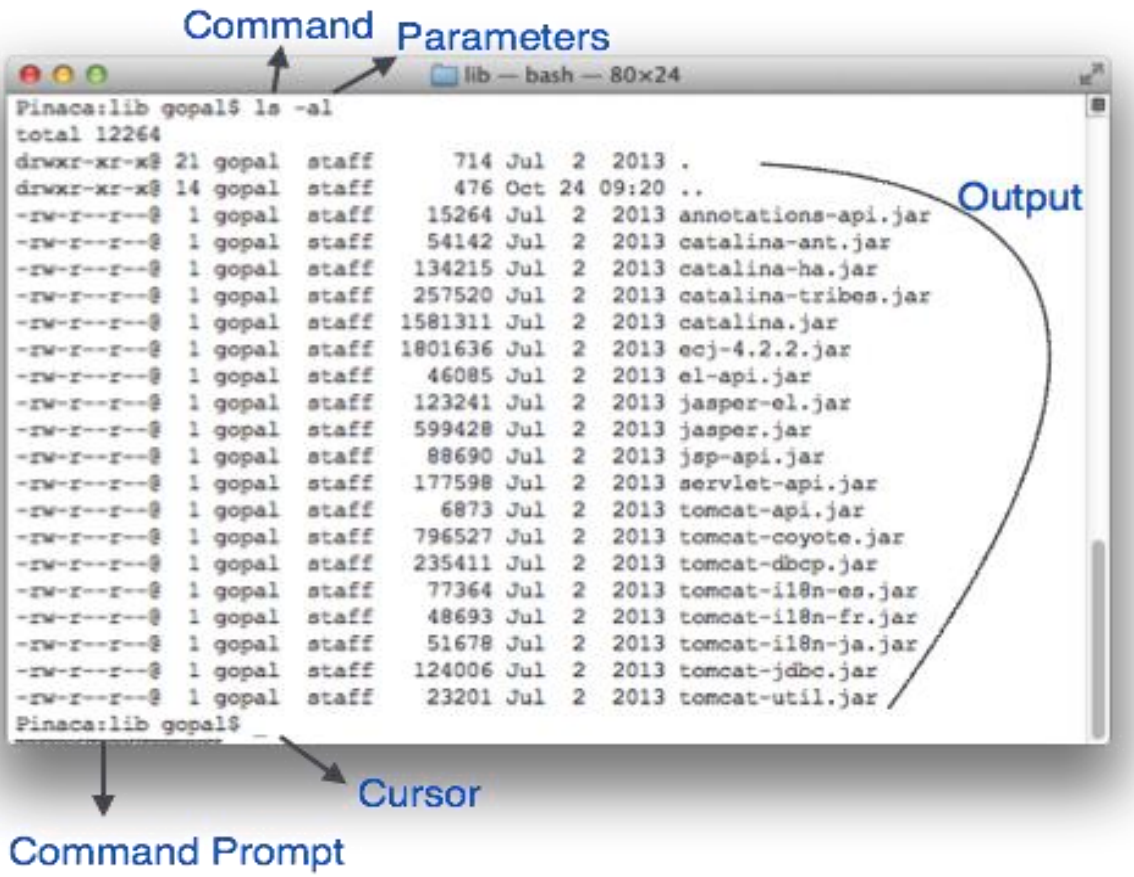
CLI has been a great tool of interaction with computers until the video display monitors came into existence. CLI is first choice of many technical users and programmers. CLI is minimum interface a software can provide to its users.

CLI provides a command prompt, the place where the user types the command and feeds to the system. The user needs to remember the syntax of command and its use. Earlier CLI were not programmed to handle the user errors effectively.

A command is a text-based reference to set of instructions, which are expected to be executed by the system. There are methods like macros, scripts that make it easy for the user to operate.

CLI uses less amount of computer resource as compared to GUI.

CLI Elements



A text-based command line interface can have the following elements:

- Command Prompt - It is text-based notifier that is mostly shows the context in which the user is working. It is generated by the software system.
- Cursor - It is a small horizontal line or a vertical bar of the height of line, to represent position of character while typing. Cursor is mostly found in blinking state. It moves as the user writes or deletes something.
- Command - A command is an executable instruction. It may have one or more parameters. Output on command execution is shown inline on the screen. When output is produced, command prompt is displayed on the next line.

Graphical User Interface

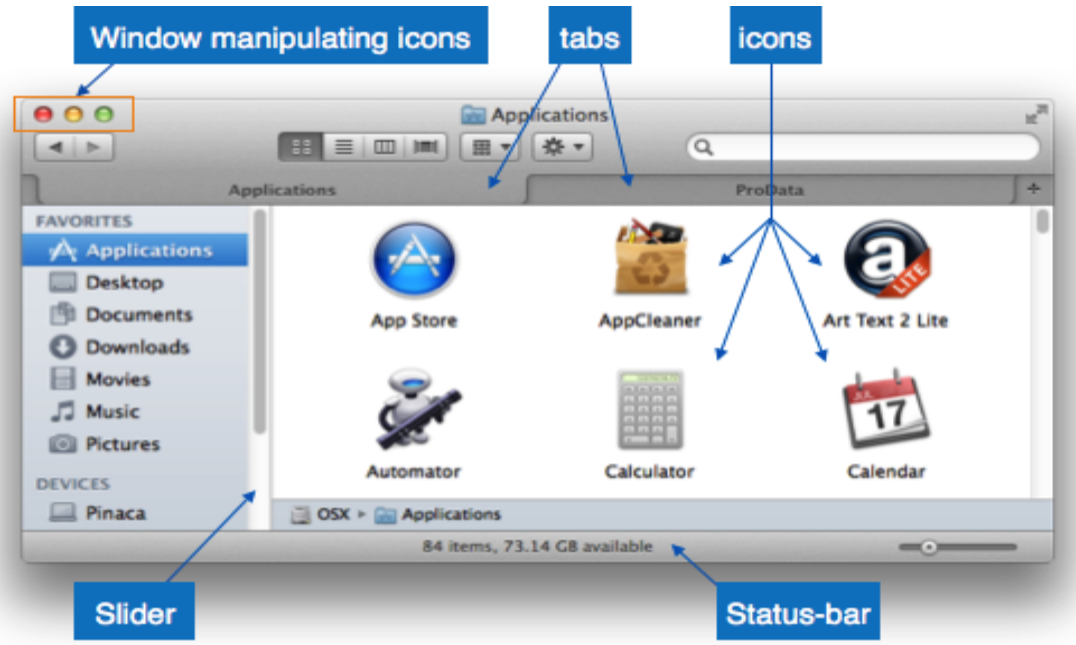
Graphical User Interface provides the user graphical means to interact with the system. GUI can be combination of both hardware and software. Using GUI, user interprets the software.

Typically, GUI is more resource consuming than that of CLI. With advancing technology, the programmers and designers create complex GUI designs that work with more efficiency, accuracy and speed.

GUI Elements

GUI provides a set of components to interact with software or hardware.

Every graphical component provides a way to work with the system. A GUI system has following elements such as:



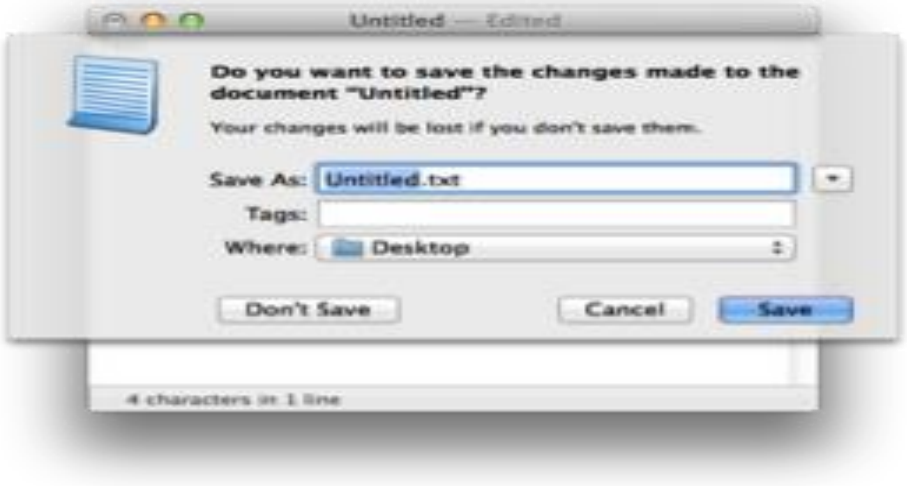
- Window - An area where contents of application are displayed. Contents in a window can be displayed in the form of icons or lists, if the window represents file structure. It is easier for a user to navigate in the file system in an exploring window. Windows can be minimized, resized or maximized to the size of screen. They can be moved anywhere on the screen. A window may contain another window of the same application, called child window.

- Tabs - If an application allows executing multiple instances of itself, they appear on the screen as separate windows. Tabbed Document Interface has come up to open multiple documents in the same window. This interface also helps in viewing preference panel in application. All modern web-browsers use this feature.
- Menu - Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.
- Icon - An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small pictures.
- Cursor - Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursors. On screen cursor follows the instructions from hardware in almost real-time. Cursors are also named pointers in GUI systems. They are used to select menus, windows and other application features.

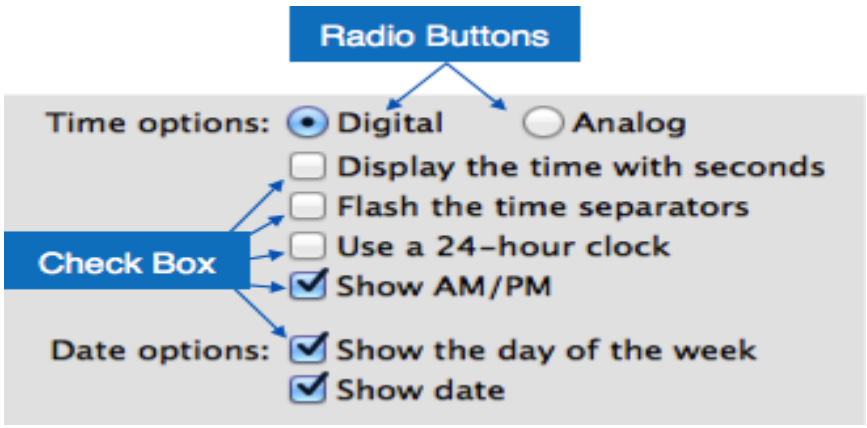
Application specific GUI components

A GUI of an application contains one or more of the listed GUI elements:

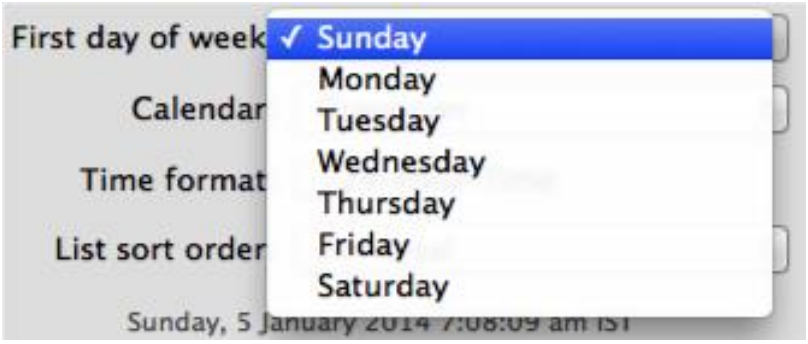
- Application Window - Most application windows uses the constructs supplied by operating systems but many use their own customer created windows to contain the contents of application.
- Dialogue Box - It is a child window that contains message for the user and request for some action to be taken. For Example: Application generate a dialogue to get confirmation from user to delete a file.



- Text-Box - Provides an area for user to type and enter text-based data.
- Buttons - They imitate real life buttons and are used to submit inputs to the software.



- Radio-button - Displays available options for selection. Only one can be selected among all offered.
- Check-box - Functions similar to list-box. When an option is selected, the box is marked as checked. Multiple options represented by check boxes can be selected.
- List-box - Provides list of available items for selection. More than one item can be selected.



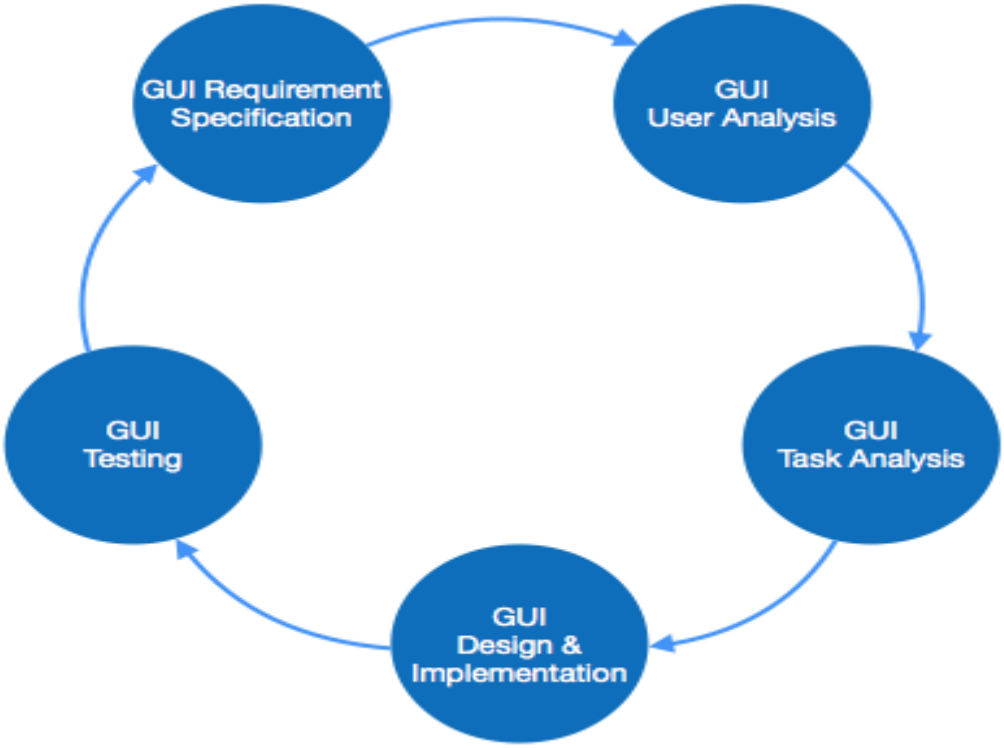
Other impressive GUI components are:

- Sliders
- Combo-box
- Data-grid
- Drop-down list

User Interface Design Activities

There are a number of activities performed for designing user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation among Waterfall, Iterative or Spiral Model.

A model used for GUI design and development should fulfill these GUI specific steps.



- GUI Requirement Gathering - The designers may like to have list of all functional and non-functional requirements of GUI. This can be taken from user and their existing software solution.
- User Analysis - The designer studies who is going to use the software GUI. The target audience matters as the design details change according to the knowledge and competency level of the user. If user is technical savvy, advanced and complex GUI can be incorporated. For a novice user, more information is included on how-to of software.
- Task Analysis - Designers have to analyze what task is to be done by the software solution. Here in GUI, it does not matter how it will be done. Tasks can be represented in hierarchical manner taking one major task and dividing it further into smaller sub-tasks. Tasks provide goals for GUI presentation. Flow of information among sub-tasks determines the flow of GUI contents in the software.
- GUI Design & implementation - Designers after having information about requirements, tasks and user environment, design the GUI and implements

into code and embed the GUI with working or dummy software in the background. It is then self-tested by the developers.

- Testing - GUI testing can be done in various ways. Organization can have in-house inspection, direct involvement of users and release of beta version are few of them. Testing may include usability, compatibility, user acceptance etc.

GUI Implementation Tools

There are several tools available using which the designers can create entire GUI on a mouse click. Some tools can be embedded into the software environment (IDE).

GUI implementation tools provide powerful array of GUI controls. For software customization, designers can change the code accordingly.

There are different segments of GUI tools according to their different use and platform.

Example

Mobile GUI, Computer GUI, Touch-Screen GUI etc. Here is a list of few tools which come handy to build GUI:

- FLUID
- AppInventor (Android)
- LucidChart
- Wavemaker
- Visual Studio

User Interface Golden rules

The following rules are mentioned to be the golden rules for GUI design, described by Shneiderman and Plaisant in their book (Designing the User Interface).

- Strive for consistency - Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employed throughout.
- Enable frequent users to use short-cuts - The user's desire to reduce the number of interactions increases with the frequency of use. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.
- Offer informative feedback - For every operator action, there should be some system feedback. For frequent and minor actions, the response must be modest, while for infrequent and major actions, the response must be more substantial.
- Design dialog to yield closure - Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and this indicates that the way ahead is clear to prepare for the next group of actions.
- Offer simple error handling - As much as possible, design the system so the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple, comprehensible mechanisms for handling the error.
- Permit easy reversal of actions - This feature relieves anxiety, since the user knows that errors can be undone. Easy reversal of actions encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.

- Support internal locus of control - Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.
- Reduce short-term memory load - The limitation of human information processing in short-term memory requires the displays to be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions

Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Software Quality Management System

A quality management system is the principal methods used by organizations to provide that the products they develop have the desired quality.

A quality system subsists of the following:

Managerial Structure and Individual Responsibilities: A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

Quality System Activities: The quality system activities encompass the following:

Auditing of projects

Review of the quality system

Development of standards, methods, and guidelines, etc.

Production of documents for the top management summarizing the effectiveness of the quality system in the organization.

Evolution of Quality Management System

Quality systems have increasingly evolved over the last five decades. Before World War II, the usual function to produce quality products was to inspect the

finished products to remove defective devices. Since that time, quality systems of organizations have undergone through four steps of evolution, as shown in the fig. The first product inspection task gave method to quality control (QC).

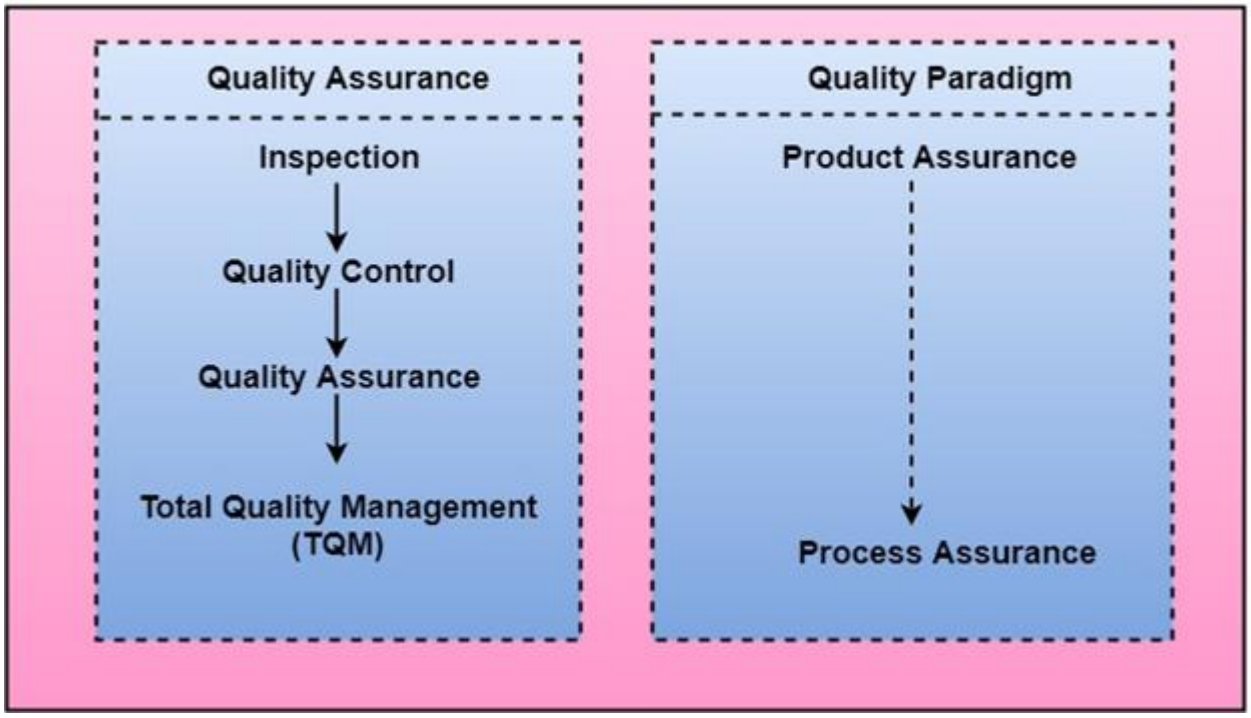
Quality control target not only on detecting the defective devices and removes them but also on determining the causes behind the defects. Thus, quality control aims at correcting the reasons for bugs and not just rejecting the products. The next breakthrough in quality methods was the development of quality assurance methods.

The primary premise of modern quality assurance is that if an organization's processes are proper and are followed rigorously, then the products are obligated to be of good quality. The new quality functions include guidance for recognizing, defining, analyzing, and improving the production process.

Total quality management (TQM) advocates that the procedure followed by an organization must be continuously improved through process measurements. TQM goes stages further than quality assurance and aims at frequently process improvement. TQM goes beyond documenting steps to optimizing them through a redesign. A term linked to TQM is Business Process Reengineering (BPR).

BPR aims at reengineering the method business is carried out in an organization. From the above conversation, it can be stated that over the years, the quality paradigm has changed from product assurance to process assurance, as shown in fig.

Evolution of quality system and corresponding shift in the quality paradigm



Software Quality Factors

The various factors, which influence the software, are termed as software factors. They can be broadly divided into two categories. The first category of the factors is of those that can be measured directly such as the number of logical errors, and the second category clubs those factors which can be measured only indirectly. For example, maintainability but each of the factors is to be measured to check for the content and the quality control.

Several models of software quality factors and their categorization have been suggested over the years. The classic model of software quality factors, suggested by McCall, consists of 11 factors (McCall et al., 1977). Similarly, models consisting of 12 to 15 factors, were suggested by Deutsch and Willis (1988) and by Evans and Marciniak (1987).

All these models do not differ substantially from McCall's model. The McCall factor model provides a practical, up-to-date method for classifying software requirements (Pressman, 2000).

McCall's Factor Model

This model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories – product operation, product revision, and product transition factors.

- **Product operation factors** − Correctness, Reliability, Efficiency, Integrity, Usability.
- **Product revision factors** − Maintainability, Flexibility, Testability.
- **Product transition factors** − Portability, Reusability, Interoperability.

Product Operation Software Quality Factors

According to McCall's model, product operation category includes five software quality factors, which deal with the requirements that directly affect the daily operation of the software. They are as follows −

Correctness

These requirements deal with the correctness of the output of the software system. They include −

- Output mission
- The required accuracy of output that can be negatively affected by inaccurate data or inaccurate calculations.
- The completeness of the output information, which can be affected by incomplete data.
- The up-to-dateness of the information defined as the time between the event and the response by the software system.
- The availability of the information.
- The standards for coding and documenting the software system.

Reliability

Reliability requirements deal with service failure. They determine the maximum allowed failure rate of the software system, and can refer to the entire system or to one or more of its separate functions.

Efficiency

It deals with the hardware resources needed to perform the different functions of the software system. It includes processing capabilities (given in MHz), its storage capacity (given in MB or GB) and the data communication capability (given in MBPS or GBPS).

It also deals with the time between recharging of the system's portable units, such as, information system units located in portable computers, or meteorological units placed outdoors.

Integrity

This factor deals with the software system security, that is, to prevent access to unauthorized persons, also to distinguish between the group of people to be given read as well as write permit.

Usability

Usability requirements deal with the staff resources needed to train a new employee and to operate the software system.

Product Revision Quality Factors

According to McCall's model, three software quality factors are included in the product revision category. These factors are as follows −

Maintainability

This factor considers the efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections.

Flexibility

This factor deals with the capabilities and efforts required to support adaptive maintenance activities of the software. These include adapting the current software to additional circumstances and customers without changing the software. This factor's requirements also support perfective maintenance activities, such as changes and additions to the software in order to improve its service and to adapt it to changes in the firm's technical or commercial environment.

Testability

Testability requirements deal with the testing of the software system as well as with its operation. It includes predefined intermediate results, log files, and also the automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the system are in working order and to obtain a report about the detected faults. Another type of these requirements deals with automatic diagnostic checks applied by the maintenance technicians to detect the causes of software failures.

Product Transition Software Quality Factor

According to McCall's model, three software quality factors are included in the product transition category that deals with the adaptation of software to other environments and its interaction with other software systems. These factors are as follows −

Portability

Portability requirements tend to the adaptation of a software system to other environments consisting of different hardware, different operating systems, and so forth. The software should be possible to continue using the same basic software in diverse situations.

Reusability

This factor deals with the use of software modules originally designed for one project in a new software project currently being developed. They may also enable future projects to make use of a given module or a group of modules of

the currently developed software. The reuse of software is expected to save development resources, shorten the development period, and provide higher quality modules.

Interoperability

Interoperability requirements focus on creating interfaces with other software systems or with other equipment firmware. For example, the firmware of the production machinery and testing equipment interfaces with the production control software.

ISO/IEC 9126

This standard deals with the following aspects to determine the quality of a software application −

- Quality model
- External metrics
- Internal metrics
- Quality in use metrics

This standard presents some set of quality attributes for any software such as −

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

ISO 9126 is an international standard for the evaluation of software. The standard is divided into four parts which addresses, respectively, the following subjects:

quality model; external metrics; internal metrics; and quality in use metrics. ISO 9126 Part one, referred to as ISO 9126-1 is an extension of previous work done by McCall (1977), Boehm (1978), FURPS and others in defining a set of software quality characteristics.

ISO9126-1 represents the latest (and ongoing) research into characterizing software for the purposes of software quality control, software quality assurance and software process improvement (SPI). This article defines the characteristics identified by ISO 9126-1. The other parts of ISO 9126, concerning metrics or measurements for these characteristics, are essential for SQC, SQA and SPI but the main concern of this article is the definition of the basic ISO 9126 Quality Model.

The ISO 9126 documentation itself, from the official ISO 9126 documentation, can only be purchased and is subject to copyright. SQA.net only reproduces the basic structure of the ISO 9126 standard and any descriptions, commentary or guidance are original material based on public domain information as well as our own experience.

The ISO 9126-1 software quality model identifies **6 main quality characteristics**, namely:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

These characteristics are broken down into subcharacteristics, a high level table is shown below. It is at the subcharacteristic level that measurement for SPI will

occur. The main characteristics of the ISO9126-1 quality model, can be defined as follows:-

Functionality

Functionality is the essential purpose of any product or service. For certain items this is relatively easy to define, for example a ship's anchor has the function of holding a ship at a given location. The more functions a product has, e.g. an ATM machine, then the more complicated it becomes to define it's functionality. For software a list of functions can be specified, i.e. a sales order processing systems should be able to record customer information so that it can be used to reference a sales order. A sales order system should also provide the following functions:

- Record sales order product, price and quantity.
- Calculate total price.
- Calculate appropriate sales tax.
- Calculate date available to ship, based on inventory.
- Generate purchase orders when stock falls below a given threshold.

The list goes on and on but the main point to note is that functionality is expressed as a totality of essential functions that the software product provides. It is also important to note that the presence or absence of these functions in a software product can be verified as either existing or not, in that it is a Boolean (either a yes or no answer). The other software characteristics listed (i.e. usability) are only present to some degree, i.e. not a simple on or off. Many people get confused between overall process functionality (in which software plays a part) and software functionality. This is partly due to the fact that Data Flow Diagrams (DFDs) and other modeling tools can depict process functionality (as a set of data in\data out conversions) and software functionality. Consider a sales order process, that has both manual and software components. A function of the sales order process could be to record the sales order but we could implement a hard copy filing cabinet for the actual orders and only use software for calculating the price, tax and ship date. In this way the functionality of the software is limited to

those calculation functions. SPI, or Software Process Improvement is different from overall Process Improvement or Process Re-engineering, ISO 9126-1 and other software quality models do not help measure overall Process costs\benefits but only the software component. The relationship between software functionality within an overall business process is outside the scope of ISO 9126 and it is only the software functionality, or essential purpose of the software component, that is of interest for ISO 9126.

Following functionality, there are 5 other software attributes that characterize the usefulness of the software in a given environment.

Each of the following characteristics can only be measured (and are assumed to exist) when the functionality of a given system is present. In this way, for example, a system can not possess usability characteristics if the system does not function correctly (the two just don't go together).

Reliability

Once a software system is functioning, as specified, and delivered the reliability characteristic defines the capability of the system to maintain its service provision under defined conditions for defined periods of time. One aspect of this characteristic is fault tolerance that is the ability of a system to withstand component failure. For example if the network goes down for 20 seconds then comes back the system should be able to recover and continue functioning.

Usability

Usability only exists with regard to functionality and refers to the ease of use for a given function. For example a function of an ATM machine is to dispense cash as requested. Placing common amounts on the screen for selection, i.e. $20.00, $40.00, $100.00 etc, does not impact the function of the ATM but addresses the Usability of the function. The ability to learn how to use a system (learnability) is also a major subcharacteristic of usability.

Efficiency

This characteristic is concerned with the system resources used when providing

the required functionality. The amount of disk space, memory, network etc. provides a good indication of this characteristic. As with a number of these characteristics, there are overlaps. For example the usability of a system is influenced by the system's Performance, in that if a system takes 3 hours to respond the system would not be easy to use although the essential issue is a performance or efficiency characteristic.

Maintainability

The ability to identify and fix a fault within a software component is what the maintainability characteristic addresses. In other software quality models this characteristic is referenced as supportability. Maintainability is impacted by code readability or complexity as well as modularization. Anything that helps with identifying the cause of a fault and then fixing the fault is the concern of maintainability. Also the ability to verify (or test) a system, i.e. testability, is one of the subcharacteristics of maintainability.

Portability

This characteristic refers to how well the software can adopt to changes in its environment or with its requirements. The subcharacteristics of this characteristic include adaptability. Object oriented design and implementation practices can contribute to the extent to which this characteristic is present in a given system.

The full table of Characteristics and Subcharacteristics for the ISO 9126-1 Quality Model is:-

Characteristics	Subcharacteristics	Definitions
	Suitability	This is the essential Functionality characteristic and refers to the appropriateness (to specification) of the functions of the software.
	Accurateness	This refers to the correctness of the

		functions, an ATM may provide a cash dispensing function but is the amount correct?
Functionality	Interoperability	A given software component or system does not typically function in isolation. This subcharacteristic concerns the ability of a software component to interact with other components or systems.
	Compliance	Where appropriate certain industry (or government) laws and guidelines need to be complied with, i.e. SOX. This subcharacteristic addresses the compliant capability of software.
	Security	This subcharacteristic relates to unauthorized access to the software functions.
Reliability	Maturity	This subcharacteristic concerns frequency of failure of the software.
	Fault tolerance	The ability of software to withstand (and recover) from component, or environmental, failure.
	Recoverability	Ability to bring back a failed system to full operation, including data and network connections.

Usability	Understandability	Determines the ease of which the systems functions can be understood, relates to user mental models in Human Computer Interaction methods.
	Learnability	Learning effort for different users, i.e. novice, expert, casual etc.
	Operability	Ability of the software to be easily operated by a given user in a given environment.
Efficiency	Time behavior	Characterizes response times for a given thru put, i.e. transaction rate.
	Resource behavior	Characterizes resources used, i.e. memory, cpu, disk and network usage.
Maintainability	Analyzability	Characterizes the ability to identify the root cause of a failure within the software.
	Changeability	Characterizes the amount of effort to change a system.
	Stability	Characterizes the sensitivity to change of a given system that is the negative impact that may be caused by system changes.
	Testability	Characterizes the effort needed to verify (test) a system change.

	Adaptability	Characterizes the ability of the system to change to new specifications or operating environments.
Portability	Installability	Characterizes the effort required to install the software.
	Conformance	Similar to compliance for functionality, but this characteristic relates to portability. One example would be Open SQL conformance which relates to portability of database used.
	Replaceability	Characterizes the plug and play aspect of software components, that is how easy is it to exchange a given software component within a specified environment.

ISO 9126 Observations

For the most part, the overall structure of ISO9126-1 is similar to past models, McCall (1977) and Boehm (1978), although there are a couple of notable differences. Compliance comes under the functionality characteristic, this can be attributed to government initiatives like SOX. In many requirements specifications all characteristics, that are specified, that are not pure functional requirements are specified as Non-Functional requirements. It is interesting to note, with ISO9126, that compliance is seen as a functional characteristic.

Using the ISO 9126 (or any other quality model) for derivation of system requirements brings clarity of definition of purpose and operating capability . For example a rules engine approach to compliance would enable greater adaptability, should the compliance rules change. The functionality for compliance could be implemented in other ways but these other implementation methods may not produce as strong an adaptability characteristic as a rules, or some other component based, architecture.

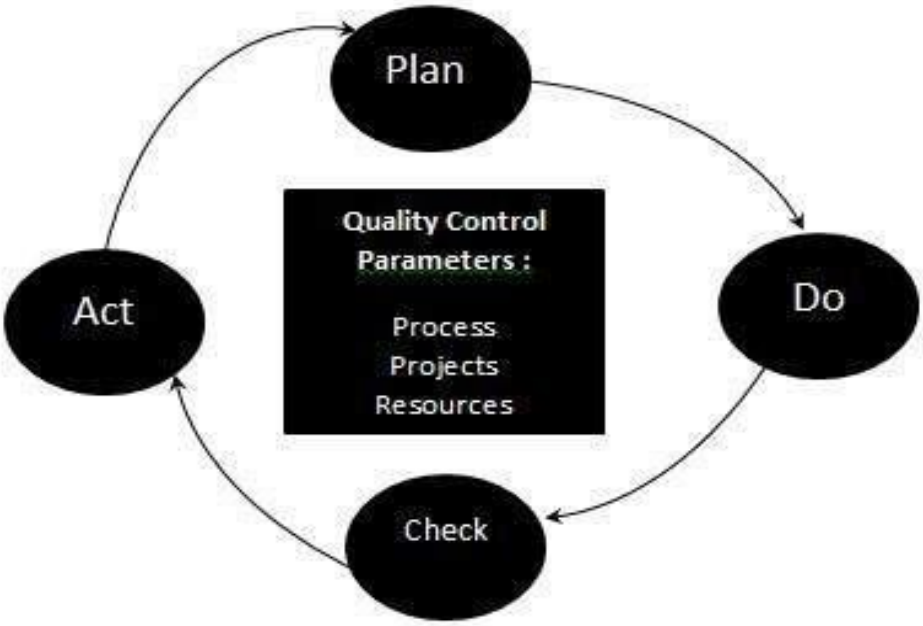
Also, a designer typically will need to make trade offs between two or more characteristics when designing the system. Consider highly modularized code, this code is usually easy to maintain, i.e. has a good changeability characteristic, but may not perform as well (for cpu resource, as unstructed program code). On a similar vein a normalized database may not perform as well as a not normalized database. These trade offs need to be identified, so that informed design decisions can be made.

Although ISO 9126-1 is the latest proposal for a useful Quality Model, of software characteristics, it is unlikely to be the last. One thing is certain, the requirements (including compliance) and operating environment of software will be continually changing and with this change will come the continuing search to find useful characteristics that facilitate measurement and control of the software production process.

Quality Control

Quality control is a set of methods used by organizations to achieve quality parameters or quality goals and continually improve the organization's ability to ensure that a software product will meet quality goals.

Quality Control Process:



The three class parameters that control software quality are:

- Products
- Processes
- Resources

The total quality control process consists of:

- Plan - It is the stage where the Quality control processes are planned
- Do - Use a defined parameter to develop the quality
- Check - Stage to verify if the quality of the parameters are met
- Act - Take corrective action if needed and repeat the work

Quality Control characteristics:

- Process adopted to deliver a quality product to the clients at best cost.
- Goal is to learn from other organizations so that quality would be better each time.

- To avoid making errors by proper planning and execution with correct review process.

Software Quality Assurance (SQA)

Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly.

Software Quality Assurance is a process which works parallel to development of a software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process.

Software Quality Assurance have:

1. A quality management approach
2. Formal technical reviews
3. Multi testing strategy
4. Effective software engineering technology
5. Measurement and reporting mechanism

Major Software Quality Assurance Activities:

1. **SQA Management Plan:**
   Make a plan how you will carry out the sqa through out the project. Think which set of software engineering activities are the best for project.check level of sqa team skills.
2. **Set The Check Points:**
   SQA team should set checkpoints. Evaluate the performance of the project on the basis of collected data on different check points.

3. **Multi testing Strategy:**
   Do not depend on single testing approach. When you have lot of testing approaches available use them.

4. **Measure Change Impact:**
   The changes for making the correction of an error sometimes re introduces more errors keep the measure of impact of change on project. Reset the new change to change check the compatibility of this fix with whole project.

5. **Manage Good Relations:**
   In the working environment managing the good relation with other teams involved in the project development is mandatory. Bad relation of sqa team with programmers team will impact directly and badly on project. Don't play politics.

Processes could be –

- Software Development Methodology
- Project Management
- Configuration Management
- Requirements Development/Management
- Estimation
- Software Design
- Testing, etc.

Once the processes have been defined and implemented, Quality Assurance has the following responsibilities –

- Identify the weaknesses in the processes
- Correct those weaknesses to continually improve the process

Benefits of Software Quality Assurance (SQA):

1. SQA produce high quality software.
2. High quality application saves time and cost.
3. SQA is beneficial for better reliability.
4. SQA is beneficial in the condition of no maintenance for long time.
5. High quality commercial software increase market share of company.
6. Improving the process of creating software.
7. Improves the quality of the software.

Disadvantage of SQA:

There are a number of disadvantages of quality assurance. Some of them include adding more resources, employing more workers to help maintain quality and so much more.

Components of SQA System

An SQA system always combines a wide range of SQA components. These components can be classified into the following six classes −

Pre-project components

This assures that the project commitments have been clearly defined considering the resources required, the schedule and budget; and the development and quality plans have been correctly determined.

Components of project life cycle activities assessment

The project life cycle is composed of two stages: the development life cycle stage and the operation–maintenance stage.

The development life cycle stage components detect design and programming errors. Its components are divided into the following sub-classes: Reviews, Expert opinions, and Software testing.

The SQA components used during the operation–maintenance phase include specialized maintenance components as well as development life cycle components, which are applied mainly for functionality to improve the maintenance tasks.

Components of infrastructure error prevention and improvement

The main objective of these components, which is applied throughout the entire organization, is to eliminate or at least reduce the rate of errors, based on the organization's accumulated SQA experience.

Components of software quality management

This class of components deal with several goals, such as the control of development and maintenance activities, and the introduction of early managerial support actions that mainly prevent or minimize schedule and budget failures and their outcomes.

Components of standardization, certification, and SQA system assessment

These components implement international professional and managerial standards within the organization. The main objectives of this class are utilization of international professional knowledge, improvement of coordination of the organizational quality systems with other organizations, and assessment of the achievements of quality systems according to a common scale. The various standards may be classified into two main groups: quality management standards and project process standards.

Organizing for SQA – the human components

The SQA organizational base includes managers, testing personnel, the SQA unit and the persons interested in software quality such as SQA trustees, SQA committee members, and SQA forum members. Their main objectives are to initiate and support the implementation of SQA components, detect deviations from SQA procedures and methodology, and suggest improvements.

Pre-project Software Quality Components

These components help to improve the preliminary steps taken before starting a project. It includes −

- Contract Review
- Development and Quality Plans

Contract Review

Normally, a software is developed for a contract negotiated with a customer or for an internal order to develop a firmware to be embedded within a hardware product. In all these cases, the development unit is committed to an agreed-upon functional specification, budget and schedule. Hence, contract review activities must include a detailed examination of the project proposal draft and the contract drafts.

Specifically, contract review activities include −

- Clarification of the customer's requirements
- Review of the project's schedule and resource requirement estimates
- Evaluation of the professional staff's capacity to carry out the proposed project
- Evaluation of the customer's capacity to fulfil his obligations
- Evaluation of development risks

Development and Quality Plans

After signing the software development contract with an organization or an internal department of the same organization, a development plan of the project and its integrated quality assurance activities are prepared. These plans include additional details and needed revisions based on prior plans that provided the basis for the current proposal and contract.

Most of the time, it takes several months between the tender submission and the signing of the contract. During these period, resources such as staff availability,

professional capabilities may get changed. The plans are then revised to reflect the changes that occurred in the interim.

The main issues treated in the project development plan are −

- Schedules
- Required manpower and hardware resources
- Risk evaluations
- Organizational issues: team members, subcontractors and partnerships
- Project methodology, development tools, etc.
- Software reuse plans

The main issues treated in the project's quality plan are −

- Quality goals, expressed in the appropriate measurable terms
- Criteria for starting and ending each project stage
- Lists of reviews, tests, and other scheduled verification and validation activities

Risk

"Tomorrow problems are today's risk." Hence, a clear definition of a "risk" is a problem that could cause some loss or threaten the progress of the project, but which has not happened yet.

These potential issues might harm cost, schedule or technical success of the project and the quality of our software device, or project team morale.

Risk Management is the system of identifying addressing and eliminating these problems before they can damage the project.

Different methods are required to address these two kinds of issues.

For example, staff storage, because we have not been able to select people with the right technical skills is a current problem, but the threat of our technical persons being hired away by the competition is a risk.

Risk Management

A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

1. Project risks
2. Technical risks
3. Business risks

1. Project risks: Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

2. Technical risks: Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

3. Business risks: This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

Other risk categories

1. **1. Known risks:** Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)
2. **2. Predictable risks:** Those risks that are hypothesized from previous project experience (e.g., past turnover)
3. **3. Unpredictable risks:** Those risks that can and do occur, but are extremely tough to identify in advance.

Principle of Risk Management

1. **Global Perspective:** In this, we review the bigger system description, design, and implementation. We look at the chance and the impact the risk is going to have.
2. **Take a forward-looking view:** Consider the threat which may appear in the future and create future plans for directing the next events.
3. **Open Communication:** This is to allow the free flow of communications between the client and the team members so that they have certainty about the risks.
4. **Integrated management:** In this method risk management is made an integral part of project management.
5. **Continuous process:** In this phase, the risks are tracked continuously throughout the risk management paradigm.

Risk Management Activities

Risk management consists of three main activities, as shown in fig:



Risk Assessment

The objective of risk assessment is to division the risks in the condition of their loss, causing potential. For risk assessment, first, every risk should be rated in two methods:

- The possibility of a risk coming true (denoted as r).
- The consequence of the issues relates to that risk (denoted as s).

Based on these two methods, the priority of each risk can be estimated:

p = r * s

Where p is the priority with which the risk must be controlled, r is the probability of the risk becoming true, and s is the severity of loss caused due to the risk becoming true. If all identified risks are set up, then the most likely and damaging risks can be controlled first, and more comprehensive risk abatement methods can be designed for these risks.

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

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

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

1. **Technology risks:** Risks that assume from the software or hardware technologies that are used to develop the system.
2. **People risks:** Risks that are connected with the person in the development team.
3. **Organizational risks:** Risks that assume from the organizational environment where the software is being developed.
4. **Tools risks:** Risks that assume from the software tools and other support software used to create the system.
5. **Requirement risks:** Risks that assume from the changes to the customer requirement and the process of managing the requirements change.
6. **Estimation risks:** Risks that assume from the management estimates of the resources required to build the system

2. Risk Analysis: During the risk analysis process, you have to consider every identified risk and make a perception of the probability and seriousness of that risk.

There is no simple way to do this. You have to rely on your perception and experience of previous projects and the problems that arise in them.

It is not possible to make an exact, the numerical estimate of the probability and seriousness of each risk. Instead, you should authorize the risk to one of several bands:

1. The probability of the risk might be determined as very low (0-10%), low (10-25%), moderate (25-50%), high (50-75%) or very high (+75%).
2. The effect of the risk might be determined as catastrophic (threaten the survival of the plan), serious (would cause significant delays), tolerable (delays are within allowed contingency), or insignificant.

Risk Control

It is the process of managing risks to achieve desired outcomes. After all, the identified risks of a plan are determined; the project must be made to include the most harmful and the most likely risks. Different risks need different containment methods. In fact, most risks need ingenuity on the part of the project manager in tackling the risk.

There are three main methods to plan for risk management:

1. **Avoid the risk:** This may take several ways such as discussing with the client to change the requirements to decrease the scope of the work, giving incentives to the engineers to avoid the risk of human resources turnover, etc.
2. **Transfer the risk:** This method involves getting the risky element developed by a third party, buying insurance cover, etc.
3. **Risk reduction:** This means planning method to include the loss due to risk. For instance, if there is a risk that some key personnel might leave, new recruitment can be planned.

Risk Leverage: To choose between the various methods of handling risk, the project plan must consider the amount of controlling the risk and the corresponding reduction of risk. For this, the risk leverage of the various risks can be estimated.

Risk leverage is the variation in risk exposure divided by the amount of reducing the risk.

Risk leverage = (risk exposure before reduction - risk exposure after reduction) / (cost of reduction)

1. Risk planning: The risk planning method considers each of the key risks that have been identified and develop ways to maintain these risks.

For each of the risks, you have to think of the behavior that you may take to minimize the disruption to the plan if the issue identified in the risk occurs.

You also should think about data that you might need to collect while monitoring the plan so that issues can be anticipated.

Again, there is no easy process that can be followed for contingency planning. It rely on the judgment and experience of the project manager.

2. Risk Monitoring: Risk monitoring is the method king that your assumption about the product, process, and business risks has not changed.

Risk mitigation

Risk mitigation planning is the process of developing options and actions to enhance opportunities and reduce threats to project objectives. Risk mitigation implementation is the process of executing risk mitigation actions. Risk mitigation progress monitoring includes tracking identified risks, identifying new risks, and evaluating risk process effectiveness throughout the project.

Keywords: risk, risk management, risk mitigation, risk mitigation implementation, risk mitigation planning, risk mitigation progress monitoring

MITRE SE Roles & Expectations: MITRE systems engineers (SEs) working on government programs develop actionable risk mitigation strategies and monitoring metrics, monitor implementation of risk mitigation plans to ensure successful project and program completion, collaborate with the government team in conducting risk reviews across projects and programs, and analyze metrics to determine ongoing risk status and identify serious risks to elevate to the sponsor or customer.

Background

Risk mitigation planning, implementation, and progress monitoring are depicted in Figure. As part of an iterative process, the risk tracking tool is used to record the results of risk prioritization analysis (step 3) that provides input to both risk mitigation (step 4) and risk impact assessment (step 2).



Figure 1. Risk Management: Fundamental Steps

The risk mitigation step involves development of mitigation plans designed to manage, eliminate, or reduce risk to an acceptable level. Once a plan is implemented, it is continually monitored to assess its efficacy with the intent of revising the course-of-action if needed.

Risk Mitigation Strategies

General guidelines for applying risk mitigation handling options are shown in Figure. These options are based on the assessed combination of the probability of

occurrence and severity of the consequence for an identified risk. These guidelines are appropriate for many, but not all, projects and programs.



Figure 2. Risk Mitigation Handling Options [3]

Risk mitigation handling options include:

- Assume/Accept: Acknowledge the existence of a particular risk, and make a deliberate decision to accept it without engaging in special efforts to control it. Approval of project or program leaders is required.
- Avoid: Adjust program requirements or constraints to eliminate or reduce the risk. This adjustment could be accommodated by a change in funding, schedule, or technical requirements.
- Control: Implement actions to minimize the impact or likelihood of the risk.
- Transfer: Reassign organizational accountability, responsibility, and authority to another stakeholder willing to accept the risk.
- Watch/Monitor: Monitor the environment for changes that affect the nature and/or the impact of the risk.

Each of these options requires developing a plan that is implemented and monitored for effectiveness. More information on handling options is discussed under best practices and lessons learned below.

From a systems engineering perspective, common methods of risk reduction or mitigation with identified program risks include the following, listed in order of increasing seriousness of the risk:

1. Intensified technical and management reviews of the engineering process
2. Special oversight of designated component engineering
3. Special analysis and testing of critical design items
4. Rapid prototyping and test feedback
5. Consideration of relieving critical design requirements
6. Initiation of fallback parallel developments

When determining the method for risk mitigation, the MITRE SE can help the customer assess the performance, schedule, and cost impacts of one mitigation strategy over another. For something like "parallel" development mitigation, MITRE SEs could help the government determine whether the cost could more than double, while time might not be extended by much (e.g., double the cost for parallel effort, but also added cost for additional program office and user engagement). For conducting rapid prototyping or changing operational requirements, MITRE SEs can use knowledge in creating prototypes and using prototyping and experimenting (see SE Guide article on Special Considerations for Conditions of Uncertainty: Prototyping and Experimentation and the Requirements Engineering topic) for projecting the cost and time to conduct a prototype to help mitigate particular risks (e.g., requirements). Implementing more engineering reviews and special oversight and testing may require changes to contractual agreements. MITRE systems engineers can help the government assess these (schedule and cost) by helping determine the basis of estimates for additional contractor efforts and providing a reality check for these estimates. MITRE's CASA [Center for Acquisition and Systems Analysis] and the CCG [Center for Connected Government] Investment Management practice department have experience and a knowledge base in many development activities across a wide

spectrum of methods and can help with realistic assessments of mitigation alternatives.

For related information, refer to the other articles in this Risk Management topic area of the SE Guide.

Best Practices and Lessons Learned

What actions are needed?

When must actions be completed?

Handling Options

- Assume/Accept. Collaborate with the operational users to create a collective understanding of risks and their implications. Risks can be characterized as impacting traditional cost, schedule, and performance parameters. Risks should also be characterized as impact to mission performance resulting from reduced technical performance or capability. Develop an understanding of all these impacts. Bringing users into the mission impact characterization is particularly important to selecting which "assume/accept" option is ultimately chosen. Users will decide whether accepting the consequences of a risk is acceptable. Provide the users with the vulnerabilities affecting a risk, countermeasures that can be performed, and residual risk that may occur. Help the users understand the costs in terms of time and money.
- Avoid. Again, work with users to achieve a collective understanding of the implications of risks. Provide users with projections of schedule adjustments needed to reduce risk associated with technology maturity or additional development to improve performance. Identify capabilities that will be delayed and any impacts resulting from dependencies on other efforts. This information better enables users to interpret the operational implications of an "avoid" option.

- Control. Help control risks by performing analyses of various mitigation options. For example, one option is to use a commercially available capability instead of a contractor developed one. In developing options for controlling risk in your program, seek out potential solutions from similar risk situations of other MITRE customers, industry, and academia. When considering a solution from another organization, take special care in assessing any architectural changes needed and their implications.
- Transfer. Reassigning accountability, responsibility, or authority for a risk area to another organization can be a double-edged sword. It may make sense when the risk involves a narrow specialized area of expertise not normally found in program offices. But, transferring a risk to another organization can result in dependencies and loss of control that may have their own complications. Position yourself and your customer to consider a transfer option by acquiring and maintaining awareness of organizations within your customer space that focus on specialized needs and their solutions. Acquire this awareness as early in the program acquisition cycle as possible, when transfer options are more easily implemented.
- Watch/Monitor. Once a risk has been identified and a plan put in place to manage it, there can be a tendency to adopt a "heads down" attitude, particularly if the execution of the mitigation appears to be operating on "cruise control." Resist that inclination. Periodically revisit the basic assumptions and premises of the risk. Scan the environment to see whether the situation has changed in a way that affects the nature or impact of the risk. The risk may have changed sufficiently so that the current mitigation is ineffective and needs to be scrapped in favor of a different one. On the other hand, the risk may have diminished in a way that allows resources devoted to it to be redirected.

Determining Mitigation Plans

- Understand the users and their needs. The users/operational decision makers will be the decision authority for accepting and avoiding risks. Maintain a close relationship with the user community throughout the system engineering life cycle. Realize that mission accomplishment is paramount to the user community and acceptance of residual risk should be firmly rooted in a mission decision.
- Seek out the experts and use them. Seek out the experts within and outside MITRE. MITRE's technical centers exist to provide support in their specialty areas. They understand what's feasible, what's worked and been implemented, what's easy, and what's hard. They have the knowledge and experience essential to risk assessment in their area of expertise. Know our internal centers of excellence, cultivate relationships with them, and know when and how to use them.
- Recognize risks that recur. Identify and maintain awareness of the risks that are "always there" — interfaces, dependencies, changes in needs, environment and requirements, information security, and gaps or holes in contractor and program office skill sets. Help create an acceptance by the government that these risks will occur and recur and that plans for mitigation are needed up front. Recommend various mitigation approaches — including adoption of an evolution strategy, prototyping, experimentation, engagement with broader stakeholder community, and the like.
- Encourage risk taking. Given all that has been said in this article and its companions, this may appear to be an odd piece of advice. The point is that there are consequences of not taking risks, some of which may be negative. Help the customer and users understand that reality and the potential consequences of being overly timid and not taking certain risks in your program. An example of a negative consequence for not taking a risk when delivering a full capability is that an adversary might realize a gain against our operational users.

Risks are not defeats, but simply bumps in the road that need to be anticipated and dealt with.

- Recognize opportunities. Help the government understand and see opportunities that may arise from a risk. When considering alternatives for managing a particular risk, be sure to assess whether they provide an opportunistic advantage by improving performance, capacity, flexibility, or desirable attributes in other areas not directly associated with the risk.
- Encourage deliberate consideration of mitigation options. This piece of advice is good anytime, but particularly when supporting a fast-paced, quick reaction government program that is juggling many competing priorities. Carefully analyze mitigation options and encourage thorough discussion by the program team. This is the form of the wisdom "go slow to go fast."
- Not all risks require mitigation plans. Risk events assessed as medium or high criticality should go into risk mitigation planning and implementation. On the other hand, consider whether some low criticality risks might just be tracked and monitored on a watch list. Husband your risk-related resources.

