**Software Engineering**

The **software** is a collection of programs. (Programs + documentation)

(**Program** is a step by step instructions to perform a specific task using a programming language.)

**Software crisis**

**Overrun development cost due to constant change in requirements which leads to increase in bugs and results in to development of poor quality software.**

**To avoid this problem we need a proper engineering approach of software development i.e. Software Engineering.**

**Software Engineering** is the process of **designing, developing, testing, and maintaining software**.

It is a **systematic** and **disciplined** approach of software development that **aims** **to create**

**high-quality,**

**reliable, and l**

**maintainable software.**

**Key Principles of Software Engineering**

1. **Modularity**: Breaking the software into smaller, reusable components that can be developed and tested independently.
2. **Abstraction**: Hiding the implementation details of a component and exposing only the necessary functionality to other parts of the software.
3. **Encapsulation**: Wrapping up the data and functions of an object into a single unit, and protecting the internal state of an object from external modifications.
4. **Reusability**: Creating components that can be used in multiple projects, which can save time and resources.
5. **Maintenance**: Regularly updating and improving the software to fix bugs, add new features, and address security vulnerabilities.
6. **Testing**: Verifying that the software meets its requirements and is free of bugs.
7. **Design Patterns**: Solving recurring problems in software design by providing templates for solving them.
8. **Agile methodologies:**Using iterative and incremental development processes that focus on customer satisfaction, rapid delivery, and flexibility.
9. **Continuous Integration & Deployment:** Continuously integrating the code changes and deploying them into the production environment.

**Software Process**

Software processes in software engineering **refer to the methods and techniques used to develop and maintain Software**.

A software process (also known 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.

Feasibility study

• Find abstract definition of the problem

• Majorly checking the financial and technical feasibility

• Analysis of cost and befit ratio

• Checking availability of infrastructure and human resource

• Examination of alternative solution strategies

**Requirement analysis and specification**

• Try to understood the exact and complete requirement of the customer and document them properly.

• Try to collect and analysis all data related to the project.

• In last, a large document will be written in the natural language which will describe what the system will do without describing it how, called Software Requirement Specification.

• Very critical phase because, a small error hear can result of sever problem in later stages.

**Designing**

• We transform the requirements into a structure that is suitable for implementation of the code in a specific programming language.

• Overall architecture and the algorithmic strategy are chosen (Coupling and cohesion).

• Lastly will prepare a document called SDD (software design description), which will describe how the system will perform functionality.

**Coding**

• Goal of coding is to translate the design of the system into a code of programming language.

• It affects both testing and maintenance, so also critical feature.

• We will be discussing some guidelines for how to write maintainable and readable code.

**Testing**

• Because of human errors there will be a bug or fault in the code and if that bug/fault is executed it become a failure.

• Software testing is a process of executing a program with the intention of finding bugs or fault in the code.

**Implementation**

• Software is installed on the user site and training of the use and h/w requirement check is done.

**Maintenance**

• Any change made in the software after its official release is called maintenance. It could be because of various reasons.

• Adaptive – same functionality in new environment(like should run in IOS)

• Corrective – not meeting the requirement fixing it.

• Perfective – Adding more functionality, improving scalability.

**SDLC**

• Software development organization follows some process when developing a software product, in mature organization this is well defined and managed. In SDLC we develop software in a systematic and disciplined manner.

• SDLC will define entry and exit for every stage. It makes assessment possible, time prediction, cost prediction, scheduling to identify faults in early is possible.

• Selection of a correct development model play an important role in cost, quality, overall success of the project.

Here's an overview of the typical phases in the SDLC:

1. **Planning**: In this phase, project stakeholders define the scope, objectives, requirements, and constraints of the project. This involves identifying the needs of the end-users, setting project goals, estimating resources, and creating a project plan.

2. **Analysis**: During the analysis phase, the development team gathers detailed requirements by studying the needs of end-users, business processes, and existing systems. This phase involves activities such as interviews, surveys, workshops, and documentation review.

3. **Design**: In this phase, the system architecture, database design, user interface, and software components are planned and documented. Designers create blueprints, diagrams, and prototypes to visualize the system's structure and functionality.

4. **Implementation**: The implementation phase involves writing code according to the specifications outlined in the design phase. Developers build, test, and integrate software components, ensuring that the system meets the defined requirements and quality standards.