Mitigation Plan Content

- Determine the appropriate risk manager. The risk manager is responsible for identifying and implementing the risk mitigation plan. He or she must have the knowledge, authority, and resources to implement the plan. Risk mitigation activities will not be effective without an engaged risk manager. It may be necessary to engage higher levels in the customer organization to ensure the need for the risk manager is addressed. This can be difficult and usually involves engaging more senior levels of the MITRE team as well.
- Develop a high-level mitigation strategy. This is an overall approach to reduce the risk impact severity and/or probability of occurrence. It

could affect a number of risks and include, for example, increasing staffing or reducing scope.

- Identify actions and steps needed to implement the mitigation strategy. Ask these key questions:

- **What actions are needed?**
  - Make sure you have the right exit criteria for each. For example, appropriate decisions, agreements, and actions resulting from a meeting would be required for exit, not merely the fact that the meeting was held.
  - Look for evaluation, proof, and validation of met criteria. Consider, for example, metrics or test events.
  - Include only and all stakeholders relevant to the step, action, or decisions.
- **When must actions be completed?**
  - Backward Planning: Evaluate the risk impact and schedule of need for the successful completion of the program and evaluate test events, design considerations, and more.
  - Forward Planning: Determine the time needed to complete each action step and when the expected completion date should be.
  - Evaluate key decision points and determine when a move to a contingency plan should be taken.
- Who is the responsible action owner?
- What resources are required? Consider, for example, additional funding or collaboration.
- How will this action reduce the probability or severity of impact?
- Develop a contingency plan ("fall back, plan B") for any high risk.

- Are cues and triggers identified to activate contingency plans and risk reviews?
- Include decision point dates to move to fallback plans. The date to move must allow time to execute the contingency plan.

- Evaluate the status of each action. Determine when each action is expected to be completed successfully.
- Integrate plans into IMS and program management baselines. Risk plans are integral to the program, not something apart from it.
- Monitoring Risk
  - Include risk monitoring as part of the program review and manage continuously. Monitoring risks should be a standard part of program reviews. At the same time, risks should be managed continuously rather than just before a program review. Routinely review plans in management meetings.
  - Review and track risk mitigation actions for progress. Determine when each action is expected to be completed successfully.
  - Refine and redefine strategies and action steps as needed.
  - Revisit risk analysis as plans and actions are successfully completed. Are the risks burning down? Evaluate impact to program critical path.
  - Routinely reassess the program's risk exposure. Evaluate the current environment for new risks or modification to existing risks.

Risk Mitigation, Monitoring and Management (RMMM)

Risk analysis support the project team in constructing a strategy to deal with risks.

There are three important issues considered in developing an effective strategy:

- **Risk avoidance or mitigation -** It is the primary strategy which is fulfilled through a plan.
- **Risk monitoring -** The project manager monitors the factors and gives an indication whether the risk is becoming more or less.
- **Risk management and planning -** It assumes that the mitigation effort failed and the risk is a reality.

RMMM Plan

- It is a part of the software development plan or a separate document.
- The RMMM plan documents all work executed as a part of risk analysis and used by the project manager as a part of the overall project plan.
- The risk mitigation and monitoring starts after the project is started and the documentation of RMMM is completed.

Software Reliability

Software Reliability means Operational reliability. It is described as the ability of a system or component to perform its required functions under static conditions for a specific period.

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.

Software Reliability is an essential connect of software quality, composed with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve because the complexity of software turn to be high. While any system with a high degree of complexity, containing software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the speedy growth of system size and ease of doing so by upgrading the software.

For example, large next-generation aircraft will have over 1 million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines; the upcoming International Space Station will have over two million lines on-board and over 10 million lines of ground support software; several significant life-critical defense systems will have over 5 million source lines of software. While the complexity of software is inversely associated with software reliability, it is directly related to other vital factors in software quality, especially functionality, capability, etc.

Estimation of Software Projects

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

Estimation determines how much money, effort, resources, and time it will take to build a specific system or product. Estimation is based on −

- Past Data/Past Experience
- Available Documents/Knowledge
- Assumptions
- Identified Risks

The four basic steps in Software Project Estimation are −

- Estimate the size of the development product.
- Estimate the effort in person-months or person-hours.
- Estimate the schedule in calendar months.
- Estimate the project cost in agreed currency.

Observations on Estimation

- Estimation need not be a one-time task in a project. It can take place during –
    - Acquiring a Project.
    - Planning the Project.
    - Execution of the Project as the need arises.
- Project scope must be understood before the estimation process begins. It will be helpful to have historical Project Data.
- Project metrics can provide a historical perspective and valuable input for generation of quantitative estimates.
- Planning requires technical managers and the software team to make an initial commitment as it leads to responsibility and accountability.
- Past experience can aid greatly.
- Use at least two estimation techniques to arrive at the estimates and reconcile the resulting values. Refer Decomposition Techniques in the next section to learn about reconciling estimates.
- Plans should be iterative and allow adjustments as time passes and more details are known.

General Project Estimation Approach

The Project Estimation Approach that is widely used is **Decomposition Technique**. Decomposition techniques take a divide and conquer approach. Size, Effort and Cost estimation are performed in a stepwise manner by breaking down a Project into major Functions or related Software Engineering Activities.

Step 1 – Understand the scope of the software to be built.

Step 2 – Generate an estimate of the software size.

- Start with the statement of scope.

- Decompose the software into functions that can each be estimated individually.
- Calculate the size of each function.
- Derive effort and cost estimates by applying the size values to your baseline productivity metrics.
- Combine function estimates to produce an overall estimate for the entire project.

Step 3 – Generate an estimate of the effort and cost. You can arrive at the effort and cost estimates by breaking down a project into related software engineering activities.

- Identify the sequence of activities that need to be performed for the project to be completed.
- Divide activities into tasks that can be measured.
- Estimate the effort (in person hours/days) required to complete each task.
- Combine effort estimates of tasks of activity to produce an estimate for the activity.
- Obtain cost units (i.e., cost/unit effort) for each activity from the database.
- Compute the total effort and cost for each activity.
- Combine effort and cost estimates for each activity to produce an overall effort and cost estimate for the entire project.

Step 4 – Reconcile estimates: Compare the resulting values from Step 3 to those obtained from Step 2. If both sets of estimates agree, then your numbers are highly reliable. Otherwise, if widely divergent estimates occur conduct further investigation concerning whether –

- The scope of the project is not adequately understood or has been misinterpreted.

- The function and/or activity breakdown is not accurate.
- Historical data used for the estimation techniques is inappropriate for the application, or obsolete, or has been misapplied.

Step 5 – Determine the cause of divergence and then reconcile the estimates.

Estimation Accuracy

Accuracy is an indication of how close something is to reality. Whenever you generate an estimate, everyone wants to know how close the numbers are to reality. You will want every estimate to be as accurate as possible, given the data you have at the time you generate it. And of course you don't want to present an estimate in a way that inspires a false sense of confidence in the numbers.

Important factors that affect the accuracy of estimates are –

- The accuracy of all the estimate's input data.
- The accuracy of any estimate calculation.
- How closely the historical data or industry data used to calibrate the model matches the project you are estimating.
- The predictability of your organization's software development process.
- The stability of both the product requirements and the environment that supports the software engineering effort.
- Whether or not the actual project was carefully planned, monitored and controlled, and no major surprises occurred that caused unexpected delays.

Following are some guidelines for achieving reliable estimates –

- Base estimates on similar projects that have already been completed.
- Use relatively simple decomposition techniques to generate project cost and effort estimates.

- Use one or more empirical estimation models for software cost and effort estimation.

Refer to the section on Estimation Guidelines in this chapter.

To ensure accuracy, you are always advised to estimate using at least two techniques and compare the results.

Estimation Issues

Often, project managers resort to estimating schedules skipping to estimate size. This may be because of the timelines set by the top management or the marketing team. However, whatever the reason, if this is done, then at a later stage it would be difficult to estimate the schedules to accommodate the scope changes.

While estimating, certain assumptions may be made. It is important to note all these assumptions in the estimation sheet, as some still do not document assumptions in estimation sheets.

Even good estimates have inherent assumptions, risks, and uncertainty, and yet they are often treated as though they are accurate.

The best way of expressing estimates is as a range of possible outcomes by saying, for example, that the project will take 5 to 7 months instead of stating it will be complete on a particular date or it will be complete in a fixed no. of months. Beware of committing to a range that is too narrow as that is equivalent to committing to a definite date.

- You could also include uncertainty as an accompanying probability value. For example, there is a 90% probability that the project will complete on or before a definite date.
- Organizations do not collect accurate project data. Since the accuracy of the estimates depend on the historical data, it would be an issue.
- For any project, there is a shortest possible schedule that will allow you to include the required functionality and produce quality output. If there is a

schedule constraint by management and/or client, you could negotiate on the scope and functionality to be delivered.

- Agree with the client on handling scope creeps to avoid schedule overruns.
- Failure in accommodating contingency in the final estimate causes issues. For e.g., meetings, organizational events.
- Resource utilization should be considered as less than 80%. This is because the resources would be productive only for 80% of their time. If you assign resources at more than 80% utilization, there is bound to be slippages.

Estimation Guidelines

One should keep the following guidelines in mind while estimating a project −

- During estimation, ask other people's experiences. Also, put your own experiences at task.
- Assume resources will be productive for only 80 percent of their time. Hence, during estimation take the resource utilization as less than 80%.
- Resources working on multiple projects take longer to complete tasks because of the time lost switching between them.
- Include management time in any estimate.
- Always build in contingency for problem solving, meetings and other unexpected events.
- Allow enough time to do a proper project estimate. Rushed estimates are inaccurate, high-risk estimates. For large development projects, the estimation step should really be regarded as a mini project.
- Where possible, use documented data from your organization’s similar past projects. It will result in the most accurate estimate. If your organization has not kept historical data, now is a good time to start collecting it.

- Use developer-based estimates, as the estimates prepared by people other than those who will do the work will be less accurate.
- Use several different people to estimate and use several different estimation techniques.
- Reconcile the estimates. Observe the convergence or spread among the estimates. Convergence means that you have got a good estimate. Wideband-Delphi technique can be used to gather and discuss estimates using a group of people, the intention being to produce an accurate, unbiased estimate.
- Re-estimate the project several times throughout its life cycle.

Project Scheduling

Scheduling in project management is the listing of activities, deliverables, and milestones within a project. A schedule also usually includes the planned start and finish date, duration, and resources assigned to each activity. Effective project scheduling is a critical component of successful time management.

Project-task scheduling is a significant project planning activity. It comprises deciding which functions would be taken up when. To schedule the project plan, a software project manager wants to do the following:

1. Identify all the functions required to complete the project.
2. Break down large functions into small activities.
3. Determine the dependency among various activities.
4. Establish the most likely size for the time duration required to complete the activities.
5. Allocate resources to activities.
6. Plan the beginning and ending dates for different activities.

7. Determine the critical path. A critical way is the group of activities that decide the duration of the project.

The first method in scheduling a software plan involves identifying all the functions required to complete the project. A good judgment of the intricacies of the project and the development process helps the supervisor to identify the critical role of the project effectively. Next, the large functions are broken down into a valid set of small activities which would be assigned to various engineers. The work breakdown structure formalism supports the manager to breakdown the function systematically after the project manager has broken down the purpose and constructs the work breakdown structure; he has to find the dependency among the activities. Dependency among the various activities determines the order in which the various events would be carried out. If an activity A necessary the results of another activity B, then activity A must be scheduled after activity B. In general, the function dependencies describe a partial ordering among functions, i.e., each service may precede a subset of other functions, but some functions might not have any precedence ordering describe between them (called concurrent function). The dependency among the activities is defined in the pattern of an activity network.

Once the activity network representation has been processed out, resources are allocated to every activity. Resource allocation is usually done using a Gantt chart. After resource allocation is completed, a PERT chart representation is developed. The PERT chart representation is useful for program monitoring and control. For task scheduling, the project plan needs to decompose the project functions into a set of activities. The time frame when every activity is to be performed is to be determined. The end of every action is called a milestone. The project manager tracks the function of a project by audit the timely completion of the milestones. If he examines that the milestones start getting delayed, then he has to handle the activities carefully so that the complete deadline can still be met.

Scheduling in project management is the listing of activities, deliverables, and milestones within a project. A schedule also usually includes the planned start and

finish date, duration, and resources assigned to each activity. Effective project scheduling is a critical component of successful time management.

In fact, when people discuss the processes for building a schedule, they are usually referring to the first six processes of time management:

1. Plan schedule management.
2. Define project activities.
3. Sequence activities.
4. Estimate resources.
5. Estimate durations.
6. Develop the project schedule.

How to do scheduling in project management

There are three main types of schedules:

1. Master project schedule. A master schedule tends to be a simplified list of tasks with a timeline or project calendar.
2. Milestone schedule or summary schedule. This type of schedule tracks major milestones and key deliverables, but not every task required to complete the project.
3. A detailed project schedule. This is the most thorough project schedule, as it identifies and tracks every project activity. If you have a complex, large, or lengthy project, it's important to have a detailed project schedule to help track everything.

The most common form of project schedule is a Gantt chart. Both a milestone schedule and a detailed project schedule can be created as a Gantt chart. When choosing a scheduling software, look for scheduling tools that allow you to create different views from the same schedule. If you create a detailed schedule with milestones as a Gantt Chart, make sure it can be summarized up to that level for a

simpler view that can be easily shared with your team or stakeholders. This gives you the ability to present the same schedule in different formats depending on the level of detail required and the target audience.

Benefits of project scheduling in project management

Project scheduling provides the following benefits:

- Assists with tracking, reporting on, and communicating progress.
- Ensures everyone is on the same page as far as tasks, dependencies, and deadlines.
- Helps highlight issues and concerns, such as a lack of resources.
- Helps identify task relationships.
- Can be used to monitor progress and identify issues early.

7 tips for creating a solid project schedule

1. The time management processes identified earlier are the key steps to creating a project schedule. However, keep these seven tips in mind to make sure your schedule is realistic.Get input from stakeholders. Make sure you don't create your schedule in isolation. It's important to use your team and other stakeholders to identify tasks, resources, dependencies, and durations.
2. Reference past projects. Looking at previous projects with similar scope and requirements can help create realistic estimates and ensure you haven't forgotten any tasks.
3. Include project milestones. Milestones are events or markers that stand for an important point in your project. They're useful for creating a summary schedule, reporting to executives, and identifying problems early. Here are some milestone examples:
4. Project kickoff

5. Design approvals
6. Completion of requirements
7. Product implementation
8. Project closeout
9. Consider any non-work time. For example, make sure vacations and holidays are reflected in your schedule so that you're not assuming people will be working when they're not.
10. Define the critical path on your project. Identifying your project's critical path allows you to prioritize and allocate resources to the most important tasks in the project.
11. Record scheduling assumptions. Write down the logic behind your scheduling predictions. For example, if you assume it will only take 10 hours to complete a task because you're going to have a senior engineer. That way, if you end up with a junior engineer, you can understand and explain why it took twice as long as planned.
12. Keep risk in mind. Identify and document any factors that pose a risk to staying on schedule. This will help your risk management efforts.

Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:

- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

1. Lines of Code (LOC): As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:

- KLOC- Thousand lines of code
- NLOC- Non comment lines of code
- KDSI- Thousands of delivered source instruction

Software Sizing

The size is estimated by comparing it with the existing systems of same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

Advantages:

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to developer's perspective.
- Simple to use.

Disadvantages:

- Different programming languages contains different number of lines.
- No proper industry standard exist for this technique.
- It is difficult to estimate the size using this technique in early stages of project.

2. Number of entities in ER diagram: ER model provides a static view of the project. It describes the entities and its relationships. The number of entities in ER model can be used to measure the estimation of size of project. Number of entities depends on the size of the project. This is because more entities needed more classes/structures thus leading to more coding.

Advantages:

- Size estimation can be done during initial stages of planning.
- Number of entities is independent of programming technologies used.

Disadvantages:

- No fixed standards exist. Some entities contribute more project size than others.
- Just like FPA, it is less used in cost estimation model. Hence, it must be converted to LOC.

3. Total number of processes in detailed data flow diagram: Data Flow Diagram(DFD) represents the functional view of a software. The model depicts the main processes/functions involved in software and flow of data between them. Utilization of number of functions in DFD to predict software size. Already existing processes of similar type are studied and used to estimate the size of the process. Sum of the estimated size of each process gives the final estimated size.

Advantages:

- It is independent of programming language.
- Each major processes can be decomposed into smaller processes. This will increase the accuracy of estimation

Disadvantages:

- Studying similar kind of processes to estimate size takes additional time and effort.
- All software projects are not required to construction of DFD.

4. Function Point Analysis: In this method, the number and type of functions supported by the software are utilized to find FPC(function point count). The steps in function point analysis are:

- Count the number of functions of each proposed type.

- Compute the Unadjusted Function Points(UFP).
- Find Total Degree of Influence(TDI).
- Compute Value Adjustment Factor(VAF).
- Find the Function Point Count(FPC).

The explanation of above points given below:

- **Count the number of functions of each proposed type:** Find the number of functions belonging to the following types:
    - External Inputs: Functions related to data entering the system.
    - External outputs:Functions related to data exiting the system.
    - External Inquiries: They leads to data retrieval from system but don't change the system.
    - Internal Files: Logical files maintained within the system. Log files are not included here.
    - External interface Files: These are logical files for other applications which are used by our system.
- **Compute the Unadjusted Function Points(UFP):** Categorise each of the five function types as simple, average or complex based on their complexity. Multiply count of each function type with its weighting factor and find the weighted sum. The weighting factors for each type based on their complexity are as follows:

FUNCTION TYPE	SIMPLE	AVERAGE	COMPLEX
External Inputs	3	4	6
External Output	4	5	7

FUNCTION TYPE	SIMPLE	AVERAGE	COMPLEX
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

- **Find Total Degree of Influence:** Use the '14 general characteristics' of a system to find the degree of influence of each of them. The sum of all 14 degrees of influences will give the TDI. The range of TDI is 0 to 70. The 14 general characteristics are: Data Communications, Distributed Data Processing, Performance, Heavily Used Configuration, Transaction Rate, On-Line Data Entry, End-user Efficiency, Online Update, Complex Processing Reusability, Installation Ease, Operational Ease, Multiple Sites and Facilitate Change.
  Each of above characteristics is evaluated on a scale of 0-5.
- **Compute Value Adjustment Factor(VAF):** Use the following formula to calculate VAF
  VAF = (TDI * 0.01) + 0.65
- **Find the Function Point Count:** Use the following formula to calculate FPC
  FPC = UFP * VAF

Advantages:

- It can be easily used in the early stages of project planning.
- It is independing on the programming language.
- It can be used to compare different projects even if they use different technologies(database, language etc).

Disadvantages:

- It is not good for real time systems and embedded systems.
- Many cost estimation models like COCOMO uses LOC and hence FPC must be converted to LOC.

Software Cost Estimation

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

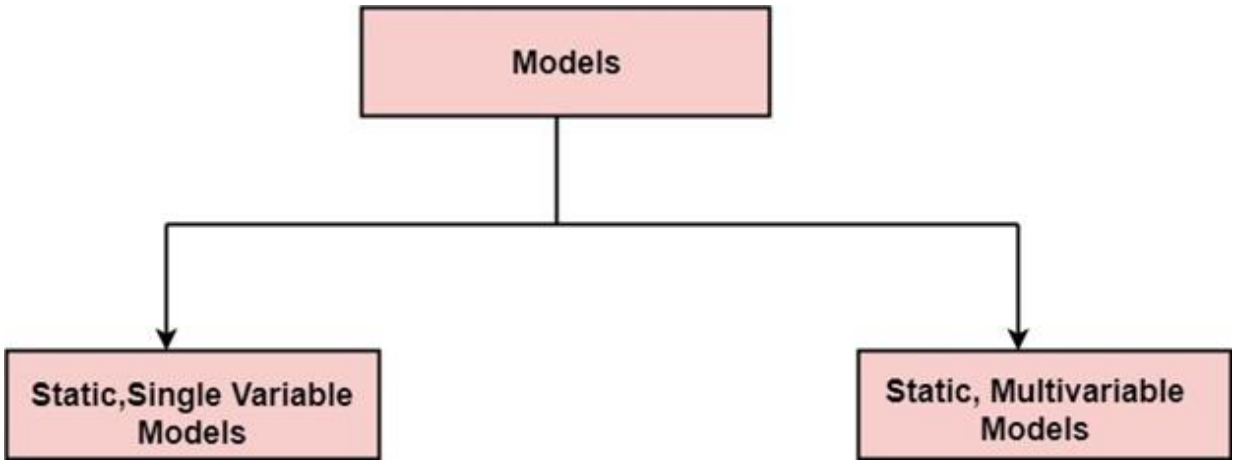
1. Project scope must be established in advanced.
2. Software metrics are used as a support from which evaluation is made.
3. The project is broken into small PCs which are estimated individually. To achieve true cost & schedule estimate, several option arise.
4. Delay estimation
5. Used symbol decomposition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

Uses of Cost Estimation

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

Cost Estimation Models

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



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

$$C=aL^b$$

Where C = Costs
L= size
a and b are constants

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

$$E=1.4L^{0.93}$$
$$DOC=30.4L^{0.90}$$
$$D=4.6L^{0.26}$$

Where E= Efforts (Person Per Month)
DOC=Documentation (Number of Pages)
D = Duration (D, in months)
L = Number of Lines per code

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

WALSTON and FELIX develop the models at IBM provide the following equation gives a relationship between lines of source code and effort:

$$E=5.2L^{0.91}$$

In the same manner duration of development is given by

$$D=4.1L^{0.36}$$

The productivity index uses 29 variables which are found to be highly correlated productivity as follows:

$$I = \sum_{i=1}^{29} W_i X_i$$

Where W_i is the weight factor for the i^{th}variable and X_i**={-1,0,+1}** the estimator gives $X_i$one of the values **-1, 0 or +1** depending on the variable decreases, has no effect or increases the productivity.

Example: Compare the Walston-Felix Model with the SEL model on a software development expected to involve 8 person-years of effort.

a. Calculate the number of lines of source code that can be produced.
b. Calculate the duration of the development.
c. Calculate the productivity in LOC/PY
d. Calculate the average manning

Solution:

The amount of manpower involved = 8PY=96persons-months

(a)Number of lines of source code can be obtained by reversing equation to give:

$$L = \left(\frac{E}{a}\right) 1/b$$

Then

L (SEL) = (96/1.4)1⁄0.93=94264 LOC
L (SEL) = (96/5.2)1⁄0.91=24632 LOC

(b)Duration in months can be calculated by means of equation

D (SEL) = 4.6 (L) 0.26
= 4.6 (94.264)0.26 = 15 months
D (W-F) = 4.1 $L^{0.36}$
= 4.1 (24.632)0.36 = 13 months

(c) Productivity is the lines of code produced per persons/month (year)

$$P\,(SEL) = \frac{94264}{8} = 11783\,\frac{LOC}{Person} - Years$$

$$P\,(Years) = \frac{24632}{8} = 3079\,\frac{LOC}{Person} - Years$$

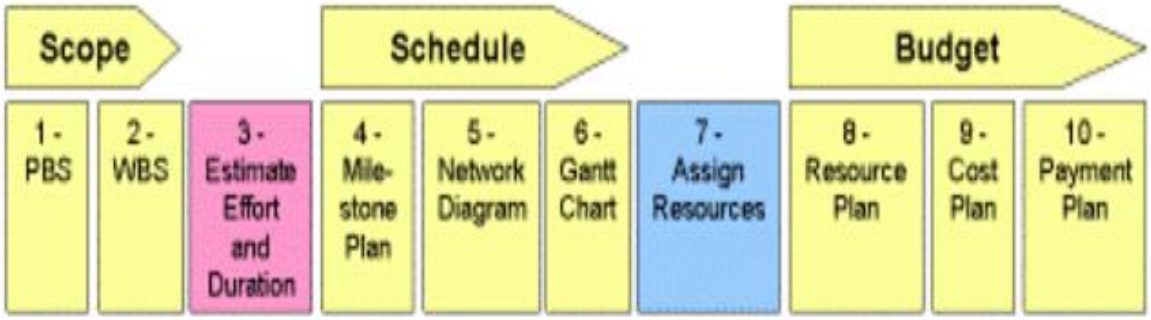
(d)Average manning is the average number of persons required per month in the project

$$M\,(SEL) = \frac{96P - M}{15M} = 6.4 Persons$$

$$M\,(W\text{-}F) = \frac{96P - M}{13M} = 7.4 Persons$$

Effort Estimation

Effort estimation represents **step 3** of the project planning process.



Before we can plan the project schedule we have to estimate effort and duration of all the work packages of the WBS. Effort estimation will generate a lot more information than only effort and duration:

- Who will be responsible for each work package?
- What is the work package specification?
- What are the expected results of each work package?
- How is the achievement of the results measured?
- What are the prerequisites for the work package?
- What are the conditions under which the work has to be done?
- What are the required start and end times?
- What and how much material is needed for each work package, at what cost?
- What tools are needed for each work package, at what rates?

Obviously, effort estimation needs expertise on a work package specific level to accomplish this transition.

There are two categories of estimating the effort of each work package: deductive and inductive methods.

Deductive or Top-down Methods

Assume the total cost for the project is given. From there we assign the cost, and thus, the effort of individual work packages based on estimated percentages derived from earlier, similar projects with similar work packages. The advantage of deductive methods is a simple and rapid cost allocation, the disadvantage is that they only work for projects that consist of work packages we already know from earlier, similar projects.

Inductive or Bottom-up Methods

The basic idea of inductive methods is to start effort estimation with the work packages individually, with support of experts, or knowledge of similar work packages of earlier projects, and then summarize bottom up, following the structure of the WBS.

(1) Factor Analysis: For a certain work package, we know all the variables or factors, how they influence the work, and how these factors correlate with each other. Then we can calculate the effort based on a mathematical formula which reflects that influences and correlations.

Effort = f (influencing variables, correlation coefficients).

For example, the effort of a work package "Develop hardware control unit" is influenced by
the number of people involved, P=4,
the number of interfaces, S=5,
the number of functional blocks, B=10.
From earlier projects with similar control units we might know the correlation coefficients. For example, cP=2.5, cS=2, cB=1.5, cU=2.0. Our formula could look like this

Effort = f(P, S, B) = cP*P + cS*S*S + cB*B =
= 10 + 50 + 15 =
= 75 (working days)

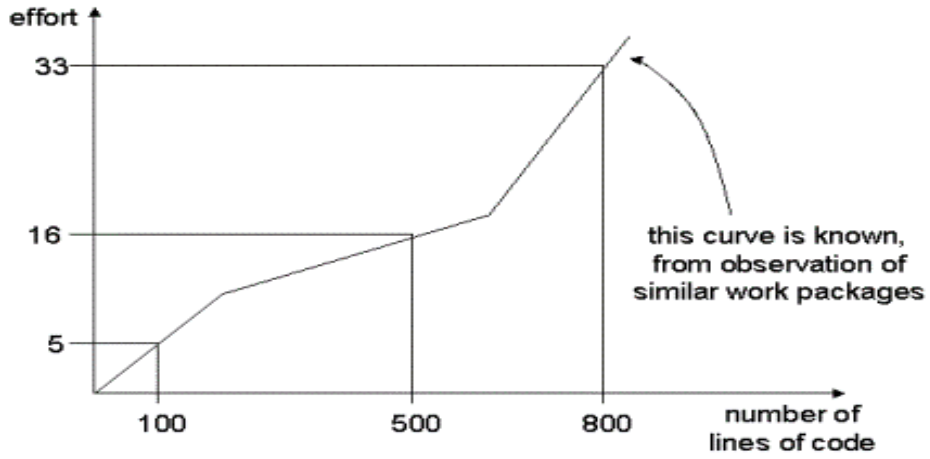
(2) Multiplication Method: If we can divide a work package into a number of equal parts then, we can estimate the total effort by estimating one part and then multiplying this value by the number of parts.

Total Effort = Effort of one part * number of parts

(3) Analogy Method: This method again applies the knowledge from similar work packages. We interpolate or extrapolate the effort for the work package from a similar one. For example, for the work package "Install electrical wiring in an apartment of 100 sqm" we could guess the effort by interpolation from similar work for apartments of 150 sqm, 10 working days, and 50 sqm, 6 working days:

Effort(100 sqm) = (10 + 6) / 2 = 8 working days.

(4) Function Point Method: For IT or software design related work packages we can apply the function point method. The prerequisite is that we need to have a lot of knowledge about the effort of work packages of similar scope and degree of difficulty, based on observation. Then the experts compile this knowledge into so called "function curves" which we can use to estimate effort for new work packages. In the following example we estimate the effort of work about creating 100, 500, and 800 lines of code. We obtain the effort via the corresponding function points on the function curve, 5, 16, and 33 working days, respectively.



(5) Delphi Method: For most of our work packages we use the Delphi method. We just ask the experts for each work package for their best guess, normal guess, and worst guess. Thus we obtain three figures for the expected

effort: **E(optimistic), E(normal), and E(pessimistic),** then combine them with this formula:

Effort = (E(optimistic)+4*E(normal)+E(pessimistic))/6.

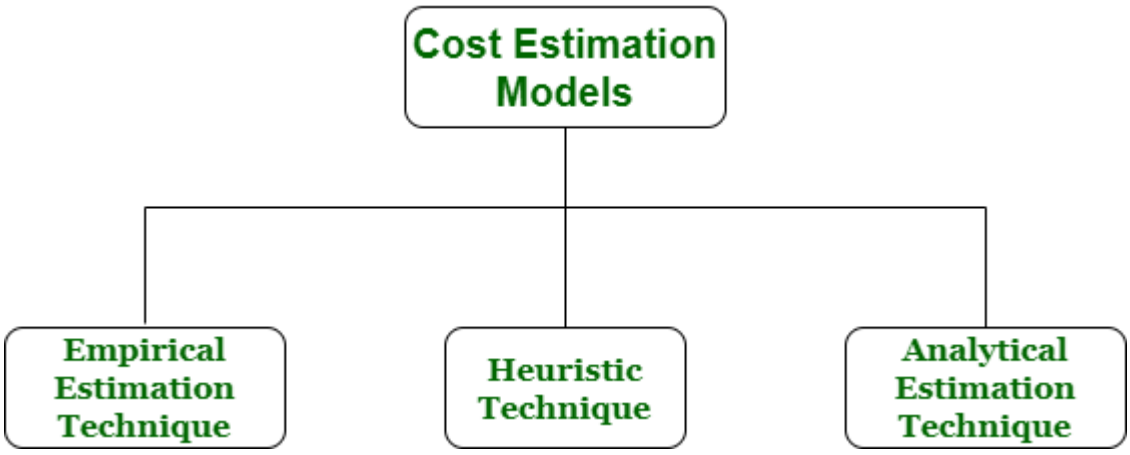
If we can ask ten or more experts, we could even calculate the mean values and apply mathematical statistics with the concept of standard deviation.

Usually, we ask all the experts to join an effort estimation workshop which can be combined with a risk management workshop, of two, three, or more days depending on the number of work packages that have to be estimated.

Cost Estimation Models

Cost estimation simply means a technique that is used to find out the cost estimates. The cost estimate is the financial spend that is done on the efforts to develop and test software in Software Engineering. Cost estimation models are some mathematical algorithms or parametric equations that are used to estimate the cost of a product or a project.

Various techniques or models are available for cost estimation, also known as Cost Estimation Models as shown below :



1. **Empirical Estimation Technique –**
   Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step. These techniques are usually

based on the data that is collected previously from a project and also based on some guesses, prior experience with the development of similar types of projects, and assumptions. It uses the size of the software to estimate the effort.

In this technique, an educated guess of project parameters is made. Hence, these models are based on common sense. However, as there are many activities involved in empirical estimation techniques, this technique is formalized. For example Delphi technique and Expert Judgement technique.

2. **Heuristic Technique –**
   Heuristic word is derived from a Greek word that means "to discover". The heuristic technique is a technique or model that is used for solving problems, learning, or discovery in the practical methods which are used for achieving immediate goals. These techniques are flexible and simple for taking quick decisions through shortcuts and good enough calculations, most probably when working with complex data. But the decisions that are made using this technique are necessary to be optimal.

1. In this technique, the relationship among different project parameters is expressed using mathematical equations. The popular heuristic technique is given by Constructive Cost Model (COCOMO). This technique is also used to increase or speed up the analysis and investment decisions.

2. **Analytical Estimation Technique –**
   Analytical estimation is a type of technique that is used to measure work. In this technique, firstly the task is divided or broken down into its basic component operations or elements for analyzing. Second, if the standard time is available from some other source, then these sources are applied to each element or component of work.

Third, if there is no such time available, then the work is estimated based on the experience of the work. In this technique, results are derived by making certain basic assumptions about the project. Hence, the analytical estimation technique

has some scientific basis. Halstead's software science is based on an analytical estimation model.

Cocomo (Constructive Cost 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 –

4. **Basic Model –**

$$E = a(KLOC)^b$$

$$time = c(Effort)^d$$

$$Personrequired = Effort/time$$

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,b,c and d for the Basic Model for the different categories of system:

SOFTWARE PROJECTS	A	B	C	D
Organic	2.4	1.05	2.5	0.38
Semi Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code.
The development time is measured in Months.

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.

Below is the C++ program for Basic COCOMO

```
// C++ program to implement basic COCOMO

#include<bits/stdc++.h>

using namespace std;
```

```
// Function for rounding off float to int
int fround(float x)
{
   int a;
   x=x+0.5;
   a=x;
   return(a);
}
// Function to calculate parameters of Basic COCOMO
void calculate(float table[][4], int n,char mode[][15], int size)
{
   float effort,time,staff;
   int model;
   // Check the mode according to size
   if(size>=2 && size<=50)
      model=0;       //organic
   else if(size>50 && size<=300)
      model=1;       //semi-detached
   else if(size>300)
      model=2;       //embedded
   cout<<"The mode is "<<mode[model];
```

```
    // Calculate Effort
    effort = table[model][0]*pow(size,table[model][1]);
    // Calculate Time
    time = table[model][2]*pow(effort,table[model][3]);
    //Calculate Persons Required
    staff = effort/time;
    // Output the values calculated
    cout<<"\nEffort = "<<effort<<" Person-Month";
    cout<<"\nDevelopment Time = "<<time<<" Months";
    cout<<"\nAverage Staff Required = "<<fround(staff)<<" Persons";
}
int main()
{
    float table[3][4]={2.4,1.05,2.5,0.38,3.0,1.12,2.5,0.35,3.6,1.20,2.5,0.32};
    char mode[][15]={"Organic","Semi-Detached","Embedded"};
    int size = 4;
    calculate(table,3,mode,size);
    return 0;
}
```

1. **Output:**
2. The mode is Organic
3. Effort = 10.289 Person-Month

4. Development Time = 6.06237 Months
5. Average Staff Required = 2 Persons

2.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

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
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:

$$\mathrm{E}= (\mathrm{a}(\mathrm{KLOC})^{b}) * EAF$$

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

6. **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:

1. Planning and requirements
2. System design
3. Detailed design
4. Module code and test
5. Integration and test

6. 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.

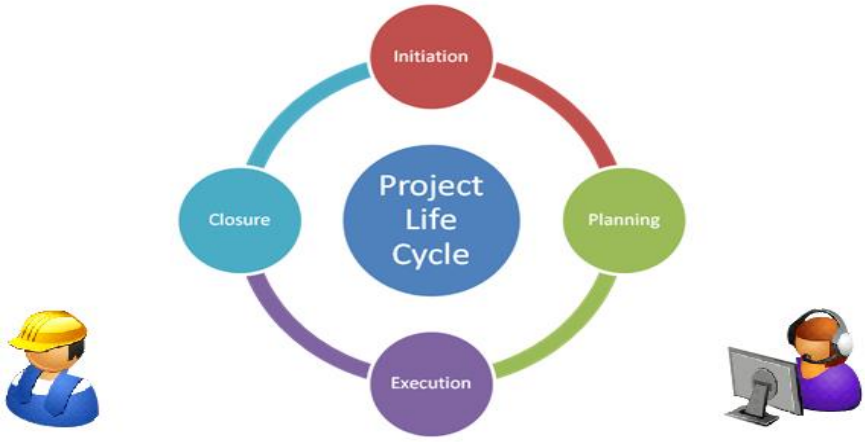
Project Staffing

Project staffing - Flexible Staffing Strategy / Solutions

Companies face multiple challenges in this global economy. To remain competitive in this demanding market, they are constantly adopting latest cutting edge technologies so as to improve services and increase their bottom-line. This technology drive has created huge demand for highly talented and skilled resources for implementing mission critical projects. To improve the profits, these technology initiatives and related projects demand optimum resourcing / staffing strategies.

IT Managers face tough staffing issues with tight deadlines for deliverables. They have insufficient staff and most of them may not be able to take up the development using latest technologies without proper training. This calls for a staffing plan/strategy to support the flexible demands of the projects during the entire life cycle using outsourced services. **The purpose of the staffing plan is to ensure that the project has sufficient resources with the right skills based on project schedule for successful completion.**

When developed as a strategy, flexible project staffing can cut cost, maintain quality and help accomplish specific goals. Better Minds Consulting has the expertise and capabilities to support mission critical project staffing needs. Our best practices approach starts off with an assessment of our client project needs, processes, and technologies. Our team, after careful evaluation, recommends a staffing plan to support the end-to end needs of the project implementation. Our staffing solutions offer flexibility, scalability, right mix and competitive pricing.



Project Staffing Plan

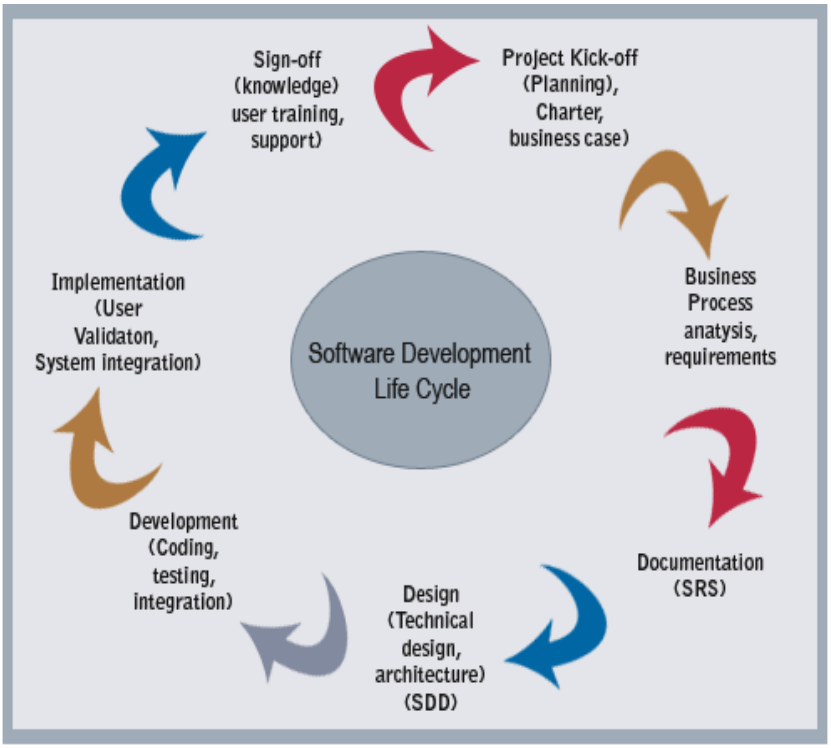
Primary task of our team is to build an effective project staffing plan to identify the following:

1. Skills Matrix - Required skills for each task in the project structure
2. Estimate man hours for each task
3. Resource matrix – required resources (internal and outsourced) - Training of in-house team, hiring, supplemental staffing using IT staffing vendors, etc. Planned hiring of new technical staff can help in transition. Critical staff has to be identified and managed carefully.
4. Resource Deployment Matrix - Resource start date and sign off date (optimum utilization) – Gradual increase / decrease of resources, rotation and re-deployment based on project delivery schedules / milestones. Right mix will ensure both internal staff and outsourced resources are deployed in a blended mode to avoid unexpected departure or absence of outside consultants.
5. Resource / Risk Management – Manage / monitor resources periodically and avoid project risks due to resource issues. Absence of skilled resources when required will lead to project delays and cost escalation. Project staffing needs careful planning as it involves lead time to source, screen, and successfully deploy the right technical and management resources. As

part of risk and continuity management, full time internal staff must be allowed to grow during the implementation cycle so that knowledge gained is not lost.

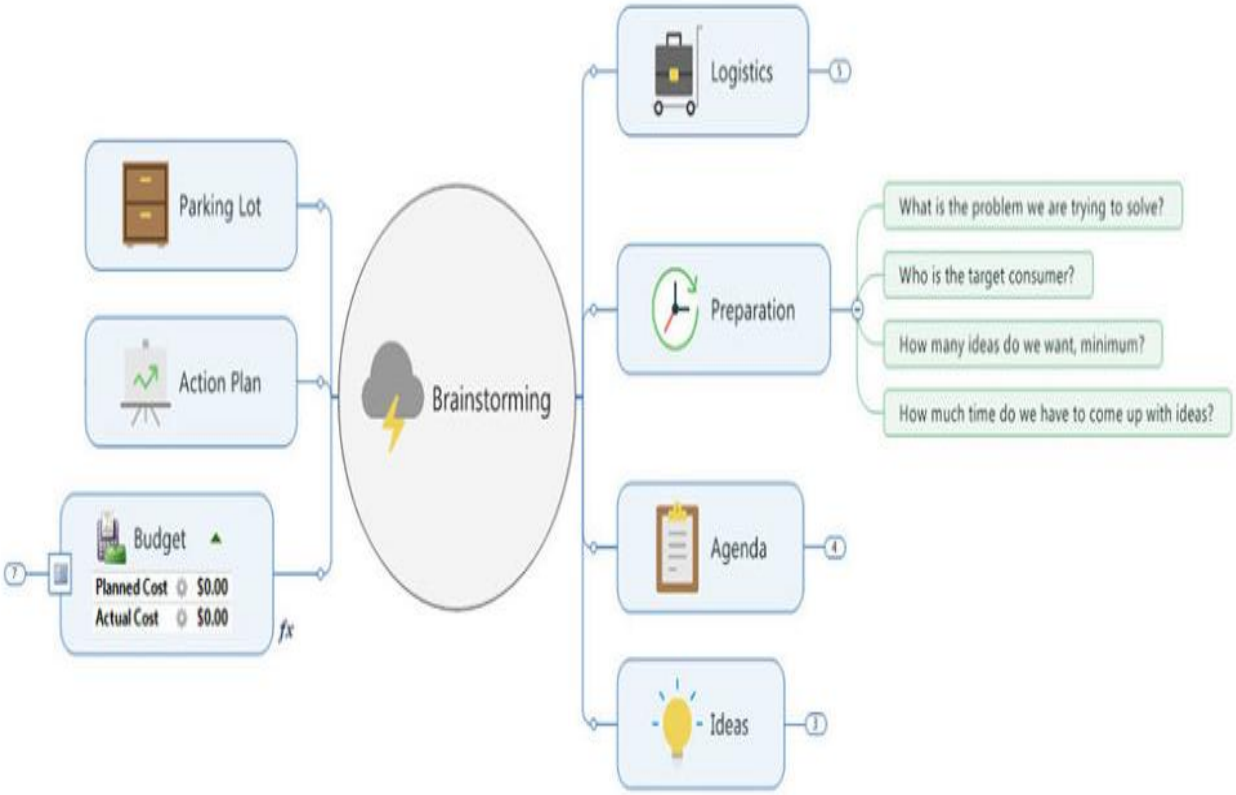
6. A check list with all the above factors will help the program managers / project managers / HR to plan the internal and outsourcing staffing requirements, manage critical staff, transition planning, knowledge transfer and deliver projects on time within budget.

Better Minds Consulting provides end-to-end staffing services for software projects from **"kick-off to sign-off", also called SDLC (Software Development Life Cycle) as shown below.**



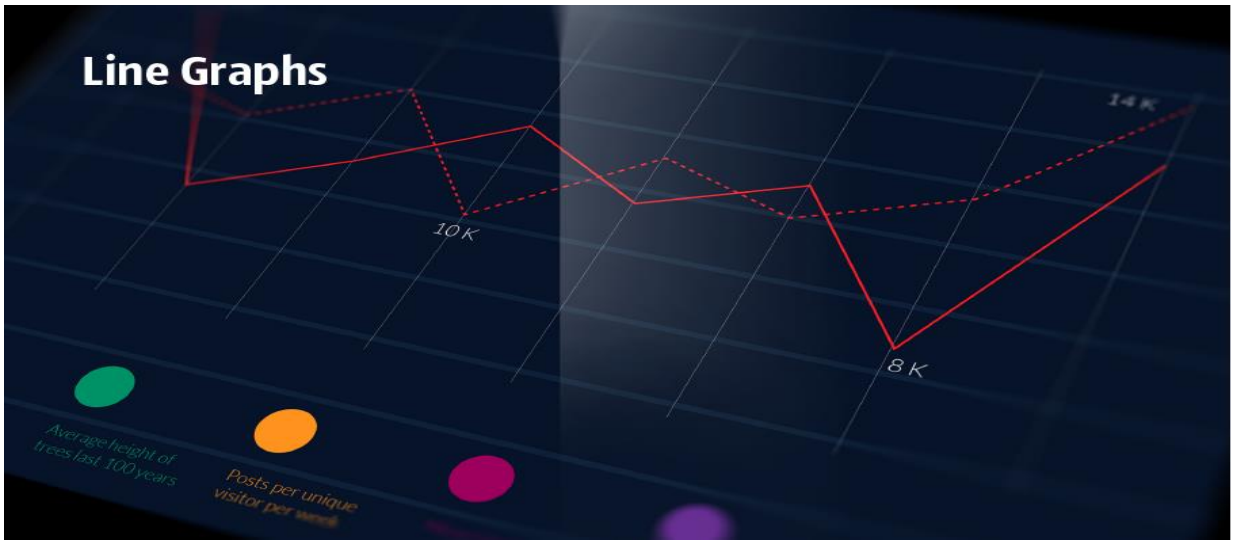
Timeline Chart

A timeline chart is an effective way to visualize a process using chronological order. Since details are displayed graphically, important points in time can be easy seen and understood.
Often used for managing a project's schedule, timeline charts function as a sort of calendar of events within a specific period of time.



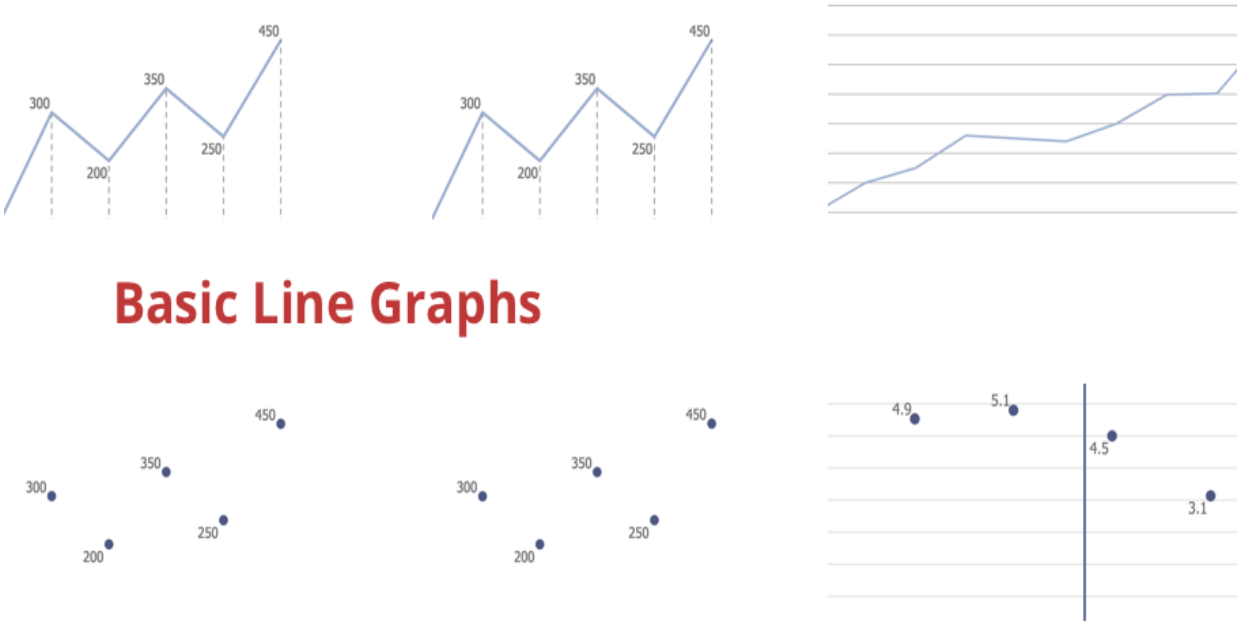
Timeline Diagrams solution extends ConceptDraw PRO diagramming software with samples, templates and libraries of specially developed smart vector design elements of timeline bars, time intervals, milestones, events, today marker, legend, and many other elements with custom properties, helpful for professional drawing clear and intuitive Timeline Diagrams. Use it to design simple Timeline Diagrams and designate only the main project stages, or highly detailed Timeline Diagrams depicting all project tasks, subtasks, milestones, deadlines, responsible persons and other description information, ideal for applying in presentations, reports, education materials, and other documentation.

Line Graphs



How to draw a Line Graph with ease? The Line Graphs solution extends the capabilities of ConceptDraw PRO v10 with professionally designed templates, samples, and a library of vector stencils for drawing perfect Line Graphs.

Basic Line Graphs

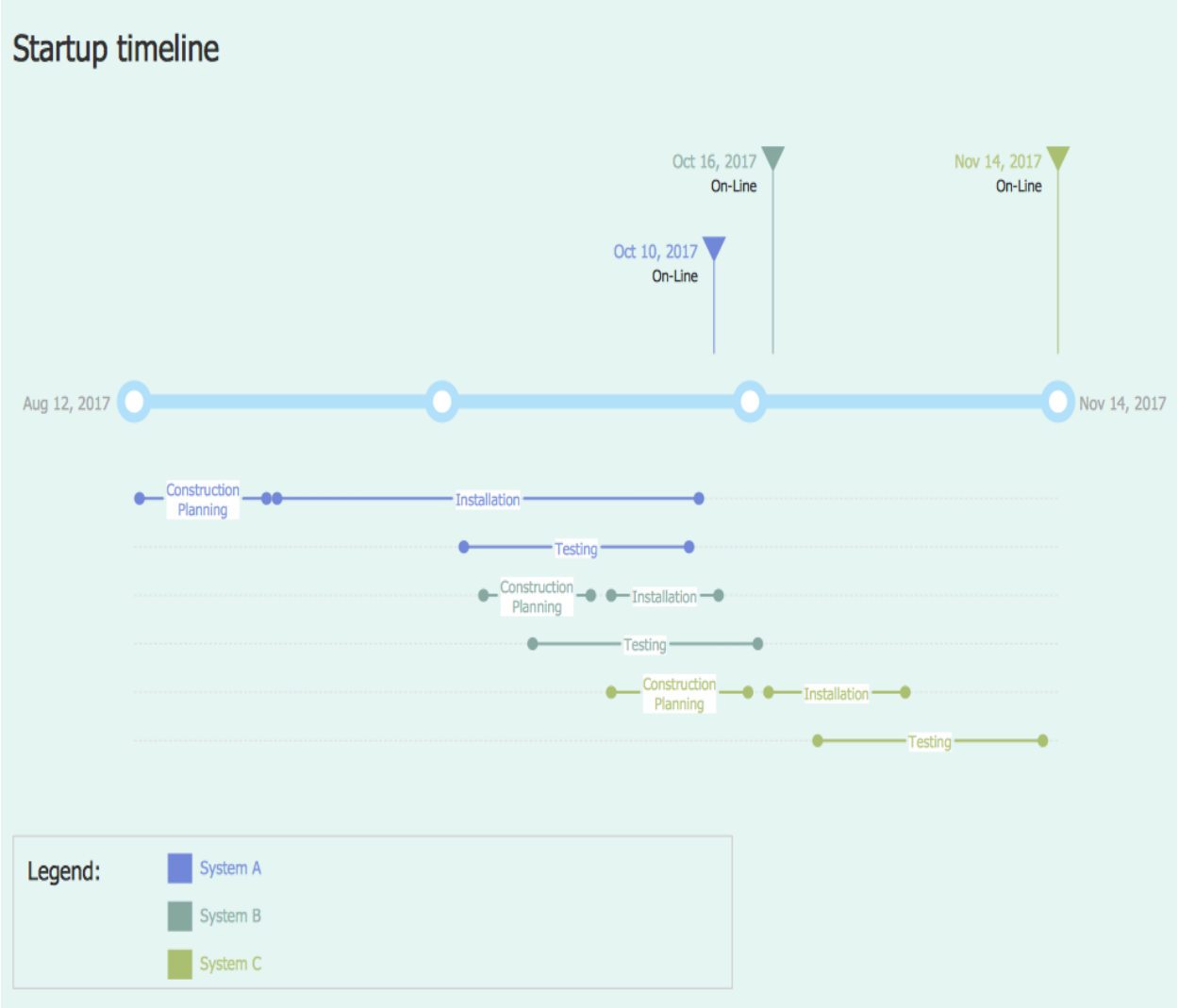


This solution extends the capabilities of ConceptDraw PRO v10.3.0 (or later) with templates, samples and a library of vector stencils for drawing Line Graphs.