5. **Testing**: Testing is conducted to identify and fix defects in the software. Different types of testing, such as unit testing, integration testing, system testing, and acceptance testing, are performed to validate the functionality, performance, and usability of the software.

6. **Deployment**: Once the software has been thoroughly tested and approved, it is deployed to the production environment. This involves installing the software on servers, configuring settings, and performing any necessary data migration.

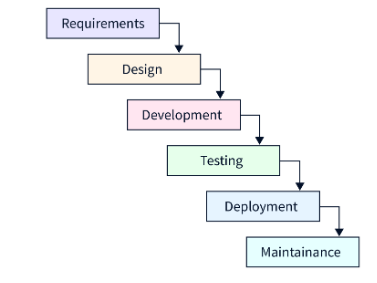
7. **Maintenance**: The maintenance phase involves making updates, enhancements, and fixes to the software to address issues discovered after deployment and to meet changing business needs. Maintenance activities may include bug fixes, performance optimization, security updates, and feature additions.

These phases are typically iterative and may overlap, especially in agile methodologies where development occurs in short cycles. The SDLC provides a structured approach to software development, ensuring that projects are completed on time, within budget, and with high quality.

**Examples of Software Processes**

Software processes, often referred to as Software Development Life Cycle (SDLC) models, provide a structured approach to design, develop, test, and maintain software applications. These processes help teams manage and control the various stages of software development. Here are some commonly used software processes:

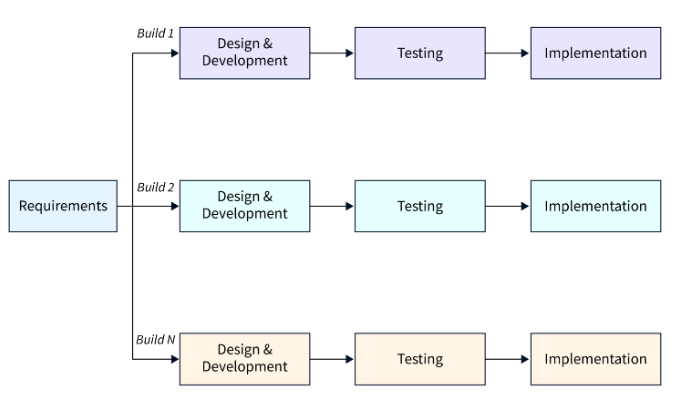
**1. Waterfall Model(Linear Sequential Development model):**



**Each phase must be completed before the next phase can begin. Output of one phase will be input of next phase**.

* **Phases:** Requirement gathering → System design → Implementation → Integration → Testing → Deployment → Maintenance.
* **Advantages:** Clear structure, well-defined stages, easy to manage.
* **Disadvantages:** Difficult to make changes once a phase is complete, not suitable for complex(big) or evolving projects.

**2. Incremental Model:**



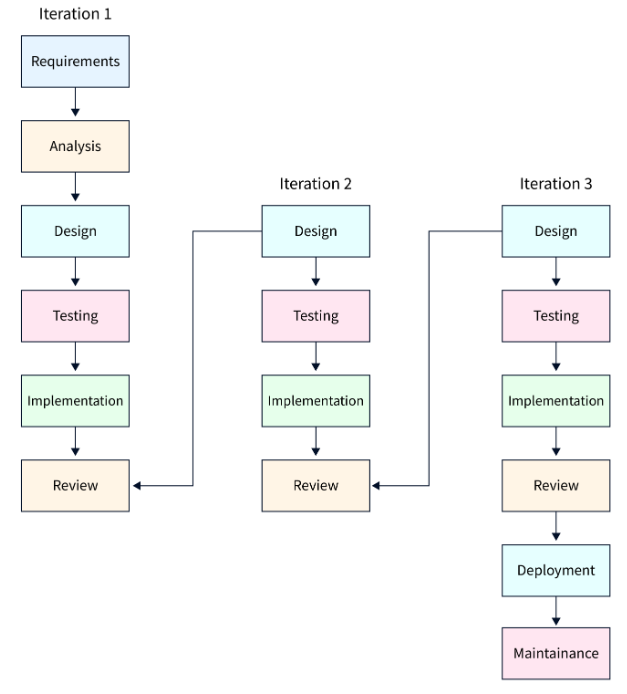
**Develop the project, module by module based on the priority and then deliver to the client(customer) after the completion of each module. This process continues until all the modules of the project(as per the SRS) are completed.**