Software Engineer Package

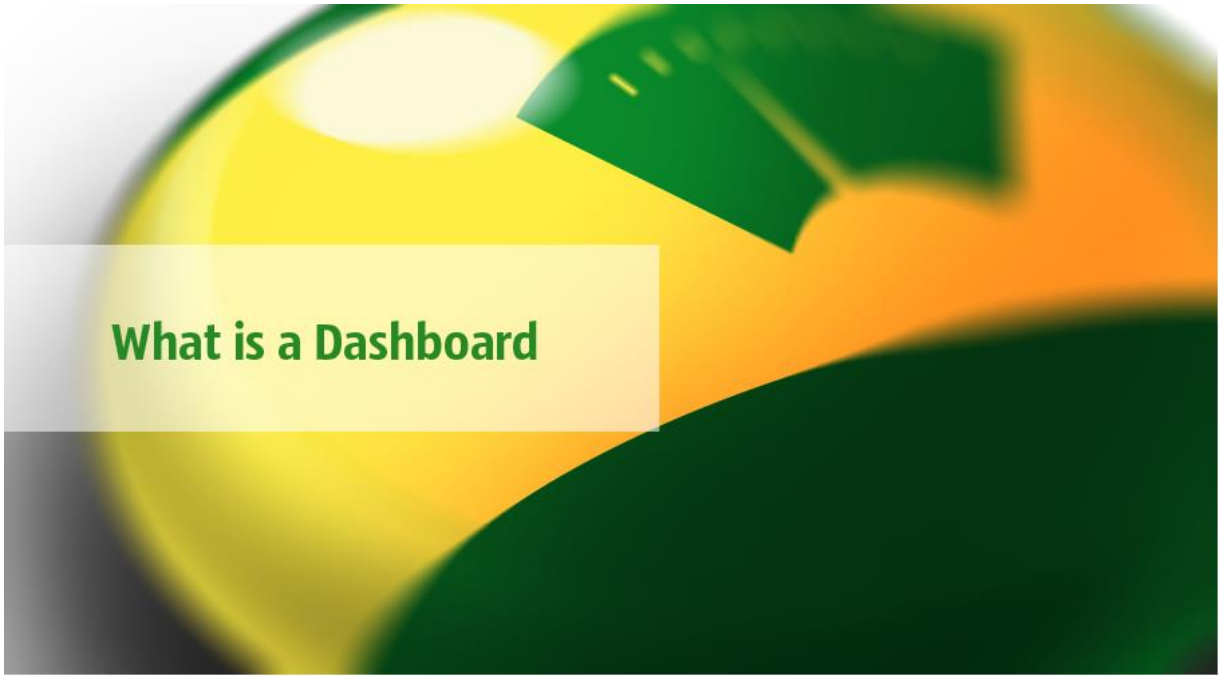


How to Create a Timeline Diagram in ConceptDraw PRO



A Timeline is a chart which displays a project plan schedule in chronological order. A Timeline is used in project management to depict project milestones and visualize project phases, and show project progress. The graphic form of a timeline makes it easy to understand critical project milestones, such as the progress of a project schedule. Timelines are particularly useful for project scheduling or project management when accompanied with a Gantt chart. It captures the main of what the project will accomplish and how it will be done. making a project timeline is one of the most important skills a project manager needs have. Making comprehensive, accurate timelines allows you getting every project off in the best way. ConceptDraw PRO allows you to draw a timeline charts using special library.

Time Series Dashboard



Time series dashboard solution extends ConceptDraw PRO software with templates, samples and vector stencils libraries with charts, graphs and plots for drawing the visual dashboards showing data time series.

Basic Area Charts

This solution extends the capabilities of ConceptDraw PRO v10.3.0 (or later) with templates, samples and a library of vector stencils for drawing Area Charts (area graphs) that visualize changes in values by filling in the area beneath the line connecting data points.

Software Testing



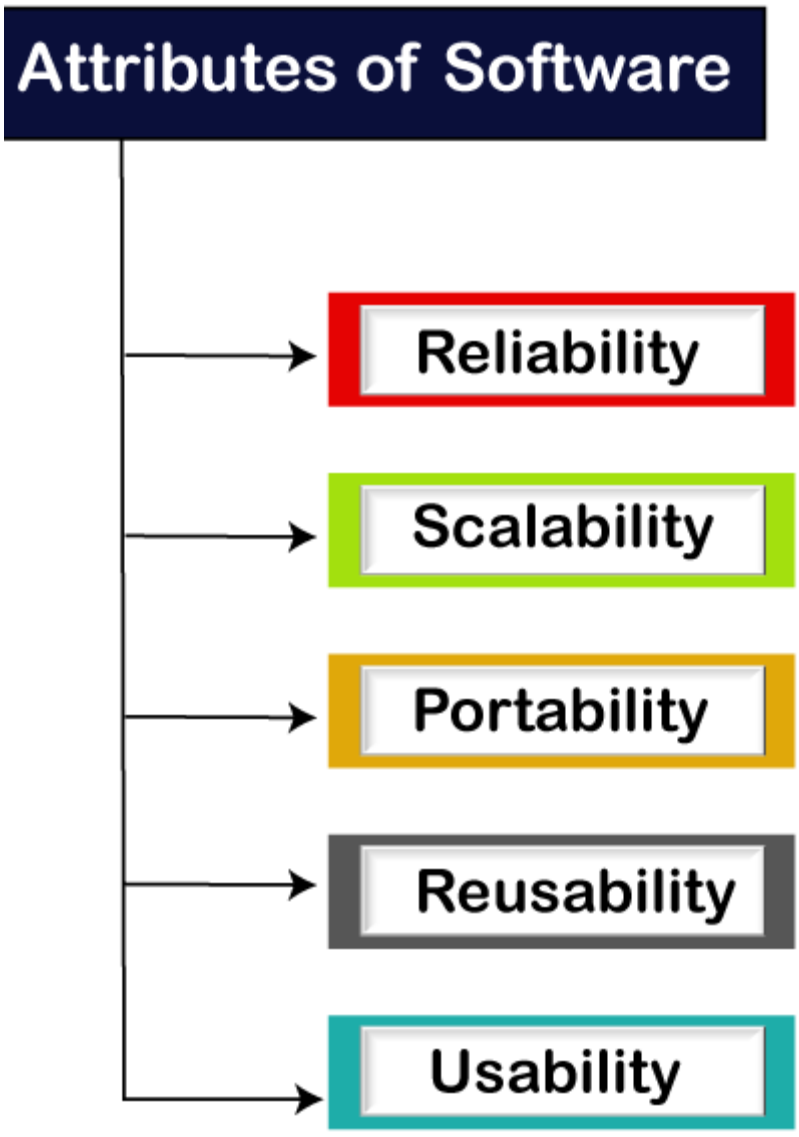
Software testing is widely used technology because it is compulsory to test each and every software before deployment.

Our Software testing tutorial includes all topics of Software

testing such as Methods such as Black Box Testing, White Box Testing, Visual Box Testing and Gray Box Testing. Levels such as Unit Testing, Integration Testing, Regression Testing, Functional Testing. System Testing, Acceptance Testing, Alpha Testing, Beta Testing, Non-Functional testing, Security Testing, Portability Testing.

Software Testing

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.



Software testing provides an independent view and objective of the software and gives surety of fitness of the software. It involves testing of all components under the required services to confirm that whether it is satisfying the specified requirements or not. The process is also providing the client with information about the quality of the software.

Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed to the end user.

What is Testing

Testing is a group of techniques to determine the correctness of the application under the predefined script but, testing cannot find all the defect of application. The main intent of testing is to detect failures of the application so that failures can be discovered and corrected. It does not demonstrate that a product functions properly under all conditions but only that it is not working in some specific conditions.

Testing furnishes comparison that compares the behavior and state of software against mechanisms because the problem can be recognized by the mechanism. The mechanism may include past versions of the same specified product, comparable products, and interfaces of expected purpose, relevant standards, or other criteria but not limited up to these.

Testing includes an examination of code and also the execution of code in various environments, conditions as well as all the examining aspects of the code. In the current scenario of software development, a testing team may be separate from the development team so that Information derived from testing can be used to correct the process of software development.

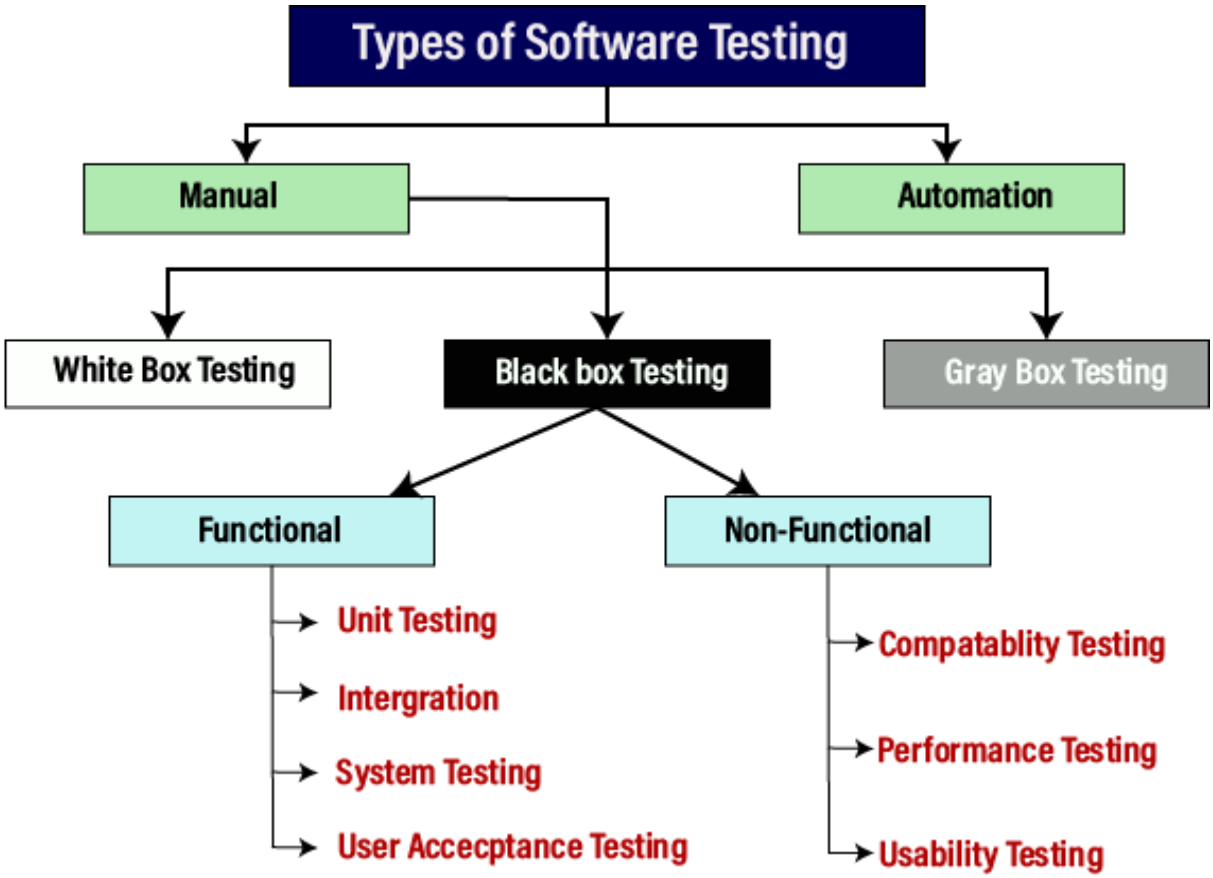
The success of software depends upon acceptance of its targeted audience, easy graphical user interface, strong functionality load test, etc. For example, the audience of banking is totally different from the audience of a video game. Therefore, when an organization develops a software product, it can assess

whether the software product will be beneficial to its purchasers and other audience.

Type of Software testing

We have various types of testing available in the market, which are used to test the application or the software.

With the help of below image, we can easily understand the type of software testing:



Manual testing

The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing. While performing the manual testing on any application, we do not need any specific knowledge of any testing tool, rather than have a proper understanding of the product so we can easily prepare the test document.

Manual testing can be further divided into three types of testing, which are as follows:

- **White box testing**
- **Black box testing**
- **Gray box testing**

Automation testing

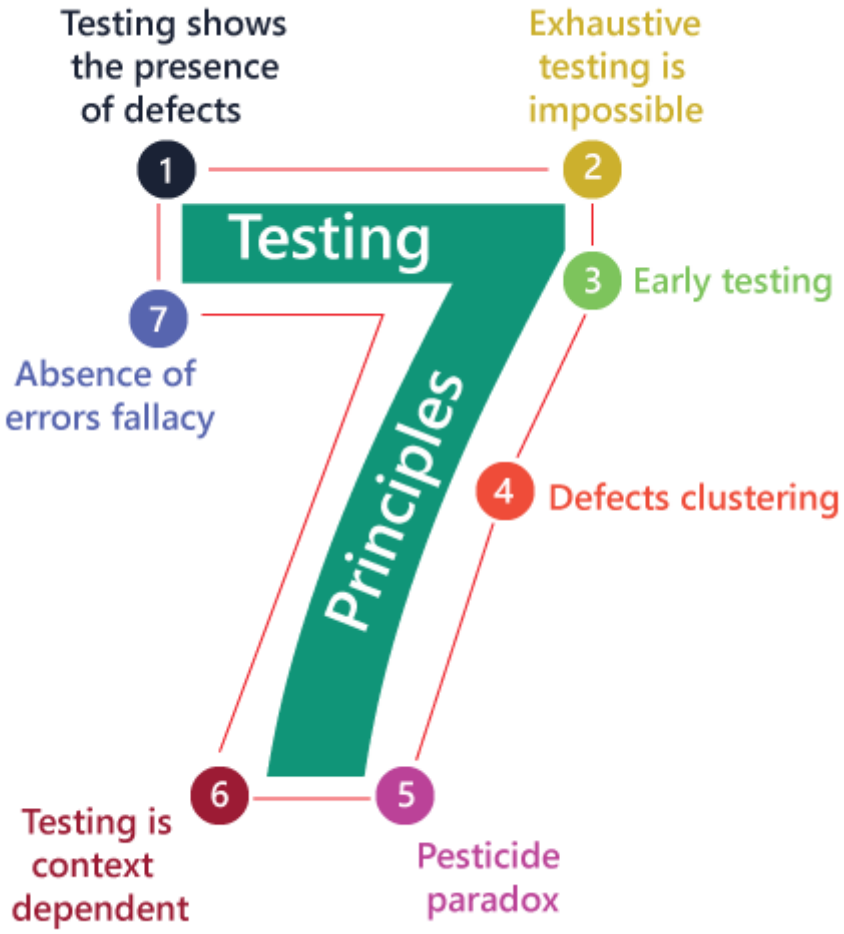
Automation testing is a process of converting any manual test cases into the test scripts with the help of automation tools, or any programming language is known as automation testing. With the help of automation testing, we can enhance the speed of our test execution because here, we do not require any human efforts. We need to write a test script and execute those scripts.

Software Testing Principles

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time. Here, in this section, we are going to learn about the seven essential principles of software testing.

The seven different testing principles, one by one:

- Testing shows the presence of defects
- Exhaustive Testing is not possible
- Early Testing
- Defect Clustering
- Pesticide Paradox
- Testing is context-dependent
- Absence of errors fallacy



Testing shows the presence of defects

The test engineer will test the application to make sure that the application is bug or defects free. While doing testing, we can only identify that the application or software has any errors. The primary purpose of doing testing is to identify the numbers of unknown bugs with the help of various methods and testing techniques because the entire test should be traceable to the customer requirement, which means that to find any defects that might cause the product failure to meet the client's needs.

By doing testing on any application, we can decrease the number of bugs, which does not mean that the application is defect-free because sometimes the software seems to be bug-free while performing multiple types of testing on it.

But at the time of deployment in the production server, if the end-user encounters those bugs which are not found in the testing process.

Exhaustive Testing is not possible

Sometimes it seems to be very hard to test all the modules and their features with effective and non- effective combinations of the inputs data throughout the actual testing process.

Hence, instead of performing the exhaustive testing as it takes boundless determinations and most of the hard work is unsuccessful. So we can complete this type of variations according to the importance of the modules because the product timelines will not permit us to perform such type of testing scenarios.

Early Testing

Here early testing means that all the testing activities should start in the early stages of the software development life cycle's **requirement analysis stage** to identify the defects because if we find the bugs at an early stage, it will be fixed in the initial stage itself, which may cost us very less as compared to those which are identified in the future phase of the testing process.

To perform testing, we will require the requirement specification documents; therefore, if the requirements are defined incorrectly, then it can be fixed directly rather than fixing them in another stage, which could be the development phase.

Defect clustering

The defect clustering defined that throughout the testing process, we can detect the numbers of bugs which are correlated to a small number of modules. We have various reasons for this, such as the modules could be complicated; the coding part may be complex, and so on.

These types of software or the application will follow the **Pareto Principle**, which states that we can identify that approx. Eighty percent of the complication is present in 20 percent of the modules. With the help of this, we can find the uncertain modules, but this method has its difficulties if the same tests are

performing regularly, hence the same test will not able to identify the new defects.

Pesticide paradox

This principle defined that if we are executing the same set of test cases again and again over a particular time, then these kinds of the test will not be able to find the new bugs in the software or the application. To get over these pesticide paradoxes, it is very significant to review all the test cases frequently. And the new and different tests are necessary to be written for the implementation of multiple parts of the application or the software, which helps us to find more bugs.

Testing is context-dependent

Testing is a context-dependent principle states that we have multiple fields such as e-commerce websites, commercial websites, and so on are available in the market. There is a definite way to test the commercial site as well as the e-commerce websites because every application has its own needs, features, and functionality. To check this type of application, we will take the help of various kinds of testing, different technique, approaches, and multiple methods. Therefore, the testing depends on the context of the application.

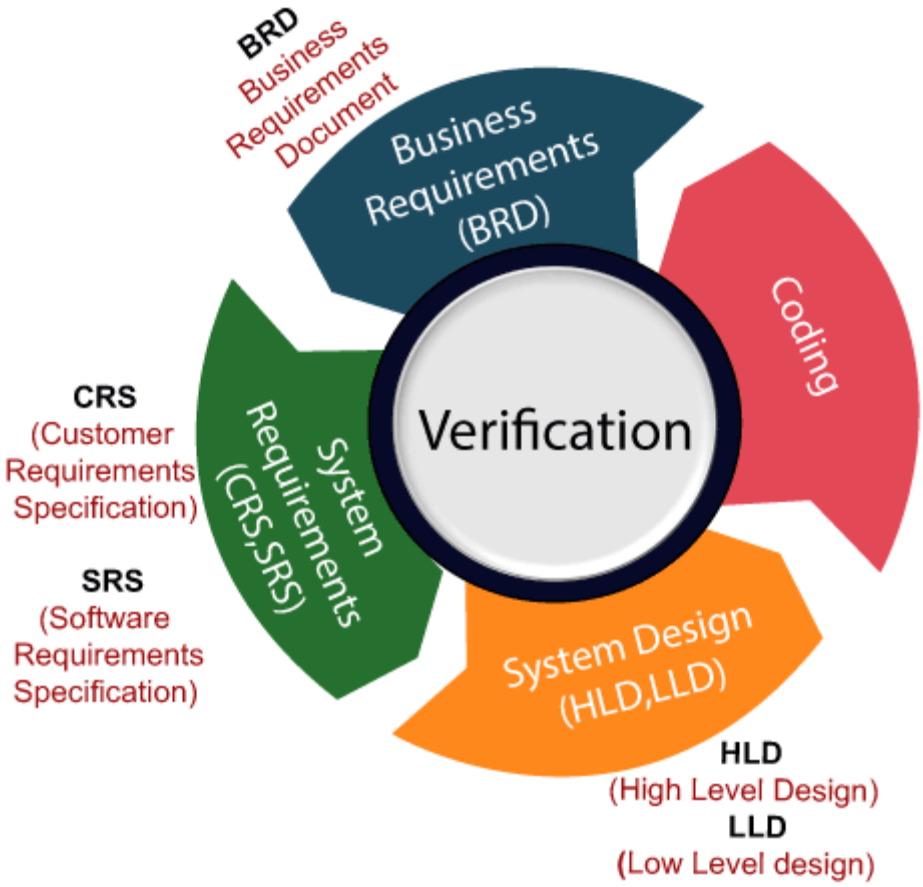
Absence of errors fallacy

Once the application is completely tested and there are no bugs identified before the release, so we can say that the application is 99 percent bug-free. But there is the chance when the application is tested beside the incorrect requirements, identified the flaws, and fixed them on a given period would not help as testing is done on the wrong specification, which does not apply to the client's requirements. The absence of error fallacy means identifying and fixing the bugs would not help if the application is impractical and not able to accomplish the client's requirements and needs.

Verification testing

Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product.

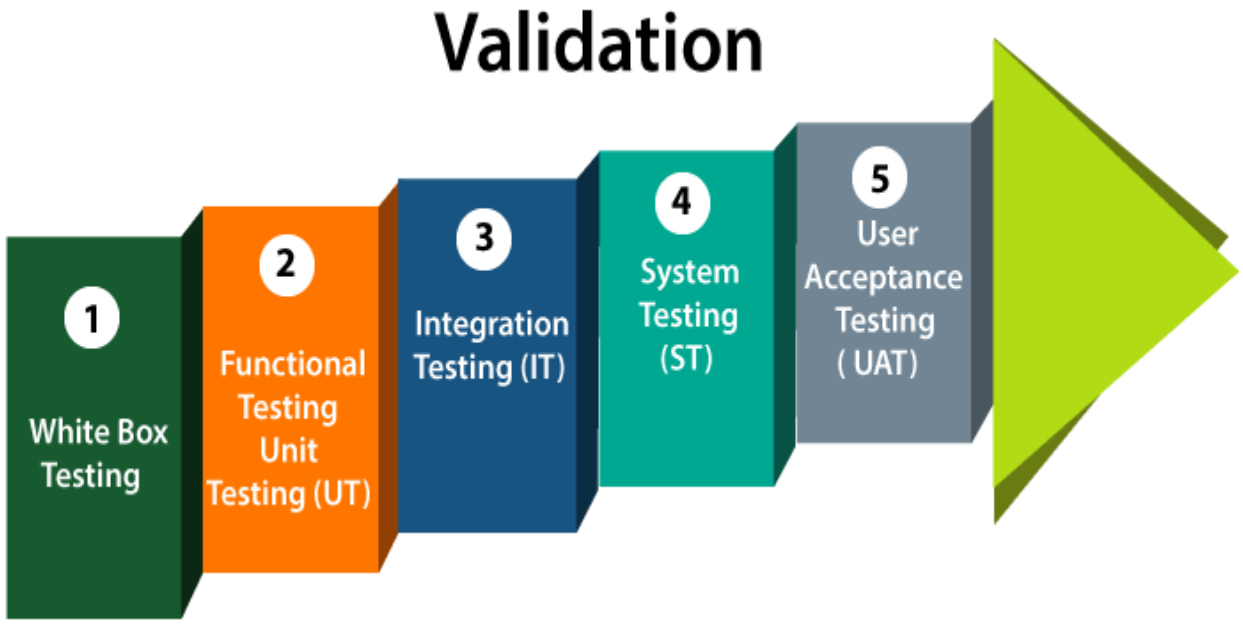
It is also known as static testing, where we are ensuring that "**we are developing the right product or not**". And it also checks that the developed application fulfilling all the requirements given by the client.



Validation testing

Validation testing is testing where tester performed functional and non-functional testing. Here **functional testing** includes Unit Testing (UT), Integration Testing (IT) and System Testing (ST), and **non-functional** testing includes User acceptance testing (UAT).

Validation testing is also known as dynamic testing, where we are ensuring that **"we have developed the product right."** And it also checks that the software meets the business needs of the client.



Note: Verification and Validation process are done under the V model of the software development life cycle.

Difference between verification and validation testing

Verification	Validation
We check whether we are developing the right product or not.	We check whether the developed product is right.
Verification is also known as **static testing**.	Validation is also known as **dynamic testing**.
Verification includes different methods like Inspections, Reviews,	Validation includes testing like functional testing, system testing, integration, and

and Walkthroughs.	User acceptance testing.
It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements.	It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements.
Quality assurance comes under verification testing.	**Quality control** comes under validation testing.
The execution of code does not happen in the verification testing.	In validation testing, the execution of code happens.
In verification testing, we can find the bugs early in the development phase of the product.	In the validation testing, we can find those bugs, which are not caught in the verification process.
Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements.	Validation testing is executed by the testing team to test the application.
Verification is done before the validation testing.	After verification testing, validation testing takes place.
In this type of testing, we can verify that the inputs follow the outputs or	In this type of testing, we can validate that the user accepts the product or not.

not.	

Defect

Software testing lays down the mechanism for finding defects in an application, product or system. A Defect is a deviation between the actual and expected outcomes. A mistake in the source code indicates Error. The error uncovered by a tester becomes a Defect. The defect recognized by the dev team turns into a Bug. If a product build doesn't pass the acceptance criteria, then it proves to be a Failure.

Defect In Manual Testing

A defect is an anomaly which causes a deviation between the expected and actual results. It could be an error discovered once the application got deployed into production. Some software could show potential issues with both the internal and external features. Hence, any variance in the behavior of the functionalities of a product or software got to be a defect in manual testing.

Defect Vs. Wrong, Miss, Error, Bug, Failure, and Fault

In Software Testing, Defect has many names such as Wrong, Miss, Error, Bug, Failure, and Fault. Let's find out what all of these are:

1. Wrong-

Wrong indicates the incorrect implementation of the customer requirements. A manual tester will record it as a defect because of the deviation from the given specification.

2. Miss-

Miss shows that one or more customer requirements got missed in the implementation. It is also a deviation from the original specifications. The tester would also log it as a defect to

3. Error-

An error could arise because of a coding mistake by the developer, he misunderstood, or the requirement was not clear enough. For example, a developer may misinterpret a design flow, or he might use an incompatible data type which results in an Error. Some actions like insufficient array limit, loop getting into the infinite loop or invalid syntax cause to errors. If a tester catches any of these, then he or she will log a defect for the same.

4. Bug-

A bug occurs because of some coding error and leads a program to malfunction. It may also lead to a functional issue in the product. These are fatal errors that could block a functionality, results in a crash, or cause performance bottlenecks. Such errors got treated as Bugs in manual testing.

5. Failure-

A failure indicates a fatal issue in software or in its module which is making the system inoperative or unresponsive. Such an error in the software which the end customer reports are a Failure. The manual testers also discover many failures during the development cycle.

6. Fault-

A fault makes an application behave in a wrong manner.

The problems like an invalid step, lack of resources or inappropriate data definition could cause a fault in a program. It may happen in software because the developer has not added the code for fault tolerance.

Unit Testing

Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing. The aim behind unit testing is to validate unit components with its performance.

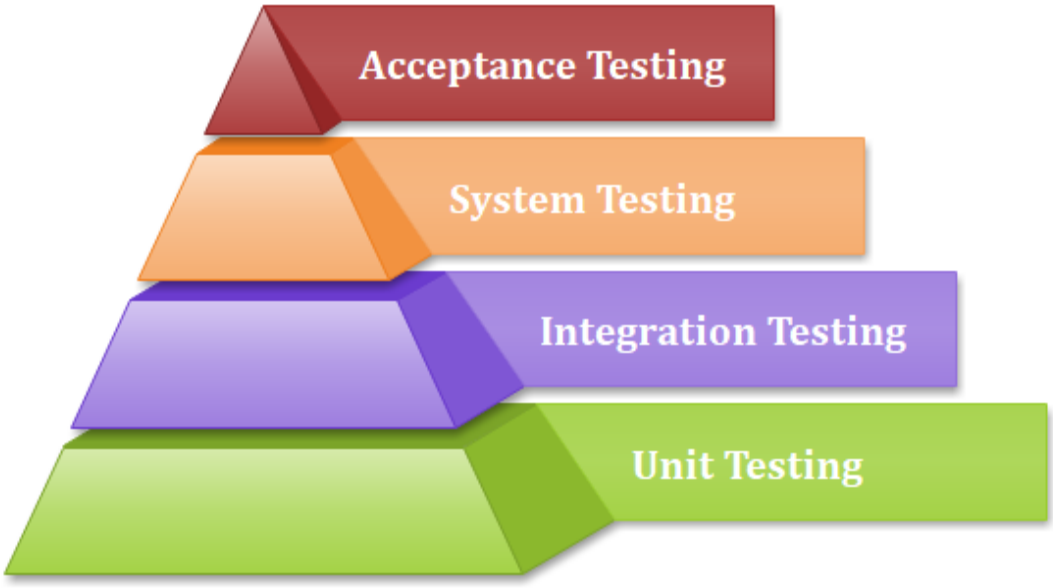
A unit is a single testable part of a software system and tested during the development phase of the application software.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. White box testing approach used for unit testing and usually done by the developers.

Whenever the application is ready and given to the Test engineer, he/she will start checking every component of the module or module of the application independently or one by one, and this process is known as **Unit testing** or **components testing**.

Why Unit Testing?

In a testing level hierarchy, unit testing is the first level of testing done before integration and other remaining levels of the testing. It uses modules for the testing process which reduces the dependency of waiting for Unit testing frameworks, stubs, drivers and mock objects are used for assistance in unit testing.



Generally, **the** software goes under four level of testing: Unit Testing, Integration Testing, System Testing, and Acceptance Testing but sometimes due to time

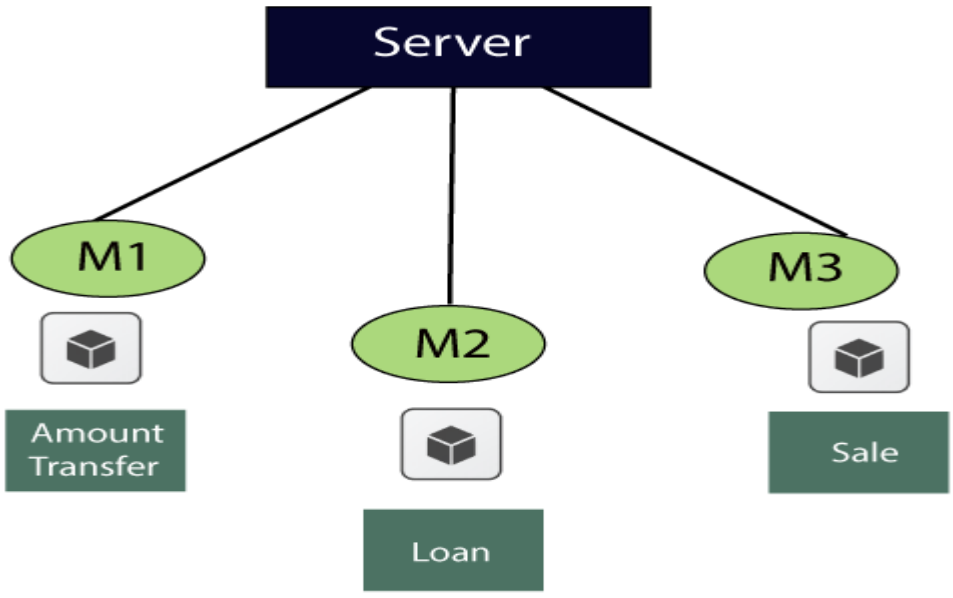
consumption software testers does minimal unit testing but skipping of unit testing may lead to higher defects during Integration Testing, System Testing, and Acceptance Testing or even during Beta Testing which takes place after the completion of software application.

Some crucial reasons are listed below:

- Unit testing helps tester and developers to understand the base of code that makes them able to change defect causing code quickly.
- Unit testing helps in the documentation.
- Unit testing fixes defects very early in the development phase that's why there is a possibility to occur a smaller number of defects in upcoming testing levels.
- It helps with code reusability by migrating code and test cases.

Example of Unit testing

one sample example for a better understanding of the concept of unit testing:



For the **amount transfer,** requirements are as follows:

1.	Amount transfer
1.1	From account number (FAN)→ Text Box
1.1.1	FAN→ accept only 4 digit
1.2	To account no (TAN)→ Text Box
1.2.1	TAN→ Accept only 4 digit
1.3	Amount→ Text Box
1.3.1	Amount → Accept maximum 4 digit
1.4	Transfer→ Button
1.4.1	Transfer → Enabled
1.5	Cancel→ Button
1.5.1	Cancel→ Enabled

Below are the application access details, which is given by the customer

- URL→ login Page
- Username/password/OK → home page
- To reach Amount transfer module follow the below

Loans → sales → Amount transfer

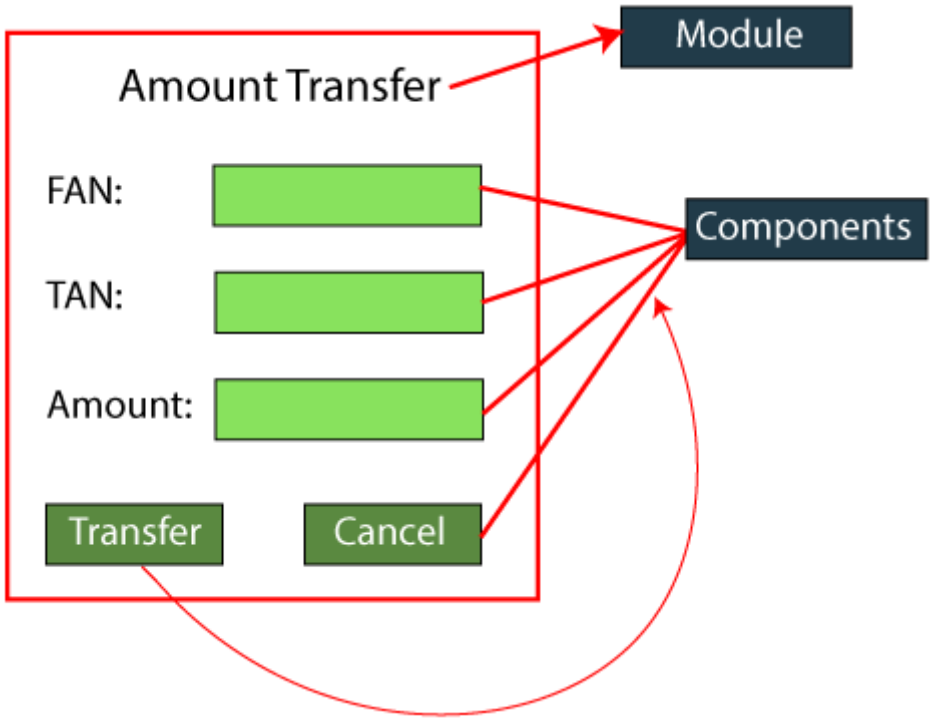
While performing unit testing, we should follow some rules, which are as follows:

- To start unit testing, at least we should have one module.
- Test for positive values
- Test for negative values
- No over testing
- No assumption required

When we feel that the **maximum test coverage** is achieved, we will stop the testing.

Now, we will start performing the unit testing on the different components such as

- From account number(FAN)
- To account number(TAN)
- Amount
- Transfer
- Cancel



For the FAN components

Values	Description
1234	accept
4311	Error message→ account valid or not
blank	Error message→ enter some values
5 digit/ 3 digit	Error message→ accept only 4 digit
Alphanumeric	Error message → accept only digit
Blocked account no	Error message

Copy and paste the value	Error message→ type the value
Same as FAN and TAN	Error message

For the TAN component

- Provide the values just like we did in **From account number** (FAN) components

For Amount component

- Provide the values just like we did in FAN and TAN components.

For Transfer component

- Enter valid FAN value
- Enter valid TAN value
- Enter the correct value of Amount
- Click on the Transfer button→ amount transfer successfully(confirmation message)

For Cancel Component

- Enter the values of FAN, TAN, and amount.
- Click on the Cancel button → all data should be cleared.

Unit Testing Tools

We have various types of unit testing tools available in the market, which are as follows:

- NUnit
- JUnit
- PHPunit

- Parasoft Jtest
- EMMA

Unit Testing Techniques:

Unit testing uses all white box testing techniques as it uses the code of software application:

- Data flow Testing
- Control Flow Testing
- Branch Coverage Testing
- Statement Coverage Testing
- Decision Coverage Testing

How to achieve the best result via Unit testing?

Unit testing can give best results without getting confused and increase complexity by following the steps listed below:

- Test cases must be independent because if there is any change or enhancement in requirement, the test cases will not be affected.
- Naming conventions for unit test cases must be clear and consistent.
- During unit testing, the identified bugs must be fixed before jump on next phase of the SDLC.
- Only one code should be tested at one time.
- Adopt test cases with the writing of the code, if not doing so, the number of execution paths will be increased.
- If there are changes in the code of any module, ensure the corresponding unit test is available or not for that module.

Advantages and disadvantages of unit testing

The pros and cons of unit testing are as follows:

Advantages

- Unit testing uses module approach due to that any part can be tested without waiting for completion of another parts testing.
- The developing team focuses on the provided functionality of the unit and how functionality should look in unit test suits to understand the unit API.
- Unit testing allows the developer to refactor code after a number of days and ensure the module still working without any defect.

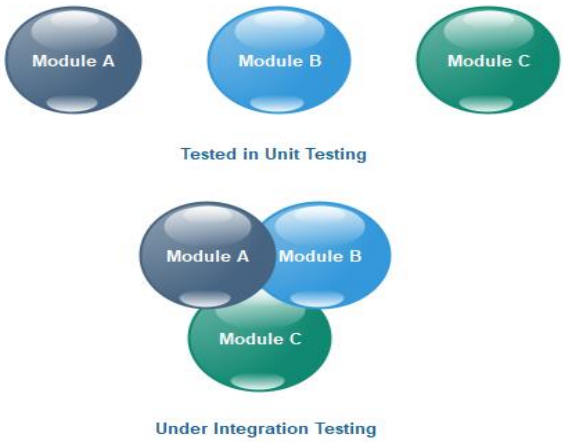
Disadvantages

- It cannot identify integration or broad level error as it works on units of the code.
- In the unit testing, evaluation of all execution paths is not possible, so unit testing is not able to catch each and every error in a program.
- It is best suitable for conjunction with other testing activities.

Integration testing

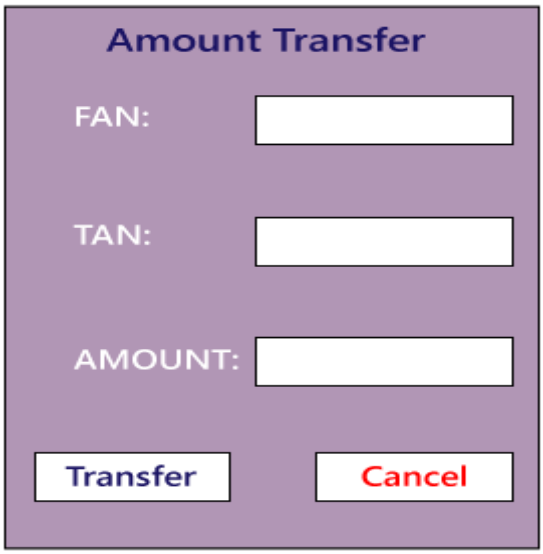
Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.



Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as **integration testing**.

one sample example of a banking application, as we can see in the below image of amount transfer.



- First, we will login as a user **P** to amount transfer and send Rs200 amount, the confirmation message should be displayed on the screen as **amount transfer successfully**. Now logout as P and login as user **Q** and go to amount balance page and check for a balance in that account = Present balance + Received Balance. Therefore, the integration test is successful.

- Also, we check if the amount of balance has reduced by Rs200 in P user account.
- Click on the transaction, in P and Q, the message should be displayed regarding the data and time of the amount transfer.

Guidelines for Integration Testing

- We go for the integration testing only after the functional testing is completed on each module of the application.
- We always do integration testing by picking module by module so that a proper sequence is followed, and also we don't miss out on any integration scenarios.
- First, determine the test case strategy through which executable test cases can be prepared according to test data.
- Examine the structure and architecture of the application and identify the crucial modules to test them first and also identify all possible scenarios.
- Design test cases to verify each interface in detail.
- Choose input data for test case execution. Input data plays a significant role in testing.
- If we find any bugs then communicate the bug reports to developers and fix defects and retest.
- Perform **positive and negative integration testing**.

Here **positive** testing implies that if the total balance is Rs15, 000 and we are transferring Rs1500 and checking if the amount transfer works fine. If it does, then the test would be a pass.

And **negative testing** means, if the total balance is Rs15, 000 and we are transferring Rs20, 000 and check if amount transfer occurs or not, if it does not

occur, the test is a pass. If it happens, then there is a bug in the code, and we will send it to the development team for fixing that bug.

Note: Any application in this world will do functional testing compulsory, whereas integration testing will be done only if the modules are dependent on each other. Each integration scenarios should compulsorily have source→ data→destination. Any scenarios can be called as integration scenario only if the data gets saved in the destination.

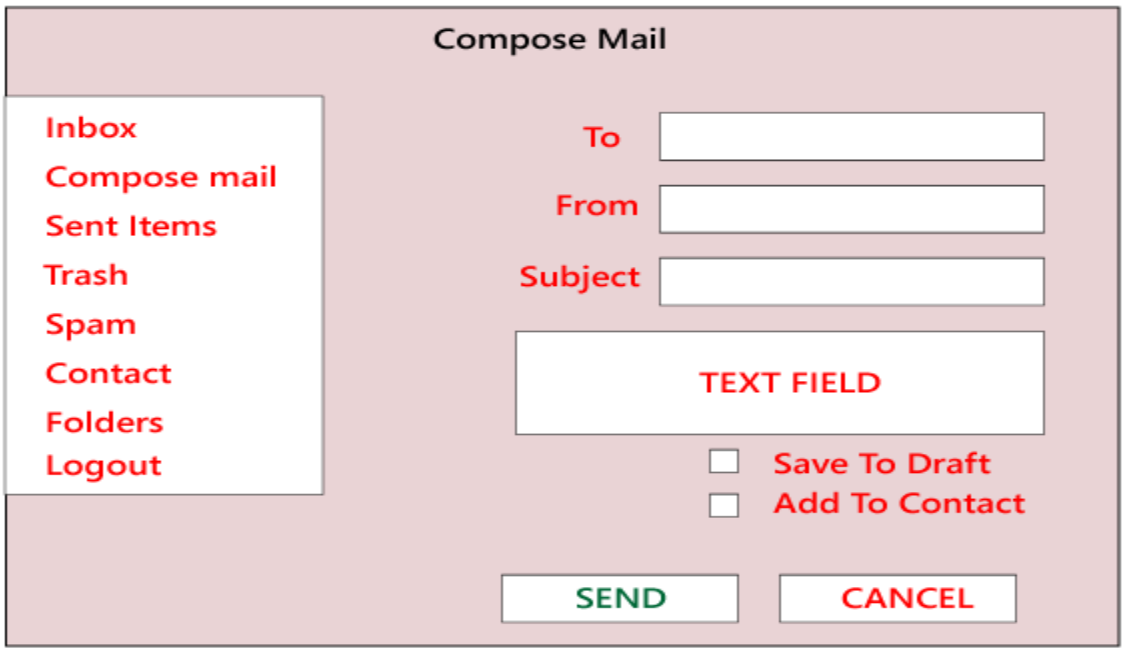
For example: In the Gmail application, the **Source** could be **Compose**, **Data** could be **Email** and the **Destination** could be the **Inbox**.

Example of integration testing

Let us assume that we have a **Gmail** application where we perform the integration testing.

First, we will do **functional testing** on **the login page**, which includes the various components such as **username, password, submit, and cancel** button. Then only we can perform integration testing.

The different integration scenarios are as follows:



Scenarios1:

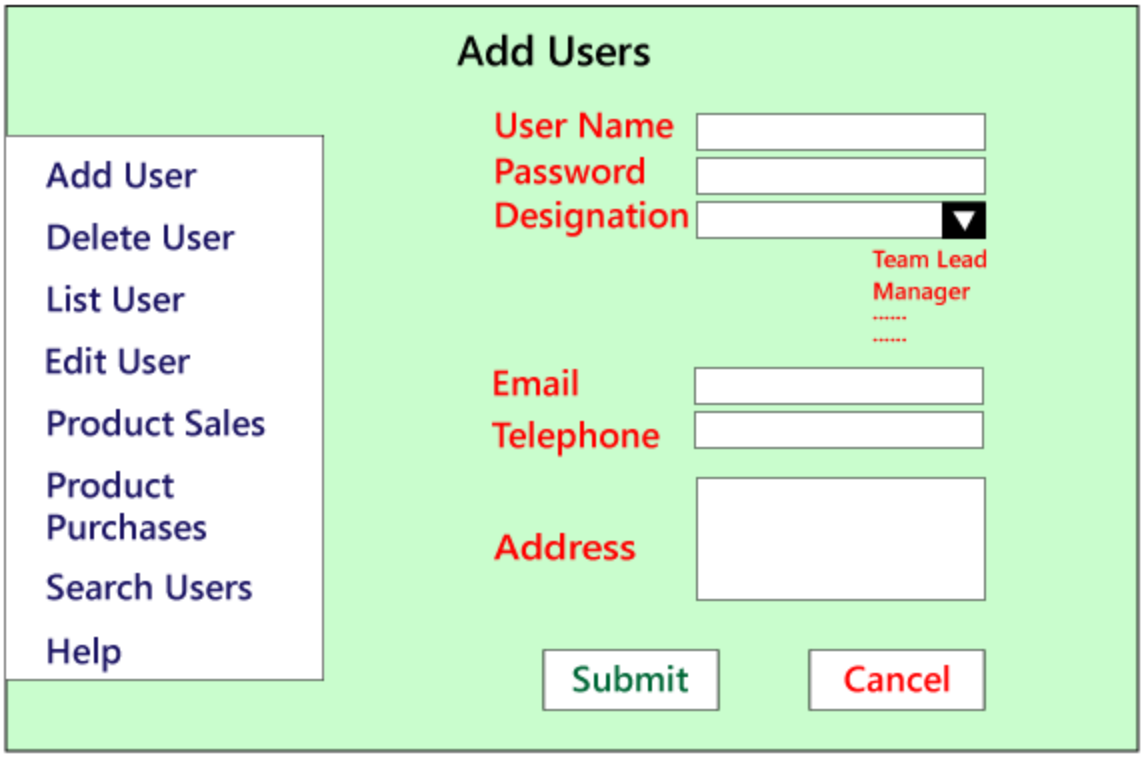
- First, we login as **P** users and click on the **Compose** mail and performing the functional testing for the specific components.
- Now we click on the **Send** and also check for **Save Drafts**.
- After that, we send a **mail** to **Q** and verify in the **Send Items** folder of P to check if the send mail is there.
- Now, we will **log out** as P and login as Q and move to the **Inbox** and verify that if the mail has reached.

Secanrios2: We also perform the integration testing on **Spam** folders. If the particular contact has been marked as spam, then any mail sent by that user should go to the spam folder and not in the inbox.

Note: We will perform functional testing for all features, such as to send items, inbox, and so on.

As we can see in the below image, we will perform the **functional testing** for all the **text fields and every feature**. Then we will perform **integration testing** for the

related functions. We first test the **add user, list of users, delete user, edit user,** and then **search user**.



Note:

- There are some features, we might be performing only the **functional testing**, and there are some features where we are performing both **functional** and **integration testing** based on the feature's requirements.
- **Prioritizing is essential,** and we should perform it at all the phases, which means we will open the application and select which feature needs to be tested first. Then go to that feature and choose which component must be tested first. Go to those components and determine what values to be entered first.
  And don't apply the same rule everywhere because testing logic varies from feature to feature.

- While performing testing, we should test one feature entirely and then only proceed to another function.
- Among the two features, we must be performing **only positive integrating testing** or both **positive and negative integration** testing, and this also depends on the features need.

Reason Behind Integration Testing

Although all modules of software application already tested in unit testing, errors still exist due to the following reasons:

1. Each module is designed by individual software developer whose programming logic may differ from developers of other modules so; integration testing becomes essential to determine the working of software modules.
2. To check the interaction of software modules with the database whether it is an erroneous or not.
3. Requirements can be changed or enhanced at the time of module development. These new requirements may not be tested at the level of unit testing hence integration testing becomes mandatory.
4. Incompatibility between modules of software could create errors.
5. To test hardware's compatibility with software.
6. If exception handling is inadequate between modules, it can create bugs.

Integration Testing Techniques

Any testing technique (Blackbox, Whitebox, and Greybox) can be used for Integration Testing; some are listed below:

Black Box Testing

- State Transition technique

- Decision Table Technique
- Boundary Value Analysis
- All-pairs Testing
- Cause and Effect Graph
- Equivalence Partitioning
- Error Guessing

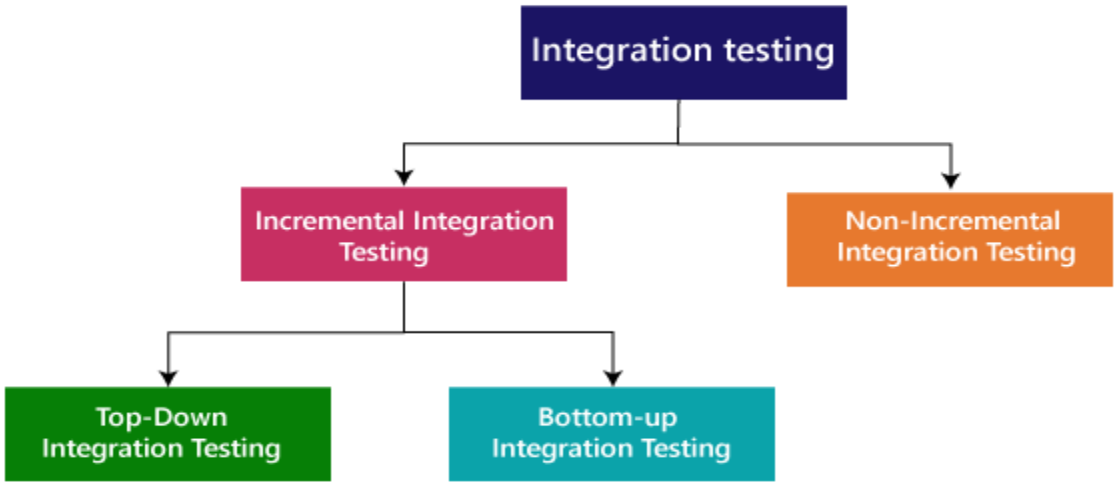
White Box Testing

- Data flow testing
- Control Flow Testing
- Branch Coverage Testing
- Decision Coverage Testing

Types of Integration Testing

Integration testing can be classified into two parts:

- **Incremental integration testing**
- **Non-incremental integration testing**

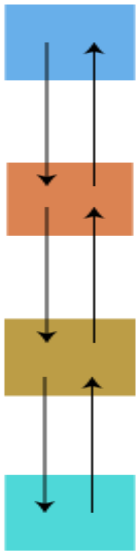


Incremental Approach

In the Incremental Approach, modules are added in ascending order one by one or according to need. The selected modules must be logically related. Generally, two or more than two modules are added and tested to determine the correctness of functions. The process continues until the successful testing of all the modules.

OR

In this type of testing, there is a strong relationship between the dependent modules. Suppose we take two or more modules and verify that the data flow between them is working fine. If it is, then add more modules and test again.



For example: Suppose we have a Flipkart application, we will perform incremental integration testing, and the flow of the application would like this:

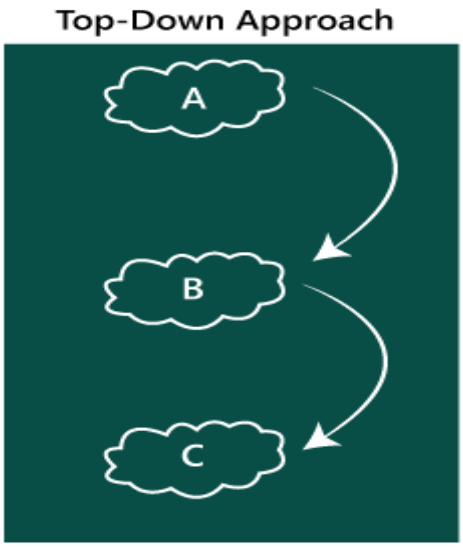
Flipkart→ Login→ Home → Search→ Add cart→Payment → Logout

Incremental integration testing is carried out by further methods:

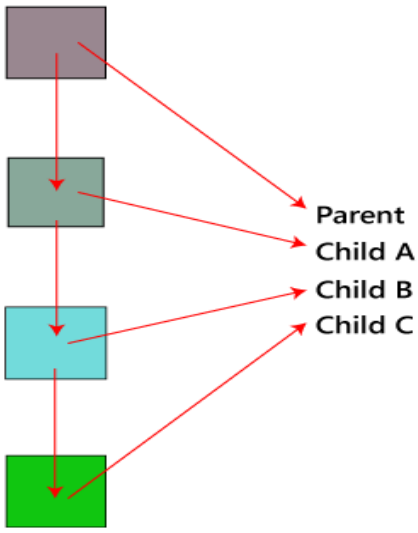
- Top-Down approach
- Bottom-Up approach

Top-Down Approach

The top-down testing strategy deals with the process in which higher level modules are tested with lower level modules until the successful completion of testing of all the modules. Major design flaws can be detected and fixed early because critical modules tested first. In this type of method, we will add the modules incrementally or one by one and check the data flow in the same order.



In the top-down approach, we will be ensuring that the module we are adding is the **child of the previous one like Child C is a child of Child B** and so on as we can see in the below image:



Advantages:

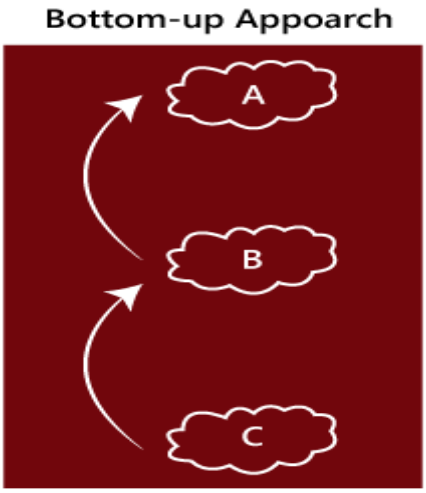
- Identification of defect is difficult.
- An early prototype is possible.