**Example: We are developing a college management system in which Teacher, Student and etc modules are there. Out of these more priority is for teacher module, which again consists of sub modules such as attendance, feedback and test.**

**Each sub module based on the priority will follow the SDLC process and gets developed and delivered to customer.**

* **Phases:** Same as the Waterfall model but repeated incrementally.
* **Advantages:** Early partial deployment of the software, better feedback and adjustment to requirements.
* **Disadvantages:** Needs clear and stable requirements for the best outcomes.

**3. Iterative Model:**



**> SDLC Cycle start.**

**• Start with some Requirements & Analysis then complete the development of the software.**

**• After the development of the first version of software, in review phase the software is reviewed,**

**• If any changes, new version is created by following the sdlc stages.**

**• Again If any changes, new version is created this is continued until all the requirements of a module is completed.**

**• This way of development is called as Iteration.**

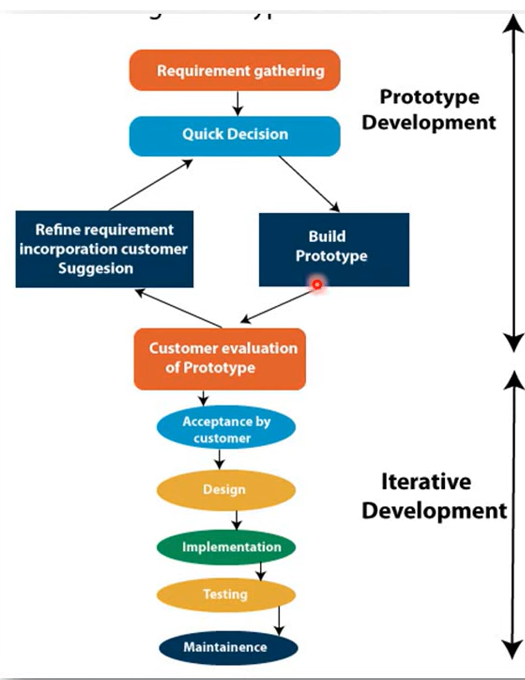
**• After finalized, Deploy the product.**

**• SDLC Cycle completed.**

**Example:**

* **Phases:** Initial implementation → Iterative development → Iterative design and implementation.
* **Advantages:** Allows refinement through iterations, more flexibility in design and development.
* **Disadvantages:** Can be difficult to manage if iterations are not well-defined.

**4. Prototype model**

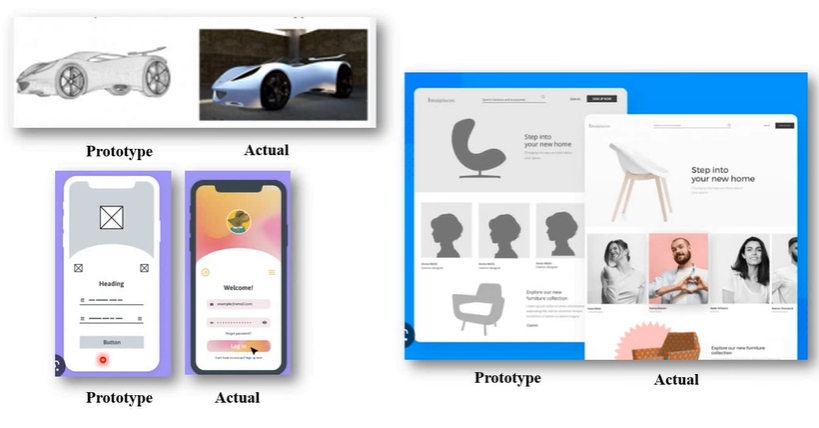


**• Prototype is not a complete product it is just functional replica or toy implementation of any idea, software or system.**

**• It is applied when customers do not know the exact project requirements.**

**• It is build before actual development of software.**

**• It is an iterative, trial and error method which takes place between developer and client.**



**Advantages:**

1. Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.