Disadvantages:

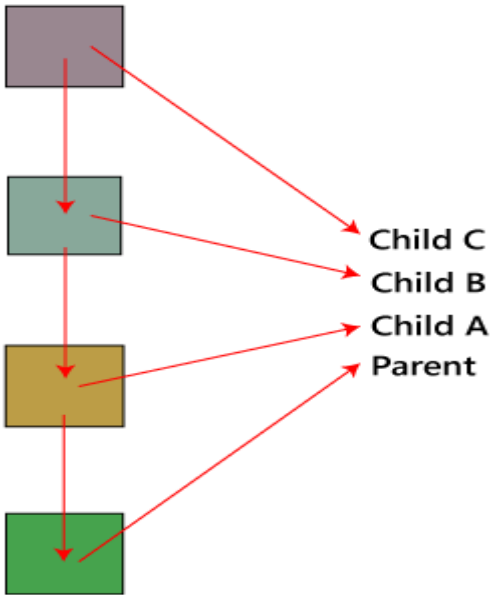
- Due to the high number of stubs, it gets quite complicated.
- Lower level modules are tested inadequately.
- Critical Modules are tested first so that fewer chances of defects.

Bottom-Up Method

The bottom to up testing strategy deals with the process in which lower level modules are tested with higher level modules until the successful completion of testing of all the modules. Top level critical modules are tested at last, so it may cause a defect. Or we can say that we will be adding the modules from **bottom to the top** and check the data flow in the same order.



In the bottom-up method, we will ensure that the modules we are adding **are the parent of the previous one** as we can see in the below image:



Advantages

- Identification of defect is easy.
- Do not need to wait for the development of all the modules as it saves time.

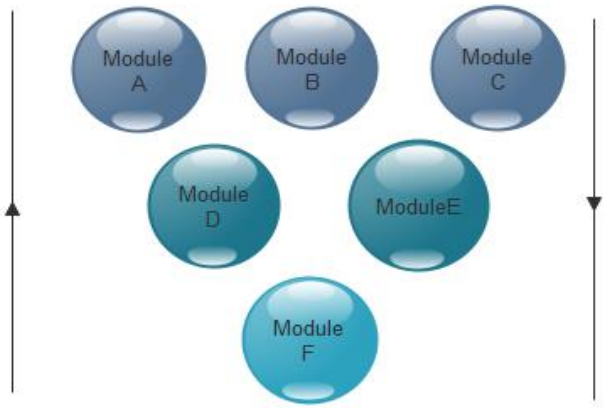
Disadvantages

- Critical modules are tested last due to which the defects can occur.
- There is no possibility of an early prototype.

In this, we have one addition approach which is known as **hybrid testing**.

Hybrid Testing Method

In this approach, both **Top-Down** and **Bottom-Up** approaches are combined for testing. In this process, top-level modules are tested with lower level modules and lower level modules tested with high-level modules simultaneously. There is less possibility of occurrence of defect because each module interface is tested.



Hybrid Method

Advantages

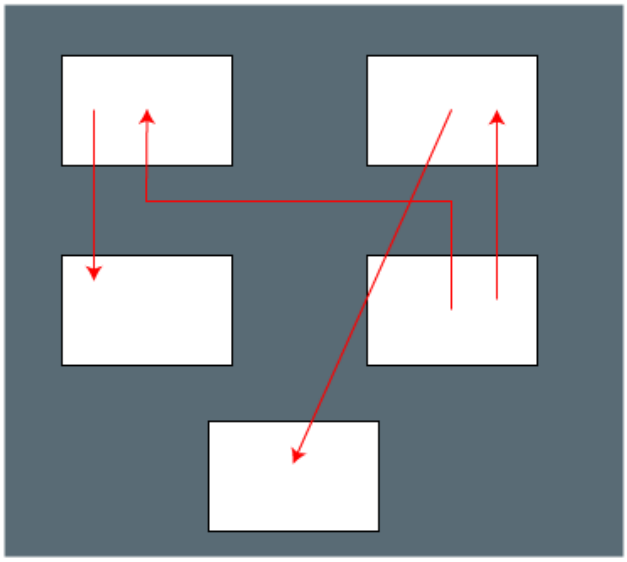
- The hybrid method provides features of both Bottom Up and Top Down methods.
- It is most time reducing method.
- It provides complete testing of all modules.

Disadvantages

- This method needs a higher level of concentration as the process carried out in both directions simultaneously.
- Complicated method.

Non- incremental integration testing

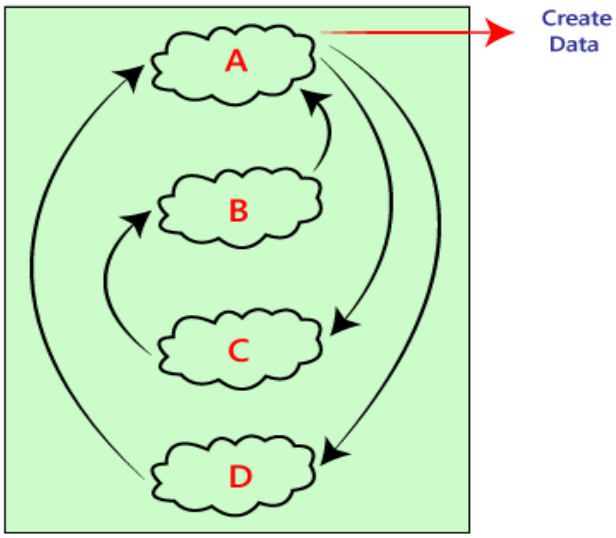
We will go for this method, when the data flow is very complex and when it is difficult to find who is a parent and who is a child. And in such case, we will create the data in any module bang on all other existing modules and check if the data is present. Hence, it is also known as the **Big bang method**.



Big Bang Method

In this approach, testing is done via integration of all modules at once. It is convenient for small software systems, if used for large software systems identification of defects is difficult.

Since this testing can be done after completion of all modules due to that testing team has less time for execution of this process so that internally linked interfaces and high-risk critical modules can be missed easily.



Advantages:

- It is convenient for small size software systems.

Disadvantages:

- Identification of defects is difficult because finding the error where it came from is a problem, and we don't know the source of the bug.
- Small modules missed easily.
- Time provided for testing is very less.
- We may miss to test some of the interfaces.

Let us see examples for our better understanding of the non-incremental integrating testing or big bang method:

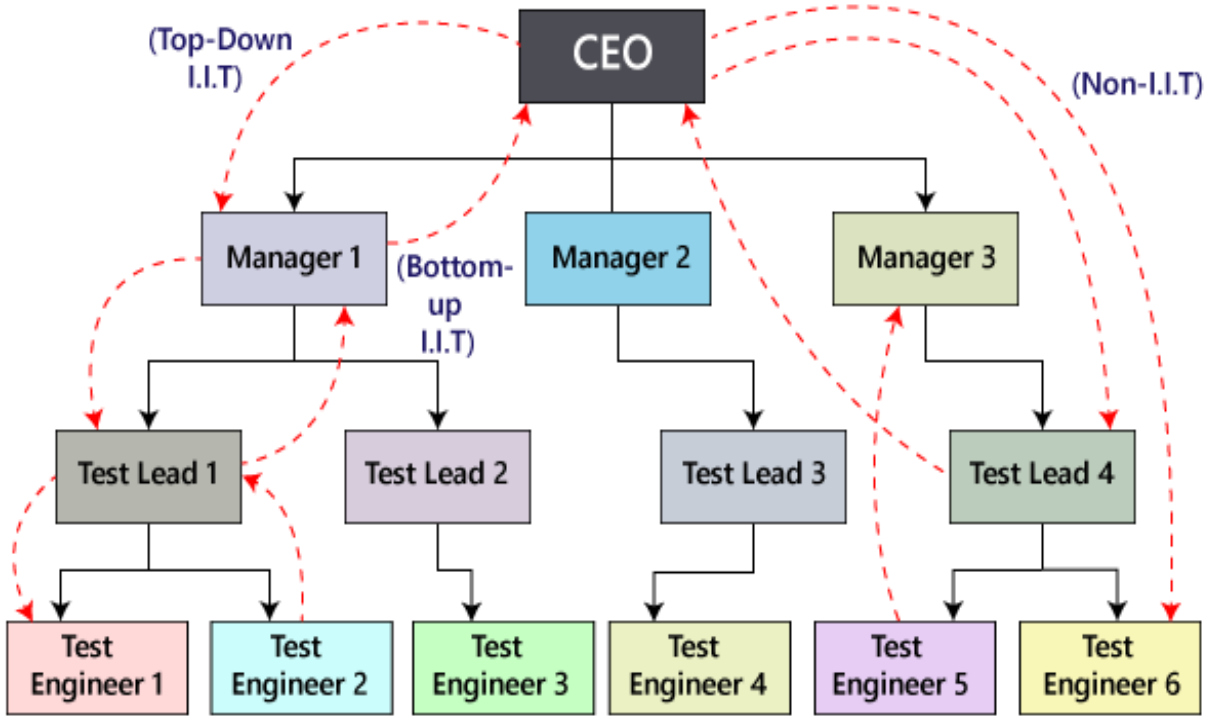
Example1

In the below example, the development team develops the application and sends it to the CEO of the testing team. Then the CEO will log in to the application and generate the username and password and send a mail to the manager. After that, the CEO will tell them to start testing the application.

Then the manager manages the username and the password and produces a username and password and sends it to the **test leads**. And the **test leads** will send it to the **test engineers** for further testing purposes. This order from the CEO to the test engineer is **top-down incremental integrating testing.**

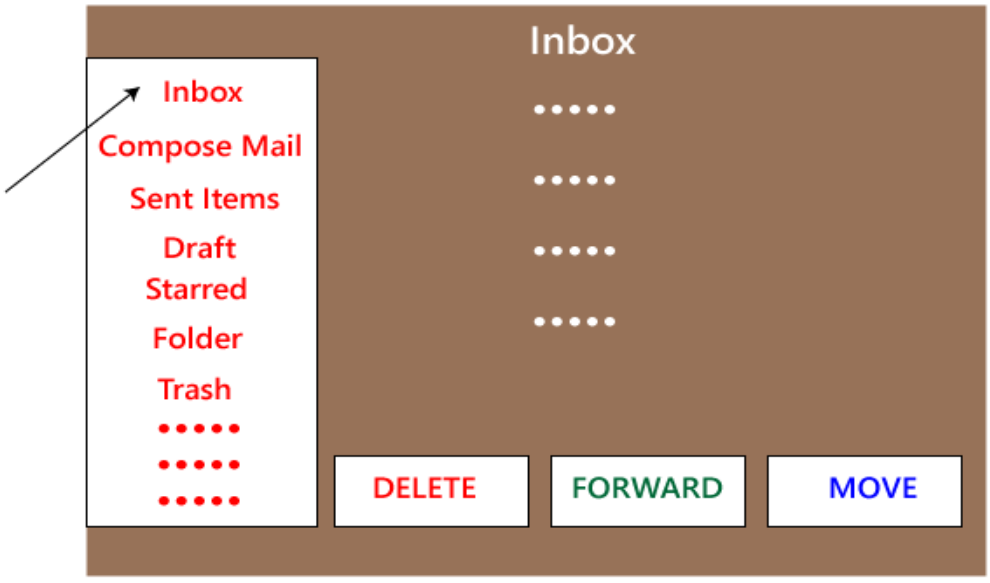
In the same way, when the test engineers are done with testing, they send a report to the **test leads**, who then submit a report to the **manager**, and the manager will send a report to the **CEO**. This process is known as **Bottom-up incremental integration testing** as we can see in the below image:

Note: The combination incremental integration testing (I.I.T) and non-incremental integration testing is known as sandwich testing.



Example2

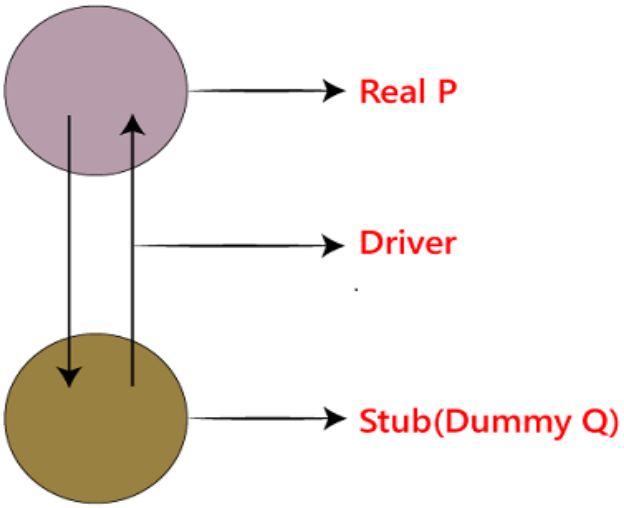
The below example demonstrates a home page of **Gmail's Inbox**, where we click on the **Inbox** link, and we are moved to the inbox page. Here we have to do **non-incremental integration testing** because there is no parent and child concept.



Note

Stub and driver

The **stub** is a dummy module that receives the data and creates lots of probable data, but it performs like a real module. When a data is sent from module P to Stub Q, it receives the data without confirming and validating it, and produce the estimated outcome for the given data.



The function of a driver is used to verify the data from P and sends it to stub and also checks the expected data from the stub and sends it to P.

The **driver** is one that sets up the test environments and also takes care of the communication, evaluates results, and sends the reports. We never use the stub and driver in the testing process.

In **White box testing, bottom-up integration testing** is ideal because writing drivers is accessible. And in **black box testing**, no preference is given to any testing as it depends on the application.

White Box Testing

The box testing approach of software testing consists of black box testing and white box testing. We are discussing here white box testing which also known as glass box is **testing, structural testing, clear box testing, open box testing and**

transparent box testing. It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.

Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the black box testing and verify the application along with the requirements and identify the bugs and sends it to the developer.

The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the particular feature is working fine on the application.

Here, the test engineers will not include in fixing the defects for the following reasons:

- Fixing the bug might interrupt the other features. Therefore, the test engineer should always find the bugs, and developers should still be doing the bug fixes.
- If the test engineers spend most of the time fixing the defects, then they may be unable to find the other bugs in the application.

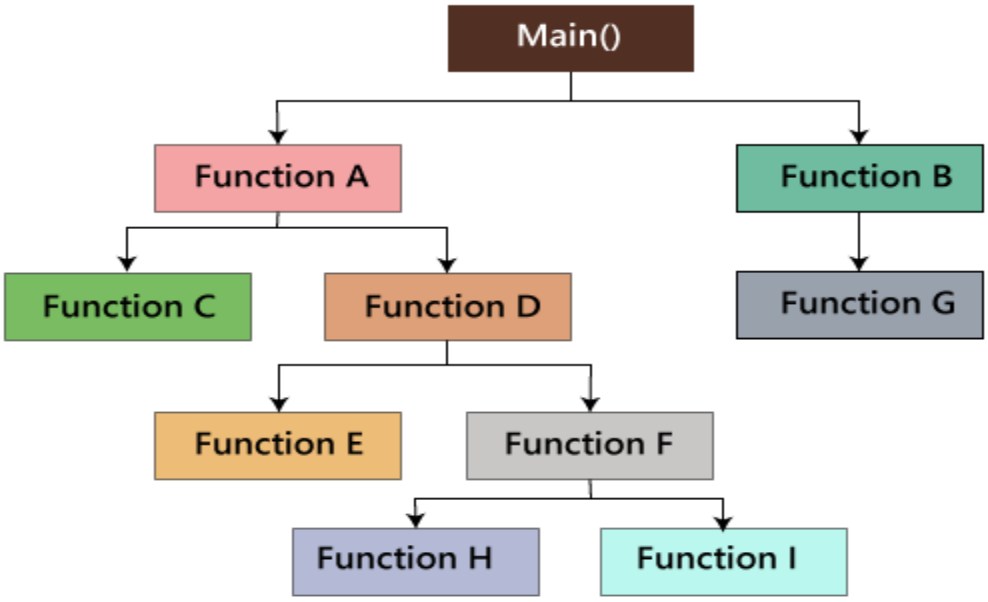
The white box testing contains various tests, which are as follows:

- Path testing
- Loop testing
- Condition testing

- Testing based on the memory perspective
- Test performance of the program

Path testing

In the path testing, we will write the flow graphs and test all independent paths. Here writing the flow graph implies that flow graphs are representing the flow of the program and also show how every program is added with one another as we can see in the below image:



And test all the independent paths implies that suppose a path from main() to function G, first set the parameters and test if the program is correct in that particular path, and in the same way test all other paths and fix the bugs.

Loop testing

In the loop testing, we will test the loops such as while, for, and do-while, etc. and also check for ending condition if working correctly and if the size of the conditions is enough.

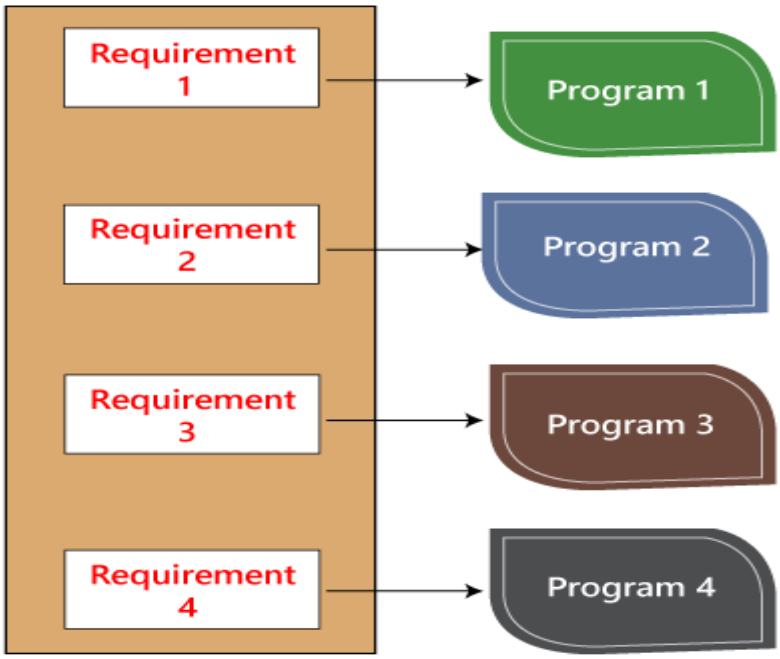
For example: we have one program where the developers have given about 50,000 loops.

1. {
2. while(50,000)
3. ……
4. ……
5. }

We cannot test this program manually for all the 50,000 loops cycle. So we write a small program that helps for all 50,000 cycles, as we can see in the below program, that test P is written in the similar language as the source code program, and this is known as a Unit test. And it is written by the developers only.

1. Test P
2. {
3. ……
4. …… }

As we can see in the below image that, we have various requirements such as 1, 2, 3, 4. And then, the developer writes the programs such as program 1,2,3,4 for the parallel conditions. Here the application contains the 100s line of codes.

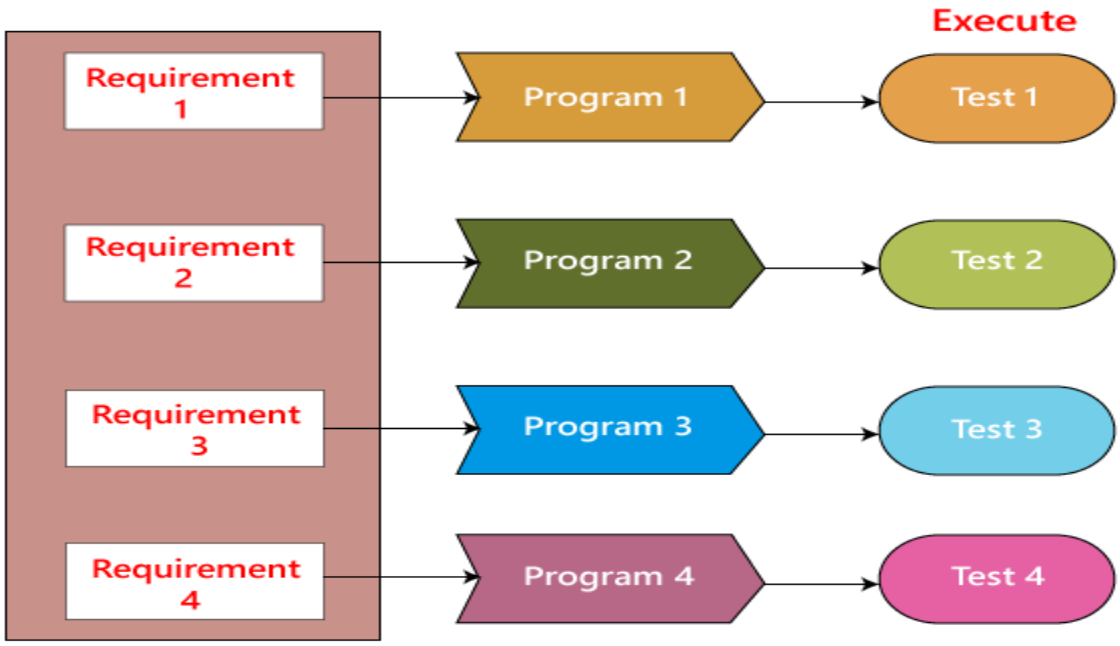


The developer will do the white box testing, and they will test all the five programs line by line of code to find the bug. If they found any bug in any of the programs, they will correct it. And they again have to test the system then this process contains lots of time and effort and slows down the product release time.

Now, suppose we have another case, where the clients want to modify the requirements, then the developer will do the required changes and test all four program again, which take lots of time and efforts.

These issues can be resolved in the following ways:

In this, we will write test for a similar program where the developer writes these test code in the related language as the source code. Then they execute these test code, which is also known as **unit test programs**. These test programs linked to the main program and implemented as programs.



Therefore, if there is any requirement of modification or bug in the code, then the developer makes the adjustment both in the main program and the test program and then executes the test program.

Condition testing

In this, we will test all logical conditions for both **true** and **false** values; that is, we will verify for both **if** and **else** condition.

For example:

1. if(condition) - true
2. {
3. .....
4. ......
5. ......
6. }
7. else - false

8. {
9. …..
10.…….
11.…….
12.}

The above program will work fine for both the conditions, which means that if the condition is accurate, and then else should be false and conversely.

Testing based on the memory (size) perspective

The size of the code is increasing for the following reasons:

- **The reuse of code is not there**: let us take one example, where we have four programs of the same application, and the first ten lines of the program are similar. We can write these ten lines as a discrete function, and it should be accessible by the above four programs as well. And also, if any bug is there, we can modify the line of code in the function rather than the entire code.
- The **developers use the logic** that might be modified. If one programmer writes code and the file size is up to 250kb, then another programmer could write a similar code using the different logic, and the file size is up to 100kb.
- The **developer declares so many functions and variables** that might never be used in any portion of the code. Therefore, the size of the program will increase.

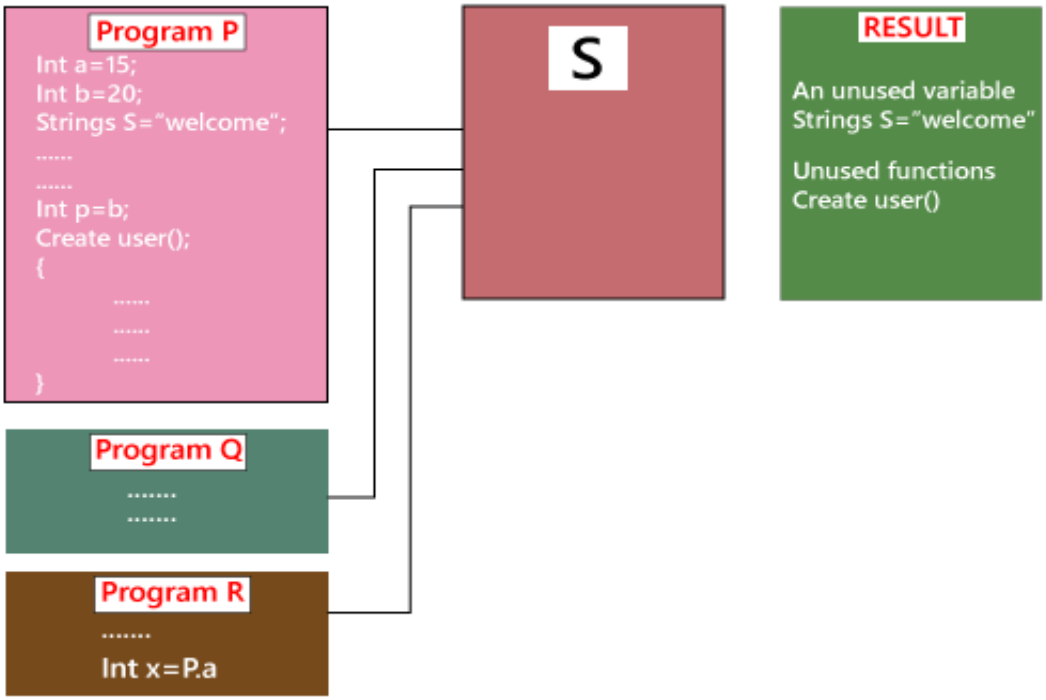
For example,

1. Int a=15;
2. Int b=20;

```
3. String S= "Welcome";
4. ….
5. …..
6. …..
7. ….
8. …..
9. Int p=b;
10.Create user()
11.{
12.…….
13.…….
14.…… 200's line of code
15.}
```

In the above code, we can see that the **integer a** has never been called anywhere in the program, and also the function **Create user** has never been called anywhere in the code. Therefore, it leads us to memory consumption.

We cannot remember this type of mistake manually by verifying the code because of the large code. So, we have a built-in tool, which helps us to test the needless variables and functions. And, here we have the tool called **Rational purify**.



Suppose we have three programs such as Program P, Q, and R, which provides the input to S. And S goes into the programs and verifies the unused variables and then gives the outcome. After that, the developers will click on several results and call or remove the unnecessary function and the variables.

This tool is only used for the C programming language and C++ programming language; for another language, we have other related tools available in the market.

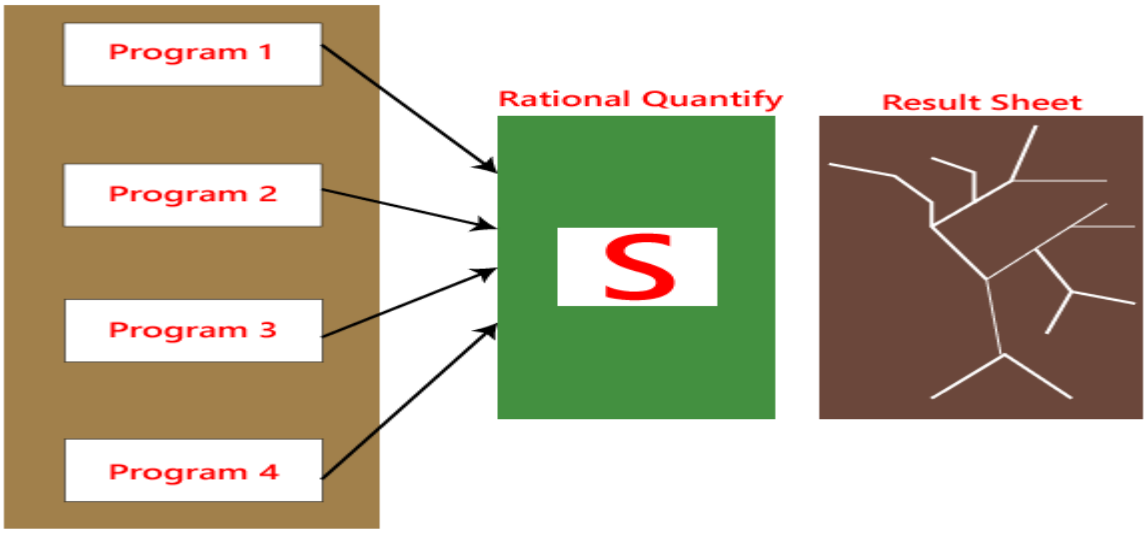
- The developer does not use the available in-built functions; instead they write the full features using their logic. Therefore, it leads us to waste of time and also postpone the product releases.

Test the performance (Speed, response time) of the program

The application could be slow for the following reasons:

- When logic is used.
- For the conditional cases, we will use **or** & **and** adequately.

- Switch case, which means we cannot use **nested if**, instead of using a switch case.



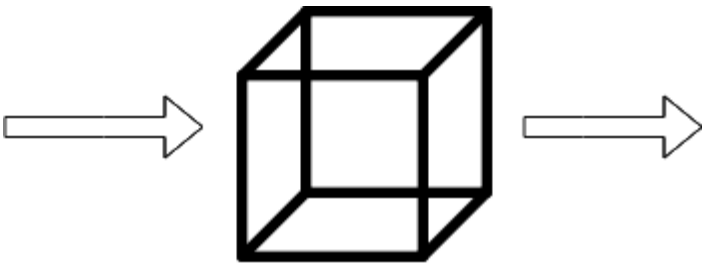
As we know that the developer is performing white box testing, they understand that the code is running slow, or the performance of the program is also getting deliberate. And the developer cannot go manually over the program and verify which line of the code is slowing the program.

To recover with this condition, we have a tool called **Rational Quantify**, which resolves these kinds of issues automatically. Once the entire code is ready, the rational quantify tool will go through the code and execute it. And we can see the outcome in the result sheet in the form of thick and thin lines.

Here, the thick line specifies which section of code is time-consuming. When we double-click on the thick line, the tool will take us to that line or piece of code automatically, which is also displayed in a different color. We can change that code and again and use this tool. When the order of lines is all thin, we know that the presentation of the program has enhanced. And the developers will perform the white box testing automatically because it saves time rather than performing manually.

Test cases for white box testing are derived from the design phase of the software development lifecycle. Data flow testing, control flow testing, path testing, branch

testing, statement and decision coverage all these techniques used by white box testing as a guideline to create an error-free software.



Whitebox Testing

White box testing follows some working steps to make testing manageable and easy to understand what the next task to do. There are some basic steps to perform white box testing.

Generic steps of white box testing

- Design all test scenarios, test cases and prioritize them according to high priority number.
- This step involves the study of code at runtime to examine the resource utilization, not accessed areas of the code, time taken by various methods and operations and so on.
- In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, interfaces are able to handle all types of data appropriately or not.
- This step focuses on testing of control statements like loops and conditional statements to check the efficiency and accuracy for different data inputs.
- In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.

Reasons for white box testing

- It identifies internal security holes.

- To check the way of input inside the code.
- Check the functionality of conditional loops.
- To test function, object, and statement at an individual level.

Advantages of White box testing

- White box testing optimizes code so hidden errors can be identified.
- Test cases of white box testing can be easily automated.
- This testing is more thorough than other testing approaches as it covers all code paths.
- It can be started in the SDLC phase even without GUI.

Disadvantages of White box testing

- White box testing is too much time consuming when it comes to large-scale programming applications.
- White box testing is much expensive and complex.
- It can lead to production error because it is not detailed by the developers.
- White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.

Techniques Used in White Box Testing

Data Flow Testing	Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.

Control Flow Testing	Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. Test cases represented by the control graph of the program.
Branch Testing	Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
Statement Testing	Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code, out of total statements present in the source code.
Decision Testing	This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Difference between white-box testing and black-box testing

Following are the significant differences between white box testing and black box testing:

White-box testing	**Black box testing**

The developers can perform white box testing.	The test engineers perform the black box testing.
To perform WBT, we should have an understanding of the programming languages.	To perform BBT, there is no need to have an understanding of the programming languages.
In this, we will look into the source code and test the logic of the code.	In this, we will verify the functionality of the application based on the requirement specification.
In this, the developer should know about the internal design of the code.	In this, there is no need to know about the internal design of the code.

Black box testing

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.



Generic steps of black box testing

- The black box test is based on the specification of requirements, so it is examined in the beginning.
- In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
- In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
- The fourth phase includes the execution of all test cases.
- In the fifth step, the tester compares the expected output against the actual output.
- In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

Test procedure

The test procedure of black box testing is a kind of process in which the tester has specific knowledge about the software's work, and it develops test cases to check the accuracy of the software's functionality.

It does not require programming knowledge of the software. All test cases are designed by considering the input and output of a particular function.A tester knows about the definite output of a particular input, but not about how the result is arising. There are various techniques used in black box testing for testing

like decision table technique, boundary value analysis technique, state transition, All-pair testing, cause-effect graph technique, equivalence partitioning technique, error guessing technique, use case technique and user story technique. All these techniques have been explained in detail within the tutorial.

Test cases

Test cases are created considering the specification of the requirements. These test cases are generally created from working descriptions of the software including requirements, design parameters, and other specifications. For the testing, the test designer selects both positive test scenario by taking valid input values and adverse test scenario by taking invalid input values to determine the correct output. Test cases are mainly designed for functional testing but can also be used for non-functional testing. Test cases are designed by the testing team, there is not any involvement of the development team of software.

Techniques Used in Black Box Testing

Decision Table Technique	Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs.
Boundary Value Technique	Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limit of a variable. It tests, while entering boundary value whether the software is producing correct output or not.
State Transition Technique	State Transition Technique is used to capture the behavior of the software application when different input values are given to the same function. This applies to those types of applications that provide the specific number of attempts to

	access the application.
All-pair Testing Technique	All-pair testing Technique is used to test all the possible discrete combinations of values. This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc.
Cause-Effect Technique	Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result.It is based on a collection of requirements.
Equivalence Partitioning Technique	Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.
Error Guessing Technique	Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software.
Use Case Technique	Use case Technique used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

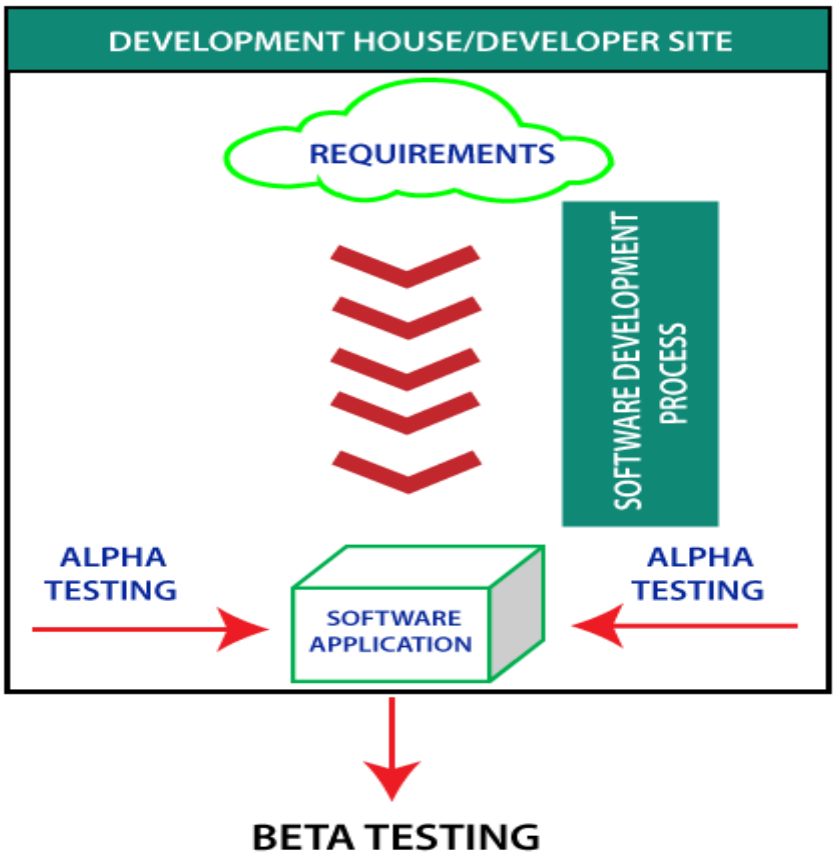
Alpha Testing



Alpha testing is conducted in the organization and tested by a representative group of end-users at the developer's side and sometimes by an independent team of testers.

Alpha testing is simulated or real operational testing at an in-house site. It comes after the unit testing, integration testing, etc. Alpha testing used after all the testing are executed.

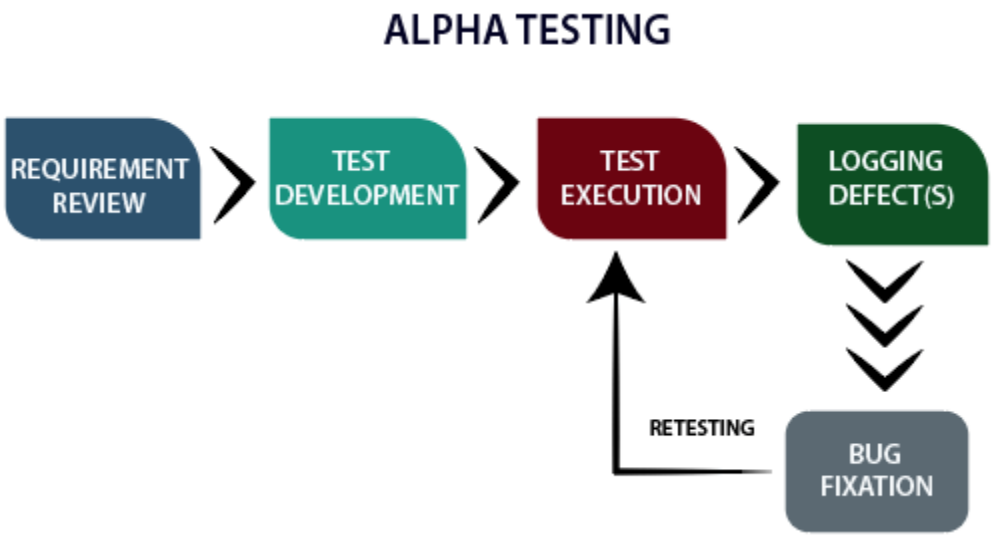
It can be a white box, or Black-box testing depends on the requirements - particular lab environment and simulation of the actual environment required for this testing.



What is the alpha testing process?

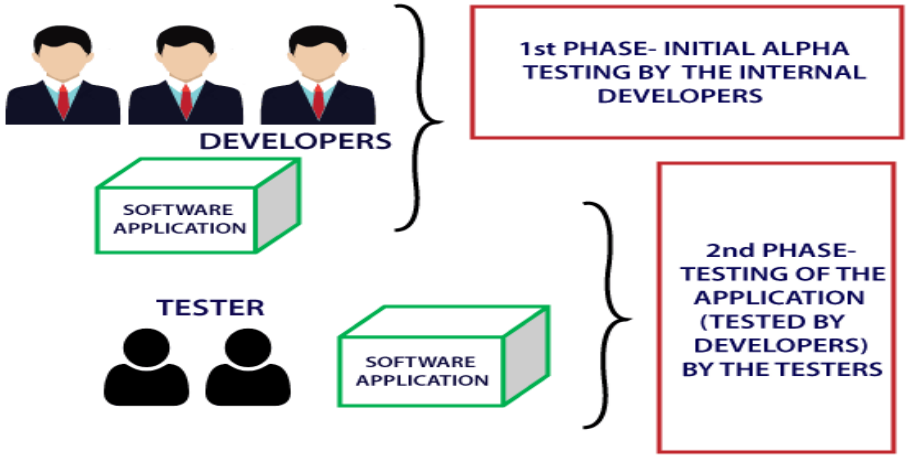
Alpha testing follows the following process:

1. **Requirement Review:** Review the design of the specification and functional requirement
2. **Test Development:** Test development is base on the outcome of the requirement review. Develop the test cases and test plan.
3. **Test case design:** Execute the test plan and test cases.
4. **Logging Defects:** Logging the identified and detected bug found in the application.
5. **Bug Fixation:** When all the bugs are identified and logged, then there is a need to fix the bug.
6. **Retesting:** When all the issues are solved, and fixed retesting is done.



What are the phases of alpha testing?

Alpha testing ensures that the software performs flawlessly and does not impact the reputation of the organization; the company implements final testing in the form of alpha testing. This testing executed into two phases.

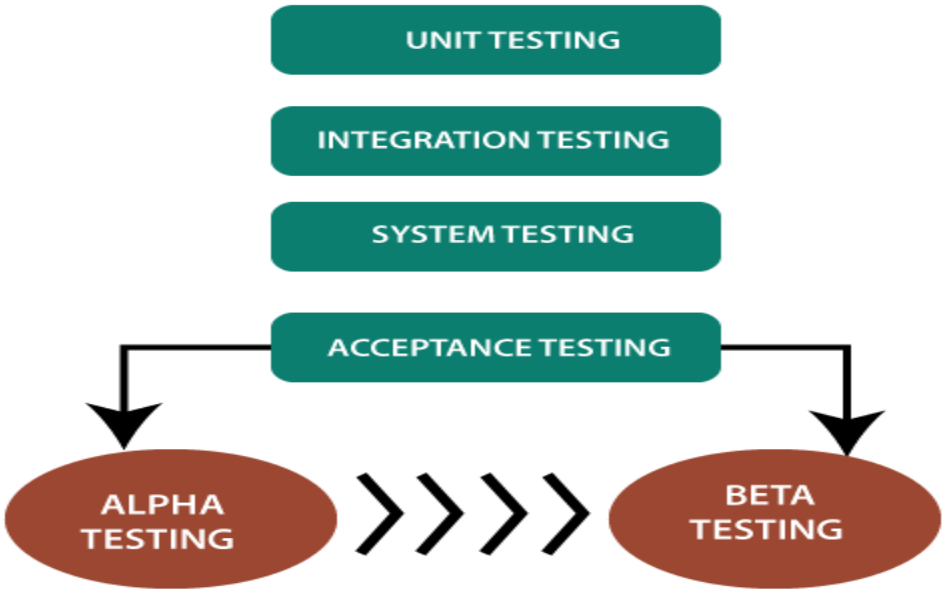


There are two phases of alpha testing.

First Phase: In-house developers of software engineers do the first phase of testing. In this phase, the tester used hardware debugger or hardware aided debugger to catches the bugs quickly. During the alpha testing, a tester finds a lot of bugs, crashes, missing features, and docs.

Second Phase: The second phase involves the quality assurance staff performs the alpha testing by involving black box and white box techniques.

When to perform alpha testing?



Alpha testing is user acceptance testing. Alpha testing performed once the product has gone through stages of testing and prepared for release. It is executing before beta testing, which is also a part of acceptance testing and can define as field testing. During this testing, we can make changes in the software to improve its quality and functionality. Alpha testing done from the developer's site where independent developers can monitor and record user experience and make necessary changes to enhance the performance.

What are the reasons to perform Alpha Testing?

Alpha testing is the final stage of the testing. Alpha testing is an essential and popular testing technique that helps the team to deliver quality and useful software. This testing performed before the release of the product. Alpha testing can define as the first round of independent testing that ensures that the software run as per the requirement plan.

Reasons for alpha testing are:

- Refines the software product by finding and rectifying bugs that weren't discovered through previous tests.
- Alpha testing allows the team to test the software in a real-world environment.
- One of the reasons to do alpha testing is to ensure the success of the software product.
- Alpha testing validates the quality, functionality of the software, and effectiveness of the software before it released in the real world.

What are the features of Alpha Testing?

- Alpha testing is a type of acceptance testing.
- Alpha testing is happening at the stage of the completion of the software product.

- Alpha testing is in the labs where we provide a specific and controlled environment.
- Alpha testing is in-house testing, which is performed by the internal developers and testers within the organization.
- There is not any involvement of the public.
- Alpha testing helps to gain confidence in the user acceptance of the software product.
- With the help of black box and white box technique, we can achieve the alpha testing.
- Alpha testing ensures the maximum possible quality of the software before releasing it to market or client for beta testing.
- Developers perform alpha testing at developer's site; it enables the developer to record the error with the ease to resolve found bugs quickly.
- Alpha testing is doing after the unit testing, integration testing, system testing but before the beta testing.
- Alpha testing is for testing the software application, products, and projects.

What are the advantages of Alpha Testing?

Advantages of alpha testing are:

- One of the benefits of alpha testing is it reduces the delivery time of the project.
- It provides a complete test plan and test cases.
- Free the team member for another project.
- Every feedback helps to improve software quality.
- It provides a better observation of the software's reliability and accountability.

What are the disadvantages of Alpha Testing?

Disadvantages of alpha testing are:

- Alpha testing does not involve in-depth testing of the software.
- The difference between the tester's tests the data for testing the software and the customer's data from their perspective may result in the discrepancy in the software functioning.
- The lab environment is used to simulate the real environment. But still, the lab cannot furnish all the requirement of the real environment such as multiple conditions, factors, and circumstances.

Wrap Up:

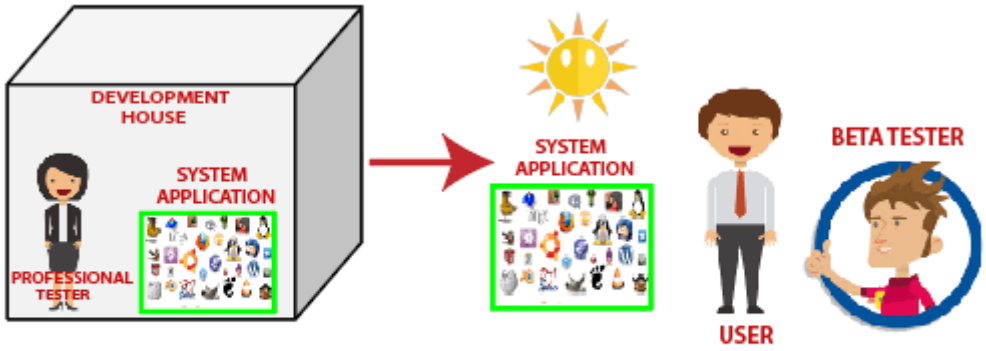
Every software product needs to undergo a vital methodology before going into a highly competitive market. Alpha testing is one of the vital testing. It needs to be considered by going through the functionality of the software and achieve the confidence in its user acceptance for the real environment, before releasing it into the market.

Beta Testing

Beta testing is a type of User Acceptance Testing among the most crucial testing, which performed before the release of the software. Beta Testing is a type of Field Test. This testing performs at the end of the software testing life cycle. This type of testing can be considered as external user acceptance testing. It is a type of salient testing. Real users perform this testing. This testing executed after the alpha testing. In this the new version, beta testing is released to a limited audience to check the accessibility, usability, and functionality, and more.



Beta testing is the last phase of the testing, which is carried out at the client's or customer's site.



BETA TESTING OF THE PRODUCT IN REAL WORLD ENVIRONMENT

What are the features of beta testing?

Testing of the product performs by the real users of the software application in the real environment. Beta version of the software is released to a restricted number of end-users to obtain the feedback of the product quality. Beta testing reduces the risk of failure and provides the quality of the product through customer validation. It is the final testing before shipping the product to the customers. Beta testing obtains direct feedback from the customers. It helps in testing to test the product in the customer's environment.

Features of beta testing are:

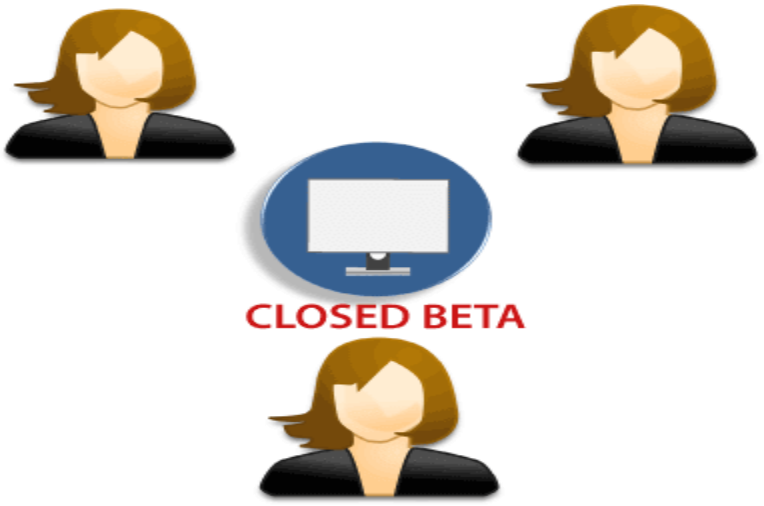
- Beta testing used in a real environment at the user's site. Beta testing helps in providing the actual position of the quality.
- Testing performed by the client, stakeholder, and end-user.
- Beta testing always is done after the alpha testing, and before releasing it into the market.
- Beta testing is black-box testing.
- Beta testing performs in the absence of tester and the presence of real users

- Beta testing is performed after alpha testing and before the release of the final product.
- Beta testing generally is done for testing software products like utilities, operating systems, and applications, etc.

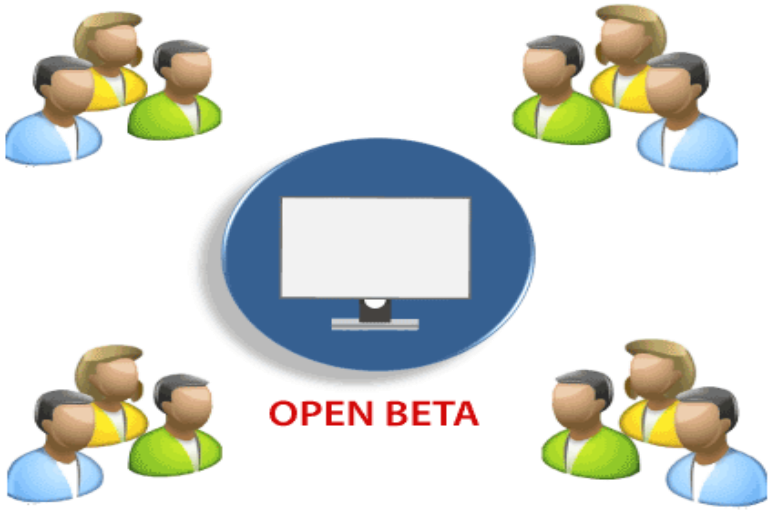
What is a beta version of the software?

The beta version of the software is delivered to a restricted number of users to accept their feedback and suggestions on quality improvement. Hence, there are two types of beta version:

1) Closed beta version: Closed beta version, also known as a private beta, it is released to a group of selected and invited people. Those people will test the software and evaluate their features and specifications. This beta version represents the software which is capable of delivering value, but it is not ready to be used by everyone. Because it shows the issues like lack of documentation or missing vital features.



2) Open beta version: Open beta is also known as a public beta. The open beta opened to the public. Any user as a tester can assess the beta version to provide the relevant feedback and reviews. Open beta version improves the quality of the final release. This version helps to find the various undetected errors and issues.



The beta testing process orients this beta version.

What is the lifecycle of Beta Testing?

A group of end-users performs beta testing. This process can't execute without any strategy or test plan. Before the testers, the end-user executes this type of testing.

The process of beta testing follows the following steps:

1. **Planning:** Like another testing process, beta testing also supports proper planning. In this stage, the team prepares a testing strategy and defines the goal of testing. In this case, the team establishes the need of users for testing, duration, and necessary details related to the process.

2. **Participant Recruitment:** This is the second stage of the beta process in which the team recruits a group of selected end-users for testing. This group can change as per the requirement of the organization and the product.

3. **Product Launch:** When a team of users (testers) recruited. The beta version of the product is launched or installed at the client or user side, and users will test the product for quality assurance.

4. **Collect and Evaluate Feedback:** When the testing finished, developers will collect the feedback provided by the testers and evaluate it. In the end, based on the feedback, issues, and bugs are fixed and resolved by the responsible individual team.

5. **Closure:** When all the problems fixed and the organization meets the exit criteria, beta testing achieved, and the rewards offered to the testing team.

What are the types of beta testing?

Beta testing has six types. Each type has different aspects of the software. All these help developers to improve the quality of the software and allow them to deliver a product that offers excellent user experience. Here are the different types of beta testing:

1. **Open Beta Testing:** Open beta testing involves testing the software product by a large number of people before the final release. The organization decides to make a software product open to the public before releasing the product. Open Beta includes the extensive participation of the public to use and evaluate software product accordingly.
   Users report the bug to the organization, along with a suggestion to improve the quality of the software.

2. **Closed Beta Testing:** Opposite to the open beta testing. Closed beta testing performed by the selective and limited number of persons. The organization recruits these. In this testing software product is not open to the public.

3. **Traditional Beta Testing:** In this testing, a software product delivered to the target market, and the feedback from the users collected. This type of testing assistance the beta testing, the quality of the software is improved, and developers can make the changes.

4. **Public Beta Testing:** This type of testing is similar to open testing. Public beta testing also allows the product is delivering to the end-users worldwide, with the aid of various online channels available in the world.

From this, the feedback and evaluated data also collected and based on the requirement changes, and the development team implements modifications.

5. **Technical Beta Testing:** Technical beta testing is also an essential type of beta testing. This testing involves delivering the software product to the internal groups of the organization. However, the data and feedback provided by the employees of the organization.

6. **Focused Beta Testing:** This type of testing focused on monitoring and evaluating a specific feature or component of the software. In focused beta testing, the software released to the market and user's experience assessed and collected to make the required changes.

7. **Post-Release Beta Testing:** In this testing, the product delivered to the market for the use of the end-users. Their feedback, reactions, and experience are collect for the future release of the software.

When to perform Beta Testing?

Acceptance testing is the final phase of the testing, which combines both alpha and beta testing to ensure that the product released flawlessly. Beta testing performed at the user's end. This testing always performed after the alpha testing, but before the product released to the market. In this stage, the product is expected to be 90% to 95% completed.

Any product undergoing to beta test should be reviewed for the entire checklist before launching it.