2. Good where requirement are changing.

3. Customers are actively involved in development.

4. Prototypes can be changed and even discarded.

5. Errors can be detected much earlier as the system is made side by side.

6. Flexibility in design.

**Disadvantages:**

1. The client involvement is more and it is not always considered by the developer.

2. It is a slow process because it takes more time for development.

3. Many changes can disturb the rhythm of the development team.

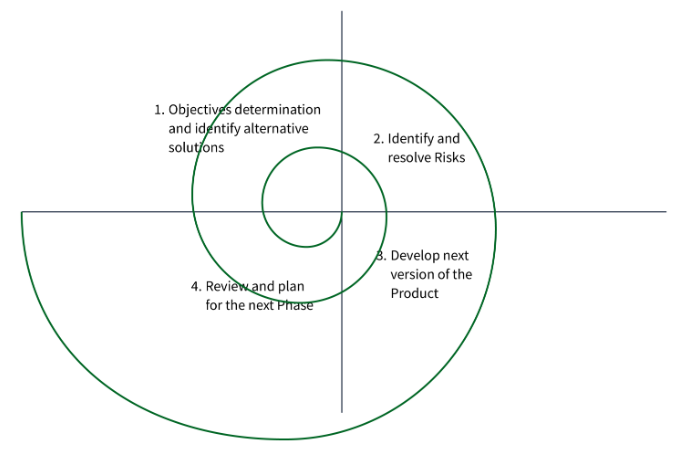
4. Documentation is poor as the requirements change frequently.

5. It is a thrown away prototype when the users are confused with it.

6. Costly with respect to time & money.

7. Prototyping tools are expensive.

**5. Spiral Model(Meta Model):**



• **Spiral Model is a risk-driven software development process model.**

**• It is a combination of Waterfall model(Step by step development approach), Iterative model(Customer feedback taken approach) & Prototyping model(first develop prototype then develop actual product).**

**• Software is developed in a series of incremental releases as per each spiral.**

**• Example: Microsoft, OS Versions, Gaming industry ex.**

**1. Planning: (Requirement Gathering & Analysis)**

**• Communication between customers & project head.**

**• Collect all the requirements from customers.**

**• Analyses estimated cost, schedule & required recourses.**

**2. Risk Analysis:**

**• Identification of all the potential risks.**

**• Risk mitigation strategy is planned for solving risks.**

**• Design a prototype of model.**

**3. Engineering & Execution:**

**• Actual development start.**

**• Designer design the product as per final prototype.**

**• Developer perform actual coding or implementation.**

**• Tester perform all testing methods.**

**• Deploy or Release product to the customer environment.**

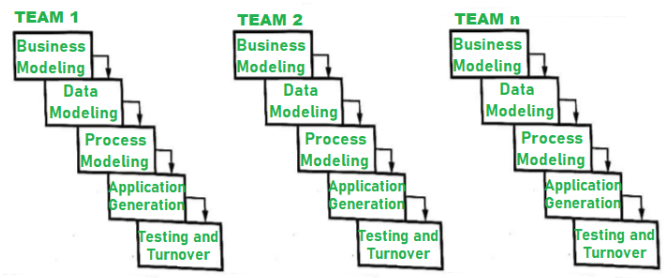
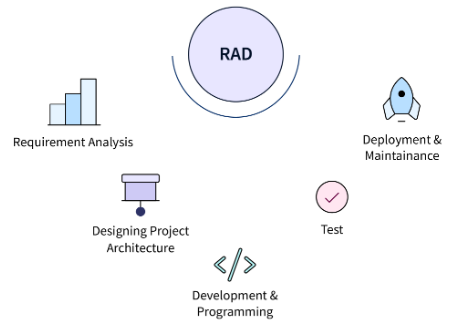
**4. Evaluation:**

**• Take a feedback from customers.**

**• If customer want any changes, goes to next planning OR next spiral iteration.**

* **Phases:** Planning → Risk analysis → Engineering → Evaluation.
* **Advantages:** Strong emphasis on risk management, iterative approach.
* **Disadvantages:** Can be complex, requires expertise in risk management.