Some of them are:

- All the component of the product is ready to start this testing.
- Documentation which is going to end-user should be kept ready - Setup, installation, usage, Uninstallation should be in detail.
- The product management team should review that all the functionality is in good condition.

- Procedure to collect bugs, feedback, etc. should be identified before publishing it.

What are the stakeholders and participants in the Beta Testing?

The product management, quality management, and user experience teams are the stakeholder in beta testing, and they closely monitor every move of the phase.

The real users who use the product are the participants.

Beta test strategy

- Business objective for the product.
- Beta test plan
- The testing approach followed by participants.
- Tools used to detect bugs, measure productivity, collect feedback.
- When and how to end this testing phase?

What is a Beta Test plan?

A beta test plan can be written in many ways,

Objective: We should have to mention the aim of the project why there is a need for beta testing even after performing the internal testing.

Scope: In this plan, we should mention the areas to be tested or not.

Test Approach: We should have to mention clearly that the testing is in the deep, what to focus on - functionality, UI, response, etc.

Schedule: We have to specify, clearly the start and ending date with time, number of cycles, and duration per cycle.

Tools: Bug logging tools and the usage of the machines should identify.

Budget: Incentive of the bugs based on the severity.

Feedback: Collecting feedback and evaluating methods.

- **Identify and review the entry and exit criteria.**

What are the entry criteria for Beta Testing?

- Sign off the document from alpha testing.
- Beta version of the software should ready.
- The environment should be ready to release the software application to the public.
- To capture the real-time faults environment should be ready.

What are the exit criteria for Beta Testing?

- All the major and minor issues resolved.
- The feedback report should prepare.
- The delivery of beta test summary report.

What are the advantages of Beta Testing?

Beta testing performed at the end of the software testing lifecycle. Beta testing offers numerous benefits to testers, software developer, as well as the users. In the assistance of this type of testing, it enables developers, testers to test the product before its release in the market. The

1. Beta testing focuses on the customer's satisfaction.
2. It helps to reduce the risk of product failure via user validations.
3. Beta testing helps to get direct feedback from users.
4. It helps to detect the defect and issues in the system, which is overlooked and undetected by the team of software testers.
5. Beta testing helps the user to install, test, and send feedback regarding the developed software.

What are the disadvantages of Beta Testing?

Disadvantages of beta testing is:

1. In this type of testing, a software engineer has no control over the process of the testing, as the users in the real-world environment perform it.
2. This testing can be a time-consuming process and can delay the final release of the product.
3. Beta testing does not test the functionality of the software in depth as software still in development.
4. It is a waste of time and money to work on the feedback of the users who do not use the software themselves properly.

Wrap Up

Keeping in mind the characteristics of beta testing can be concluded that beta testing may be considered desirable for the organization. Beta testing provides the feedback of the real-users, which helps improve the software quality before the product released in the market.

Regression testing

Regression testing is a black box testing techniques. It is used to authenticate a code change in the software does not impact the existing functionality of the product. Regression testing is making sure that the product works fine with new functionality, bug fixes, or any change in the existing feature.

Regression testing is a type of software testing. Test cases are re-executed to check the previous functionality of the application is working fine, and the new changes have not produced any bugs.

Regression testing can be performed on a new build when there is a significant change in the original functionality. It ensures that the code still works even when the changes are occurring. Regression means Re-test those parts of the application, which are unchanged.

Regression tests are also known as the Verification Method. Test cases are often automated. Test cases are required to execute many times and running the same test case again and again manually, is time-consuming and tedious too.

Example of Regression testing

Here we are going to take a case to define the regression testing efficiently:

Consider a product Y, in which one of the functionality is to trigger confirmation, acceptance, and dispatched emails. It also needs to be tested to ensure that the change in the code not affected them. Regressing testing does not depend on any programming language like Java, C++, C#, etc. This method is used to test the product for modifications or any updates done. It ensures that any change in a product does not affect the existing module of the product. Verify that the bugs fixed and the newly added features not created any problem in the previous working version of the Software.

When can we perform Regression Testing?

We do regression testing whenever the production code is modified.

We can perform regression testing in the following scenario, these are:

1. When new functionality added to the application.

Example:

A website has a login functionality which allows users to log in only with Email. Now providing a new feature to do login using Facebook.

2. When there is a Change Requirement.

Example:

Remember password removed from the login page which is applicable previously.

3. When the defect fixed

Example:

Assume login button is not working in a login page and a tester reports a bug stating that the login button is broken. Once the bug fixed by developers, tester tests it to make sure Login Button is working as per the expected result. Simultaneously, tester tests other functionality which is related to the login button.

4. When there is a performance issue fix

Example:

Loading of a home page takes 5 seconds, reducing the load time to 2 seconds.

5. When there is an environment change

Example:

When we update the database from MySql to Oracle.

How to perform Regression Testing?

The need for regression testing comes when software maintenance includes enhancements, error corrections, optimization, and deletion of existing features. These modifications may affect system functionality. Regression Testing becomes necessary in this case.

Regression testing can be performed using the following techniques:



1. Re-test All:

Re-Test is one of the approaches to do regression testing. In this approach, all the test case suits should be re-executed. Here we can define re-test as when a test fails, and we determine the cause of the failure is a software fault. The fault is reported, we can expect a new version of the software in which defect fixed. In this case, we will need to execute the test again to confirm that the fault fixed. This is known as re-testing. Some will refer to this as confirmation testing.

The re-test is very expensive, as it requires enormous time and resources.

2. Regression test Selection:

- In this technique, a selected test-case suit will execute rather than an entire test-case suit.
- The selected test case suits divided in two cases
    1. Reusable Test cases.
    2. Obsolete Test cases.
- Reusable test cases can use in succeeding regression cycle.
- Obsolete test cases can't use in succeeding regression cycle.

3. Prioritization of test cases:

Prioritize the test case depending on business impact, critical and frequently functionality used. Selection of test cases will reduce the regression test suite.

What are the Regression Testing tools?

Regression Testing is a vital part of the QA process; while performing the regression we may face the below challenges:

- **Time Consuming**
  Regression Testing consumes a lot of time to complete. Regression testing involves existing tests again, so testers are not excited to re-run the test.

- **Complex**
  Regression Testing is complex as well when there is a need to update any product; lists of the test are also increasing.
- **Communicating business rule**
  Regression Testing ensures the existing product features are still in working order. Communication about regression testing with a non-technical leader can be a difficult task. The executive wants to see the product move forward and making a considerable time investment in regression testing to ensure existing functionality working can be hard.
- **Identify Impact Area**
- **Test Cases Increases Release by Release**
- **Less Resources**
- **No Accuracy**
- **Repetitive Task**
- **Monotonous Job**

Regression testing process

The regression testing process can be performed across the **builds** and the **releases**.

Regression testing across the builds

Whenever the bug fixed, we retest the Bug, and if there is any dependent module, we go for a Regression Testing.



For example, How we perform the regression testing if we have different builds as **Build 1, Build 2, and Build 3**, which having different scenarios.

Build1

- Firstly the client will provide the business needs.
- Then the development team starts developing the features.
- After that, the testing team will start writing the test cases; for example, they write 900 test cases for the release#1 of the product.
- And then, they will start implementing the test cases.
- Once the product is released, the customer performs one round of acceptance testing.
- And in the end, the product is moved to the production server.

Build2

- Now, the customer asks for 3-4 extra (new) features to be added and also provides the requirements for the new features.
- The development team starts developing new features.
- After that, the testing team will start writing the test case for the new features, and they write about 150 new test cases. Therefore, the total number of the test case written is 1050 for both the releases.

- Now the testing team starts testing the new features using 150 new test cases.
- Once it is done, they will begin testing the old features with the help of 900 test cases to verify that adding the new feature has damaged the old features or not.
- Here, testing the old features is known as **Regression Testing**.
- Once all the features (New and Old) have been tested, the product is handed over to the customer, and then the customer will do the acceptance testing.
- Once the acceptance testing is done, the product is moved to the production server.

Build3

- After the second release, the customer wants to remove one of the features like Sales.
- Then he/she will delete all the test cases which are belonging to the sales module (about 120 test cases).
- And then, test the other feature for verifying that if all the other features are working fine after removing the sales module test cases, and this process is done under the regression testing.

Note:

- Testing the stable features to ensure that it is broken because of the changes. Here changes imply that the **modification, addition, bug fixing, or the deletion**.
- Re-execution of the same test cases in the different builds or releases is to ensure that changes (modification, addition, bug fixing, or the deletion) are not introducing bugs in stable features.

Regression testing across the release

The regression testing process starts whenever there is a new Release for same project because the new feature may affect the old elements in the previous releases.

To understand the regression testing process, we will follow the below steps:

Step1

There is no regression testing in **Release#1** because there is no modification happen in the Release#1 as the release is new itself.

Step2

The concept of Regression testing starts from **Release#2** when the customer gives some **new requirements**.

Step3

After getting the new requirements (modifying features) first, they (the developers and test engineers) will understand the needs before going to the **impact analysis**.

Step4

After understanding the new requirements, we will perform one round of **impact analysis** to avoid the major risk, but here the question arises who will do the Impact analysis?

Step5

The impact analysis is done by the **customer** based on their **business knowledge**, the **developer** based on their **coding knowledge**, and most importantly, it is done by the **test engineer** because they have the **product knowledge**.

Note: If a single person does, he/she may not cover all the impact areas, so we include all persons so that we may cover a maximum impact area, and Impact Analysis should be done at the early stages of the releases.

Step6

Once we are done with the **impact area**, then the developer will prepare the **impact area (document)**, and the **customer** will also prepare the **impact area document** so that we can achieve the **maximum coverage of impact analysis**.

Step7

After completing the impact analysis, the developer, the customer, and the test engineer will send the **Reports#** of the impact area documents to the **Test Lead**. And in the meantime, the test engineer and the developer are busy working on the new test case.

Step8

Once the Test lead gets the Reports#, he/she will **consolidate** the reports and stored in the **test case requirement repository** for the release#1.

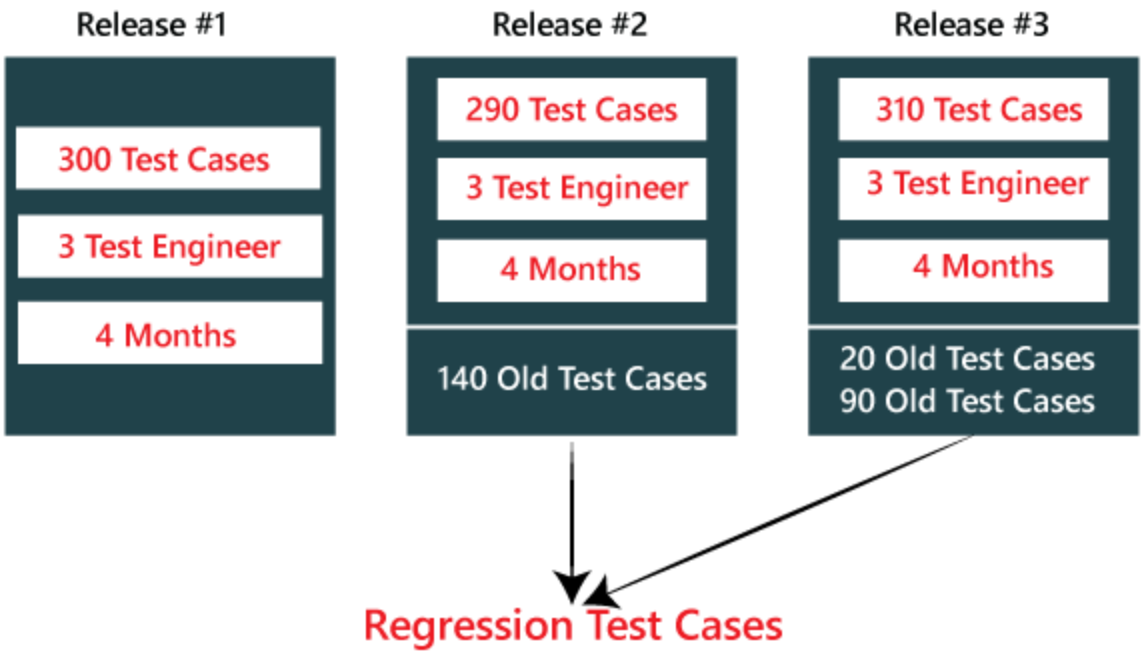
Note: Test case Repository: Here, we will save all the test case of releases.

Step9

After that, the Test Lead will take the help of RTM and pick the necessary **regression test case** from the **test case repository**, and those files will be placed in the **Regression Test Suite**.

Note:

- The test lead will store the regression test case in the regression test suite for no further confusion.
- **Regression test suite**: Here, we will save all the impact area test documents.
- **Regression Test Cases**: These are the test cases of the old releases text document which need to be re-executed as we can see in the below image:

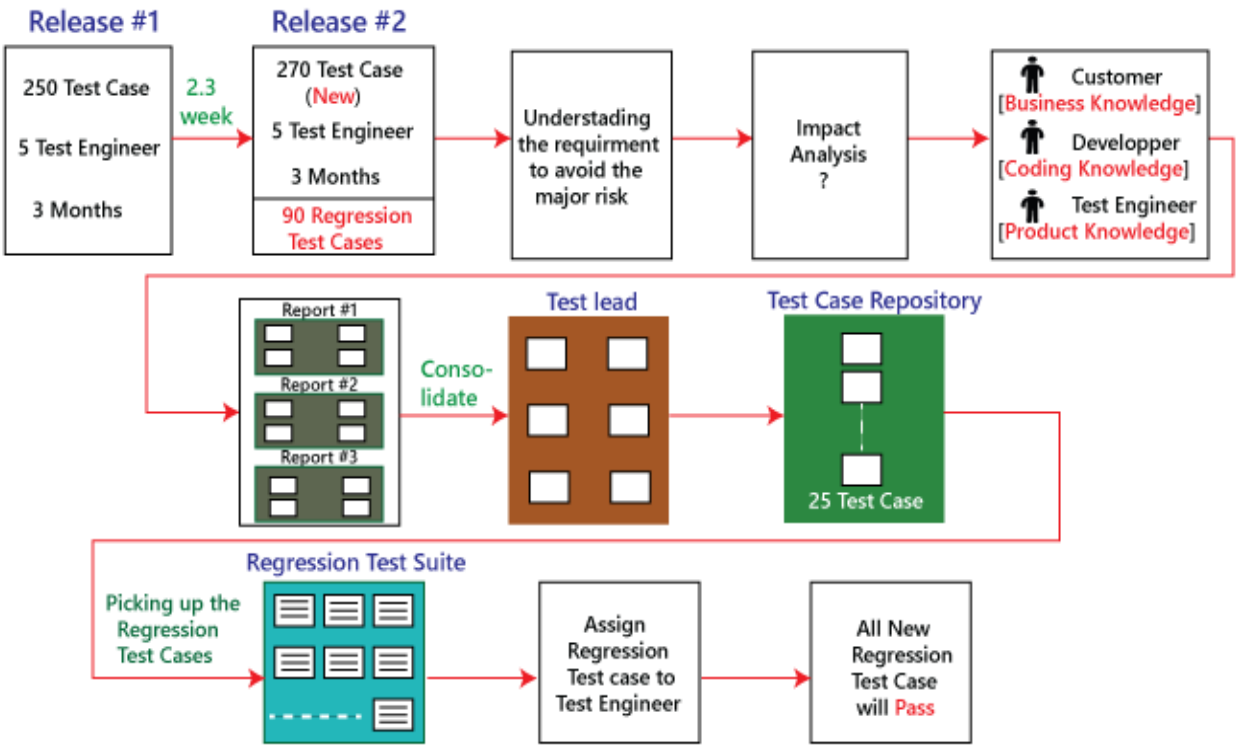


Step10

After that, when the test engineer has done working on the new test cases, the test lead will **assign the regression test case** to the test engineer.

Step11

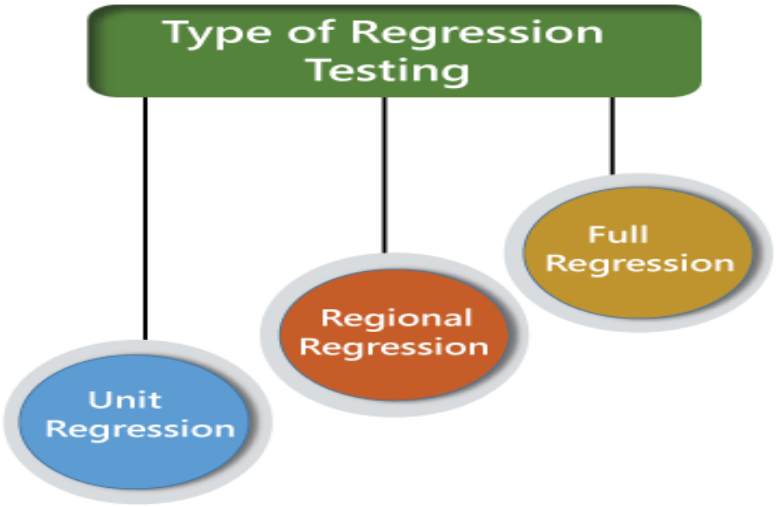
When all the regression test cases and the new features are **stable and pass**, then check the **impact area using the test case** until it is durable for old features plus the new features, and then it will be handed over to the customer.



Types of Regression Testing

The different types of Regression Testing are as follows:

- Unit Regression Testing [URT]
- Regional Regression Testing[RRT]
- Full or Complete Regression Testing [FRT]

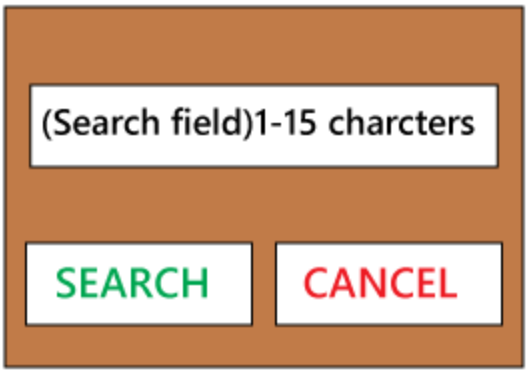


Unit Regression Testing [URT]

In this, we are going to test only the changed unit, not the impact area, because it may affect the components of the same module.

Example1

In the below application, and in the first build, the developer develops the **Search** button that accepts **1-15 characters**. Then the test engineer tests the Search button with the help of the **test case design technique**.



Build 1-B001



Build 1-B002

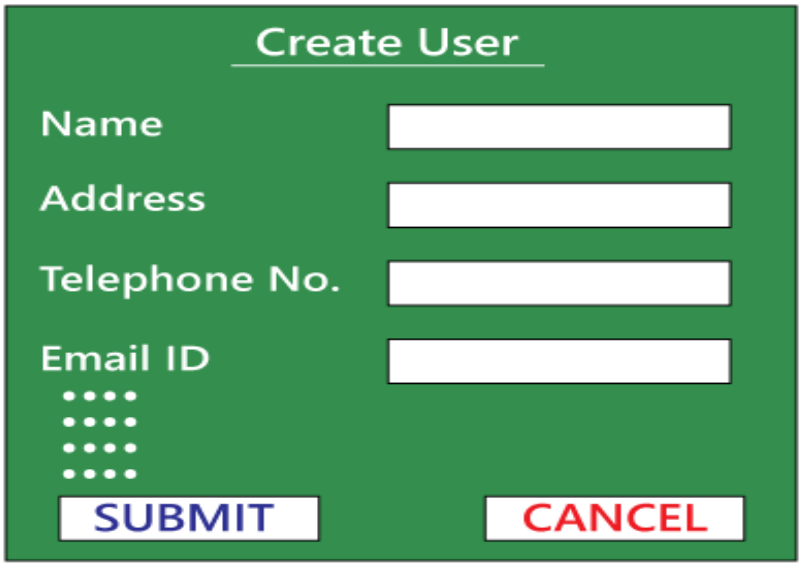
Now, the client does some modification in the requirement and also requests that the **Search button** can accept the **1-35 characters**. The test engineer will test only the Search button to verify that it takes 1-35 characters and does not check any further feature of the first build.

Example2

Here, we have **Build B001**, and a defect is identified, and the report is delivered to the developer. The developer will fix the bug and sends along with some new features which are developed in the second **Build B002**. After that, the test engineer will test only after the defect is fixed.

- The test engineer will identify that clicking on the **Submit** button goes to the blank page.
- And it is a defect, and it is sent to the developer for fixing it.

- When the new build comes along with the bug fixes, the test engineer will test only the Submit button.
- And here, we are not going to check other features of the first build and move to test the new features and sent in the second build.
- We are sure that fixing the **Submit**button is not going to affect the other features, so we test only the fixed bug.



Therefore, we can say that by testing only the changed feature is called the **Unit Regression Testing**.

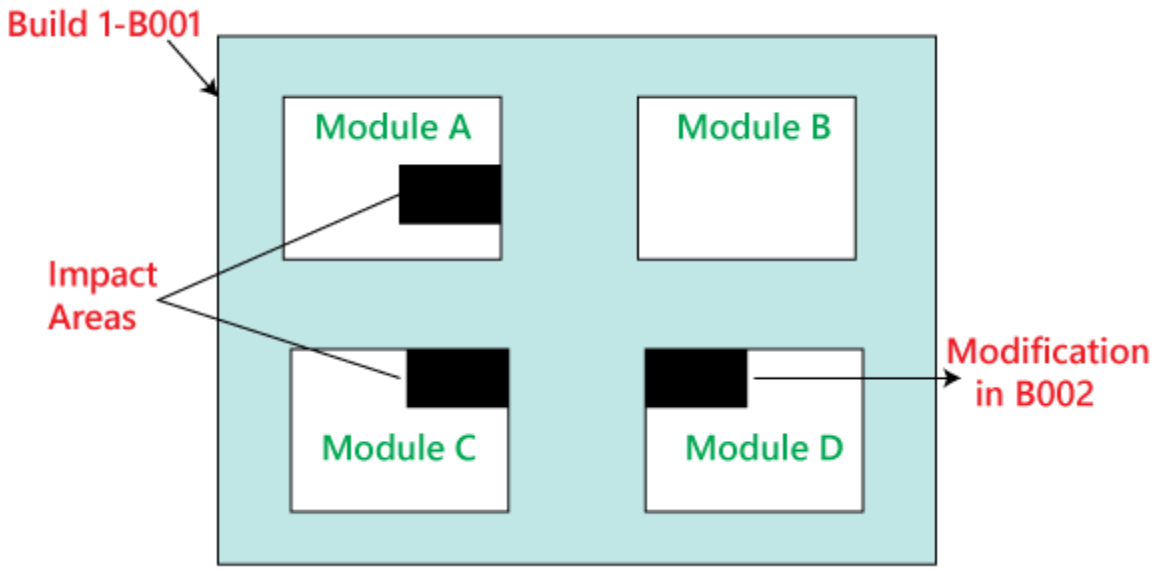
Regional Regression testing [RRT]

In this, we are going to test the modification along with the impact area or regions, are called the **Regional Regression testing**. Here, we are testing the impact area because if there are dependable modules, it will affect the other modules also.

For example:

In the below image as we can see that we have four different modules, such as **Module A, Module B, Module C, and Module D**, which are provided by the developers for the testing during the first build. Now, the test engineer will

identify the bugs in **Module D**. The bug report is sent to the developers, and the development team fixes those defects and sends the second build.



In the second build, the previous defects are fixed. Now the test engineer understands that the bug fixing in Module D has impacted some features in **Module A and Module C**. Hence, the test engineer first tests the Module D where the bug has been fixed and then checks the impact areas in **Module A and Module C**. Therefore, this testing is known as **Regional regression testing.**

While performing the regional regression testing, we may face the below problem:

Problem:

In the first build, the client sends some modification in requirement and also wants to add new features in the product. The needs are sent to both the teams, i.e., development and testing.

After getting the requirements, the development team starts doing the modification and also develops the new features based on the needs.

Now, the test lead sends mail to the clients and asks them that all are the impact areas that will be affected after the necessary modification have been done.

Therefore, the customer will get an idea, which all features are needed to be tested again. And he/she will also send a mail to the development team to know which all areas in the application will be affected as a result of the changes and additions of new features.

And similarly, the customer sends a mail to the testing team for a list of impact areas. Hence, the test lead will collect the impact list from the client, development team, and the testing team as well.

This **Impact list** is sent to all the test engineers who look at the list and check if their features are modified and if yes, then they do **regional regression testing**. The impact areas and modified areas are all tested by the respective engineers. Every test engineer tests only their features that could have been affected as a result of the modification.

The problem with this above approach is that the test lead may not get the whole idea of the impact areas because the development team and the client may not have so much time to revert his/her mails.

Solution

To resolve the above problem, we will follow the below process:

When a new build comes along with the latest features and bug fixes, the testing team will arrange the meeting where they will talk about if their features are affecting because of the above modification. Therefore, they will do one round of **Impact Analysis** and generate the **Impact List**. In this particular list, the test engineer tries to enclose the maximum probable impact areas, which also decreases the chance of getting the defects.

When a new build comes, the testing team will follow the below procedure:

- They will do smoke testing to check the basic functionality of an application.
- Then they will test new features.
- After that, they will check the changed features.

- Once they are done with checking the changed features, the test engineer will re-test the bugs.
- And then they will check the impact area by performing the regional regression testing.

Disadvantage of using Unit and Regional Regression testing

Following are some of the drawbacks of using unit and Regional regression testing:

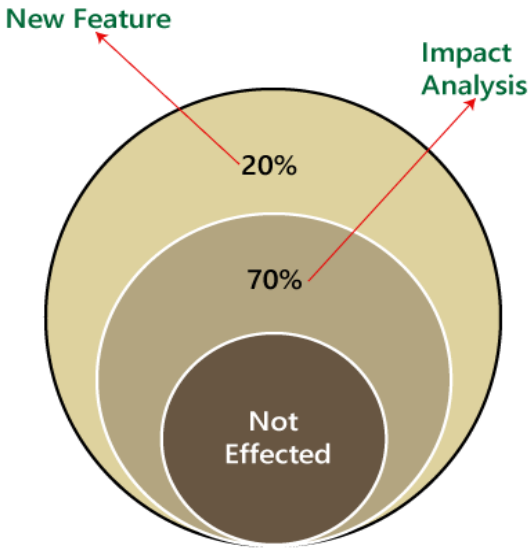
- We may miss some impact area.
- It is possible that we may identify the wrong impact area.

Note: We can say that the major work we do on the regional regression testing will lead us to get more number of defects. But, if we will perform the same dedication to work on the full regressing testing, we will get less number of defects. Therefore, we can determine here that enhancement in the testing effort will not help us to get more defects.

Full Regression testing [FRT]

During the second and the third release of the product, the client asks for adding 3-4 new features, and also some defects need to be fixed from the previous release. Then the testing team will do the Impact Analysis and identify that the above modification will lead us to test the entire product.

Therefore, we can say that testing the **modified features** and **all the remaining (old) features** is called the **Full Regression testing**.



When we perform Full Regression testing?

We will perform the FRT when we have the following conditions:

- When the modification is happening in the source file of the product. **For example**, JVM is the root file of the JAVA application, and if any change is going to happen in JVM, then the entire JAVA program will be tested.
- When we have to perform n-number of changes.

Note:

The regional regression testing is the ideal approach of regression testing, but the issue is, we may miss lots of defects while performing the Regional Regression testing.

And here we are going to solve this issue with the help of the following approach:

- When the application is given for the testing, the test engineer will test the first 10-14 cycle, and will do the **RRT**.
- Then for the 15th cycle, we do FRT. And again, for the next 10-15 cycle, we do **Regional regression testing**, and for the 31th cycle, we do the **full regression testing**, and we will continue like this.

- But for the last ten cycle of the release, we will perform only **complete regression testing**.

Therefore, if we follow the above approach, we can get more defects.

The drawback of doing regression testing manually repeatedly:

- Productivity will decrease.
- It is a difficult job to do.
- There is no consistency in test execution.
- And the test execution time is also increased.

Hence, we will go for the automation to get over with these issues; when we have n-number of the regression test cycle, we will go for the **automation regression testing process**.

Automated Regression testing process

Generally we go for the automation whenever there are multiple releases or multiple regression cycle or there is the repetitive task.

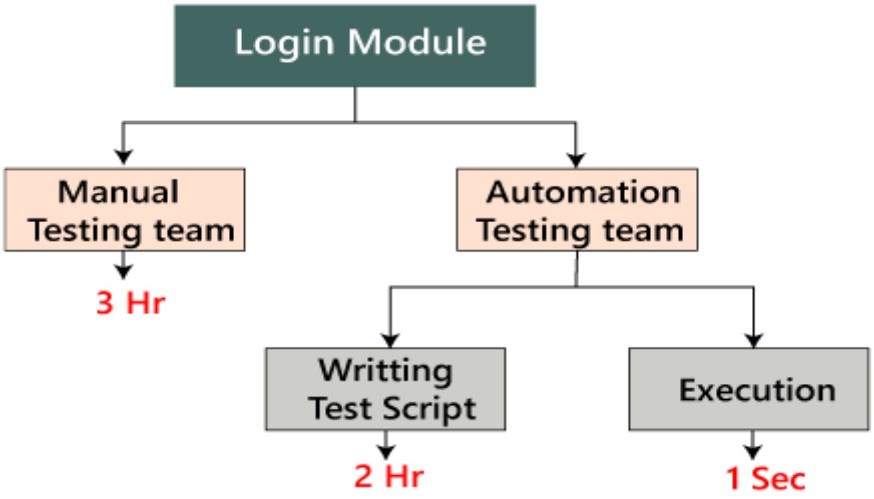
The automation regression testing process can be done in the following steps:

Note1:

The process of testing the application by using some tools is known as automation testing.

Suppose if we take one sample example of a **Login module**, then how we can perform the regression testing.

Here, the Login can be done in two ways, which are as follows:



Manually: In this, we will perform regression only one and twice.

Automation: In this, we will do the automation multiple times as we have to write the test scripts and do the execution.

Note2: In real-time, if we have faced some issues such as:

Issues	Handle by
New features	Manual test engineer
Regressing testing features	Automation test engineer
Remaining (110 feature + Release#1)	Manual test engineer

Step1

When the new release starts, we don't go for the automation because there is no concept of regression testing and regression test case as we understood this in the above process.

Step2

When the new release and the enhancement starts, we have two teams, i.e., manual team and the automation team.

Step3

The manual team will go through the requirements and also identify the impact area and hand over the **requirement test suite** to the automation team.

Step4

Now, the manual team starts working on the new features, and the automation team will start developing the test script and also start automating the test case, which means that the regression test cases will be converted into the test script.

Step5

Before they (automation team) start automating the test case, they will also analyze which all cases can be automated or not.

Step6

Based on the analysis, they will start the automation i.e., converting every regression test cases into the test script.

Step7

During this process, they will take help of the **Regression cases** because they don't have product knowledge as well as the **tool** and the **application**.

Step8

Once the test script is ready, they will start the execution of these scripts on the new application [old feature]. Since, the test script is written with the help of the regression feature or the old feature.

Step9

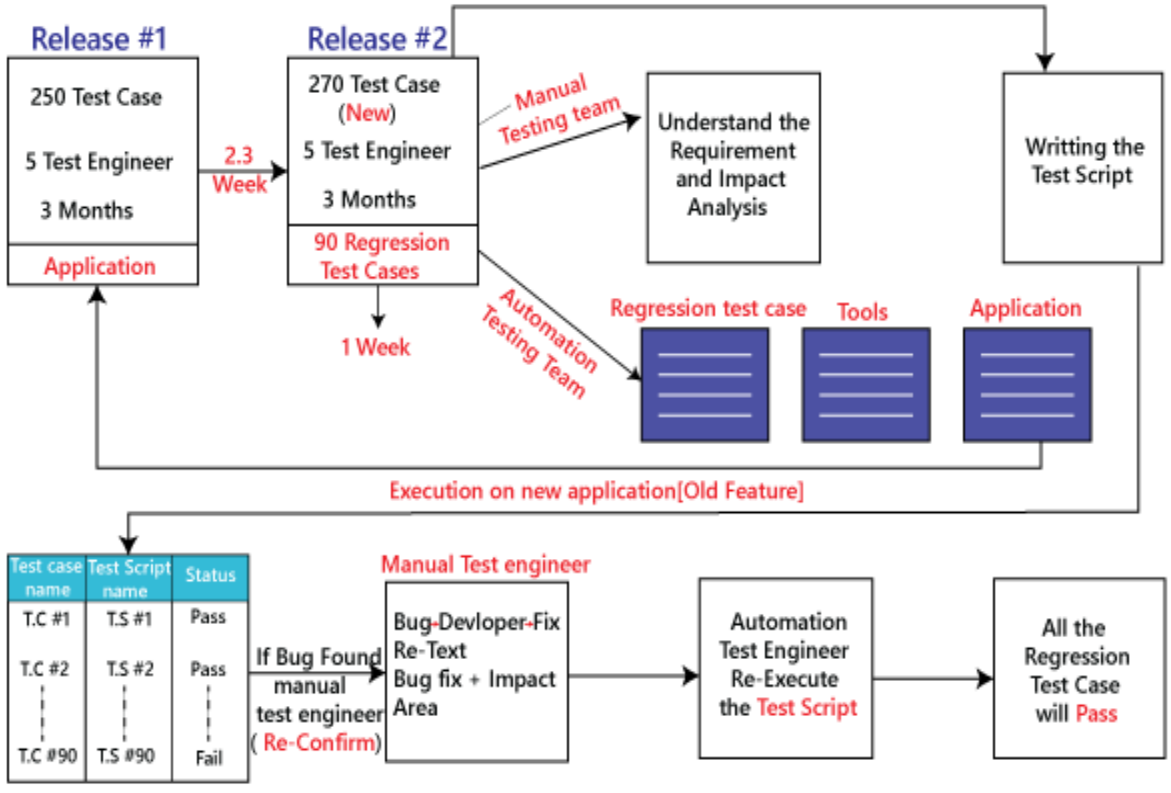
Once the execution is completed, we get a different status like **Pass/fail**.

Step10

If the status is failed, which means it needs to be re-confirmed manually, and if the Bug exists, then it will report to the concerned developer. When the developer fixes that bug, the Bug needs to be re-tested along with the Impact area by the manual test engineer, and also the script needs to be re-executed by the automation test engineer.

Step11

This process goes on until all the new features, and the regression feature will be passed.



Benefits of doing regression testing by the automation testing:

- **Accuracy** always exists because the task is done by the tools and tools never get bored or tired.
- The test script can be re-used across multiple releases.

- **Batch execution** is possible using the automation i.e.; all the written test scripts can be executed parallel or simultaneously.
- Even though the number of regression test case increase release per release, and we don't have to increase the automation resource since some regression case are already automated from the previous release.
- It is a **time-saving process** because the execution is always faster than the manual method.

How to select test cases for regression testing?

It was found from industry inspection. The several defects reported by the customer were due to last-minute bug fixes. These creating side effects and hence selecting the Test Case for regression testing is an art, not an easy task.

Regression test can be done by:

- A test case which has frequent defects
- Functionalities which are more visible to users.
- Test cases verify the core features of the product.
- All integration test cases
- All complex test cases
- Boundary value test cases
- A sample of successful test cases
- Failure of test cases

Regression testing tools

If Software undergoes frequent changes, regression testing costs also increase. In those cases, manual execution of test cases increases test execution time as well as costs. In that case, automation testing is the best choice. The duration of

automation depends on the number of test cases that remain reusable for successive regression cycles.

Following are the essential tools used for regression testing:

Selenium

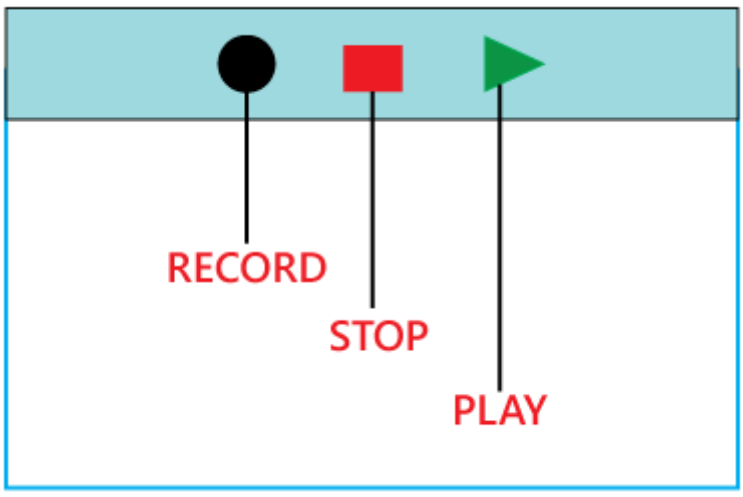
Selenium is an open-source tool. This tool used for automated testing of a web application. For browser-based regression testing, selenium used. Selenium used for UI level regression test for web-based application.

Ranorex Studio

All in one regression test automation for desktop, web, and mobile apps with built-in Selenium Web Driver. Ranorex Studio includes full IDE plus tools for codeless automation.

Quick Test Professional (QTP)

QTP is an automated testing tool used for Regression and Functional Testing. It is a Data-Driven, keyword-based tool. It used VBScript language for automation. If we open the QTP tool, we see the three buttons which are **Record, Play and Stop**. These buttons help to record every click and action performed on the computer system. It records the actions and play it back.



Rational Functional Tester (RTF)

Rational functional tester is a Java tool used to automate the test cases of software applications. RTF used for automating regression test cases, and it also integrates with the rational functional tester.

What are the Regression Testing and Configuration Management?

Configuration Management in the regression testing becomes imperative in Agile Environments, where a code is continuously modified. To ensure a valid regression test, we must follow the steps:

- Changes are not allowed in the code during the regression testing phase.
- A regression test case must be unaffected developer changes.
- The database used for regression testing must be isolated; changes are not allowed in the database.

What are the differences between Retesting and Regression Testing?

Re-testing Testing means testing the functionality or bug again to ensure the code fixed. If not set, defects need not be re-opened. If fixed, the defect closed.

Re-testing is a type of testing which performed to check the test-cases that were unsuccessful in the final execution are successfully pass after the defects repaired.

Regression Testing means testing the software application when it undergoes a code change to ensure that new code has not affected other parts of the Software.

Regression testing is a type of testing executed to check whether a code has not changed the existing functionality of the application.

Differences between the Re-testing and Regression Testing are as follows:

Re-testing	Regression Testing
Re-testing is performed to ensure	Regression Testing is done to confirm

that the test cases that are failed in the final execution are passing after the defects fixed.	whether the code change has not affected the existing features.
Re-Testing works on defect fixes.	The purpose of regression testing is to ensure that the code changes adversely not affect the existing functionality.
Defect verification is the part of the Retesting.	Regression testing does not include defect verification
The priority of Retesting is higher than Regression Testing, so it is done before the Regression Testing.	Based on the project type and availability of resources, regression testing can be parallel to Retesting.
Re-Test is a planned Testing.	Regression testing is a generic Testing.
We cannot automate the test-cases for Retesting.	We can do automation for regression testing; manual testing could be expensive and time-consuming.
Re-testing is for failed test-cases.	Regression testing is for passed Test-cases.
Re-testing make sure that the original fault is corrected.	Regression testing checks for unexpected side effect.

Retesting executes defects with the same data and the same environment with different input with a new build.	Regression testing is when there is a modification or changes become mandatory in an existing project.
Re-testing cannot do before start testing.	Regression testing can obtain test cases from the functional specification, user tutorials and manuals, and defects reports in regards to the corrected problem.

What are the advantages of Regression Testing?

Advantages of Regression Testing are:

- Regression Testing increases the product's quality.
- It ensures that any bug fix or changes do not impact the existing functionality of the product.
- Automation tools can be used for regression testing.
- It makes sure the issues fixed do not occur again.

What are the disadvantages of Regression Testing?

There are several advantages of Regression Testing though there are disadvantages as well.

- Regression Testing should be done for small changes in the code because even a slight change in the code can create issues in the existing functionality.
- If in case automation is not used in the project for testing, it will time consuming and tedious task to execute the test again and again.

Conclusion

Regression Testing is one of the essential aspects as it helps to deliver a quality product which saves organizations time and money. It helps to provide a quality product by making sure that any change in the code does not affect the existing functionality.

For automating the regression test cases, there are several automation tools available. A tool should have the ability to update the **test suite** as the regression test suit needs to be updated frequently.

Performance Testing

In this section, we will learn about performance testing, why we need it, types of performance testing, and the performance testing process.

Following are the topics, which we will understand in this section:

What is performance testing?

It is the most important part of non-functional testing.

Checking the behavior of an application by applying some load is known as performance testing.

Generally, this testing defines how quickly the server responds to the user's request.

While doing performance testing on the application, we will concentrate on the various factors like **Response time, Load, and Stability** of the application.

Response time: Response time is the time taken by the server to respond to the client's request.

Load: Here, Load means that when **N-number** of users using the application simultaneously or sending the request to the server at a time.

Stability: For the stability factor, we can say that, when N-number of users using the application simultaneously for a particular time.

When we use performance testing?

We will do performance testing once the software is stable and moved to the production, and it may be accessed by the multiple users concurrently, due to this reason, some performance issues may occur. To avoid these performance issues, the tester performs one round of performance testing.

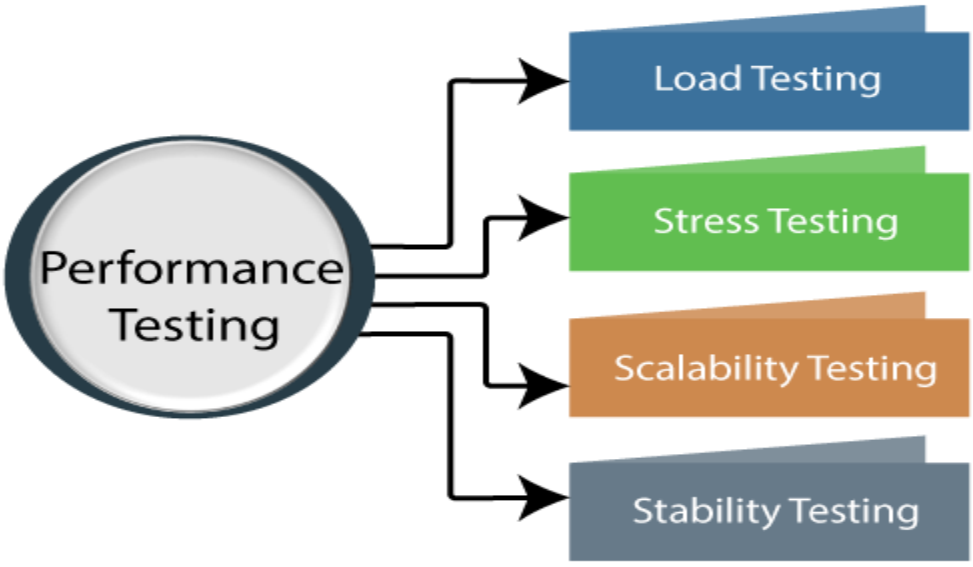
Since it is non-functional testing which doesn't mean that we always use performance testing, we only go for performance testing when the application is functionally stable.

Note: Performance testing cannot be done manually since its costly and accurate result can't be maintained.

Types of Performance Testing

Following are the types of performance testing:

- **Load testing**
- **Stress testing**
- **Scalability testing**
- **Stability testing**



discuss one by one to give you a complete understanding of **Load, Stress, Scalability,** and **Stability** performance testing.

Load testing

The load testing is used to check the performance of an application by applying some load which is either less than or equal to the desired load is known as load testing.

For example: In the below image, **1000 users** are the **desired load**, which is given by the customer, and **3/second** is the **goal** which we want to achieve while performing a load testing.



Stress Testing

The stress testing is testing, which checks the behavior of an application by applying load greater than the desired load.

For example: If we took the above example and increased the desired load 1000 to 1100 users, and the goal is 4/second. While performing the stress testing in this scenario, it will pass because the load is greater (100 up) than the actual desired load.



Scalability Testing

Checking the performance of an application by increasing or decreasing the load in particular scales (no of a user) is known as **scalability testing**. Upward scalability and downward scalability testing are called scalability testing.

Scalability testing is divided into two parts which are as follows:

- **Upward scalability testing**
- **Downward scalability testing**

Upward scalability testing

It is testing where we **increase the number of users on a particular scale** until we get a crash point. We will use upward scalability testing to find the maximum capacity of an application.

Downward scalability testing

The downward scalability testing is used when the load testing is not passed, then start **decreasing the no. of users in a particular interval** until the goal is achieved. So that it is easy to identify the bottleneck (bug).

Stability Testing

Checking the performance of an application by **applying the load for a particular duration of time** is known as **Stability Testing**.

Performance testing example

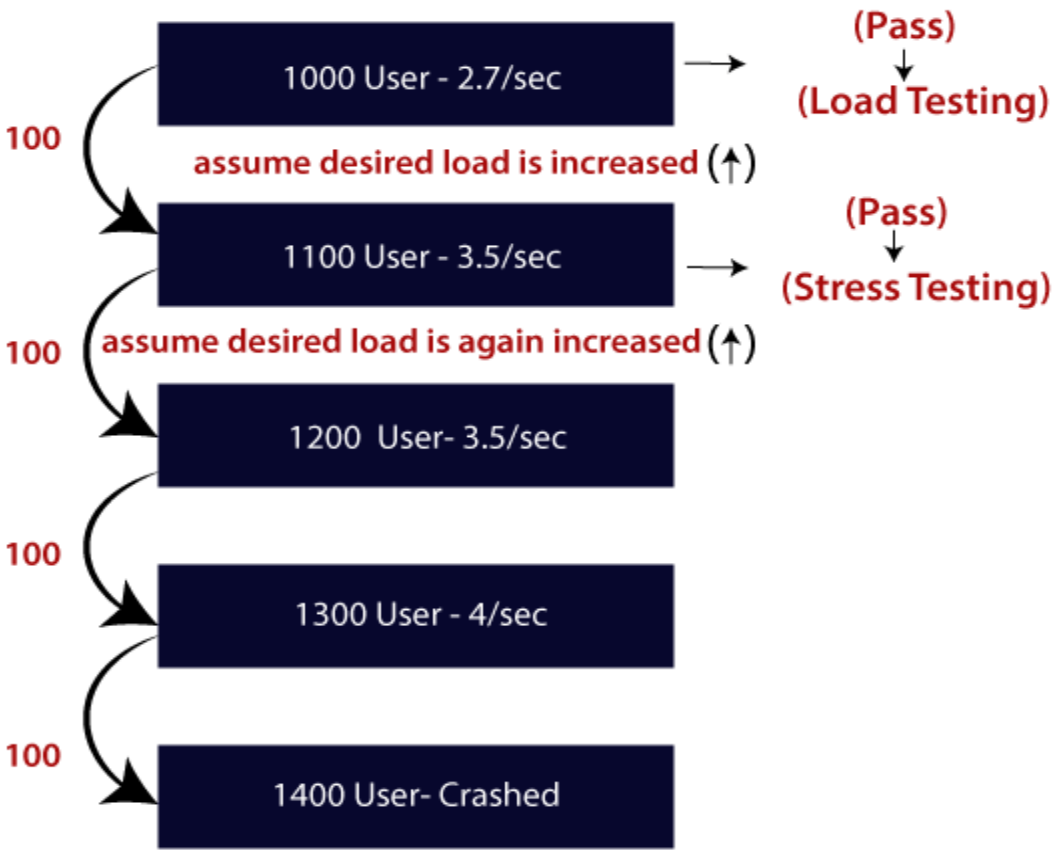
Let us take one example where we will **test the behavior of an application where the desired load is either less than 1000 or equal to 1000 users**.

In the below image, we can see that the **100 up** users are increased continuously to check the **maximum load**, which is also called **upward scalability testing**.

- **Scenario1:** When we have the 1000 users as desired load, and the 2.7/sec is goal time, these scenarios will pass while performing the load test because

in load testing, we will concentrate on the no. of users, and as per the requirement it is equal to 1000 user.

- **Scenario2:** In the next scenario, we will increase the desired load by 100 users, and goal time will go up to 3.5\sec. This scenario will pass if we perform stress testing because here, the actual load is greater than (1100) the desired load (1000).
- **Scenario3:** In this, if we increase the desired load three times as
  1200 → 3.5\sec: [it is not less than or equal to the desired load that's why it will **Fail**]
  1300 → 4\sec: [it is not less than or equal to the desired load. i.e., **Fail**]
  1400 → Crashed



Note1: Volume and soak testing is a type of testing but not performance testing.

Volume testing

Volume testing is testing, which helps us to check the behavior of an application by inserting a massive volume of the load in terms of data is known as volume testing, and here, we will concentrate on the number of data rates than the number of users.

Note2:
Volume is a capacity while Load is a quantity, i.e., load testing means no. of users, and volume testing means amount of data.

Soak testing

In this type of testing, we will check the behavior of an application on the environment, which is unsupportive for a long duration of time is known as soak testing.

Generally, soak testing is a negative type of testing since we already know that the server or environment is not supportive.

Performance testing process

The performance testing cannot be done manually since:

- We need a lot of resources, and it became a costlier approach.
- And the accuracy cannot maintain when we track response time manually.

The Performance testing process will be completed in the following steps:

- Identify performance scenarios
- Plan and design performance test script
- Configure the test environment & distribute the load
- Execute test scripts
- Result

- Analysis result
- Identify the Bottleneck
- Re-run test



If we perform a **positive flow** of the performance testing process, it could follow the below process:

Identify performance scenarios

Firstly, we will identify the performance scenarios based on these below factors:

Most commonly scenarios: It means that we can find the performance scenarios based on the scenarios, which commonly used like in the **Gmail application;** we will perform **login, inbox, send items, and compose a mail and logout**.

Most critical scenarios: Critical scenarios mean regularly used and important for the business-like in Gmail application **login, compose, inbox, and logout**.

Huge data transaction: If we have huge data means that n-number of the users using the application at the same time.

Once we identify the performance scenarios, we will move to the next step.

Plan and design performance test script

In this step, we will install the tools in the Test Engineer Machine and access the test server and then we write some script according to the test scenarios and run the tool.

Once we are done with writing the script, we will go to the next step.

Configure the test environment & distribute the load

After writing the test scripts, we will arrange the testing environment before the execution. And also, manage the tools, other resources and distribute the load according to the "Usage Pattern" or mention the duration and stability.

Execute test scripts

Once we are done with distributing the load, we will execute, validate, and monitor the test scripts.

Result

After executing the test scripts, we will get the test result. And check that the result meeting the goal in the given response time or not, and the response time could be maximum, average, and minimum.

If the response is not meeting the required time response, then we will go for the **negative flow** where will perform the below steps:

Analysis result

First, we will analyze the test result whether it meets with the response time or not.

Identify the Bottleneck

After that, we will identify the **bottleneck (bug or performance issue**). And the bottleneck could occur because of these aspects like the **problem in code, hardware issue (hard disk, RAM Processor), network issues,** and the **software issue (operating system)**. And after finding the bottleneck, we will perform **tuning (fix or adjustment)** to resolve this bottleneck.

Re-run test

Once we fix the bottlenecks, re-run the test scripts and checks the result whether it meets the required goal or not.

The problem occurs in performance testing

While performing performance testing on the application, some problems may occur, and these problems are also called the **performance issue**.

The performance issues are as follows:

- **Response time issue**
- **Scalability issue**
- **Bottleneck**
- **Speed issue**

Response time issue

The response time means how quickly the server respond to the client's request. If the user's request does not complete in the given response time, it might have possible that the user may be lost his/her interest in the particular software or application. That's why the application or software should have a perfect response time for responding user's request quickly.

Scalability issue

The scalability issues occur when the application cannot take the n-numbers of users and expected user requests at the same time. That's why we will do **upward scalability testing** (check the maximum capacity of the application) and **downward scalability testing** (when expected time is not matched with the actual time).

Bottleneck

The Bottleneck is the informal name of bug, which occurs when the application is limited by a single component and creates a bad impact on the system performance.

The main causes of bottlenecking are **software issues (issue related to the operating system), hardware issues (issues related to the hard disk, RAM and the processor),** and **coding issue,** etc.

Following are the most common performance bottlenecks:

- Memory utilization
- Disk usage
- CPU utilization
- Operating System limitations
- Network utilization

Speed issues

When we perform performance testing on the application, the application should be faster in speed to get the user's interest and attention because if the application's speed is slow, it may lose the user interest in the application.

Performance test tools

We have various types of performance testing tools available in the market, where some are commercial tools and open-source tool.

Commercial tools: LoadRunner[HP], WebLOAD, NeoLoad

Open-source tool: JMeter

LoadRunner

It is one of the most powerful tools of performance testing, which is used to support the performance testing for the extensive range of protocols, number of technologies, and application environments.

It quickly identifies the most common causes of performance issues. And also accurately predict the application scalability and capacity.

JMeter

The Apache JMeter software is an open- source tool, which is an entirely a Java application designed to load the functional test behavior and measure the performance.

Generally, it was designed for testing the Web Applications but now expanded to other test functions also.

Apache JMeter is used to test performance for both static and dynamic resources and dynamic web applications.
It can be used to reproduce the heavy load on a server, network or object, group of servers to test its strength or to analyze overall performance under different load types.

WebLOAD

WebLOAD testing tool used to test the load testing, performance testing, and stress test web applications.

The WebLOAD tool combines performance, scalability, and integrity as a single process for the verification of web and mobile applications.

NeoLoad

Neotys develop a testing tool which is called NeoLoad. The NeoLoad is used to test the performance test scenarios. With the help of NeoLoad, we can find the bottleneck areas in the web and the mobile app development process.

The NeoLoad testing tool is faster as compared to traditional tools.

Apart from them, some other tools are **Electric load, web stress tool, LoadUI Pro, StresStimulus, LoadView, LoadNinja, and RedLine13,** which helps to test the performance of the software or an application.