**6. RAD (Rapid Application Development):**



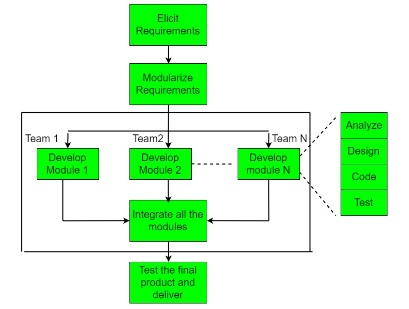
**The RAD process model enables a development team, to create a fully functional system with in very short time**

**periods it allows multiple s/w team work in parallel on different functions. The major goals of RAD is reduces the communication gap between the customer and developer.**

**objectives:-**

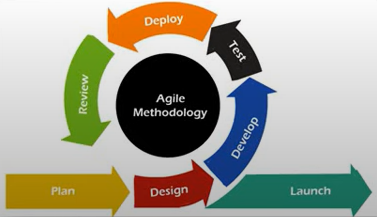
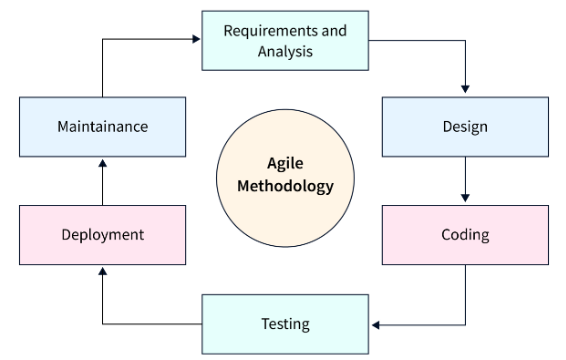
**(a) s/w development in very short time period**

**(b) Guides the s/w engineer through a set of frame work activities.**



* **Phases: Planning → User design → Construction → Cutover.**
* **Advantages: Faster development, high user involvement.**
* **Disadvantages: Requires highly skilled developers, not suitable for larger projects.**

**7. Agile Model:**



**• Mostly used model in todays digital era.**

**• Agile means "The ability to respond to changes from requirements, technology & people".**

**• It follows both incremental and iterative process of software development.**

**Traditional VS Agile Model Working with Example**

**For Example:**

**“Instagram Social Application” , Requirements are:**

**1. Follow - Unfollow Option**

**2. Edit profile**

**3. Search**

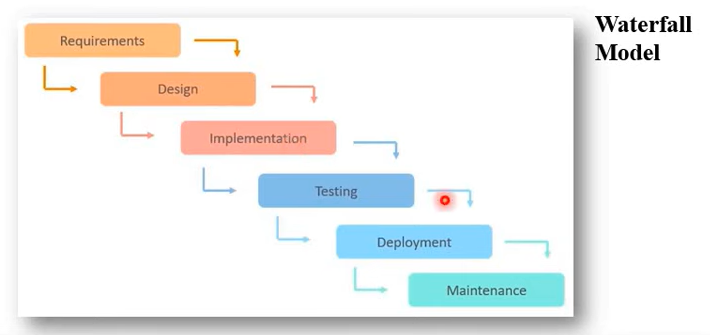
**4. Messaging**

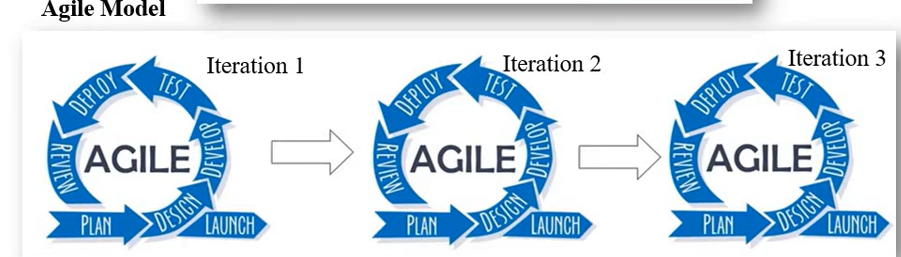
**5. Post photos**

**6. Upload story**

**7. To make reels**

**8. Go Live**





**The above example (Instagram Social Application) contains 8 requirements and the application looks big enough to implement. So we will compare the implementation in traditional software process model with Advanced process model.**

**Observations:**

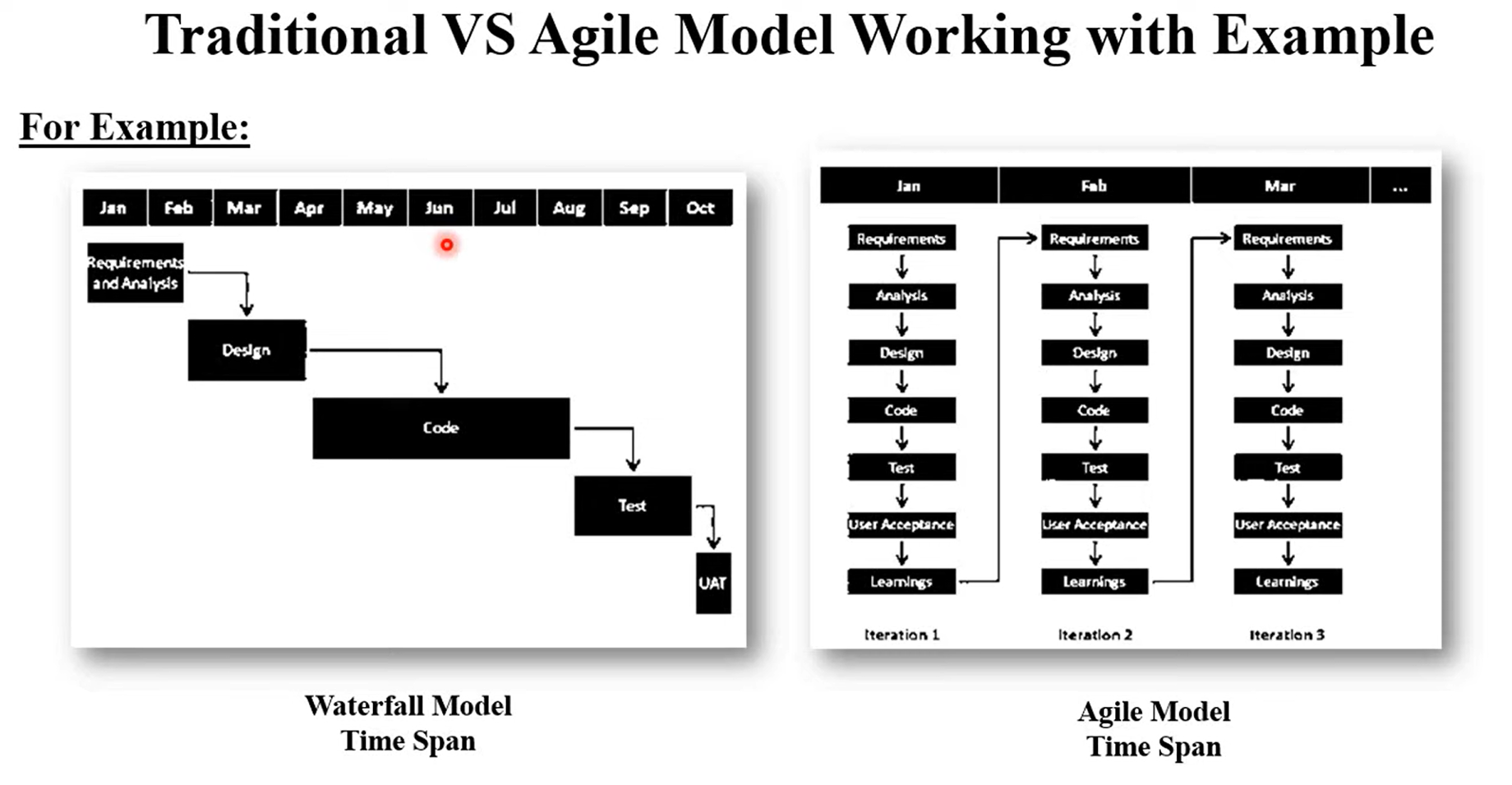
**If we develop the Instagram social application using waterfall model then all the 8 requirements should be elicited and analyzed at a time in Requirements stage and produce SRS document as output to Design phase, in this phase system design, database design & Ui design must be completed to proceed for next phase i.e. implementation or develop phase in this phase after complete code is developed then only it will be proceeded for next phase i.e. Testing phase, after the entire code is tested then the tested code which is free form bugs is delivered to client in deployment phase. Once the application is deployed then the application is proceeded to maintenance phase.**

**Waterfall model is linear fashioned approach and time consuming. So the latest model in the market is Agile Model (respond to changes from requirements, technology and people).**