Basis Path Testing

Basis path testing, a structured testing or white box testing technique used for designing test cases intended to examine all possible paths of execution at least once. Creating and executing tests for all possible paths results in 100% statement coverage and 100% branch coverage.

Basis Path Testing is a white-box testing technique based on the control structure of a program or a module. Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing. Therefore, by definition,

Basis path testing is a technique of selecting the paths in the control flow graph, that provide a basis set of execution paths through the program or module.

Since this testing is based on the control structure of the program, it requires complete knowledge of the program's structure. To design test cases using this technique, four steps are followed :

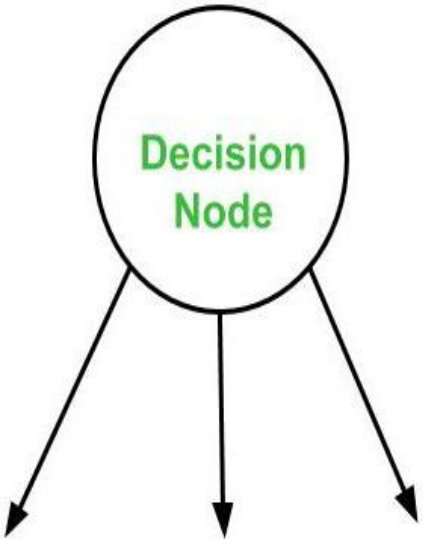
1. Construct the Control Flow Graph
2. Compute the Cyclomatic Complexity of the Graph
3. Identify the Independent Paths
4. Design Test cases from Independent Paths

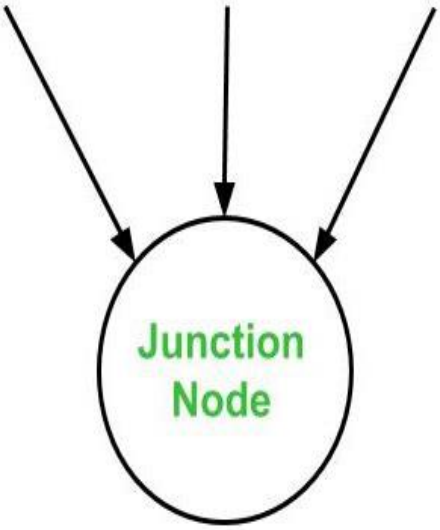
Understand each step one by one.

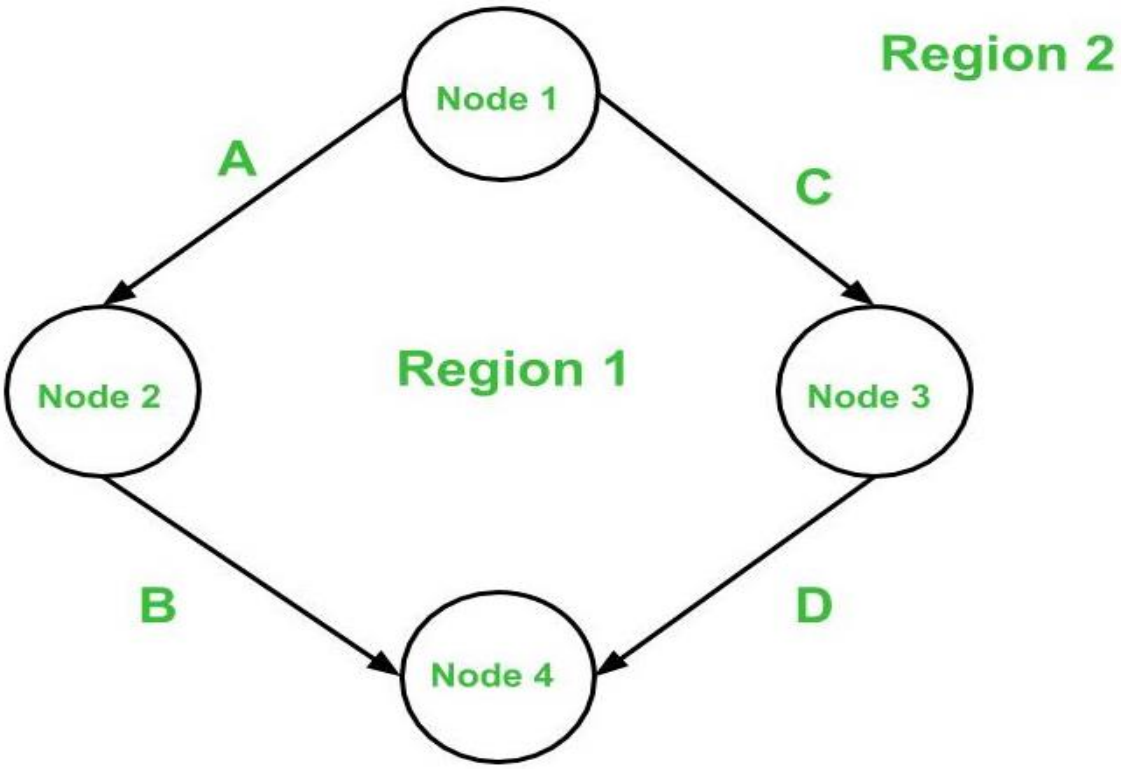
1. Control Flow Graph –

A control flow graph (or simply, flow graph) is a directed graph which represents the control structure of a program or module. A control flow graph (V, E) has V number of nodes/vertices and E number of edges in it. A control graph can also have :

- **Junction Node –** a node with more than one arrow entering it.
- **Decision Node –** a node with more then one arrow leaving it.
- **Region –** area bounded by edges and nodes (area outside the graph is also counted as a region.).

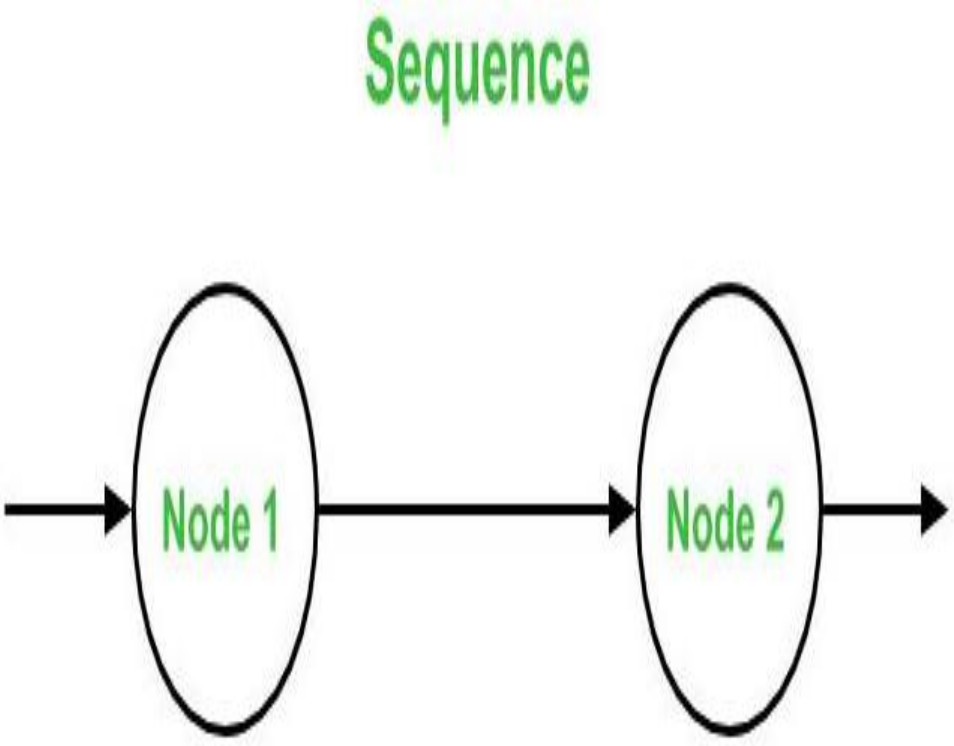




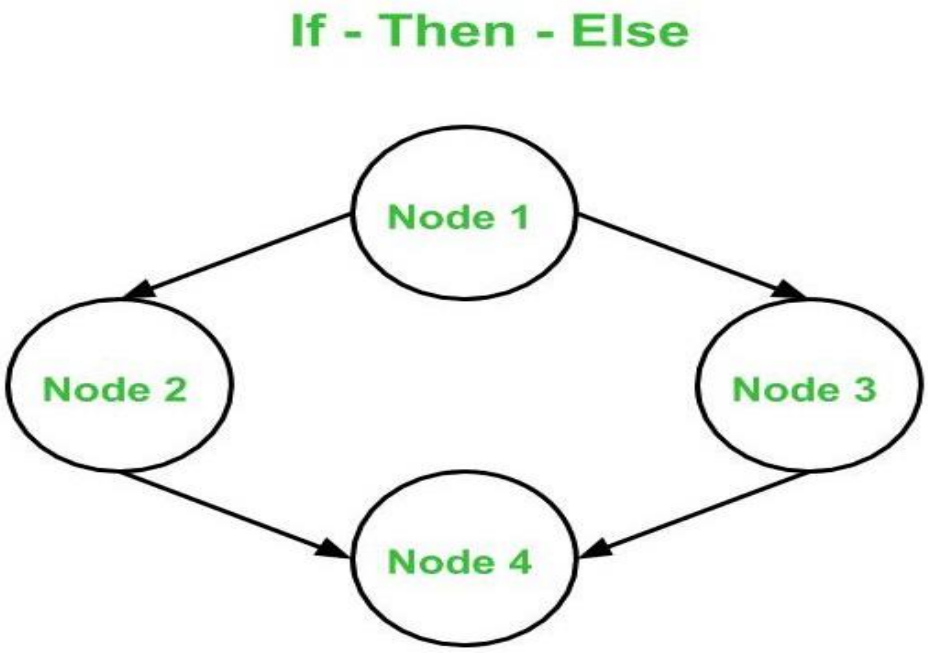


Below are the **notations** used while constructing a flow graph :

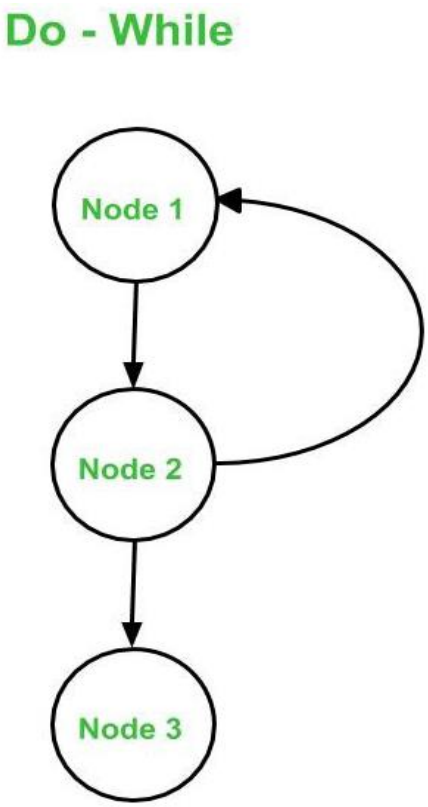
- **Sequential Statements –**



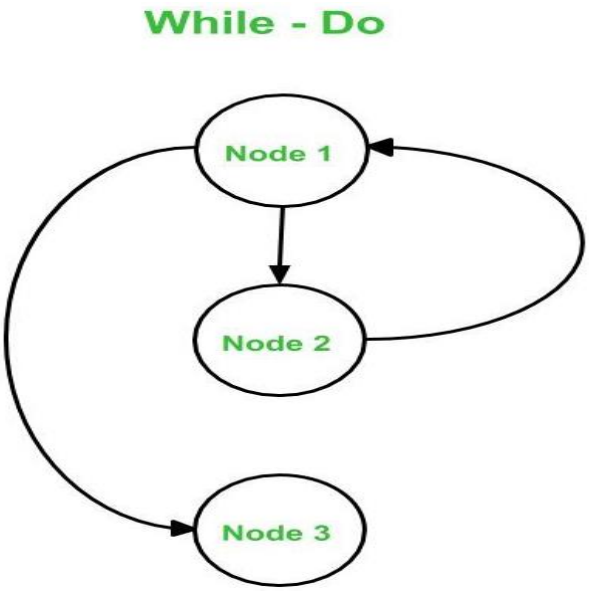
- **If – Then – Else –**



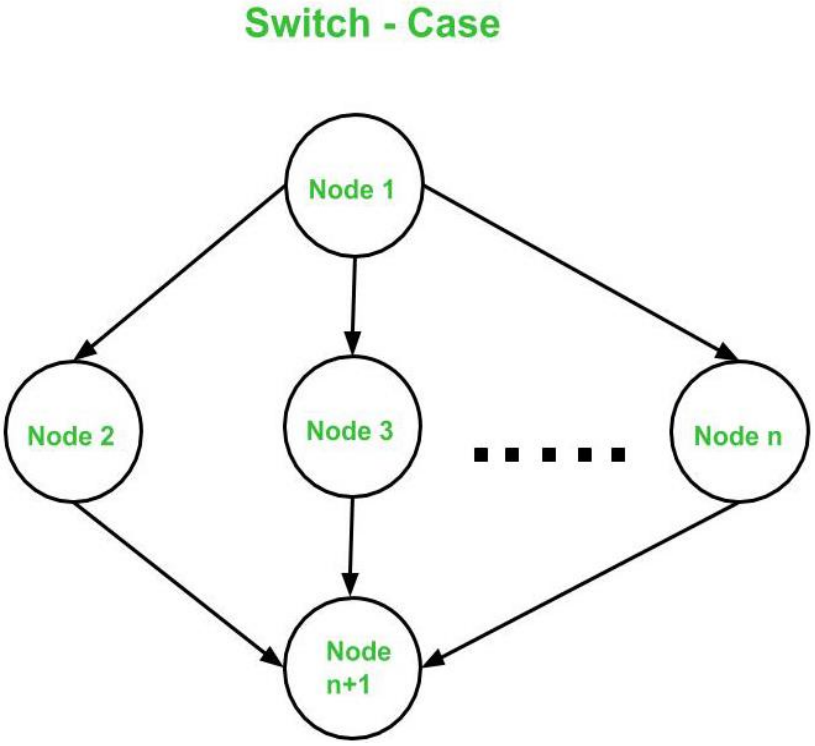
- **Do – While –**



- **While – Do –**



- **Switch – Case –**



Cyclomatic Complexity –

The cyclomatic complexity V(G) is said to be a measure of the logical complexity of a program. It can be calculated using three different formulae :

1. **Formula based on edges and nodes :**

V(G) = e - n + 2*P

Where,
e is number of edges,
n is number of vertices,
P is number of connected components.

For example, consider first graph given above,

where, e = 4, n = 4 and p = 1

So,

Cyclomatic complexity V(G)

= 4 - 4 + 2 * 1

= 2

2. **Formula based on Decision Nodes :**

V(G) = d + P

where,
d is number of decision nodes,
P is number of connected nodes.

For example, consider first graph given above,

where, d = 1 and p = 1

So,

Cyclomatic Complexity V(G)

= 1 + 1

= 2

3. **Formula based on Regions :**

V(G) = number of regions in the graph

For example, consider first graph given above,

Cyclomatic complexity V(G)

= 1 (for Region 1) + 1 (for Region 2)

= 2

Hence, using all the three above formulae, the cyclomatic complexity obtained remains same. All these three formulae can be used to compute and verify the cyclomatic complexity of the flow graph.

Note –

1. For one function [e.g. Main( ) or Factorial( )], only one flow graph is constructed. If in a program, there are multiple functions, then a separate flow graph is constructed for each one of them. Also, in the cyclomatic complexity formula, the value of 'p' is set depending of the number of graphs present in total.
2. If a decision node has exactly two arrows leaving it, then it is counted as one decision node. However, if there are more than 2 arrows leaving a decision node, it is computed using this formula :

$$d = k - 1$$

Here, k is number of arrows leaving the decision node.

Independent Paths :
An independent path in the control flow graph is the one which introduces at least one new edge that has not been traversed before the path is defined. The

cyclomatic complexity gives the number of independent paths present in a flow graph. This is because the cyclomatic complexity is used as an upper-bound for the number of tests that should be executed in order to make sure that all the statements in the program have been executed at least once.

Consider first graph given above here the independent paths would be 2 because number of independent paths is equal to the cyclomatic complexity.
So, the independent paths in above first given graph :

- **Path 1:**

  A -> B

- **Path 2:**

  C -> D

Note –
Independent paths are not unique. In other words, if for a graph the cyclomatic complexity comes out be N, then there is a possibility of obtaining two different sets of paths which are independent in nature.

Design Test Cases :
Finally, after obtaining the independent paths, test cases can be designed where each test case represents one or more independent paths.

Advantages :
Basis Path Testing can be applicable in the following cases:

1. **More Coverage –**
   Basis path testing provides the best code coverage as it aims to achieve maximum logic coverage instead of maximum path coverage. This results in an overall thorough testing of the code.

2. **Maintenance Testing –**
   When a software is modified, it is still necessary to test the changes made in the software which as a result, requires path testing.

3. **Unit Testing –**
   When a developer writes the code, he or she tests the structure of the program or module themselves first. This is why basis path testing requires enough knowledge about the structure of the code.

4. **Integration Testing –**
   When one module calls other modules, there are high chances of Interface errors. In order to avoid the case of such errors, path testing is performed to test all the paths on the interfaces of the modules.

5. **Testing Effort –**
   Since the basis path testing technique takes into account the complexity of the software (i.e., program or module) while computing the cyclomatic complexity, therefore it is intuitive to note that testing effort in case of basis path testing is directly proportional to the complexity of the software or program.

Example:

```
Function fn_delete_element (int value, int array_size, int array[])
{
        1 int i;
        location = array_size + 1;
        2 for i = 1 to array_size
        3 if ( array[i] == value )
        4 location = i;
         end if;
         end for;
```

```
5 for i = location to array_size

6 array[i] = array[i+1];

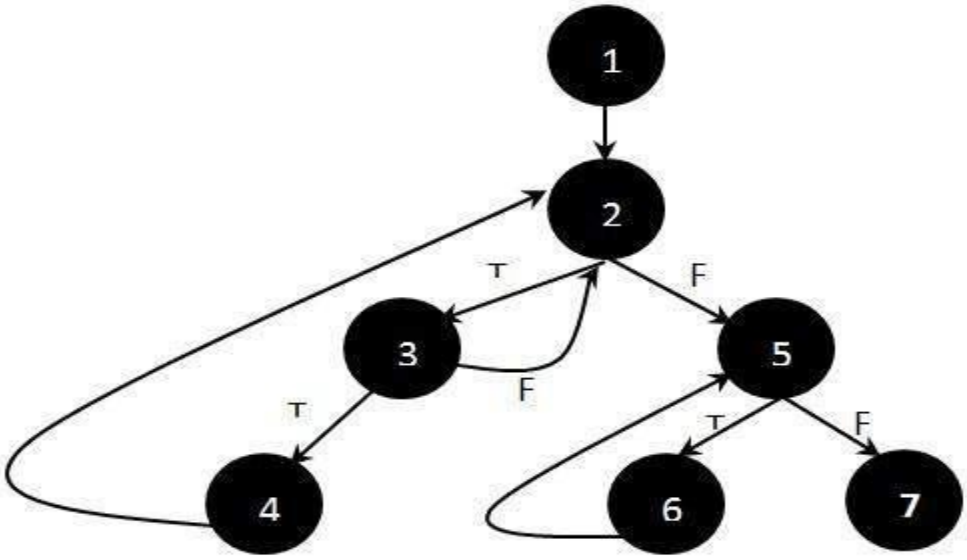
end for;

7 array_size --;

}
```

Steps to Calculate the independent paths

Step 1 : Draw the Flow Graph of the Function/Program under consideration as shown below:



Step 2 : Determine the independent paths.

Path 1: 1 - 2 - 5 - 7

Path 2: 1 - 2 - 5 - 6 - 7

Path 3: 1 - 2 - 3 - 2 - 5 - 6 - 7

Path 4: 1 - 2 - 3 - 4 - 2 - 5 - 6 - 7

Control structure testing

Control structure testing is used to increase the coverage area by testing various control structures present in the program. The different types of testing performed under control structure testing are as follows-

1. Condition Testing

2. Data Flow Testing

3. Loop Testing

1. Condition Testing :
Condition testing is a test cased design method, which ensures that the logical condition and decision statements are free from errors. The errors present in logical conditions can be incorrect boolean operators, missing parenthesis in a booleans expression, error in relational operators, arithmetic expressions, and so on.

The common types of logical conditions that are tested using condition testing are-

1. A relation expression, like E1 op E2 where 'E1' and 'E2' are arithmetic expressions and 'OP' is an operator.
2. A simple condition like any relational expression preceded by a NOT (~) operator.
   For example, (~E1) where 'E1' is an arithmetic expression and 'a' denotes NOT operator.
3. A compound condition consists of two or more simple conditions, Boolean operator, and parenthesis.
   For example, (E1 & E2)|(E2 & E3) where E1, E2, E3 denote arithmetic expression and '&' and '|' denote AND or OR operators.
4. A Boolean expression consists of operands and a Boolean operator like 'AND', OR, NOT.

For example, 'A|B' is a Boolean expression where 'A' and 'B' denote operands and | denotes OR operator.

2. Data Flow Testing :

The data flow test method chooses the test path of a program based on the locations of the definitions and uses all the variables in the program.

The data flow test approach is depicted as follows suppose each statement in a program is assigned a unique statement number and that theme function cannot modify its parameters or global variables.
For example, with S as its statement number.

DEF (S) = {X | Statement S has a definition of X}

USE (S) = {X | Statement S has a use of X}

If statement S is an if loop statement, them its DEF set is empty and its USE set depends on the state of statement S. The definition of the variable X at statement S is called the line of statement S' if the statement is any way from S to statement S' then there is no other definition of X.

A definition use (DU) chain of variable X has the form [X, S, S'], where S and S' denote statement numbers, X is in DEF(S) and USE(S'), and the definition of X in statement S is line at statement S'.

A simple data flow test approach requires that each DU chain be covered at least once. This approach is known as the DU test approach. The DU testing does not ensure coverage of all branches of a program.

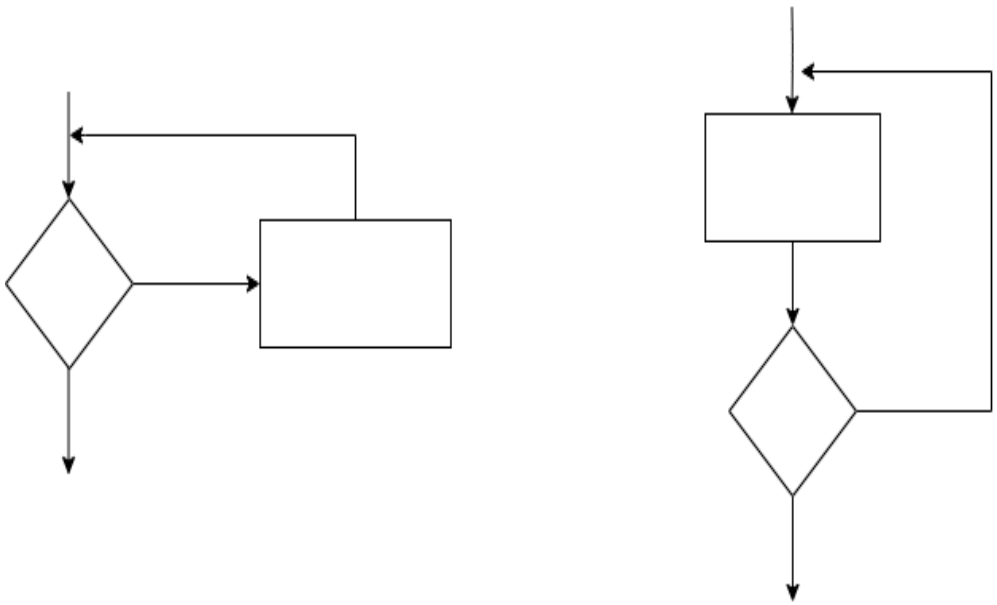
However, a branch is not guaranteed to be covered by DU testing only in rar cases such as then in which the other construct does not have any certainty of any variable in its later part and the other part is not present. Data flow testing strategies are appropriate for choosing test paths of a program containing nested if and loop statements.

3. Loop Testing :

Loop testing is actually a white box testing technique. It specifically focuses on the

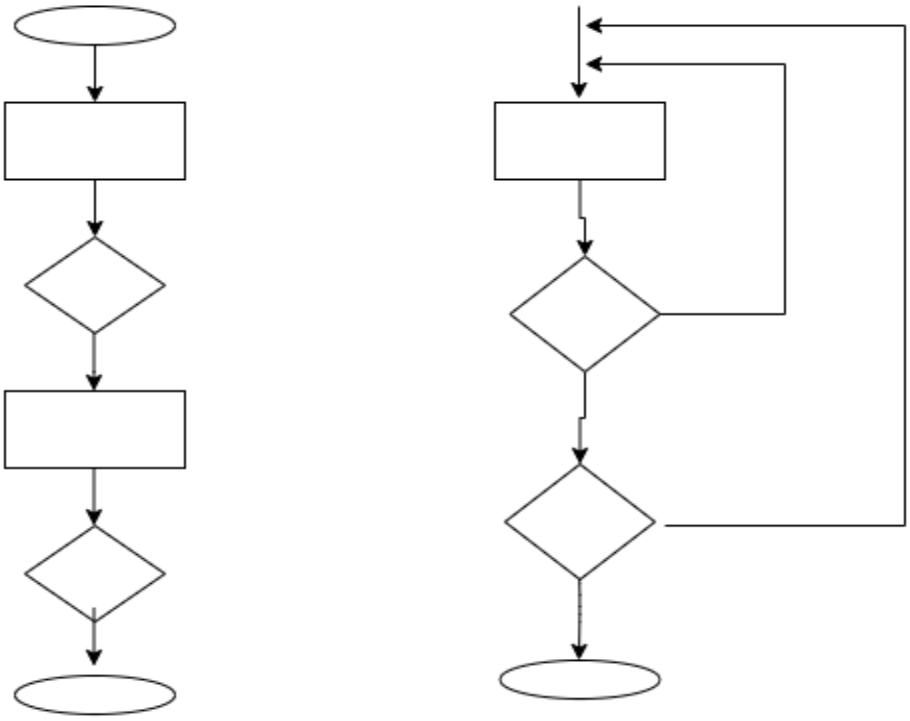
validity of loop construction.
Following are the types of loops.

1. **Simple Loop –** The following set of test can be applied to simple loops, where the maximum allowable number through the loop is n.
    1. Skip the entire loop.
    2. Traverse the loop only once.
    3. Traverse the loop two times.
    4. Make p passes through the loop where p<n.
    5. Traverse the loop n-1, n, n+1 times.



SIMPLE LOOPS

2. **Concatenated Loops –** If loops are not dependent on each other, contact loops can be tested using the approach used in simple loops. if the loops are interdependent, the steps are followed in nested loops.

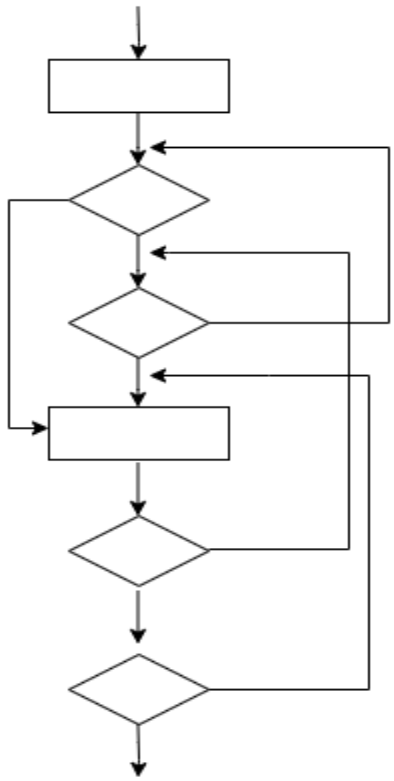


Concatenated Loops

3. **Nested Loops –** Loops within loops are called as nested loops. when testing nested loops, the number of tested increases as level nesting increases. The following steps for testing nested loops are as follows-

    1. Start with inner loop. set all other loops to minimum values.
    2. Conduct simple loop testing on inner loop.
    3. Work outwards.
    4. Continue until all loops tested.

4. **Unstructured loops –** This type of loops should be redesigned, whenever possible, to reflect the use of unstructured the structured programming constructs.



Unstructured Loops

Deriving Test Cases from Use Cases: A Four-Step Process

Use cases and test cases come from different origins and serve different, albeit related , purposes. Moving from one to another is a nontrivial but reasonably stepwise process. Now that we've defined the concept of use-case scenarios, we can prescribe a four-step process to accomplish this objective.

1. **Identify the use-case scenarios.**
2. For each scenario, identify one or more test cases.
3. For each test case, identify the conditions that will cause it to execute.
4. Complete the test case by adding data values.

Step 1: Identify the Use-Case Scenarios

Since we have a oneto-many relationship between use cases and scenarios, we need some organizing technique to manage this potential information explosion.

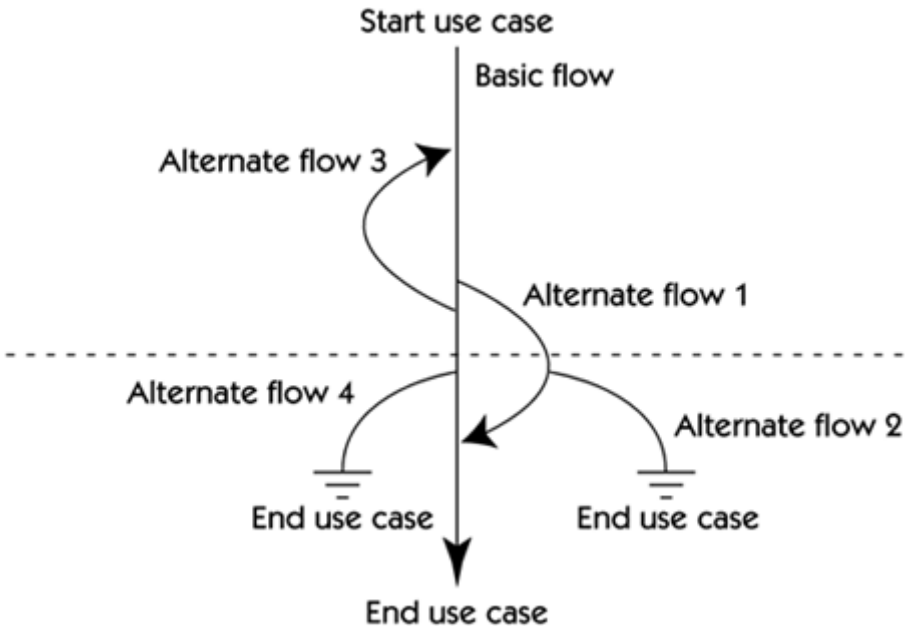
We'll use a simple matrix that can be implemented in a spreadsheet, database, or test management tool. In order to keep track of things, we also need to number each scenario and define exactly what combination of basic and alternate flows that particular scenario represents. Returning to Figure 2, we could construct a matrix similar to the one in Table 1.

In Table 26-1 we've identified eight possible scenarios for the sample use case. Note that the use case we've described is not an overly complex one, and yet a significant number of scenarios may result. As the use cases grow more complex, more and more scenarios will result. In many situations, the tester will need to devise a testing strategy that recognizes that it is impractical to test all possible scenarios but still assures that adequate testing is achieved. Even then, this technique can be used to identify all the possible scenarios that could be tested .

Table1. Scenario Matrix for Figure 26-2

Scenario Number	**Originating Flow**	**Alternate Flow**	**Next Alternate**	**Next Alternate**
1	Basic flow			
2	Basic flow	Alternate flow 1		
3	Basic flow	Alternate flow 1	Alternate flow 2	
4	Basic flow	Alternate flow 3		
5	Basic flow	Alternate flow 3	Alternate flow 1	
6	Basic flow	Alternate flow 3	Alternate flow 1	Alternate flow 2

Scenario Number	Originating Flow	Alternate Flow	Next Alternate	Next Alternate
7	Basic flow	Alternate flow 4		
8	Basic flow	Alternate flow 3	Alternate flow 4	



In addition, the tester should be aware that not all scenarios may be described in the original use case and that this scenario discovery process may well need to be conducted interactively with the development team. There are two reasons for this.

1. **The use cases developed for implementation are not 100 percent exhaustive and are written at a level of detail that may be insufficient for testing.**
2. The test team's review process will create new discoveries and additional scenarios that may result from executing the use case. Some of these may not even have been considered in the design and may thus require design modifications.

This is yet another reason we've adopted the iterative model in our lifecycle approach; it allows us to effectively plan for and manage this process. In any event, the process is a beneficial one, and a better system should result as the testing team reviews the use cases and finds holes and additional alternate scenarios.

Step 2: Identify the Test Cases

Testing processes vary from company to company and even from project to project, but in each situation a test case documents a number of common items. Whether in a spreadsheet, document, database, or testing tool, the test case should contain the parameters of the test to be conducted, including the conditions of the test and the expected results. One common format, illustrated in Table2, is to use a matrix in which each row represents a specific test case and the columns represent scenarios, conditions, data values, and expected and actual results.

Table2. Matrix for Testing Specific Scenarios

Test Case ID	**Scenario/ Condition**	**Data Value 1**	**Data Value 2**	**Data Value N**	**Expected Result**	**Actual Result**
1	Scenario 1					
2	Scenario 2					
3	Scenario 2					

Note in Table that more than one test case can result from a specific scenario (see test case IDs 2 and 3, both for scenario 2). Typically, this arises because of various logical constructs identified in a single step in a use case. For example, consider the following step in a HOLIS use case.

The homeowner enters the desired lighting sequence for each day of the week up to a maximum of seven different daily settings. The system confirms acceptance of each daily entry with a beep.

This single step in the use case will produce two test cases from this step (Table 3).

In addition, note that in this process we've discovered an ambiguity that will have to be resolved: "What does the system do if the user attempts to enter more than seven settings?" The test team will have to caucus with the development team on that one. Such is the nature of our iterative discovery process.

Table3. Two Test Cases for One Scenario

Test Case ID	**Scenario/ Condition**	**Description**	**Expected Result**
1	Scenario 6	Less than seven sequences entered	Sequence saved System beeps
2	Scenario 6	Attempt to enter an eighth sequence	Error?

Step 3: Identify the Test Conditions

The next step is to identify the specific conditions in the test case that would cause it to execute. In other words, what conditions cause a user to execute a specific event or sequence of events within a use case? During this process, the tester searches the use-case steps for the various data conditions, branches, and so on that would cause a specific test case to occur. For each such condition identified, the tester enters a new column in the matrix, representing the specific condition identified. In this initial pass, it is adequate to simply identify that a condition exists, create the column entry, and then indicate which of the three states that could occur for that condition (valid, invalid, or not applicable) is appropriate.

1. **Valid (V) indicates a condition that must be true for the basic flow to execute.**

2. Invalid (I) indicates a condition that will invoke the alternate flow, causing a specific scenario to occur.
3. Not applicable (N/A) indicates that an identified condition is not applicable to that specific test case ID.

To illustrate , let's look at another HOLIS example. Consider the Control Light use case described in Table 4.

In this use case, we have three separate conditions to consider (button pressed for less than one second, button pressed for greater than one second, and button released after being pressed for more than one second) that trigger changes in the behavior of the system. Specifically, they trigger scenario 1, scenario 2, and scenario 3, which in this case translates to the basic flow and alternate flows plus a branch on the alternate flow.

We can now create a matrix of the test cases, Table 26-5, which identifies each of these conditions as valid or invalid, and we can document the expected result in each case.

Table4. Sample HOLIS Use Case: Control Light

Item	**Value**
Use-case name	Control Light
Actors	Resident and Light Bank
Brief description	This use case prescribes the way in which lights are turned on and off and also how they are dimmed and brightened in accordance with how long the user presses a button on the Control Switch.
Flow of events	Basic flow begins when the Resident presses the On/Off/Dim button on the Control Switch. When the Resident removes pressure on the On/Off/Dim button within the timer period, the system "toggles" the state of the

Item	Value
	light as follows . ○ If the light is On, the light is then turned Off, and there is no illumination . ○ If the light is Off, the light is then turned On to the last remembered brightness level.
Alternative flow of events	When the Resident holds down the On/Off/Dim button for more than 1 second, the system initiates a brightening/ dimming activity for the room's Light Bank. While the Resident continues to press the On/Off/Dim button: 1. **The brightness of the controlled light is smoothly increased to a system-wide maximum value at a rate of 10 percent per second.** 2. When the brightness reaches its maximum value, the brightness of the controlled light is then smoothly decreased to a system-wide minimum value at a rate of 10 percent per second. 3. When the brightness reaches its minimum value, the use case continues at subflow step 1. When the Resident releases the On/Off/Dim button: 4. The use case terminates and the brightness stays at the current level.
Pre-conditions	The selected On/Off/Dim button must be Dim Enabled. The selected On/Off/Dim button must be preprogrammed to control a Light Bank.

Item	Value
Post-condition	On leaving this use case, the system remembers the current brightness level for the selected On/Off/Dim button.
Special requirements	Performance: For any action that is perceptible to the Resident, the response time from a control panel action to system response must be less than 50 milliseconds .

Step 4: Add Data Values to Complete the Test Case

We've made good progress. We have now identified all the conditions that need to be tested to test a specific use case fully. We're not quite done, howeverwithout real data, test cases are merely descriptions of conditions, scenarios, and paths without concrete values to identify them succinctly. Without such data, it's not possible to execute the test case and determine the results. In many cases, the use cases themselves will not directly identify these data values, and you will have to look to supplementary specifications to find performance specs , valid data ranges for input forms and interface protocols, and so on. However, this is not a new problem to the tester; just use your normal techniques for finding the data ranges.

Table5. Control Light Test Cases with Identified Conditions

Test Case ID	Scenario	Description	Condition: Button Pressed < Timer Period	Condition: Button Pressed > Timer Period	Condition: Button Released After Being Held	Condition	Expected Result
1	1	Basic flow : Resident releases button before timer	V	I	N/A	Light on	Light goes off

Test Case ID	Scenario	Description	Condition: Button Pressed < Timer Period	Condition: Button Pressed > Timer Period	Condition: Button Released After Being Held	Condition	Expected Result
		period ends					
2	1	Basic flow : Resident releases button before timer period ends	V	I	N/A	Light off	Light goes on
3	2	Alternate flow : Resident continuously presses button for longer than timer period	I	V	N/A	N/A	Light level goes up and down continuously
4	3	Resident releases switch after continuously pressing button	I	I	V	N/A	Light stays at last brightness

This is also the time to make sure that test cases address any special requirements defined for the use case. These include such things as minimum/maximum performance, sometimes combined with

minimum/maximum loads, or data volumes expected during the execution of the use cases.

Once you have identified the data ranges, you can finalize the test matrix with those values. Then you are ready to execute the test cases. For our HOLIS Control Light test case, the result might look like Table 26-6.

Table6. Control Light Test Cases with Data Values

Test Case ID	Scenario	Description	Condition: Button Pressed < Timer Period	Condition: Button Pressed > Timer Period	Condition: Button Released After Being Held	Condition	Expected Result
1	1	Basic flow : Resident releases button before timer period ends	< 1 sec. in .1-sec. intervals	I	N/A	Light on	Light goes off
2	1	Basic flow : Resident releases button before timer period ends	< 1 sec. in .1-sec. intervals	I	N/A	Light off	Light goes on
3	2	Alternate flow : Resident continuously presses	I	160 sec.	N/A	N/A	Light level goes up and down continuously

Test Case ID	Scenario	Description	Condition: Button Pressed < Timer Period	Condition: Button Pressed > Timer Period	Condition: Button Released After Being Held	Condition	Expected Result
		button for longer than timer period					
4	3	Resident releases switch after continuously pressing button	I	I	V	N/A	Light stays at last brightness

Software Configuration Management

When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.

Several individuals (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.

The elements that comprise all information produced as a part of the software process are collectively called a software configuration.

As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.

These are handled and controlled by SCM. This is where we require software configuration management.

A configuration of the product refers not only to the product's constituent but also to a particular version of the component.

Therefore, SCM is the discipline which

- Identify change
- Monitor and control change
- Ensure the proper implementation of change made to the item.
- Auditing and reporting on the change made.

Configuration Management (CM) is a technic of identifying, organizing, and controlling modification to software being built by a programming team.

The objective is to maximize productivity by minimizing mistakes (errors).

CM is used to essential due to the inventory management, library management, and updation management of the items essential for the project.

Why do we need Configuration Management?

Multiple people are working on software which is consistently updating. It may be a method where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently. It changes in user requirements, and policy, budget, schedules need to be accommodated.

Importance of SCM

It is practical in controlling and managing the access to various SCIs e.g., by preventing the two members of a team for checking out the same component for modification at the same time.

It provides the tool to ensure that changes are being properly implemented.

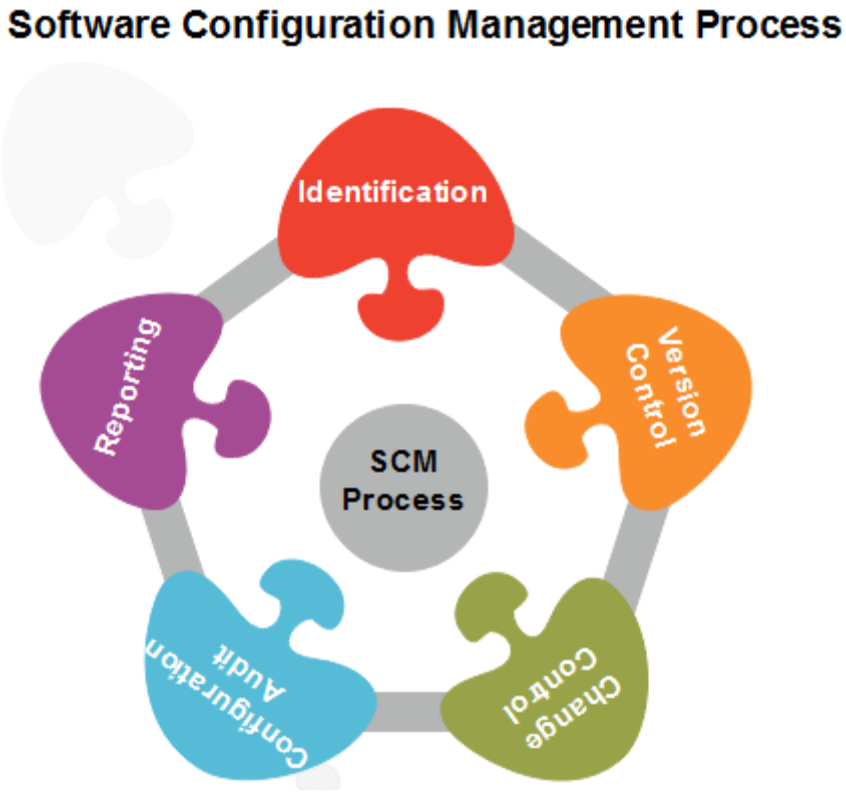
It has the capability of describing and storing the various constituent of software.

SCM is used in keeping a system in a consistent state by automatically producing derived version upon modification of the same component.

SCM Process

It uses the tools which keep that the necessary change has been implemented adequately to the appropriate component. The SCM process defines a number of tasks:

- Identification of objects in the software configuration
- Version Control
- Change Control
- Configuration Audit
- Status Reporting



Identification

Basic Object: Unit of Text created by a software engineer during analysis, design, code, or test.

Aggregate Object: A collection of essential objects and other aggregate objects. Design Specification is an aggregate object.

Each object has a set of distinct characteristics that identify it uniquely: a name, a description, a list of resources, and a "realization."

The interrelationships between configuration objects can be described with a Module Interconnection Language (MIL).

Version Control

Version Control combines procedures and tools to handle different version of configuration objects that are generated during the software process.

Clemm defines version control in the context of SCM: Configuration management allows a user to specify the alternative configuration of the software system through the selection of appropriate versions. This is supported by associating attributes with each software version, and then allowing a configuration to be specified [and constructed] by describing the set of desired attributes.

Change Control

James Bach describes change control in the context of SCM is: Change Control is Vital. But the forces that make it essential also make it annoying.

We worry about change because a small confusion in the code can create a big failure in the product. But it can also fix a significant failure or enable incredible new capabilities.

We worry about change because a single rogue developer could sink the project, yet brilliant ideas originate in the mind of those rogues, and

A burdensome change control process could effectively discourage them from doing creative work.

A change request is submitted and calculated to assess technical merit; potential side effects, the overall impact on other configuration objects and system functions, and projected cost of the change.

The results of the evaluations are presented as a change report, which is used by a change control authority (CCA) - a person or a group who makes a final decision on the status and priority of the change.

The "check-in" and "check-out" process implements two necessary elements of change control-**access control** and **synchronization control**.

Access Control governs which software engineers have the authority to access and modify a particular configuration object.

Synchronization Control helps to ensure that parallel changes, performed by two different people, don't overwrite one another.

Configuration Audit

SCM audits to verify that the software product satisfies the baselines requirements and ensures that what is built and what is delivered.

SCM audits also ensure that traceability is maintained between all CIs and that all work requests are associated with one or more CI modification.

SCM audits are the "**watchdogs**" that ensures that the integrity of the project's scope is preserved.

Status Reporting

Configuration Status reporting (sometimes also called status accounting) providing accurate status and current configuration data to developers, testers, end users, customers and stakeholders through admin guides, user guides, FAQs, Release Notes, Installation Guide, Configuration Guide, etc.

Software Reuse

A definition of software reuse is the process of creating software systems from predefined software components.

The advantage of software reuse:

- The systematic development of reusable components.
- The systematic reuse of these components as building blocks to create new systems.

A reusable component may be code, but the bigger benefits of reuse come from a broader and higher-level view of what can be reused. Software specifications, designs, tests cases, data, prototypes, plans, documentation, frameworks, and templates are all candidates for reuse.

Software reuse can cut software development time and costs. The major advantages for software reuse are to:

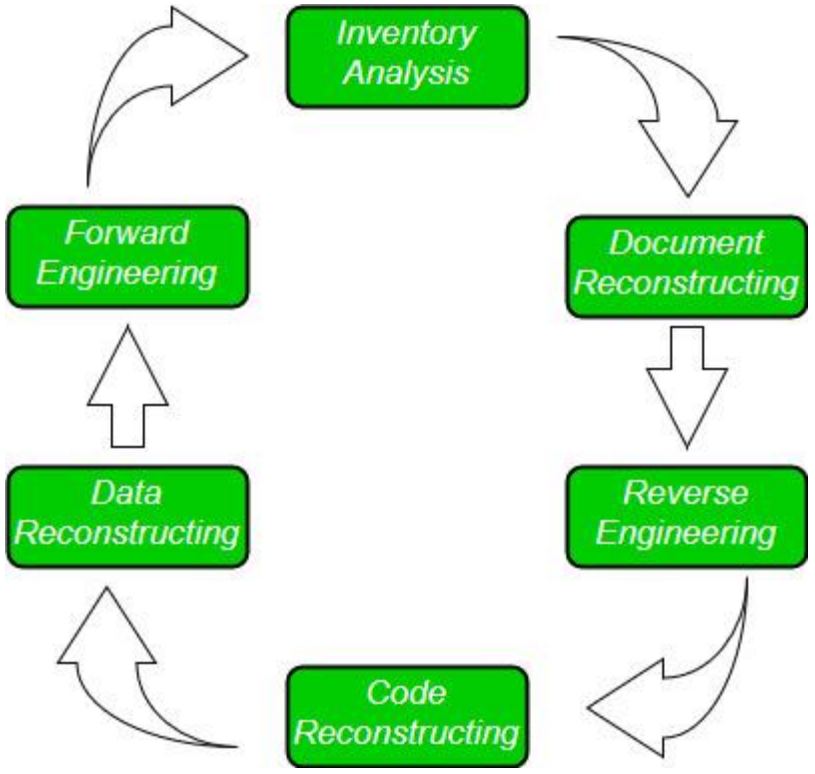
- Increase software productivity.
- Shorten software development time.
- Improve software system interoperability.
- Develop software with fewer people.
- Move personnel more easily from project to project.
- Reduce software development and maintenance costs.
- Produce more standardized software.
- Produce better quality software and provide a powerful competitive advantage.

Software Re-Engineering

Software Re-Engineering is the examination and alteration of a system to reconstitute it in a new form. The principles of Re-Engineering when applied to the software development process is called software re-engineering. It affects positively at software cost, quality, service to the customer and speed of delivery. In Software Re-engineering, we are improving the software to make it more efficient and effective.

Re-Engineering cost factors:

- The quality of the software to be re-engineered.
- The tool support availability for engineering.
- Extent of the data conversion which is required.
- The availability of expert staff for Re-engineering.



Software Re-Engineering Activities:

1. Inventory Analysis:

Every software organisation should have an inventory of all the applications.

- Inventory can be nothing more than a spreadsheet model containing information that provides a detailed description of every active application.
- By sorting this information according to business criticality, longevity, current maintainability and other local important criteria, candidates for re-engineering appear.
- Resource can then be allocated to candidate application for re-engineering work.

2. Document reconstructing:

Documentation of a system either explains how it operate or how to use it.

- Documentation must be updated.
- It may not be necessary to fully document an application.
- The system is business critical and must be fully re-documented.

3. Reverse Engineering:

Reverse engineering is a process of design recovery. Reverse engineering tools extracts data, architectural and proccedural design information from an existing program.

4. Code Reconstructing:

- To accomplish code reconstructing, the source code is analysed using a reconstructing tool. Violations of structured programming construct are noted and code is then reconstruct.
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced.

5. Data Restructuring:

- Data restructuring begins with the reverse engineering activity.
- Current data architecture is dissecred, and necessary data models are defined.
- Data objects and attributes are identified, and existing data structure are reviewed for quality.

6. Forward Engineering:

Forward Engineering also called as renovation or reclamation not only for recovers design information from existing software but uses this information to alter or reconstitute the existing system in an effort to improve its overall quality.

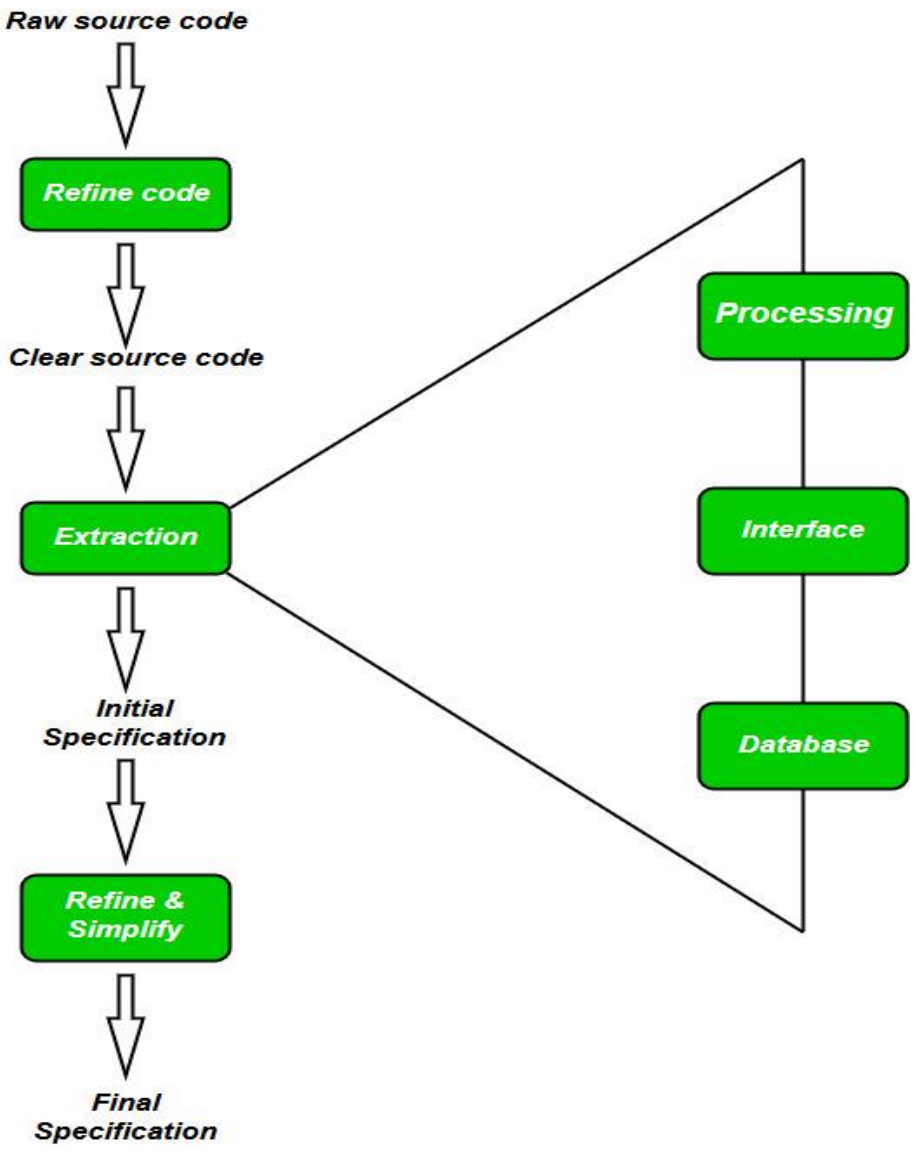
Software Reverse Engineering

Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.

The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

Reverse Engineering Goals:

- Cope with Complexity.
- Recover lost information.
- Detect side effects.
- Synthesise higher abstraction.
- Facilitate Reuse.



Steps of Software Reverse Engineering:

1. **Collection Information:**
   This step focuses on collecting all possible information (i.e., source design documents etc.) about the software.
2. **Examining the information:**
   The information collected in step-1 as studied so as to get familiar with the system.

3. **Extracting the structure:**
   This step concerns with identification of program structure in the form of structure chart where each node corresponds to some routine.

4. **Recording the functionality:**
   During this step processing details of each module of the structure, charts are recorded using structured language like decision table, etc.

5. **Recording data flow:**
   From the information extracted in step-3 and step-4, set of data flow diagrams are derived to show the flow of data among the processes.

6. **Recording control flow:**
   High level control structure of the software is recorded.

7. **Review extracted design:**
   Design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.

8. **Generate documentation:**
   Finally, in this step, the complete documentation including SRS, design document, history, overview, etc. are recorded for future use.

DIWAKAR EDUCATION HUB

Software Engineering Unit – 6 MCQs

As per updated syllabus

DIWAKAR EDUCATION HUB

2020

THE LEARN WITH EXPERTIES

1. Build & Fix Model is suitable for programming exercises of _____ LOC (Line of Code).
a) 100-200
b) 200-400
c) 400-1000
d) above 1000

Answer: a
Explanation: Build & Fix Model is suitable for small projects & programming exercises of 100 or 200 lines.

2. RAD stands for
a) Relative Application Development
b) Rapid Application Development
c) Rapid Application Document
d) None of the mentioned

Answer: b
Explanation: None.

3. Which one of the following models is not suitable for accommodating any change?
a) Build & Fix Model
b) Prototyping Model
c) RAD Model
d) Waterfall Model

Answer: d
Explanation: Real projects rarely follow the sequential flow that the Waterfall Model proposes.

4. Which is not one of the types of prototype of Prototyping Model?
a) Horizontal Prototype
b) Vertical Prototype
c) Diagonal Prototype
d) Domain Prototype

Answer: c
Explanation: Their is no such thing as Diagonal Prototype whereas other options have their respective definitions.

5. Which one of the following is not a phase of Prototyping Model?
a) Quick Design
b) Coding
c) Prototype Refinement
d) Engineer Product

Answer: b
Explanation: A prototyping model generates only a working model of a system.

6. Which of the following statements regarding Build & Fix Model is wrong?
a) No room for structured design
b) Code soon becomes unfixable & unchangeable
c) Maintenance is practically not

DIWAKAR EDUCATION HUB

possible
d) It scales up well to large projects

Answer: d
Explanation: Build & Fix Model is suitable for 100-200 LOC

7. RAD Model has
a) 2 phases
b) 3 phase
c) 5 phases
d) 6 phases

Answer: c
Explanation: RAD Model consists of five phases namely:Business modeling,Data modeling,Process modeling,Application generation and Testing & Turnover.

8. What is the major drawback of using RAD Model?
a) Highly specialized & skilled developers/designers are required
b) Increases reusability of components
c) Encourages customer/client feedback
d) Increases reusability of components, Highly specialized & skilled developers/designers are required

Answer: d
Explanation: The client may create an unrealistic product vision leading a team to over or under-develop functionality.Also, the specialized & skilled developers are not easily available.

9. SDLC stands for
a) Software Development Life Cycle
b) System Development Life cycle
c) Software Design Life Cycle
d) System Design Life Cycle

Answer: a
Explanation: None.

10. Which model can be selected if user is involved in all the phases of SDLC?
a) Waterfall Model
b) Prototyping Model
c) RAD Model
d) both Prototyping Model & RAD Model

Answer: c
Explanation: None.

11. Which one of the following is not an Evolutionary Process Model?
a) WINWIN Spiral Model
b) Incremental Model

DIWAKAR EDUCATION HUB

c) Concurrent Development Model
d) All of the mentioned

Answer: d
Explanation: None.

12. The Incremental Model is a result of combination of elements of which two models?
a) Build & FIX Model & Waterfall Model
b) Linear Model & RAD Model
c) Linear Model & Prototyping Model
d) Waterfall Model & RAD Model

Answer: c
Explanation: Each linear sequence produces a deliverable "increment" of the software and particularly when we have to quickly deliver a limited functionality system.

13. What is the major advantage of using Incremental Model?
a) Customer can respond to each increment
b) Easier to test and debug
c) It is used when there is a need to get a product to the market early
d) Easier to test and debug & It is used when there is a need to get a product to the market early

Answer: d
Explanation: Incremental Model is generally easier to test and debug than other methods of software development because relatively smaller changes are made during each iteration and is popular particularly when we have to quickly deliver a limited functionality system.However, option "a" can be seen in other models as well like RAD model,hence option "d" answers the question.

14. The spiral model was originally proposed by
a) IBM
b) Barry Boehm
c) Pressman
d) Royce

Answer: b
Explanation: None.

15. The spiral model has two dimensions namely _____ and _____
a) diagonal, angular
b) radial, perpendicular
c) radial, angular
d) diagonal, perpendicular

Answer: c
Explanation: The radial dimension of

DIWAKAR EDUCATION HUB

the model represents the cumulative costs and the angular dimension represents the progress made in completing each cycle. Each loop of the spiral from X-axis clockwise through 360° represents one phase.

16. How is WINWIN Spiral Model different from Spiral Model?
a) It defines tasks required to define resources, timelines, and other project related information
b) It defines a set of negotiation activities at the beginning of each pass around the spiral
c) It defines tasks required to assess both technical and management risks
d) It defines tasks required to construct, test, install, and provide user support

Answer: b
Explanation: Except option “b” all other tasks/activities are present in Spiral Model as well.

17. Identify the disadvantage of Spiral Model.
a) Doesn’t work well for smaller projects
b) High amount of risk analysis
c) Strong approval and documentation control
d) Additional Functionality can be added at a later date

Answer: a
Explanation: All other options are the advantages of Spiral Model.

18. Spiral Model has user involvement in all its phases.
a) True
b) False

Answer: b
Explanation: None.

19. How is Incremental Model different from Spiral Model?
a) Progress can be measured for Incremental Model
b) Changing requirements can be accommodated in Incremental Model
c) Users can see the system early in Incremental Model
d) All of the mentioned

Answer: a
Explanation: None.

20. If you were to create client/server applications, which model would you go for?
a) WINWIN Spiral Model
b) Spiral Model
c) Concurrent Model
d) Incremental Model

DIWAKAR EDUCATION HUB

Answer: c
Explanation: When applied to client/server applications, the concurrent process model defines activities in two dimensions: a system dimension and a component dimension.Thus Concurrency is achieved by system and component activities occurring simultaneously and can be modeled using the state-oriented approach.

21. Selection of a model is based on
a) Requirements
b) Development team & Users
c) Project type and associated risk
d) All of the mentioned

Answer: d
Explanation: Each model has to have some requirements, a team of developers, users and the risk involved in developing a project.

22. Which two models doesn't allow defining requirements early in the cycle?
a) Waterfall & RAD
b) Prototyping & Spiral
c) Prototyping & RAD
d) Waterfall & Spiral

Answer: b
Explanation: Prototyping Model starts with a requirements analysis phase including techniques like FAST, QFD, Brainstorming.In case of Spiral model the first phase involves activities related to customer communication like determining objectives.

23. Which of the following life cycle model can be chosen if the development team has less experience on similar projects?
a) Spiral
b) Waterfall
c) RAD
d) Iterative Enhancement Model

Answer: a
Explanation: Relying on risk assessment/analysis provides more flexibility than required for many applications which overcomes the criteria of less experienced developers.

24. If you were a lead developer of a software company and you are asked to submit a project/product within a stipulated time-frame with no cost barriers, which model would you select?
a) Waterfall
b) Spiral

DIWAKAR EDUCATION HUB

c) RAD
d) Incremental

Answer: c
Explanation: RAD model is inapplicable to develop cheaper products/software/projects as the cost of modeling, hiring highly skilled developers/designers and automated code generation is very high.But here the cost is not an issue, so one can select this model as it reduces development time.

25. Which two of the following models will not be able to give the desired outcome if user's participation is not involved?
a) Waterfall & Spiral
b) RAD & Spiral
c) RAD & Waterfall
d) RAD & Prototyping

Answer: d
Explanation: Active Participation of user is involved in all the four phases of RAD model and in case of the Prototyping model we need user's presence/involvement every time a new prototype is build or designed.

26. A company is developing an advance version of their current software available in the market, what model approach would they prefer ?
a) RAD
b) Iterative Enhancement
c) Both RAD & Iterative Enhancement
d) Spiral

Answer: c
Explanation: None.

27. One can choose Waterfall Model if the project development schedule is tight.
a) True
b) False

Answer: b
Explanation: Real projects rarely follow the sequential flow and iterations in this model are handled indirectly. This changes can cause confusion as the project proceeds thereby delaying the delivery date.

28. Choose the correct option from given below:
a) Prototyping Model facilitates reusability of components
b) RAD Model Model facilitates reusability of components
c) Both RAD & Prototyping Model facilitates reusability of components
d) None

DIWAKAR EDUCATION HUB

Answer: c
Explanation: None.

29. Spiral Model has high reliability requirements.
a) True
b) False

Answer: a
Explanation: None.

30. RAD Model has high reliability requirements.
a) True
b) False

Answer: b
Explanation: None.

31. Identify a fourth generation language(4GL) from the given below.
a) FORTRAN
b) COBOL
c) Unix shell
d) C++

Answer: c
Explanation: Rest all are third generation languages(3GL).

32. Arrange the following activities for making a software product using 4GT.
i. Design strategy
ii. Transformation into product
iii. Implementation
iv. Requirement gathering
a) 1, 4, 3, 2
b) 4, 3, 1, 2
c) 4, 1, 3, 2
d) 1, 3, 4, 2

Answer: c
Explanation: The sequence of activities mentioned in option c represents the Fourth Generation Techniques(4GT)Model.

33. 4GL is an example of _____ processing.
a) White Box
b) Black Box
c) Functional
d) Both Black Box & Functional

Answer: d
Explanation: Functional processing/testing is also referred to as black box testing in which contents of the black box are not known.Almost anything might be referred to as a black box:an algorithm or the human mind.Functionality of the black box is understood in terms of its inputs and outputs.

34. The 4GT Model is a package of _____

DIWAKAR EDUCATION HUB

a) CASE Tools
b) Software tools
c) Software Programs
d) None of the mentioned

Answer: b
Explanation: 4GT encompasses a broad array of software tools enabling the software engineer to specify the characteristics at a high level leading to an automatically generated source code based on these specifications.

35. Which of the following is not a type of a 4GL? One originating _____
a) on Lisp machine
b) on report generators
c) from database query languages
d) from GUI creators

Answer: a
Explanation: Fifth-generation programming language are built on LISP.

36. In 4GT, we can specify the user requirements in graphic notation or small abbreviated language form.
a) True
b) False

Answer: a
Explanation: None.

37. Productivity of software engineers is reduced in using a 4GT.
a) True
b) False

Answer: b
Explanation: 4GLs are more programmer-friendly and enhance programming efficiency with usage of English-like words and phrases, thereby increasing the productivity of professionals able to engage in software development.

38. Which of the following 4GLs invented at IBM and subsequently adopted by ANSI and ISO as the standard language for managing structured data?
a) SQL
b) PROLOG
c) C
d) JAVA

Answer: a
Explanation: C & JAVA are third generation languages(3GLs) whereas PROLOG is a 5GL.

39. What is a major advantage of using a 4GT Model for producing

DIWAKAR EDUCATION HUB

small scale products,
applications or programs ?
a) Improved productivity of software engineers
b) Reduction in software development time
c) 4GT helped by CASE tools and code generators offers a credible solution to many software problems
d) None of the mentioned

Answer: b
Explanation: Since automated coding is done using CASE tools & code generators proponents claim a dramatic reduction in software development time.

40. Which of the following model has a major disadvantage in terms of the coding phase of a software life cycle model ?
a) Spiral Model
b) Waterfall Model
c) Rad Model
d) 4GT Model

Answer: d
Explanation: Since coding phase is eliminated in 4GT Model,more expertise is required for analysis,design and testing activities.

41. Which one of the following is not a software process quality?
a) Productivity
b) Portability
c) Timeliness
d) Visibility

Answer: b
Explanation: Portability is a software product quality which means software can run on different hardware platforms or software environments.

42._____&_____ are two kinds of software products.
a) CAD, CAM
b) Firmware, Embedded
c) Generic, Customised
d) None of the mentioned

Answer: c
Explanation: rest all are sub categories/applications of option c.

43. Software costs more to maintain than it does to develop.
a) True
b) False

Answer: a
Explanation: For systems with a long life, maintenance costs may be several times development costs.

DIWAKAR EDUCATION HUB

44. Which one of the following is not an application of embedded software product?
a) keypad control of a security system
b) pattern recognition game playing
c) digital function of dashboard display in a car
d) none of the mentioned

Answer: b
Explanation: Pattern recognition uses Artificial Intelligence (AI) software.

45. Purpose of process is to deliver software
a) in time
b) with acceptable quality
c) that is cost efficient
d) both in time & with acceptable quality

Answer: d
Explanation: Cost of a software is a management issue & is not related to process activities.

46. The work associated with software engineering can be categorized into three generic phases,regardless of application area, project size, or complexity namely the_____ phase which focuses on *what*, the_____ phase which focuses on *how* and the_____ phase which focuses on *change*.
i. support
ii. development
iii. definition
a) 1, 2, 3
b) 2, 1, 3
c) 3, 2, 1
d) 3, 1, 2

Answer: c
Explanation: None.

47. Which of the following activities of a Generic Process framework provides a feedback report?
a) Communication
b) Planning
c) Modeling & Construction
d) Deployment

Answer: d
Explanation: In Deployment the product is delivered to the customer who evaluates the product and provides feedback based on the evaluation.

48. Process adopted for one project is same as the process adopted from another project.
a) True
b) False

DIWAKAR EDUCATION HUB

Answer: b
Explanation: the overall flow of activities, actions,tasks,the level of autonomy given to the software team and the inter dependencies among two process can never be the same.

49. Which one of the following is not an Umbrella Activity that complements the five process framework activities and help team manage and control progress, quality, change, and risk.
a) Reusability management
b) Risk management
c) Measurement
d) User Reviews

Answer: d
Explanation: None.

50. Four types of change are encountered during the support phase.Which one of the following is not one that falls into such category?
a) Translation
b) Correction
c) Adaptation
d) Prevention

Answer: a
Explanation: Translation is done in the development phase.

51. If a software production gets behind schedule, one can add more programmers and catch up.
a) True
b) False

Answer: b
Explanation: As new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort.

52. Choose an internal software quality from given below:
a) scalability
b) usability
c) reusability
d) reliability

Answer: c
Explanation: rest all are external qualities which are visible to the user.

53. RUP stands for_____ created by a division of _____
a) Rational Unified Program, IBM
b) Rational Unified Process, Infosys
c) Rational Unified Process, Microsoft
d) Rational Unified Process, IBM

DIVAKAR EDUCATION HUB

Answer: d
Explanation: None.

54. The RUP is normally described from three perspectives-dynamic, static & practice.What does static perspective do ?
a) It shows the process activities that are enacted
b) It suggests good practices to be used during the process
c) It shows the phases of the model over time
d) All of the mentioned

Answer: a
Explanation: None.

55. The only deliverable work product for a successful project is the working program.
a) True
b) False

Answer: b
Explanation: A working program is only one part of a software configuration that includes many elements. Documentation provides a foundation for successful engineering and, more important, guidance for software support.

56. Which phase of the RUP is used to establish a business case for the system ?
a) Transition
b) Elaboration
c) Construction
d) Inception

Answer: d
Explanation: None.

57. Which one of the following is not a fundamental activity for software processes in software engineering ?
a) Software Verification
b) Software Validation
c) Software design and implementation
d) Software evolution

Answer: a
Explanation: Software Verification is accounted for in implementation & testing activity.

58. A general statement of objectives is the major cause of failed software efforts.
a) True
b) False

Answer: a
Explanation: A formal and detailed description of the information

DIWAKAR EDUCATION HUB

domain, function, behavior, performance, interfaces, design constraints and validation criteria is essential which can be determined only after thorough communication between customer and developer.

59. The longer a fault exists in software
a) the more tedious its removal becomes
b) the more costly it is to detect and correct
c) the less likely it is to be properly corrected
d) All of the mentioned

Answer: d
Explanation: None.

60. Component-based Software Engineering allows faster delivery.
a) True
b) False

Answer: a
Explanation: Due to using previously tested components they produce more reliable system at a faster rate.

61. Arrange the following steps to form a basic/general Engineering Process Model.
i. Test
ii. Design
iii. Install
iv. Specification
v. Manufacture
vi. Maintain
a) 2, 4, 5, 1, 6, 3
b) 4, 2, 5, 1, 3, 6
c) 2, 4, 5, 1, 3, 6
d) 4, 2, 5, 1, 6, 3

Answer: b
Explanation: None.

62. Select the option that suits the Manifesto for Agile Software Development
a) Individuals and interactions
b) Working software
c) Customer collaboration
d) All of the mentioned

Answer:d
Explanation: None.

63. Agile Software Development is based on
a) Incremental Development
b) Iterative Development
c) Linear Development
d) Both Incremental and Iterative Development

Answer:d
Explanation: The software is

DIWAKAR EDUCATION HUB

developed in increments with the customer specifying the requirements to be included in each increment and the highest priority is to satisfy the customer through early and continuous delivery of valuable software. They are iterative because they work on one iteration followed by improvements in next iteration

64. Which on of the following is not an agile method?
a) XP
b) 4GT
c) AUP
d) All of the mentioned

Answer:b
Explanation: The 4GT approach does not incorporate iteration and the continuous feedback,which is the fundamental aspect of an agile method.

65. Agility is defined as the ability of a project team to respond rapidly to a change.
a) True
b) False

Answer:b
Explanation: The aim of agile methods is to reduce overheads in the software process and to be able to respond quickly to changing requirements without excessive rework.

66. How is plan driven development different from agile development ?
a) Outputs are decided through a process of negotiation during the software development process
b) Specification, design, implementation and testing are interleaved
c) Iteration occurs within activities
d) All of the mentioned

Answer:c
Explanation: A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.

67. How many phases are there in Scrum ?
a) Two
b) Three
c) Four
d) Scrum is an agile method which means it does not have phases

Answer:b
Explanation: There are three phases in Scrum.The initial phase is an

DIWAKAR EDUCATION HUB

outline planning phase followed by a series of sprint cycles and project closure phase.

68. Agile methods seem to work best when team members have a relatively high skill level.
a) True
b) False

Answer:a
Explanation: None.

69. Which of the following does not apply to agility to a software process?
a) Uses incremental product delivery strategy
b) Only essential work products are produced
c) Eliminate the use of project planning and testing
d) All of the mentioned

Answer:c
Explanation: Testing is a major part of each software development process which can't be avoided.

70. Which three framework activities are present in Adaptive Software Development(ASD) ?
a) analysis, design, coding
b) requirements gathering, adaptive cycle planning, iterative development
c) speculation, collaboration, learning
d) all of the mentioned

Answer:c
Explanation: None.

71. In agile development it is more important to build software that meets the customers' needs today than worry about features that might be needed in the future.
a) True
b) False

Answer:a
Explanation: None.

72. Incremental development in Extreme Programming (XP) is supported through a system release once every month.
a) True
b) False

Answer: b
Explanation: Incremental development is supported through small, frequent system releases.

73. In XP, as soon as the work on a task is complete, it is integrated into the whole system.
a) True
b) False

DIWAKAR EDUCATION HUB

Answer: a
Explanation: XP follows a continuous integration approach.After any such integration, all the unit tests in the system must pass.

74. In XP Increments are delivered to customers every _____ weeks.
a) One
b) Two
c) Three
d) Four

Answer: b
Explanation: Extreme Programming (XP) takes an 'extreme' approach to iterative development.New versions may be built several times per day, hence delivering the increment for approval every 2nd week after testing the new version.

75. User requirements are expressed as _____ in Extreme Programming.
a) implementation tasks
b) functionalities
c) scenarios
d) none of the mentioned

Answer: c
Explanation: User requirements are expressed as scenarios or user stories.These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.

76. Is a customer involved test development and validation in XP ?
a) Yes
b) No
c) It may vary from Customer to Customer
d) None of the mentioned

Answer: c
Explanation: The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

77. Programmers prefer programming to testing and sometimes they take shortcuts when writing tests. For example, they may write incomplete tests that do not

DIWAKAR EDUCATION HUB

check for all possible exceptions that may occur.
a) True
b) False

Answer: a
Explanation: In XP Some tests can be very difficult to write incrementally.For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.

78. Tests are automated in Extreme Programming.
a) True
b) False

Answer: a
Explanation: Automated test harnesses are used to run all component tests each time that a new release is built.

79. In XP an automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
a) True
b) False

Answer: a
Explanation: XP follows Test-first development approach.

80. Developers work individually on a release and they compare their results with other developers before forwarding that release to customers.
a) True
b) False

Answer: b
Explanation: XP follows the principle of pair programming which means developers work in pairs, checking each other's work and providing the support to always do a good job.

81. Which four framework activities are found in the Extreme Programming(XP) ?
a) analysis, design, coding, testing
b) planning, analysis, design, coding
c) planning, design, coding, testing
d) planning, analysis, coding, testing

Answer: c
Explanation: XP involves the mentioned four activities, and in the same in order.

82. What are the types of requirements ?
a) Availability

DIWAKAR EDUCATION HUB

b) Reliability
c) Usability
d) All of the mentioned

Answer: d
Explanation: All the mentioned traits are beneficial for an effective product to be developed.

83. Select the developer-specific requirement ?
a) Portability
b) Maintainability
c) Availability
d) Both Portability and Maintainability

Answer: d
Explanation: Availability is user specific requirement.

84. Which one of the following is not a step of requirement engineering?
a) elicitation
b) design
c) analysis
d) documentation

Answer: b
Explanation: Requirement Elicitation, Requirement Analysis, Requirement Documentation and Requirement Review are the four crucial process steps of requirement engineering.Design is in itself a different phase of Software Engineering.

85. FAST stands for
a) Functional Application Specification Technique
b) Fast Application Specification Technique
c) Facilitated Application Specification Technique
d) None of the mentioned

Answer: c
Explanation: None.

86. QFD stands for
a) quality function design
b) quality function development
c) quality function deployment
d) none of the mentioned

Answer: c
Explanation: None.

87. A Use-case actor is always a person having a role that different people may play.
a) True
b) False

Answer: b
Explanation: Use-case Actor is anything that needs to interact with

DIWAKAR EDUCATION HUB

the system, be it a person or another (external) system.

88. The user system requirements are the parts of which document ?
a) SDD
b) SRS
c) DDD
d) SRD

Answer: b
Explanation: Software requirements specification (SRS), is a complete description of the behaviour of a system to be developed and may include a set of use cases that describe interactions the users will have with the software.

89. A stakeholder is anyone who will purchase the completed software system under development.
a) True
b) False

Answer: b
Explanation: Stakeholders are anyone who has an interest in the project. Project stakeholders are individuals and organizations that are actively involved in the project, or whose interests may be affected as a result of project execution or project completion.

90. Conflicting requirements are common in Requirement Engineering, with each client proposing his or her version is the right one.
a) True
b) False

Answer: a
Explanation: This situation is seen in every field of work as each professional has his/her way of looking onto things & would argue to get his/her point approved.

91. Which is one of the most important stakeholder from the following ?
a) Entry level personnel
b) Middle level stakeholder
c) Managers
d) Users of the software

Answer: d
Explanation: Users are always the most important stakeholders.After all, without users or customers, what's the point of being in business?.

92. Which one of the following is a functional requirement ?
a) Maintainability
b) Portability

DIWAKAR EDUCATION HUB

c) Robustness
d) None of the mentioned

Answer: d
Explanation: All are non-functional requirements representing quality of the system. Functional requirements describe what the software has to do.

93. Which one of the following is a requirement that fits in a developer's module ?
a) Availability
b) Testability
c) Usability
d) Flexibility

Answer: b
Explanation: A developer needs to test his product before launching it into the market.

94. "Consider a system where, a heat sensor detects an intrusion and alerts the security company." What kind of a requirement the system is providing ?
a) Functional
b) Non-Functional
c) Known Requirement
d) None of the mentioned

Answer: a
Explanation: Functional requirements describe what the software has to do.

95. Which of the following statements explains portability in non-functional requirements?
a) It is a degree to which software running on one platform can easily be converted to run on another platform
b) It cannot be enhanced by using languages, OS' and tools that are universally available and standardized
c) The ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which the system was intended
d) None of the mentioned

Answer: a
Explanation: Option c is termed as reliability and option e refers to efficiency.

96. Functional requirements capture the intended behavior of the system.
a) True
b) False

Answer: a
Explanation: The behavior of functional requirements may be expressed as services, tasks or

DIWAKAR EDUCATION HUB

functions the system is required to perform.

97. Choose the incorrect statement with respect to Non-Functional Requirement(NFR).
a) Product-oriented Approach – Focus on system (or software) quality
b) Process-oriented Approach – Focus on how NFRs can be used in the design process
c) Quantitative Approach – Find measurable scales for the functionality attributes
d) Qualitative Approach – Study various relationships between quality goals

Answer: c
Explanation: Quantitative Approaches in NFRs are used to find measurable scales for the quality attributes like efficiency, flexibility, integrity, usability etc.

98. How many classification schemes have been developed for NFRs ?
a) Two
b) Three
c) Four
d) Five

Answer: d
Explanation: Software Quality Tree [Boehm 1976], Roman [IEEE Computer 1985], Process-Product-External considerations [Sommerville 1992], Mc Call's NFR list and Dimensions of Quality–Components of FURPS+ are the five classification schemes for NFRs.

99. According to components of FURPS+, which of the following does not belong to S ?
a) Testability
b) Speed Efficiency
c) Serviceability
d) Installability

Answer: b
Explanation: Speed Efficiency belong to Performance (P) in FURPS+ .

100. Does software wear & tear by decomposition ?
a) Yes
b) No

Answer: b
Explanation: Unlike hardware, software is reliable.

101. What are the four dimensions of Dependability ?
a) Usability, Reliability, Security, Flexibility
b) Availability, Reliability,

DIVAKAR EDUCATION HUB

Maintainability, Security
c) Availability, Reliability, Security, Safety
d) Security, Safety, Testability, Usability

Answer: c
Explanation: All the traits of option c sync with dependability.

102. Choose the correct statement on how NFRs integrates with Rational Unified Process ?
a) System responds within 4 seconds on average to local user requests and changes in the environment
b) System responds within 4 seconds on average to remote user requests and changes in the environment
c) All of the mentioned
d) None of the mentioned

Answer: b
Explanation: System response to a local user is 2 seconds on average

103. What is the first step of requirement elicitation ?
a) Identifying Stakeholder
b) Listing out Requirements
c) Requirements Gathering
d) All of the mentioned

Answer: a
Explanation: Stakeholders are the one who will invest in and use the product, so its essential to chalk out stakeholders first.

104. Starting from least to most important, choose the order of stakeholder.
i. Managers
ii. Entry level Personnel
iii. Users
iv. Middle level stakeholder
a) i, ii, iv, iii
b) i, ii, iii, iv
c) ii, iv, i, iii
d) All of the mentioned

Answer: c
Explanation: Users are your customers, they will be using your product, thus making them most important of all.

105. Arrange the tasks involved in requirements elicitation in an appropriate manner.
i. Consolidation
ii. Prioritization
iii. Requirements Gathering
iv. Evaluation
a) iii, i, ii, iv
b) iii, iv, ii, i

DIWAKAR EDUCATION HUB

c) iii, ii, iv, i
d) ii, iii, iv, i

Answer: b
Explanation: Requirements gathering captures viewpoint from different users followed by evaluation of those view points.Now comes the task of checking the relative importance of the requirements and finally to consolidate or bind together the information collected.

106. What are the types of requirement in Quality Function Deployment(QFD) ?
a) Known, Unknown, Undreamed
b) User, Developer
c) Functional, Non-Functional
d) Normal, Expected, Exciting

Answer: d
Explanation: According to QFD, Normal, Expected and Exciting requirements maximizes customer satisfaction from the Software Engineering Process.

107. What kind of approach was introduced for elicitation and modelling to give a functional view of the system ?
a) Object Oriented Design (by Booch)
b) Use Cases (by Jacobson)
c) Fusion (by Coleman)
d) Object Modeling Technique (by Rumbaugh)

Answer: b
Explanation: Use Case captures who does what with the system, for what purpose, without dealing with system internals.

108. What are the kinds of actors used in OOSE ?
a) Primary
b) Secondary
c) Ternary
d) Both Primary and Secondary

Answer: d
Explanation: A primary actor is one having a goal requiring the assistance of the system whereas, a secondary actor is one from which system needs assistance.There is no such thing as ternary actor in Software Engineering.

109. Why is Requirements Elicitation a difficult task ?
a) Problem of scope
b) Problem of understanding
c) Problem of volatility
d) All of the mentioned

DIWAKAR EDUCATION HUB

Answer: d
Explanation: Users specify unnecessary technical detail that may confuse, rather than clarify overall system objectives.Also, the customers/users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment and they do not understand that the requirements change over time.

110. What requirement gathering method developed at IBM in 1970s is used for managing requirement elicitation ?
a) JAD
b) Traceability
c) FAST
d) Both JAD and Traceability

Answer: d
Explanation: Joint application design (JAD) is a process used to collect business requirements while developing new information systems for a company. Requirements traceability is concerned with documenting the life of a requirement and providing bi-directional traceability between various associated requirements.

111. Requirements elicitation is a cyclic process
a) True
b) False

Answer: a
Explanation: Requirements traceability provides bi-directional traceability between various associated requirements.

112. How many Scenarios are there in elicitation activities ?
a) One
b) Two
c) Three
d) Four

Answer: d
Explanation: As-is Scenario, Visionary Scenario, Evaluation Scenario and Training Scenario are the four scenarios in requirement elicitation activities.

113. Which of the following elicitation techniques is a viewpoint based method?
a) FODA
b) QFD
c) CORE
d) IBIS

DIWAKAR EDUCATION HUB

Answer: c
Explanation: Controlled Requirements Expression(CORE) says that any system can be viewed from a number of view points and that a complete picture of system requirements can only emerge by putting together the various viewpoints.

114. _____ and _____ are the two view points discussed in Controlled Requirements Expression (CORE).
a) Functional, Non-Functional
b) User, Developer
c) Known, Unknown
d) All of the mentioned

Answer: a
Explanation: The CORE sessions includes the discussion of functional and non-functional requirements.

115. What is the major drawback of CORE ?
a) Requirements are comprehensive
b) NFRs are not given enough importance
c) Role of analyst is passive
d) All of the mentioned

Answer: c
Explanation: In CORE the requirement specification are put together by all users, customers and analysts, so a passive analyst will not get the requirements properly.

116. Choose a framework that corresponds to Issue Based Information System (IBIS).
a) Idea -> Question -> Argument
b) Question -> Idea -> Argument
c) Issue -> Position -> Justification
d) Both Question -> Idea -> Argument and Issue -> Position -> Justification

Answer: d
Explanation: IBIS is a simple and non-intrusive method that provides a framework for resolving issues and gathering requirements.

117. How is CORE different from IBIS ?
a) Iterative in nature
b) Redundancies are removed
c) It is simple and an easier method to use
d) Consistency problems are addressed in CORE

Answer: d
Explanation: Preliminary data collection is done in CORE to get some broad level data on each view point to structure the view point and

DIWAKAR EDUCATION HUB

to check consistency from within and outside the viewpoints.

118. Which of the following Requirement Elicitation Techniques removes the poor understanding of application domain and lack of common terminology between the users and the analysts ?
a) FODA
b) CORE
c) IBIS
d) Prototyping

Answer: a
Explanation: Feature Oriented Domain Analysis (FODA) is defined as the process of identifying, collecting, organizing and representing relevant information in a domain .

119. How many steps are involved in Feature Oriented Domain Analysis (FODA) ?
a) Two
b) Three
c) Four
d) Five

Answer: b
Explanation: Context Analysis, Domain Modeling and Architecture Modeling are the three steps involved in Feature Oriented Domain Analysis (FODA).

120. IBIS is a more structured approach than CORE.
a) True
b) False

Answer: a
Explanation: IBIS is a more structured approach as it captures information which is consistent and important.On the other hand CORE gives importance to every view point even if it is obsolete.

121. Which one of the following is not an actor in JAD sessions ?
a) User
b) Tester
c) Scribe
d) Sponsor

Answer: b
Explanation: A Tester's role is seen in after coding phase rather than in elicitation phase.

122. What of the following is not an output of a JAD session ?
a) Context Diagrams
b) DFDs
c) ER model
d) UML diagrams

DIWAKAR EDUCATION HUB

Answer: d
Explanation: Unified Modeling Language (UML) diagrams are constructed during the design phase of the SDLC.

123. How is brainstorming different from JAD ? Brainstorming sessions
a) last for about 2-3 hours
b) last for about 2-3 days
c) cover the technology used for the development
d) all of the mentioned

Answer: a
Explanation: Brainstorming is a group or individual creativity technique by which efforts are made to find a conclusion for a specific problem by gathering a list of ideas spontaneously contributed by its member(s).The idea is to quickly reach to an approved solution ASAP.

124. How is throwaway prototype different from evolutionary prototype ?
a) It involves successive steps
b) It involves just one task
c) The prototype is built with the idea that it will eventually be converted into final system
d) It has a shorter development time

Answer: b
Explanation: Except option b all other options represent the characteristics of an evolutionary prototype.

125. Keeping the requirements of QFD in mind which of the following is not an example of an Expected Requirement ?
a) Ease of software installation
b) Overall operational correctness and reliability
c) Specific system functions
d) Quality graphical display

Answer: c
Explanation: Expected requirements are so fundamental that a customer does not explicitly state them.System functions comes under the category of Normal requirements in QFD which is compulsory to be defined,hence is not an expected requirement.

126. QFD works best if it has management commitment.
a) True
b) False

Answer: a
Explanation: QFD involves heavy investment in initial stages, thus bounding the management to

DIWAKAR EDUCATION HUB

provide appropriate funding for the development process .

127. Which of the following Requirement Elicitation Techniques is applicable to messy, changing and ill-defined problem situations ?
a) Quality Function Deployment (QFD)
b) Prototyping
c) Soft Systems Methodology (SSM)
d) Controlled Requirements Expression (CORE)

Answer: c
Explanation: Soft systems methodology (SSM) is a systemic approach for tackling real-world problematic situations.It is a common misunderstanding that SSM is a methodology for dealing solely with 'soft problems' (problems which involve psychological, social, and cultural elements). SSM does not differentiate between 'soft' and 'hard' problems, it merely provides a different way of dealing with situations perceived as problematic.

128. To ensure that a given root definition is rigorous and comprehensive, The Lancaster team proposed several criteria that are summarized in the mnemonic CATWOE in Soft Systems Methodology (SSM).Which of the following alphabet is representing an entirely different meaning to SSM ?
a) C – Customer
b) A – Actor
c) T – Transformation
d) E – ER Model

Answer: d
Explanation: 'E' in CATWOE stands for Environmental constraints.

129. Choose the disadvantage of using SSM as an elicitation technique.
a) It incorporates human element into design
b) SSM is in its infant stage
c) SSM is suitable for new systems
d) Standard methodologies like Role Exploration, Issue Resolution and Reorganization support SSM

Answer: b
Explanation: SSM is still in its infancy.It is evolving and its industrial usage is low.

130. How many phases are there in Brainstorming ?
a) Two
b) Three

DIWAKAR EDUCATION HUB

c) Four
d) All of the mentioned

Answer: b
Explanation: Preparation, Execution and Follow up are the three phases to be achieved for a successful brainstorming session.

131. Who controls the FAST (Facilitated Application Specification Techniques) meeting ?
a) System Analyst
b) Scribe
c) Facilitator
d) Manager

Answer: c
Explanation: A Facilitator (a customer/developer/an outsider) controls the FAST meeting.His role is to ensure that the meeting is productive.

132. Arrange the steps in order to represent the conducting of Wideband Delphi Technique.
i. Conduct a group discussion
ii. Conduct another group discussion
iii. Present experts with a problem
iv. Collect expert opinion anonymously
v. Iterate until consensus is reached
vi. Feedback a summary of result to each expert
a) i, iii, ii, iv, v, vi
b) iii, i, ii, iv, v, vi
c) i, ii, iii, iv, vi, v
d) iii, i, iv, vi, ii, v

Answer: d
Explanation: The sequence represents the working steps of a Wideband Delphi technique .

133. Which of the following is not a diagram studied in Requirement Analysis ?
a) Use Cases
b) Entity Relationship Diagram
c) State Transition Diagram
d) Activity Diagram

Answer: d
Explanation: Activity Diagram comes under the design phase of SDLC.

134. How many feasibility studies is conducted in Requirement Analysis ?
a) Two
b) Three
c) Four
d) None of the mentioned

Answer: b
Explanation: Economic feasibility (cost/benefit analysis), Technical feasibility

DIWAKAR EDUCATION HUB

(hardware/software/people, etc.) and Legal feasibility studies are done in Requirement Analysis.

135. How many phases are there in Requirement Analysis ?
a) Three
b) Four
c) Five
d) Six

Answer: c
Explanation: Problem Recognition, Evaluation and Synthesis (focus is on what not how), Modeling, Specification and Review are the five phases.

136. Traceability is not considered in Requirement Analysis.
a) True
b) False

Answer: b
Explanation: Requirements traceability is concerned with documenting the life of a requirement and providing bi-directional traceability between various associated requirements, hence requirements must be traceable.

137. Requirements analysis is critical to the success of a development project.
a) True
b) False
c) Depends upon the size of project
d) None of the mentioned

Answer: a
Explanation: Requirements must be actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

138. _____ and _____ are the two issues of Requirement Analysis.
a) Performance, Design
b) Stakeholder, Developer
c) Functional, Non-Functional
d) None of the mentioned

Answer: b
Explanation: Option a and c are the types of requirements and not the issues of requirement analysis..

139. The requirements that result from requirements analysis are typically expressed from one of three perspectives or views.WhaT is that perspective or view ?

DIWAKAR EDUCATION HUB

a) Developer
b) User
c) Non-Functional
d) Physical

Answer: d
Explanation: The perspectives or views have been described as the Operational, Functional, and Physical views.All three are necessary and must be coordinated to fully understand the customers' needs and objectives.

140. Requirements Analysis is an Iterative Process.
a) True
b) False

Answer: a
Explanation: Requirements analysis is conducted iteratively with functional analysis to optimize performance requirements for identified functions, and to verify that synthesized solutions can satisfy customer requirements.

141. Coad and Yourdon suggested _____ selection characteristics that should be used as an analyst considers each potential object for inclusion in the requirement analysis model.
a) Three
b) Four
c) Five
d) Six

Answer: d
Explanation: Retained information, Needed services, Multiple attributes, Common attributes, Common operations and Essential requirements are the six criterion mentioned by Coad and Yourdon.

142. Requirements should specify 'what' but not 'how'.
a) True
b) False

Answer: a
Explanation: 'What' refers to a system's purpose, while 'How' refers to a system's structure and behavior.

143. Which of the following property does not correspond to a good Software Requirements Specification (SRS) ?
a) Verifiable
b) Ambiguous
c) Complete
d) Traceable

Answer: b
Explanation: The SRS should be

DIWAKAR EDUCATION HUB

unambiguous in nature which means each sentence in SRS should have a unique interpretation.

144. Which of the following property of SRS is depicted by the statement : "Conformity to a standard is maintained" ?
a) Correct
b) Complete
c) Consistent
d) Modifiable

Answer: b
Explanation: The SRS is complete full labeling and referencing of all figures, tables etc. and definition of all terms and units of measure is defined.

145. The SRS is said to be *consistent* if and only if
a) its structure and style are such that any changes to the requirements can be made easily while retaining the style and structure
b) every requirement stated therein is one that the software shall meet
c) every requirement stated therein is verifiable
d) no subset of individual requirements described in it conflict with each other

Answer: d
Explanation: Real world object may conflict with each other for example one requirement says that all lights should be red while the other states that all lights should green.

146. Which of the following statements about SRS is/are true ?
i. SRS is written by customer
ii. SRS is written by a developer
iii. SRS serves as a contract between customer and developer
a) Only i is true
b) Both ii and iii are true
c) All are true
d) None of the mentioned

Answer: c
Explanation: The SRS acts as a communication media between the Customer, Analyst, system developers, maintainers etc. Thus it is a contract between Purchaser and Supplier. It is essentially written by a developer on the basis of customer' need but in some cases it may be written by a customer as well.

147. The SRS document is also known as _____ specification.
a) black-box
b) white-box

DIWAKAR EDUCATION HUB

c) grey-box
d) none of the mentioned

Answer: a
Explanation: The system is considered as a black box whose internal details are not known that is, only its visible external (input/output) behavior is documented.

148. Which of the following is included in SRS ?
a) Cost
b) Design Constraints
c) Staffing
d) Delivery Schedule

Answer: b
Explanation: Design constraints include standards to be incorporated in the software, implementation language, resource limits, operating environment etc.

149. Which of the following is not included in SRS ?
a) Performance
b) Functionality
c) Design solutions
d) External Interfaces

Answer: c
Explanation: The SRS document concentrates on:"what needs to be done" and carefully avoids the solution ("how to do") aspects.

150. Arrange the given sequence to form a SRS Prototype outline as per IEEE SRS Standard.
i. General description
ii. Introduction
iii. Index
iv. Appendices
v. Specific Requirements
a) iii, i, ii,v, iv
b) iii, ii, i, v, iv
c) ii, i, v, iv, iii
d) iii, i, ii

Answer: c
Explanation: The given sequence correctly resemble a standard SRS prototype as per IEEE.

151. Consider the following Statement: "The output of a program shall be given within 10 secs of event X 10% of the time."What characteristic of SRS is being depicted here ?
a) Consistent
b) Verifiable
c) Non-verifiable
d) Correct

Answer: b
Explanation: An SRS is verifiable, if

DIWAKAR EDUCATION HUB

and only if, every requirement stated therein is verifiable.Here the given condition can be verified during testing phase.

152. Consider the following Statement: “The data set will contain an end of file character.”What characteristic of SRS is being depicted here ?
a) Consistent
b) Non-verifiable
c) Correct
d) Ambiguous

Answer: b
Explanation: An SRS is unambiguous if and only if, every requirement stated therein has only one unique interpretation. The given statement does not answer the question: “which data set will have an end of file character ?”.

153. Consider the following Statement: “The product should have a good human interface.”What characteristic of SRS is being depicted here ?
a) Consistent
b) Non-Verifiable
c) Correct
d) Ambiguous

Answer: b
Explanation: An SRS is verifiable, if and only if, every requirement stated therein is verifiable. The statement can only be answered on completion of the software and customer evaluation but still human interface will vary from person to person.

154. Narrative essay is one of the best types of specification document ?
a) True
b) False

Answer:b
Explanation: Narrative essay is one of the worst types of specification document as it is difficult to change, difficult to be precise, has scope for contradictions, etc.

155. Which two requirements are given priority during Requirement Management of a product ?
a) User and Developer
b) Functional and Non-functional
c) Enduring and Volatile
d) All of the mentioned

Answer: c
Explanation: Enduring requirements are core requirements & are related to main activity of the organization

DIWAKAR EDUCATION HUB

while volatile requirements are likely to change during software development life cycle or after delivery of the product.

156. Considering the example of issue/return of a book, cataloging etc. in a library management.What type of management requirement is being depicted here?
a) Enduring
b) Volatile
c) Both Enduring & Volatile
d) All of the mentioned

Answer: a
Explanation: For library management system issue/return of a book, cataloging etc. are core activities and are stable for any system.

157. Why is Requirements Management Important ? It is due to the changes
a) to the environment
b) in technology
c) in customer's expectations
d) in all of the mentioned.

Answer: d
Explanation: Systems continue to be built as the advancement of new products being launched in the market and so does the market changes, the technology and in turn customer's expectation.

158. Requirements Management is a prerequisite for Quality-Oriented Development.
a) True
b) False

Answer: a
Explanation: Quality makes no sense without reference to requirements, which means quality-oriented development is requirements-driven development, thus requirements management is a prerequisite for quality-oriented development.

159. Requirements traceability is one of the most important part requirement management. It may also be referred to as the heart of requirement management.
a) True
b) False

Answer: a
Explanation: Requirements traceability refers to the ability to describe and follow the life of a requirement in both forwards and backwards direction. Requirements can be traced from its origins, through its development and

DIVAKAR EDUCATION HUB

specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases.

160. Requirements Management has a high initial start-up cost but does not need ongoing funding throughout a project.
a) True
b) False

Answer: b
Explanation: Requirements Management needs continued funding throughout a project.Project funding is often limited at the onset of a project, restricted to those aspects of the project which are tangible and visible, and subsequently allocated in a phase-by-phase manner.

161. Which of the following is not a Requirement Management workbench tool ?
a) RTM
b) DOORS
c) Rational Suite
d) RDD 100

Answer: c
Explanation: Rational Suite is an environment tool for requirement management.

162. Which of the following is a requirement management activity ?
a) Investigation
b) Design
c) Construction and Test
d) All of the mentioned

Answer: d
Explanation: All the options are the activities of requirement management.

163. What functionality of Requirement Management Tool (RMT) is depicted by the statement: “the tool should be able to automatically detect relations between artifacts. For example information retrieval techniques, monitoring of change history, naming schemas or model transformations.”
a) Automatic Link Detection
b) Documentation Support
c) Graphical Representation
d) Automatic Link Creation and Change

Answer: a
Explanation: DOORS is one such tool that supports Automatic Link Detection.

DIWAKAR EDUCATION HUB

164. According to a statistical report: "over 30% of all software projects are cancelled before completion and over 70% of the remainder fail to deliver expected features". What must be the reason for such a situation ?
a) Poor change management
b) Poor requirements management
c) Poor quality control
d) All of the mentioned

Answer: b
Explanation: Fundamental to the problem mentioned in the statistical report is poor requirements management. Option a and c are its sub parts.

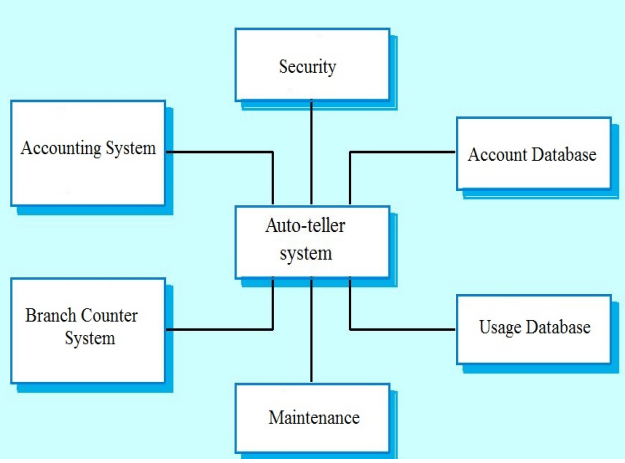
165. Which model in system modelling depicts the static nature of the system ?
a) Behavioral Model
b) Context Model
c) Data Model
d) Structural Model

Answer: d
Explanation: Structural models show the organization and architecture of a system. These are used to define the static structure of classes in a system and their associations.

166. Which perspective in system modelling shows the system or data architecture.
a) Structural perspective
b) Behavioral perspective
c) External perspective
d) All of the mentioned

Answer: a
Explanation: Structural perspective is used to define the static structure of classes in a system and their associations.

167. Which system model is being depicted by the ATM operations shown below:



a) Structural model
b) Context model
c) Behavioral model
d) Interaction model

DIWAKAR EDUCATION HUB

Answer: b
Explanation: Context models are used to illustrate the operational context of a system.They show what lies outside the system boundaries.

168. Activity diagrams are used to model the processing of data.
a) True
b) False

Answer: a
Explanation: The statement mentioned is true and each activity represents one process step.

169. Model-driven engineering is just a theoretical concept. It cannot be converted into a working/executable code.
a) True
b) False

Answer: b
Explanation: Model-driven engineering is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.

170. Which of the following diagram is not supported by UML considering Data-driven modeling ?
a) Activity
b) Data Flow Diagram (DFD)
c) State Chart
d) Component

Answer: b
Explanation: DFDs focus on system functions and do not recognize system objects.

171. The two dimensions of spiral model are
a) diagonal, angular
b) radial, perpendicular
c) radial, angular
d) diagonal, perpendicular

Answer: c
Explanation: The radial dimension depicts the cumulative costs and the angular dimension depicts the progress made in completing each cycle. Each loop of the spiral model represents a phase.

172. The Incremental Model is combination of elements of
a) Build & FIX Model & Waterfall Model
b) Linear Model & RAD Model
c) Linear Model & Prototyping Model
d) Waterfall Model & RAD Model

DIWAKAR EDUCATION HUB

Answer: c
Explanation: Each linear sequence produces a deliverable "increment" of the software system, particularly needed in case of quick delivery of a limited functionality system..

173. Model preferred to create client/server applications is
a) WINWIN Spiral Model
b) Spiral Model
c) Concurrent Model
d) Incremental Model

Answer: c
Explanation: In case of client/server applications, the concurrent process model specifies activities in two dimensions: a system dimension and a component dimension. Hence Concurrency is achieved by these two activities occurring simultaneously and can be modeled using the state-oriented approach.

174. Identify the correct statement with respect to Evolutionary development:
a) Evolutionary development usually has two flavors; exploratory development, and throw-away prototyping
b) Very large projects are usually done using evolutionary development based approach
c) It facilitates easy project management, through the high volume of documentation it generates
d) Sometimes the construction of a throw-away prototype is not followed by a re- implementation of the software system using a more structured approach

Answer: a
Explanation: Evolutionary development usually has two flavors; exploratory development, and throw-away prototyping.

175. Spiral model was developed by
a) Victor Bisili
b) Berry Boehm
c) Bev Littlewood
d) Roger Pressman

Answer: b
Explanation: Berry Boehm in 1986 in his Article "A spiral model of software development and enhancement".

176. Software evolution does not comprises:
a) Development activities
b) Negotiating with client

DIVAKAR EDUCATION HUB

c) Maintenance activities
d) Re-engineering activities

Answer: b
Explanation: Software evolution refers to the study and management of the process of making changes to software over time. Thus it comprises rest three options.

177. Processes for evolving a software product depend on:
a) Type of software to be maintained
b) Development processes used
c) Skills and experience of the people involved
d) All of the mentioned

Answer: d
Explanation: Processes used for software evolution depend on all these factors.

178. Which technique is applied to ensure the continued evolution of legacy systems ?
a) Forward engineering
b) Reverse Engineering
c) Reengineering
d) Reverse Engineering and Reengineering

Answer: d
Explanation: Processes used for software evolution depend rely on these two techniques.

179. Program modularization and Source code translation are the activities of_____
a) Forward engineering
b) Reverse Engineering
c) Reengineering
d) Reverse Engineering and Reengineering

Answer: c
Explanation: Reengineering is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.

180. Reverse engineering is the last activity in a reengineering project.
a) True
b) False

Answer: b
Explanation: Reverse engineering is often the initial activity in a reengineering project.

181. The cost of re-engineering is often significantly less than the costs of developing new software.
a) True
b) False

DIWAKAR EDUCATION HUB

Answer: a
Explanation: There is a high risk in new software development. There may be development problems, staffing problems and specification problems, thereby increasing the cost.

182. A characteristic of a software system that can lead to a system error is known as?
a) Human error or mistake
b) System fault
c) System error
d) System failure

Answer: b
Explanation: None.

183. An erroneous system state that can lead to system behavior that is unexpected by system users is known as?
a) Human error or mistake
b) System fault
c) System error
d) System failure

Answer: c
Explanation: None.

184. An event that occurs at some point in time when the system does not deliver a service as expected by its users is called _____
a) Human error or mistake
b) System fault
c) System error
d) System failure

Answer: d
Explanation: None.

185. A chemical plant system may detect excessive pressure and open a relief valve to reduce these pressures before an explosion occurs. What kind of dependability and security issue the example states?
a) Hazard avoidance
b) Damage limitation
c) Hazard detection
d) Hazard detection and removal

Answer: d
Explanation: The system is designed so that hazards are detected and removed before they result in an accident.

186. An aircraft engine normally includes automatic fire extinguishers.What kind of dependability and security issue the example states?
a) Hazard avoidance
b) Damage limitation

DIWAKAR EDUCATION HUB

c) Hazard detection
d) Hazard detection and removal

Answer: b
Explanation: The system may include protection features that minimize the damage that may result from an accident.

187. Which reliability metric sets out the probable number of system failures that are likely to be observed relative to a certain time period?
a) POFOD
b) ROCOF
c) AVAIL
d) None of the mentioned

Answer: b
Explanation: Rate of occurrence of failures (ROCOF) sets out the probable number of system failures that are likely to be observed relative to the number of system executions.

188. Which of the following is not a functional reliability requirement for a system?
a) Checking requirements
b) Recovery requirements
c) Redundancy requirements
d) Ambiguous requirements

Answer: d
Explanation: All the options are correct except option d.

189. To specify security requirements, one should identify the risks that are to be dealt with.
a) True
b) False

Answer: b
Explanation: To specify security requirements, one should identify the assets that are to be dealt with.

190. The aim of preliminary risk analysis and assessment process is to derive security requirements for the system as a whole.
a) True
b) False

Answer: a
Explanation: In preliminary risk analysis stage, decisions on the detailed system requirements, the system design, or the implementation technology have not been made.

191. At which stage of risk analysis specification, the additional security requirements take account of the technologies used in building the system and system design and

DIWAKAR EDUCATION HUB

implementation decisions?
a) Preliminary risk analysis
b) Life-cycle risk analysis
c) Operational risk analysis
d) All of the mentioned

Answer: b
Explanation: This risk assessment takes place during the system development life cycle after design choices have been made..

192. Which reliability requirements are concerned with maintaining copies of the system?
a) Checking requirements
b) Recovery requirements
c) Redundancy requirements
d) Ambiguous requirements

Answer: b
Explanation: These requirements are geared to helping the system recover after a failure has occurred.

193. What are security controls ?
a) Controls that are intended to ensure that attacks are unsuccessful
b) Controls that are intended to detect and repel attacks
c) Controls that are intended to support recovery from problems
d) All of the mentioned

Answer:d
Explanation: All the options define a security control property.

194. Controls that are intended to repel attacks is analogous to _____ in dependability engineering.
a) Fault avoidance
b) Fault tolerance
c) Fault detection
d) None of the mentioned

Answer:b
Explanation: Here the system is designed so that faults in the delivered software do not result in system failure.

195. Controls that are intended to ensure that attacks are unsuccessful is analogous to _____ in dependability engineering.
a) Fault avoidance
b) Fault tolerance
c) Fault detection
d) Fault Recovery

Answer:a
Explanation: In Fault avoidance the system is developed in such a way that human error is avoided and thus system faults are minimised.

DIWAKAR EDUCATION HUB

196. What is Life cycle risk assessment ?
a) Risk assessment before the system has been deployed
b) Risk assessment while the system is being developed
c) All of the mentioned
d) None of the mentioned

Answer:c
Explanation: None.

197. Which of the following is incorrect with respect to Model Checking?
a) Model checking is particularly valuable for verifying concurrent systems
b) Model checking is computationally very inexpensive
c) The model checker explores all possible paths through the model
d) All of the mentioned

Answer: b
Explanation: Model checking is very expensive.It is only practical to use it in the verification of small to medium sized critical systems.

198. Which is the first step in the software development life cycle ?
a) Analysis
b) Design
c) Problem/Opportunity Identification
d) Development and Documentation

Answer: c
Explanation: None.

199. Which tool is use for structured designing ?
a) Program flowchart
b) Structure chart
c) Data-flow diagram
d) Module

Answer: b
Explanation: A Structure Chart (SC) in software engineering and organizational theory, is a chart which shows the breakdown of a system to its lowest manageable levels.

200. A step by step instruction used to solve a problem is known as
a) Sequential structure
b) A List
c) A plan
d) An Algorithm

Answer: d
Explanation: None.

201. In the Analysis phase, the development of the _____ occurs, which is a clear statement of the goals and objectives of the

DIWAKAR EDUCATION HUB

project.
a) documentation
b) flowchart
c) program specification
d) design

Answer: c
Explanation: Program specification is the definition of what a computer program is expected to do.

202. Actual programming of software code is done during the _____ step in the SDLC.
a) Maintenance and Evaluation
b) Design
c) Analysis
d) Development and Documentation

Answer: d
Explanation: The developer has to find in the technical documentation enough information to start coding.

203. Who designs and implement database structures.
a) Programmers
b) Project managers
c) Technical writers
d) Database administrators

Answer: d
Explanation: The role of database administrators includes the development and design of database strategies, system monitoring and improving database performance and capacity, and planning for future expansion requirements.

204. _____ is the process of translating a task into a series of commands that a computer will use to perform that task.
a) Project design
b) Installation
c) Systems analysis
d) Programming

Answer: d
Explanation: None.

205. Debugging is:
a) creating program code
b) finding and correcting errors in the program code
c) identifying the task to be computerized
d) creating the algorithm

Answer: b
Explanation: Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware, thus making it behave as expected.

DIWAKAR EDUCATION HUB

206. In Design phase, which is the primary area of concern ?
a) Architecture
b) Data
c) Interface
d) All of the mentioned

Answer: d
Explanation: Part of the design phase is to create structural and behavioral models of the system which is covered by architecture, data and the interface of the product.

207. The importance of software design can be summarized in a single word which is:
a) Efficiency
b) Accuracy
c) Quality
d) Complexity

Answer: c
Explanation: Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications.

208. Cohesion is a qualitative indication of the degree to which a module
a) can be written more compactly
b) focuses on just one thing
c) is able to complete its function in a timely manner
d) is connected to other modules and the outside world

Answer: b
Explanation: Cohesion of a single module/component is the degree to which its responsibilities form a meaningful unit.

209. Coupling is a qualitative indication of the degree to which a module
a) can be written more compactly
b) focuses on just one thing
c) is able to complete its function in a timely manner
d) is connected to other modules and the outside world

Answer: d
Explanation: Coupling between modules/components is their degree of mutual interdependence.

210. Java packages and Fortran subroutine are examples of_____
a) Functions
b) Modules
c) Classes
d) Sub procedures

DIWAKAR EDUCATION HUB

Answer: b
Explanation: A modular system consist of well defined manageable units with well defined interfaces among the units.

211. Which of the property of software modularity is incorrect with respect to benefits software modularity?
a) Modules are robust
b) Module can use other modules
c) Modules Can be separately compiled and stored in a library
d) Modules are mostly dependent

Answer: d
Explanation: Modularity cannot bring benefits unless the modules are autonomous or independent.

212. _____ is a measure of the degree of interdependence between modules.
a) Cohesion
b) Coupling
c) None of the mentioned
d) All of the mentioned

Answer: b
Explanation: Coupling or dependency is the degree to which each program module relies on each one of the other modules.

213. Which of the following is the best type of module coupling?
a) Control Coupling
b) Stamp Coupling
c) Data Coupling
d) Content Coupling

Answer: c
Explanation: The dependency between module A and B is said to be data coupled if their dependency is based on the fact they communicate by only passing of data.

214. Which of the following is the worst type of module coupling?
a) Control Coupling
b) Stamp Coupling
c) External Coupling
d) Content Coupling

Answer: c
Explanation: Content coupling occurs when module A changes data of module B or when control is passed from one module to the middle of another.

215. Which of the following is the worst type of module cohesion?
a) Logical Cohesion
b) Temporal Cohesion
c) Functional Cohesion
d) Coincidental Cohesion

DIVAKAR EDUCATION HUB

Answer: d
Explanation: Coincidental cohesion exists in modules that contain instructions that have little or no relationship to one another.

216. Which of the following is the best type of module cohesion?
a) Functional Cohesion
b) Temporal Cohesion
c) Functional Cohesion
d) Sequential Cohesion

Answer: a
Explanation: Functional Cohesion is a type of cohesion in which the tasks performed by a software module all contribute to the performance of a single function.

217. A software engineer must design the modules with the goal of high cohesion and low coupling.
a) True
b) False

Answer: a
Explanation: If the software is not properly modularized, a host of seemingly trivial enhancement or changes will result into death of the project.

218. In what type of coupling, the complete data structure is passed from one module to another?
a) Control Coupling
b) Stamp Coupling
c) External Coupling
d) Content Coupling

Answer: b
Explanation: None.

219. If all tasks must be executed in the same time-span, what type of cohesion is being exhibited?
a) Functional Cohesion
b) Temporal Cohesion
c) Functional Cohesion
d) Sequential Cohesion

Answer: b
Explanation: A Module exhibits temporal cohesion when it contains tasks that are related by the fact that all tasks must be executed in the same time-span.

220. Choose the option that does not define Function Oriented Software Design.
a) It consists of module definitions
b) Modules represent data abstraction
c) Modules support functional

DIWAKAR EDUCATION HUB

abstraction
d) None of the mentioned

Answer:b
Explanation: Option b defines an Object Oriented Design.

221. Which of the following is a complementary approach to function-oriented approach ?
a) Object oriented analysis
b) Object oriented design
c) Structured approach
d) Both Object oriented analysis and design

Answer:d
Explanation: None.

222. Function-oriented design techniques starts with functional requirements specified in
a) SDD
b) SRS
c) All of the mentioned
d) None of the mentioned

Answer:b
Explanation: None.

223. Structured Analysis is based on the principles of
a) Top-down decomposition approach
b) Divide and conquer principle
c) Graphical representation of results using DFDs
d) All of the mentioned

Answer:d
Explanation: None.

224. Which of the following is/are true with respect to functions ?
a) A function such as “search-book” is represented using a circle
b) Functions represent some activity
c) Function symbol is known as a process symbol or a bubble in DFD
d) All of the mentioned

Answer:d
Explanation: All the options are correct with respect to Function Oriented Software Design.

225. Which of the following is not a use of a CASE tool ?
a) Support structured analysis and design (SA/SD)
b) Maintains the data dictionary
c) Checks whether DFDs are balanced or not
d) It complies with the available system

Answer:d
Explanation: It takes long time to

DIWAKAR EDUCATION HUB

establish the system in order to comply with the available system.

226. What DFD notation is represented by the Rectangle?
a) Transform
b) Data Store
c) Function
d) None of the mentioned

Answer:b
Explanation: None.

227. Structural decomposition is concerned with function calls.
a) True
b) False

Answer:a
Explanation: Structural decomposition is concerned with developing a model of the design which shows the dynamic structure.

228. A function-oriented design focuses on the entities in the system rather than the data processing activities.
a) True
b) False

Answer:b
Explanation: It is an object oriented design which focus on entities.

229. In DFDs, user interactions with the system is denoted by
a) Circle
b) Arrow
c) Rectangle
d) Triangle

Answer:a
Explanation: None.

230. SA/SD features are obtained from which of the methodologies?
a) Constantine and Yourdon methodology
b) DeMarco and Yourdon methodology
c) Gane and Sarson methodology
d) All of the mentioned

Answer: d
Explanation: None.

231. Which of the following is not an activity of Structured Analysis (SA) ?
a) Functional decomposition
b) Transformation of a textual problem description into a graphic model
c) All the functions represented in the DFD are mapped to a module structure
d) All of the mentioned

DIWAKAR EDUCATION HUB

Answer: c
Explanation: The module structure is the software architecture.

232. To arrive at a form which is suitable for implementation in some programming language is the purpose of
a) Structured Analysis (SA)
b) Structured Design (SD)
c) Detailed Design (DD)
d) None of the mentioned

Answer: b
Explanation: None.

233. The results of structured analysis can be easily understood by ordinary customers.
a) True
b) False

Answer: a
Explanation: The results of structured analysis directly represents customer's perception of the problem and uses customer's terminology for naming different functions and data.

234. Structured Analysis is based on the principle of Bottom-Up Approach.
a) True
b) False

Answer: b
Explanation: Structured Analysis follows uses decomposition approach.

235. The context diagram is also known as
a) Level-0 DFD
b) Level-1 DFD
c) Level-2 DFD
d) All of the mentioned

Answer: a
Explanation: Context diagram captures the various entities external to the system interacting with it and data flow occurring between the system and the external entities.

236. A directed arc or line in DFD represents
a) Data Store
b) Data Process
c) Data Flow
d) All of the mentioned

Answer: c
Explanation: It resembles data flow in the direction of the arrow.

237. A DFD is always accompanied by a data dictionary.
a) True
b) False

DIWAKAR EDUCATION HUB

Answer: a
Explanation: A data dictionary lists all data items appearing in a DFD including definition and data names.

238. Which of the following is a function of CASE Tool?
a) Supporting Structured analysis and design (SA/SD)
b) Maintaining the data dictionary
c) Checking whether DFDs are balanced or not
d) All of the mentioned

Answer: a
Explanation: None.

239. Data Store Symbol in DFD represents a
a) Physical file
b) Data Structure
c) Logical file
d) All of the mentioned

Answer: d
Explanation: A logical file can be a data structure or a physical file on disk.

240. Choose the incorrect statement in terms of Objects.
a) Objects are abstractions of real-world
b) Objects can't manage themselves
c) Objects encapsulate state and representation information
d) All of the mentioned

Answer: b
Explanation: Objects are independent.

241. What encapsulates both data and data manipulation functions ?
a) Object
b) Class
c) Super Class
d) Sub Class

Answer: a
Explanation: None.

242. Which of the following is a mechanism that allows several objects in an class hierarchy to have different methods with the same name?
a) Aggregation
b) Polymorphism
c) Inheritance
d) All of the mentioned

Answer: b
Explanation: In polymorphism instances of each subclass will be free to respond to messages by calling their own version of the method.

DIWAKAR EDUCATION HUB

243. Inherited object classes are self-contained.
a) True
b) False

Answer: b
Explanation: Inherited object classes are not self-contained. They cannot be understood without reference to their super-classes.

244. Which of the following points related to Object-oriented development (OOD) is true?
a) OOA is concerned with developing an object model of the application domain
b) OOD is concerned with developing an object-oriented system model to implement requirements
c) All of the mentioned
d) None of the mentioned

Answer: c
Explanation: The answer is in support with the OOD.

245. How is generalization implemented in Object Oriented programming languages?
a) Inheritance
b) Polymorphism
c) Encapsulation
d) Abstract Classes

Answer: a
Explanation: None.

246. Which of the following is a disadvantage of OOD ?
a) Easier maintenance
b) Objects may be understood as stand-alone entities
c) Objects are potentially reusable components
d) None of the mentioned

Answer: d
Explanation: All the options define the characteristics of OOD.

247. Which of the following describes"Is-a-Relationship" ?
a) Aggregation
b) Inheritance
c) Dependency
d) All of the mentioned

Answer: b
Explanation: None.

248. Object that collects data on request rather than autonomously is known as
a) Active Object
b) Passive Object
c) Multiple instance
d) None of the mentioned

DIVAKAR EDUCATION HUB

Answer: b
Explanation: A passive object holds data, but does not initiate control.

249. Objects are executed
a) sequentially
b) in Parallel
c) sequentially & Parallel
d) none of the mentioned

Answer: c
Explanation: Objects may be distributed and may execute sequentially or in parallel.

250. How many layers are present in the OO design pyramid?
a) three
b) four
c) five
d) one

Answer: b
Explanation: The four layers are: Subsystem layer, class and object layer, message layer and responsibilities layer

251. A design description in OOD includes
a) Protocol Description
b) Implementation Description
c) Type Description
d) both Protocol and Implementation Description

Answer: d
Explanation: None.

252. Which of the following is not an operation as per OOD algorithms and data structures?
a) operations that manipulate data in some way
b) operations that perform a computation
c) operations that check for syntax errors
d) operations that monitor an object for the occurrence of a controlling event

Answer: c
Explanation: Operations that check for syntax errors is concerned with the programming language used, so it will be handled by the compiler.

253. Throughout the OOD process, a software engineer should look for every opportunity for creating new design process.
a) True
b) False

Answer: b
Explanation: A software engineer

DIWAKAR EDUCATION HUB

should look for every opportunity to reuse existing design patterns whenever they meet the needs of the design rather than creating new ones.

254. The user has no control over the contents of a static web page.
a) True
b) False

Answer: a
Explanation: Static web pages are just for information purposes.

255. Which metric gives the idea about the contents on a web page ?
a) Word Token
b) Word Count
c) Word Size
d) Word Length

Answer: b
Explanation: The word count metric gives the total number of words on a web page.

256. How is the complexity of a web page related to link count ?
a) Directly
b) Indirectly
c) No relation
d) All of the mentioned

Answer: a
Explanation: If link count is more, complexity will be more.

256. It is expected to have less number of connections for a good web application.
a) True
b) False

Answer: a
Explanation: More the link count, more the complexity and the web page dependence factor will increase.

257. Number of dynamic web pages provides an idea about_____ for a web page that is to be built.
a) size
b) complexity
c) effort
d) all of the mentioned

Answer: d
Explanation: None.

258. Which of the following web engineering metric measures the extent of relatedness between two or more web pages ?
a) Number of Static Content Objects
b) Number of Dynamic Content Objects

DIWAKAR EDUCATION HUB

c) Web Page Similarity
d) Number of Internal Page Links

Answer: c
Explanation: None.

259. Which of the following is not a classification of the web engineering metric, Web Page Similarity ?
a) Content based
b) Link based
c) Usage based
d) Traffic based

Answer: d
Explanation: Similarity between two web pages is not judged upon its traffic activity.

260. Link based measures rely on _____ structure of a web graph to obtain related pages.
a) Embedded
b) Hyperlink
c) Dynamic
d) All of the mentioned

Answer: b
Explanation: Only option b answers the blank, rest are not in accordance to the question.

261. Which of the following is not a web engineering project metric ?
a) Number of Static Content Objects
b) Number of Dynamic Content Objects
c) Number of Inherited Objects
d) Word Count

Answer: c
Explanation: There is no such metric as an inherited object's count.

262. Which of the following is not project management goal?
a) Keeping overall costs within budget
b) Delivering the software to the customer at the agreed time
c) Maintaining a happy and well-functioning development team
d) Avoiding customer complaints

Answer: d
Explanation: Projects need to be managed because professional software engineering is always subject to organizational budget and schedule constraints.

263. Project managers have to assess the risks that may affect a project.
a) True
b) False

Answer: b
Explanation: Risk management involves anticipating risks that might

DIWAKAR EDUCATION HUB

affect the project schedule or the quality of the software being developed, and then taking action to avoid these risks.

264. Which of the following is not considered as a risk in project management?
a) Specification delays
b) Product competition
c) Testing
d) Staff turnover

Answer: c
Explanation: Testing is a part of project, thus it can't be categorized as risk.

265. The process each manager follows during the life of a project is known as
a) Project Management
b) Manager life cycle
c) Project Management Life Cycle
d) All of the mentioned

Answer: c
Explanation: A proven methodical life cycle is necessary to repeatedly implement and manage projects successfully.

266. Quality planning is the process of developing a quality plan for
a) team
b) project
c) customers
d) project manager

Answer: b
Explanation: The quality plan should set out the desired software qualities and describe how these are to be assessed.

267. Which of the following is incorrect activity for the configuration management of a software system?
a) Internship management
b) Change management
c) Version management
d) System management

Answer: a
Explanation: Configuration management policies and processes define how to record and process proposed system changes, how to decide what system components to change, how to manage different versions of the system and its components, and how to distribute changes to customers.

268. Identify the sub-process of process improvement
a) Process introduction

DIWAKAR EDUCATION HUB

b) Process analysis
c) De-processification
d) Process distribution

Answer: b
Explanation: The current process is assessed, and process weaknesses and bottlenecks are identified.

269. Which of the following is an important factor that can affect the accuracy and efficacy of estimates?
a) Project size
b) Planning process
c) Project complexity
d) Degree of structural uncertainty

Answer: a
Explanation: As size increases, the interdependence among various elements of the software grows rapidly.

270. What describes the data and control to be processed?
a) Planning process
b) Software scope
c) External hardware
d) Project complexity

Answer: b
Explanation: Functions described in the statement of scope are evaluated and in some cases refined to provide more detail prior to the beginning of estimation.

271. A number of independent investigators have developed a team-oriented approach to requirements gathering that can be applied to establish the scope of a project called
a) JAD
b) CLASS
c) FAST
d) None of the mentioned

Answer: c
Explanation: Facilitated application specification techniques
(FAST), this approach encourages the creation of a joint team of customers and developers who work together to identify the problem, propose elements
of the solution, negotiate different approaches, and specify a preliminary set of
requirements.

272. The project planner examines the statement of scope and extracts all important software functions which is known as
a) Association
b) Decomposition

DIWAKAR EDUCATION HUB

c) Planning process
d) All of the mentioned

Answer: b
Explanation: None

273. Which of the following is not an option to achieve reliable cost and effort estimate?
a) Base estimates on similar projects that have already been completed
b) Use one or more empirical models for software cost and effort estimation
c) Use relatively simple decomposition techniques to generate project cost and effort estimates
d) The ability to translate the size estimate into human effort, calendar time, and dollars

Answer: d
Explanation: None.

274. What can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right?
a) Automated estimation tools
b) Empirical estimation models
c) Decomposition techniques
d) Both Automated estimation tools and Empirical estimation models

Answer: b
Explanation: An estimation model for computer software uses empirically derived formulas to predict effort as a function of LOC or FP.

275. Software project estimation can never be an exact science, but a combination of good historical data and systematic techniques can improve estimation accuracy.
a) True
b) False

Answer: a
Explanation: None.

176. Which of the following are parameters involved in computing the total cost of a software development project?
a) Hardware and software costs
b) Effort costs
c) Travel and training costs
d) All of the mentioned

Answer: d
Explanation: All these are accounted for in estimating a software development cost.

DIWAKAR EDUCATION HUB

277. Which of the following costs is not part of the total effort cost?
a) Costs of networking and communications
b) Costs of providing heating and lighting office space
c) Costs of lunch time food
d) Costs of support staff

Answer: c
Explanation: This is a incurred by the employees.

278. It is often difficult to estimate size at an early stage in a project when only a specification is available
a) True
b) False

Answer: a
Explanation: Function-point and object-point estimates are easier to produce than estimates of code size but are often still inaccurate.

279. Which technique is applicable when other projects in the same analogy application domain have been completed?
a) Algorithmic cost modelling
b) Expert judgement
c) Estimation by analogy
d) Parkinson's Law

Answer: c
Explanation: The cost of a new project is estimated by analogy with these completed projects.

280. Which model assumes that systems are created from reusable components, scripting or database programming?
a) An application-composition model
b) A post-architecture model
c) A reuse model
d) An early design model

Answer: a
Explanation: It is designed to make estimates of prototype development.

281. Which model is used to compute the effort required to integrate reusable components or program code that is automatically generated by design or program translation tools?
a) An application-composition model
b) A post-architecture model
c) A reuse model
d) An early design model

Answer: c
Explanation: None.

282. The COCOMO model takes into account different approaches to

DIWAKAR EDUCATION HUB

software development, reuse, etc.
a) True
b) False

Answer: b
Explanation: Its the COCOMO-2 model. COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.

283. COCOMO stands for
a) Constructive cost model
b) Comprehensive cost model
c) Constructive cost estimation model
d) Complete cost estimation model

Answer: a
Explanation: None.

284. Which version of COCOMO states that once requirements have been stabilized, the basic software architecture has been established?
a) Early design stage model
b) Post-architecture-stage model
c) Application composition model
d) All of the mentioned

Answer: a
Explanation: None.

285. Which model was used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity were paramount.
a) Early design stage model
b) Post-architecture-stage model
c) Application composition model
d) All of the mentioned

Answer: c
Explanation: None.

286.COCOMO was developed initially by
a) B.Beizer
b) Rajiv Gupta
c) B.W.Bohem
d) Gregg Rothermal

Answer: c
Explanation: Barry Boehm introduced a hierarchy of software estimation models bearing the name COCOMO, for COnstructive COst MOdel.

287. Estimation of size for a project is dependent on
a) Cost
b) Time
c) Schedule
d) None of the mentioned

DIWAKAR EDUCATION HUB

Answer: d
Explanation: Estimation depends on factors such as Function points and LOC.

288. COCOMO-II was developed at
a) University of Texas
b) University of Southern California
c) MIT
d) IIT-Kanpur

Answer: b
Explanation: None.

289. Which one is not a stage of COCOMO-II?
a) Early design estimation model
b) Application Composition estimation model
c) Comprehensive cost estimation model
d) Post architecture estimation model

Answer: a
Explanation: It was a part of COCOMO.

290. Size and Complexity are a part of
a) Product Metrics
b) Process Metrics
c) Project Metrics
d) All of the mentioned

Answer: a
Explanation: Product Metrics describe the characteristics of product.

291. Cost and schedule are a part of
a) Product Metrics
b) Process Metrics
c) Project Metrics
d) All of the mentioned

Answer: c
Explanation: Project Metrics describe the project characteristics and execution.

292. Number of errors found per person hours expended is an example of a
a) measurement
b) measure
c) metric
d) all of the mentioned

Answer: c
Explanation: Metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.

293. Which of the following is not categorized under Product Operation of McCall's Software Quality Factors?
a) Flexibility
b) Reliability

DIWAKAR EDUCATION HUB

c) Usability
d) Integrity

Answer: a
Explanation: Flexibility is a part of Product revision as per McCall's Software Quality Factors.

195. Which of the following is not categorized under Component-Level Design Metrics?
a) Complexity Metrics
b) Cohesion Metrics
c) Morphology Metrics
d) Coupling Metrics

Answer: c
Explanation: Morphology metrics are a part of High level design metrics.

296. Percentage of modules that were inspected is a part of
a) Product Metrics
b) Process Metrics
c) Project Metrics
d) All of the mentioned

Answer: b
Explanation: None.

297. MTTC falls the the category of
a) correctness
b) integrity
c) maintainability
d) all of the mentioned
View Answer

298. Identify the correct option with reference to Software Quality Metrics.
a) Integrity = [Sigma(1 – threat)] * (1 – security)
b) Integrity = [1 – Sigma(threat)] * (1 – security)
c) Integrity = [1 – threat * Sigma(1 – security)].
d) Integrity = Sigma[1 – threat * (1 – security)].

Answer: d
Explanation: None.

299. Which one is not a risk management activity?
a) Risk assessment
b) Risk generation
c) Risk control
d) None of the mentioned

Answer: b
Explanation: Risk management activities would never want a new risk to be generated.

300. Risk management is one of the most important jobs for a
a) Client
b) Investor

DIWAKAR EDUCATION HUB

c) Production team
d) Project manager

Answer: d
Explanation: Risk management involves anticipating risks that might affect the project schedule or the quality of the software being developed, and then taking action to avoid these risks.

301. Which of the following risk is the failure of a purchased component to perform as expected?
a) Product risk
b) Project risk
c) Business risk
d) Programming risk

Answer: a
Explanation: Risks that affect the quality or performance of the software being developed.

302. Which of the following term is best defined by the statement: "There will be a change of organizational management with different priorities."?
a) Staff turnover
b) Technology change
c) Management change
d) Product competition

Answer: c
Explanation: None.

303. What assess the risk and your plans for risk mitigation and revise these when you learn more about the risk?
a) Risk monitoring
b) Risk planning
c) Risk analysis
d) Risk identification

Answer: a
Explanation: None.

304. Which of the following risks are derived from the organizational environment where the software is being developed?
a) People risks
b) Technology risks
c) Estimation risks
d) Organizational risks

Answer: d
Explanation: These risks are at management level.

305. Which of the following risks are derived from the software or hardware technologies that are used to develop the system?
a) Managerial risks
b) Technology risks

DIWAKAR EDUCATION HUB

c) Estimation risks
d) Organizational risks

Answer: b
Explanation: The risks associated with technology might affect the product development.

306. Which of the following strategies means that the impact of the risk will be reduced?
a) Avoidance strategies
b) Minimization strategies
c) Contingency plans
d) All of the mentioned

Answer: b
Explanation: None.

307. Risk management is now recognized as one of the most important project management tasks.
a) True
b) False

Answer: a
Explanation: None.

308. Why is decomposition technique required?
a) Software project estimation is a form of problem solving
b) Developing a cost and effort estimate for a software project is too complex
c) All of the mentioned
d) None of the mentioned

Answer: c
Explanation: For these reasons, we decompose the problem, re-characterizing it as a set of smaller problems.

309. Cost and effort estimation of a software uses only one forms of decomposition, either decomposition of the problem or decomposition of the process.
a) True
b) False

Answer: b
Explanation: Estimation uses one or both forms of partitioning.

310. If a Direct approach to software project sizing is taken, size can be measured in
a) LOC
b) FP
c) LOC and FP
d) None of the mentioned

Answer: a
Explanation: LOC or Line of Code is a direct measure to estimate project size.

DIWAKAR EDUCATION HUB

311. Which software project sizing approach develop estimates of the information domain characteristics?
a) Function point sizing
b) Change sizing
c) Standard component sizing
d) Fuzzy logic sizing

Answer: a
Explanation: None.

312. The expected value for the estimation variable (size), S, can be computed as a weighted average of the optimistic(Sopt), most likely (Sm), and pessimistic (Spess) estimates given as
a) EV = (Sopt + 4Sm + Spess)/4
b) EV = (Sopt + 4Sm + Spess)/6
c) EV = (Sopt + 2Sm + Spess)/6
d) EV = (Sopt + 2Sm + Spess)/4

Answer: b
Explanation: This assumes that there is a very small probability that the actual size result will fall outside the optimistic or pessimistic values.

313. How many forms exists of Barry Boehm's COCOMO Model?
a) Two
b) Three
c) Four
d) No form exists

Answer: b
Explanation: The three forms include the basic, intermediate and advanced COCOMO model.

314. In many cases, it is often more cost-effective to acquire, rather than develop, computer software.
a) True
b) False

Answer: a
Explanation: Managers are faced with a make-buy decision in such situations.

315. A make-buy decision is based on whether
a) The software may be purchased off-the-shelf
b) "Full-experience" or "Partial-experience" software components should be used
c) Customer-built software should be developed
d) All of the mentioned

Answer: d
Explanation: None..

316. Which of the following is not one of the five information domain characteristics of Function Point (FP) decomposition?

DIVAKAR EDUCATION HUB

a) External inputs
b) External outputs
c) External process
d) External inquiries

Answer: c
Explanation: External inputs, external outputs, external inquiries, internal logical files, external interface files are the five domains.

317. The project planner must reconcile the estimates based on decomposition techniques to produce a single estimate of effort.
a) True
b) False

Answer: b
Explanation: The planner must determine the cause of divergence and then reconcile the estimates.

318. Programming language experience is a part of which factor of COCOMO cost drivers?
a) Personnel Factor
b) Product Factor
c) Platform Factor
d) Project Factor

Answer: a
Explanation: None.

319. If an Indirect approach is taken, then the sizing approach is represented as
a) LOC
b) FP
c) Fuzzy Logic
d) LOC and FP

Answer: b
Explanation: A function point (FP) is a unit of measurement to express the amount of business functionality an information system provides to a user.

320. Ensuring that no more than the allocated number of people are allocated at any given time in Software Scheduling is known as
a) Time Allocation
b) Effort Validation
c) Defined Milestone
d) Effort Distribution

Answer: b
Explanation: None.

321. What is used to determine the recommended degree of rigor with which the software process should be applied on a project?
a) Degree of Rigor
b) Adaptation criteria
c) Task Set

DIWAKAR EDUCATION HUB

d) Both degree of Rigor and adaptation criteria

Answer: b
Explanation: Four different degrees of rigor are: casual, structured, strict, and quick reaction.

322. What evaluates the risk associated with the technology to be implemented as part of project scope?
a) Concept scoping
b) Preliminary concept planning
c) Technology risk assessment
d) Customer reaction to the concept

Answer: b
Explanation: None.

323. Which of the following is not an adaptation criteria for software projects?
a) Size of the project
b) Customers Complaints
c) Project staff
d) Mission criticality

324. Which of the following is a project scheduling method that can be applied to software development?
a) PERT
b) CPM
c) CMM
d) Both PERT and CPM

Answer: d
Explanation: Program evaluation and review technique (PERT) and critical path method (CPM) are two project scheduling methods that can be applied to software development.

325. A technique for performing quantitative analysis of progress is known as
a) BCWS
b) EVA
c) BAC
d) CBSE

Answer: b
Explanation: The earned value system provides a common value scale for every task, regardless of the type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total.

326. What is the recommended distribution of effort for a project?
a) 40-20-40
b) 50-20-30
c) 30-40-30
d) 50-30-20

DIWAKAR EDUCATION HUB

Answer: a
Explanation: A recommended distribution of effort across the software process is 40% (analysis and design), 20% (coding), and 40% (testing).

327. A project usually has a timeline chart which was developed by
a) Henry Gantt
b) Barry Boehm
c) Ivar Jacabson
d) None of the mentioned

Answer: a
Explanation: Timeline chart, also called a Gantt chart was invented by Henry Gantt, an industrial engineer in 1917 .

328. Which of the following is not needed to develop a system design from concept to detailed object-oriented design?
a) Designing system architecture
b) Developing design models
c) Specifying interfaces
d) Developing a debugging system

Answer: d
Explanation: The debugging system is a part of testing phase.

329. Which of the following come under system control?
a) Reconfigure
b) Shutdown
c) Powersave
d) All of the mentioned

Answer: d
Explanation: Functionalities are governed by the system.

330. We use _____ where various parts of system use are identified and analyzed in turn.
a) tangible entities
b) scenario-based analysis
c) design-based analysis
d) none of the mentioned

Answer: b
Explanation: Use a scenario-based analysis where various scenarios of system use are identified and analyzed in turn.

331. Which model describes the static structure of the system using object classes and their relationships?
a) Sequence model
b) Subsystem model
c) Dynamic model
d) Structural model

DIVAKAR EDUCATION HUB

Answer: d
Explanation: Important relationships that may be documented at this stage are generalization (inheritance) relationships, uses/used-by relationships, and composition relationships.

332. Which model shows the flow of object interactions?
a) Sequence model
b) Subsystem model
c) Dynamic model
d) Both Sequence and Dynamic model

Answer: a
Explanation: Sequence model are represented using a UML sequence or a collaboration diagram and are dynamic models.

333. If the system state is Shutdown then it can respond to which of the following message?
a) restart()
b) reconfigure()
c) powerSave()
d) all of the mentioned

Answer: d
Explanation: A restart() message causes a transition to normal operation. Both the powerSave() and reconfigure() messages cause a transition to a state in which the system reconfigures itself.

334. Which message is received so that the system moves to the Testing state, then the Transmitting state, before returning to the Running state?
a) signalStatus()
b) remoteControl()
c) reconfigure()
d) reportStatus()

Answer: d
Explanation: None.

335. Which of the following is not a construct?
a) sequence
b) condition
c) repetition
d) selection

Answer: d
Explanation: Sequence implements processing steps that are essential in the specification of any algorithm. Condition provides the facility for selected processing based on some logical occurrence, and repetition allows for looping.

DIWAKAR EDUCATION HUB

336. Which of the following term is best defined by the statement:"The ability to represent local and global data is an essential element of component-level design."?
a) Data representation
b) Logic verification
c) "Code-to" ability
d) Automatic processing

Answer: a
Explanation: None.

337. A software component
a) Implements some functionality
b) Has explicit dependencies through provides and required interfaces
c) Communicates through its interfaces only
d) All of the mentioned

Answer: d
Explanation: All the options identify with features of a software component.

338. Which diagram evolved from a desire to develop a procedural design representation that would not allow violation of the structured constructs?
a) State transition diagram
b) Box diagram
c) ER diagram
d) None of the mentioned

Answer: b
Explanation: None.

339. Which of the following is golden rule for interface design?
a) Place the user in control
b) Reduce the user's memory load
c) Make the interface consistent
d) All of the mentioned

Answer: d
Explanation: These golden rules actually form the basis for a set of user interface design principles that guide this important software design activity.

340. Which of the following is not a design principle that allow the user to maintain control?
a) Provide for flexible interaction
b) Allow user interaction to be interrupt-able and undo-able
c) Show technical internals from the casual user
d) Design for direct interaction with objects that appear on the screen

Answer: c
Explanation: The user interface

DIWAKAR EDUCATION HUB

should move the user into the virtual world of the application.

341. Which of the following is not a user interface design process?
a) User, task, and environment analysis and modeling
b) Interface design
c) Knowledgeable, frequent users
d) Interface validation

Answer: c
Explanation: These are the end user for whom the product is being built.

342. When users are involved in complex tasks, the demand on _____ can be significant.
a) short-term memory
b) shortcuts
c) objects that appear on the screen
d) all of the mentioned

Answer: a
Explanation: The interface should be designed to reduce the requirement to remember past actions and results.

343. Which of the following option is not considered by the Interface design?
a) the design of interfaces between software components
b) the design of interfaces between the software and human producers and consumers of information
c) the design of the interface between two computers
d) all of the mentioned

Answer: c
Explanation: None

344. A software might allow a user to interact via
a) keyboard commands
b) mouse movement
c) voice recognition commands
d) all of the mentioned

Answer: d
Explanation: All the mentioned input mediums are available today.

345. A software engineer designs the user interface by applying an iterative process that draws on predefined design principles.
a) True
b) False

Answer: a
Explanation: The statement is true.

346. What incorporates data, architectural, interface, and procedural representations of the software?

DIWAKAR EDUCATION HUB

a) design model
b) user's model
c) mental image
d) system image

Answer: a
Explanation: The requirements specification may establish certain constraints that help to define the user of the system, but the interface design is often only incidental to the design model.

347. What establishes the profile of end-users of the system?
a) design model
b) user's model
c) mental image
d) system image

Answer: b
Explanation: To build an effective user interface, all design should begin with an understanding of the intended users, including their profiles of their age, physical abilities, education, etc.

348. What do you understand by V&V in software testing?
a) Verified Version
b) Version Validation
c) Verification and Validation
d) Version Verification

Answer: c
Explanation: V&V generally refers to any activity that attempts to ensure that the software will function as required.

349. In static test techniques, behavioral and performance properties of the program are observed.
a) True
b) False

Answer: b
Explanation: Static Analysis Techniques are based solely on the (manual or automated) examination of project documentation of software models and code.

350. Which granularity level of testing checks the behavior of module cooperation?
a) Unit Testing
b) Integration Testing
c) Acceptance Testing
d) Regression Testing

Answer: b
Explanation: Integration testing is the phase in software testing in which individual software modules are combined and tested as a group.

DIWAKAR EDUCATION HUB

351. Which test refers to the retesting of a unit, integration and system after modification, in order to ascertain that the change has not introduced new faults?
a) Regression Test
b) Smoke Test
c) Alpha Test
d) Beta Test

Answer: a
Explanation: Regression test seeks to uncover new software bugs in existing functional and non-functional areas of a system after changes have been made to them.

352. Which of the following is a black box testing strategy?
a) All Statements Coverage
b) Control Structure Coverage
c) Cause-Effect Graphs
d) All Paths Coverage

Answer: c
Explanation: Rest are test strategies of white box testing.

353. A set of inputs, execution preconditions and expected outcomes is known as a
a) Test plan
b) Test case
c) Test document
d) Test Suite

Answer: b
Explanation: None.

354. When does the testing process stops?
a) When resources (time and budget) are over
b) When some coverage is reached
c) When quality criterion is reached
d) Testing never ends

Answer: c
Explanation: As software testing is an exhaustive process, when the quality assurance is established and the product is ready to be delivered, testing is stopped.

355. Which of the following is not a part of a test design document?
a) Test Plan
b) Test Design Specification
c) Test Case Specification
d) Test Log

Answer: d
Explanation: Test log is a part of testing result document.

356. Specifying a set of test cases or test paths for each item to be tested at that level is known as

DIWAKAR EDUCATION HUB

a) Test case generation
b) Test case design
c) ALL of the mentioned
d) None of the mentioned

Answer: c
Explanation: None.

357. Acceptance & system test planning are a part of architectural design.
a) True
b) False

Answer: b
Explanation: They are a part of requirements engineering, while integration & unit test planning come under architectural design.

358. PRD stands for
a) Product Requirement Document
b) Project Requirement Document
c) Product Restrictions Document
d) None of the mentioned

Answer: a
Explanation: A product requirements document (PRD) is a document written by a company that defines a product they are making, or the requirements for one or more new features for an existing product.

359. Which mechanism is applied to use a design pattern in an OO system?
a) Inheritance
b) Composition
c) All of the mentioned
d) None of the mentioned

Answer: c
Explanation: Using inheritance, an existing design pattern becomes a template for a new subclass.Composition is a concept that leads to aggregate objects.

360. Design patterns does not follow the concept of software reuse.
a) True
b) False

Answer: b
Explanation: Design patterns allow the designer to create the system architecture by integrating reusable components.

361. The use of design patterns for the development of object-oriented software has important implications for
a) Component-based software engineering
b) Reusability in general

DIWAKAR EDUCATION HUB

c) All of the mentioned
d) None of the mentioned

Answer: c
Explanation: None.

362. Which of the following is a design pattern?
a) Behavioral
b) Structural
c) Abstract Factory
d) All of the mentioned

Answer: d
Explanation: All the options are design patterns so option d.

363. You want to minimize development cost by reusing methods? Which design pattern would you choose?
a) Adapter Pattern
b) Singleton Pattern
c) Delegation pattern
d) Immutable Pattern

Answer: c
Explanation: The delegation pattern is a design pattern in OOP where an object, instead of performing one of its stated tasks, delegates that task to an associated helper object.

364. You want to avoid multiple inheritance. Which design pattern would you choose?
a) Abstraction-Occurrence Pattern
b) Player-Role Pattern
c) General Hierarchy Pattern
d) Singleton Pattern

Answer: b
Explanation: The answer is self-explanatory.

365. The recurring aspects of designs are called design
a) patterns
b) documents
c) structures
d) methods

Answer: a
Explanation: A pattern is the outline of a reusable solution to a general problem encountered in a particular context.

366. Design pattern is a solution to a problem that occurs repeatedly in a variety of contexts.
a) True
b) False

Answer: a
Explanation: Each design pattern has a name and use of each pattern has consequences.

DIWAKAR EDUCATION HUB

367. Which pattern prevents one from creating more than one instance of a variable?
a) Factory Method
b) Singleton
c) Observer
d) None of the mentioned

Answer: b
Explanation: In singleton pattern, the class itself is made responsible for keeping track of its instance.Thus it ensures that no more than one instance is created.

368. Facade pattern promotes weak coupling between subsystem and its clients.
a) True
b) False

Answer: a
Explanation: It is one of the patterns benefit.The facade pattern shields clients from subsystem classes and reduces the number of objects that clients deal with.

369. Which design pattern defines one-to-many dependency among objects?
a) Singleton pattern
b) Facade Pattern
c) Observer pattern
d) Factory method pattern

Answer: c
Explanation: Observer pattern defines one-to-many dependency among objects so that when one object changes its state, all its dependents are notified.

370. Facade pattern couples a subsystem from its clients.
a) True
b) False

Answer: b
Explanation: A facade can be a single entry point to each subsystem level. It decouples the subsystem.

371. In factory method pattern, the framework must instantiate classes but it only knows about the abstract classes, which it cannot initiate. How would one solve this problem?
a) encapsulating the knowledge of which document subclass to is to be created and
b) moving this knowledge out of the framework
c) instantiating the application specific documents without knowing their class
d) all of the mentioned

DIWAKAR EDUCATION HUB

Answer: d
Explanation: Following all the options in order will solve the factory method problem.

372. Reuse-based software engineering is a software engineering strategy where the development process is geared to reusing existing software.
a) True
b) False

Answer: a
Explanation: None.

373. The open source movement has meant that there is a huge reusable code base available at
a) free of cost
b) low cost
c) high cost
d) short period of time

Answer: b
Explanation: The open source movement has meant that there is a huge reusable code base available at low cost. This may be in the form of program libraries or entire applications.

374. Consider the example and categorize it accordingly, “A pattern-matching system developed as part of a text-processing system may be reused in a database management system”.
a) Application system reuse
b) Component reuse
c) Object and function reuse
d) None of the mentioned

Answer: b
Explanation: Components of an application, ranging in size from subsystems to single objects, may be reused.

376. COTS product reuse means
a) Class and function libraries that implement commonly used abstractions are available for reuse
b) Shared components are woven into an application at different places when the program is compiled
c) Large-scale systems that encapsulate generic business functionality and rules are configured for an organization
d) Systems are developed by configuring and integrating existing application systems

Answer: d
Explanation: None.

DIWAKAR EDUCATION HUB

377. .NET are specific to which platform?
a) Java
b) Mac-OS
c) Microsoft
d) LINUX

Answer: c
Explanation: NET Framework (pronounced dot net) is a software framework developed by Microsoft that runs primarily on Microsoft Windows.

378. Which of the following is a generic structure that is extended to create a more specific subsystem or application?
a) Software reuse
b) Object-oriented programming language
c) Framework
d) None of the mentioned

Answer: c
Explanation: Frameworks are implemented as a collection of concrete and abstract object classes in an object-oriented programming language.

379. "An ordering system may be adapted to cope with a centralized ordering process in one company and a distributed process in another." Which category the example belong to?
a) Process specialization
b) Platform specialization
c) Environment specialization
d) Functional specialization

Answer: a
Explanation: In process specialization, the system is adapted to cope with specific business processes.

380. Which of the following is not an advantages of software reuse?
a) lower costs
b) faster software development
c) high effectiveness
d) lower risks

Answer: c
Explanation: Effectiveness depends on how one reuses the existing product.

381. Which framework class include standards and classes that support component communication and information exchange?
a) System infrastructure frameworks
b) Middleware integration frameworks
c) Enterprise application frameworks
d) MVC

DIWAKAR EDUCATION HUB

Answer: b
Explanation: None.

382. Which of the following is not a benefit of software reuse?
a) Standards compliance
b) Increased Reliability
c) Reduced Process risk
d) Maintaining a component library

Answer: c
Explanation: There can be thousands of components in a frameworks whose maintenance is quite difficult.

383. In which of the following language the frameworks will not work?
a) C#
b) Ruby
c) PHP
d) Java

Answer: c
Explanation: Frameworks available in all of the commonly used object-oriented programming languages.

384. Which frameworks support the development of system infrastructures such as communications, user interfaces, and compilers?
a) Middleware integration frameworks
b) System infrastructure framework
c) Enterprise application frameworks
d) Web application frameworks

Answer: b
Explanation: None.

385. The MVC pattern was originally proposed in the 1980s as an approach to
a) Web application frameworks
b) Middleware integration frameworks
c) Web application frameworks
d) GUI design

Answer: d
Explanation: The MVC pattern was originally proposed in the 1980s as an approach to GUI design that allowed for multiple presentations of an object and separate styles of interaction with each of these presentations.

386. MVC framework includes
a) Observer pattern
b) Strategy pattern
c) Composite pattern
d) All of the mentioned

Answer: d
Explanation: MVC framework

DIWAKAR EDUCATION HUB

includes the Observer pattern, the Strategy pattern, the Composite pattern, and a number of others .

387. Which category the following statement belongs,"Classes to create and manage sessions are usually part of a WAF"?
a) Session management
b) Security
c) User interaction
d) Database support

Answer: a
Explanation: None.

388. Which framework's applications are difficult to deal with?
a) MVC pattern
b) Web application frameworks
c) Debugging framework
d) None of the mentioned

Answer: c
Explanation: Debugging framework based applications is difficult because you may not understand how the framework methods interact. This is a general problem with reusable software .

389. Which of the following option is not provided by formal methods?
a) providing frameworks
b) verifying systems
c) provide investors
d) both providing frameworks and verifying systems

Answer: d
Explanation: A method is formal if it has a sound mathematical basis, typically given by a formal specification language.

390. Which of the following provides a concise, unambiguous, and consistent method for documenting system requirements?
a) CMM
b) ISO-9001
c) CASE tools
d) Formal methods

Answer: d
Explanation: Formal methods provide a concise, unambiguous, and consistent method for documenting system requirements.

391. It is generally not necessary to apply formal methods to every aspect of a major system.
a) True
b) False

Answer: a
Explanation: Those components that

DIWAKAR EDUCATION HUB

are safety critical are first choices, followed by components whose failure cannot be tolerated.

392. Service Oriented Architecture (SOA) is
a) Strongly Coupled
b) Loosely Coupled
c) Strongly Cohesive
d) Loosely Cohesive

Answer: b
Explanation: SOA is the architectural style that supports loosely coupled services to enable business flexibility.

393. Which of the following is an essential principle of an architecture?
a) Consistency
b) Reliability
c) Scalability
d) All of the mentioned

Answer: d
Explanation: Architecture implies a consistent and coherent design approach.

394. Which of the following is not a phase of "bathtub curve" of hardware reliability?
a) Useful Life
b) Burn-in
c) Wear-out
d) Time

Answer: d
Explanation: Time is the horizontal dimension on which the bathtub curve is built and not the phase.

395. How is reliability and failure intensity related to each other?
a) direct relation
b) inverse relation
c) no relation
d) none of the mentioned

Answer: b
Explanation: As the reliability increases, failure intensity decreases.

396. How many product quality factors are proposed in McCall quality model?
a) 2
b) 3
c) 11
d) 8

Answer: b
Explanation: McCall quality model has three product quality factors namely: Product revision, Product operation, Product Transition .

397. Which one of the following is not a software quality model?

DIWAKAR EDUCATION HUB

a) ISO 9000
b) McCall model
c) Boehm model
d) ISO 9126

Answer: a
Explanation: ISO-9000 series of standards is a set of document dealing with quality systems that can be used for quality assurance purposes.

398. What is MTTF ?
a) Maximum time to failure
b) Mean time to failure
c) Minimum time to failure
d) None of the mentioned

Answer: b
Explanation: None.

399. How is software reliability defined?
a) time
b) efficiency
c) quality
d) speed

Answer: a
Explanation: Software Reliability mainly concerned with the time component. It can be seen in various models like Basic Execution Time Model and Logarithmic Poisson Execution Time Model.

400. Suitability, Accuracy, Interoperability, and security are what type quality attribute of ISO 9126 ?
a) Reliability
b) Efficiency
c) Functionality
d) Usability

Answer: c
Explanation: All the Characteristics mentioned in the question are related to achievement of the basic purpose for which the software is being engineered, which is functionality.

401. Time Behavior and Resource Behavior fall under which quality attribute of ISO 9126 ?
a) Reliability
b) Efficiency
c) Functionality
d) Usability

Answer: b
Explanation: The Characteristics mentioned in the question are related to the relationship between the level of performance of the software and the amount of

DIWAKAR EDUCATION HUB

resources used, under stated conditions.

402. NHPP stands for
a) Non Homogeneous Poisson Product
b) Non-Hetrogeneous Poisson Product
c) Non-Hetrogeneous Poisson Process
d) Non Homogeneous Poisson Process

Answer: d
Explanation: None.

403. The CMM model is a technique to
a) automatically maintain the software reliability
b) improve the software process.
c) test the software
d) all of the mentioned

Answer: b
Explanation: Capability Maturity Model (CMM) is a strategy for improving the software process, irrespective of the actual life cycle model used.

404. Software Testing with real data in real environment is known as
a) alpha testing
b) beta testing
c) regression testing
d) none of the mentioned

Answer: b
Explanation: Beta testing is the last stage of testing, and normally can involve sending the product to beta test sites outside the company for real-world exposure or offering the product for a free trial download over the Internet.

405. Which of the following testing tools examine program systematically & automatically ?
a) Code Inspector
b) Static Analyzer
c) Standard Enforcer
d) Coverage Analyzer

Answer: b
Explanation: A static analyzer operates from a pre-computed database o descriptive information derived from the source text of the program.

406. Which testing tool is responsible for documenting programs ?
a) Test/File Generator
b) Test Harness System
c) Test Archiving Systems
d) Coverage Analyzer

DIWAKAR EDUCATION HUB

Answer: c
Explanation: The answer is self-explanatory.

407. Beta Testing is done by
a) Developers
b) Testers
c) Users
d) All of the mentioned

Answer: c
Explanation: None.

408. Standard enforcer tool looks at the whole program.
a) True
b) False

Answer: b
Explanation: This tool looks at only single statements.

409. Debugging Program is a program which runs concurrently with the program under test & provide commands to
a) examine memory & registers
b) stop execution at a particular point
c) search for references for particular variables, constant and registers
d) all of the mentioned

Answer: d
Explanation: Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware, thus making it behave as expected.

410. Execution Verifier is a dynamic tool that is also known as
a) Test File Generator
b) Coverage Analyzer
c) Output Comparator
d) Test Harness System

Answer: b
Explanation: None.

411. Why is software difficult to build ?
a) Controlled changes
b) Lack of reusability
c) Lack of monitoring
d) All of the mentioned

Answer: c
Explanation: Monitoring is a key aspect which requires much attention for a successful build.

412. Which of the following is not a conflict in software development team?
a) Simultaneous updates
b) Shared and common code
c) Versions
d) Graphics issues

DIWAKAR EDUCATION HUB

Answer: d
Explanation: These are part of design, which can be handled by the design team.

413. Which of the following lasts for the duration of the project and covers the development process?
a) Monitoring all key parameters like cost, schedule, risks
b) Taking corrective actions when needed
c) Providing information on the development process in terms of metrics
d) All of the mentioned

Answer: a
Explanation: None.

414. Which of the following is not a typical environment in communication facilitation ?
a) Multiple teams
b) Multiple user groups
c) Multiple fests
d) Multiple locations

Answer: c
Explanation: The answer is not related to the question.

415. Which of the following is a software process ?
a) Analysis and design
b) Configuration and management
c) Business modeling
d) All of the mentioned

Answer: d
Explanation: None.

416. Which of the following is not included in Issues Meetings?
a) Issues gathered the day before
b) Regular schedule of meeting
c) Discussion with business
d) Attendance

Answer: c
Explanation: Discussion with business is planning in QA Meetings.

417. What is a collection of software elements treated as a unit for the purposes of SCM?
a) Software Configuration Item
b) Baseline
c) Configuration
d) Configuration Control Board

Answer: a
Explanation: Software Configuration Item is a collection of software elements treated as a unit for the purposes of SCM.

418. What is one or more software configuration items that have been

DIWAKAR EDUCATION HUB

formally reviewed and agreed upon and serve as a basis for further development?
a) Configuration
b) Baseline
c) Software
d) All of the mentioned

Answer: b
Explanation: Baseline – One or more software configuration items that have been formally reviewed and agreed upon and serve as a basis for further development.

419. What is validating the completeness of a product?
a) Identification
b) Software
c) Auditing and Reviewing
d) Status Accounting

Answer: c
Explanation: Auditing and Reviewing is validating the completeness of a product and that SCM procedures are being followed.

420. What is group with the responsibility for reviewing and approving changes to baselines?
a) Software Configuration Item
b) Baseline
c) Configuration
d) Configuration Control Board

Answer: d
Explanation: Configuration Control Board (CCB) is the group with the responsibility for reviewing and approving changes to baselines.

421. In many settings PM is a center of communication hub
a) True
b) False

Answer: a
Explanation: None.

422. What is a specific instance of a baseline or configuration item?
a) Software
b) Configuration
c) Version
d) Status Accounting

Answer: c
Explanation: Even the smallest development projects should utilize some sort of version and baseline control tool.

DIWAKAR EDUCATION HUB