**Agile model is the combination of incremental and iterative process models.**

**Lets take the same example “Instagram Social Application” in this first customer gives priority to all the modules. According to the priority the modules will be analyzed and an SRS document will be sent to the design phase where system, UI, and Database design takes place and sent to development phase where code is developed according to the designs. Then the code is sent to testers for testing, after successfully completion of testing then customer will review the software product that was developed and tested and customer gives feedback according to the feedback the same process continues by following entire sdlc phases and again software product is sent for customer feedback this process continues until no feedback is received from customer.**

**And then remaining modules are implemented according to the priority followed by the SDLC phases with associated increments and iterations, followed as usual for development of the Final software product/service.**



* **Phases: Planning → Design → Development → Testing → Review → Deploy.**
* **Advantages: large project size, High flexibility, faster delivery of small features, better collaboration and communication.**
* **Disadvantages: Less emphasis on documentation, needs experienced team members.**

**Comparing plan-driven vs. Agile methodologies**

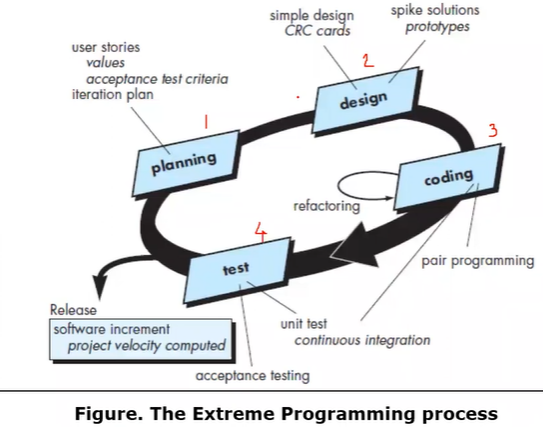
Plan-driven and Agile methodologies represent two contrasting approaches to project management and software development. Here's a comparison of the two:

1. **Approach**:
   * Plan-driven methodologies, such as Waterfall, follow a sequential approach where each phase of the project is planned in advance, and the project progresses through these phases in a linear fashion. Requirements are typically gathered upfront, and changes are costly to implement once the project is underway.
   * Agile methodologies, such as Scrum or Kanban, are iterative and incremental. They emphasize flexibility and adaptability by breaking the project into smaller, manageable increments called iterations or sprints. Requirements and solutions evolve through collaboration between self-organizing, cross-functional teams and stakeholders.
2. **Flexibility**:
   * Plan-driven methodologies are less flexible when it comes to accommodating changes. Once the project plan is established, changes to requirements or scope can be challenging to implement and may require significant effort to revise.
   * Agile methodologies are highly flexible and welcome changes even late in the development process. Changes are expected and are often incorporated in subsequent iterations, allowing for quicker adaptation to evolving requirements or market conditions.
3. **Delivery**:
   * Plan-driven methodologies often deliver the entire product at once, typically after a long development cycle. This can lead to longer lead times before stakeholders see any tangible results.
   * Agile methodologies deliver increments of the product at regular intervals, allowing stakeholders to provide feedback early and frequently. This incremental delivery enables faster time-to-market and allows for more immediate validation of the product's direction.
4. **Risk Management**:
   * Plan-driven methodologies attempt to mitigate risks upfront through extensive planning and documentation. However, there is a risk of missing critical requirements or changes that emerge later in the project.
   * Agile methodologies manage risks through frequent inspection and adaptation. By delivering working software iteratively, risks are identified and addressed early, reducing the likelihood of major issues later in the project lifecycle.
5. **Customer Involvement**:
   * Plan-driven methodologies often involve less customer involvement during the development process, with requirements typically being gathered upfront and signed off before development begins.
   * Agile methodologies prioritize customer collaboration throughout the project. Customers are frequently engaged to provide feedback on incremental deliveries, ensuring that the final product meets their needs and expectations.
6. **Documentation**:
   * Plan-driven methodologies typically require extensive documentation, including detailed project plans, requirements specifications, design documents, and test plans.
   * Agile methodologies prioritize working software over comprehensive documentation, although sufficient documentation is still produced to support development and maintenance efforts.

Both methodologies have their strengths and weaknesses, and the choice between them depends on factors such as project size, complexity, and the level of uncertainty in requirements. Many organizations today adopt a hybrid approach, blending elements of both plan-driven and Agile methodologies to suit their specific needs.

**Extreme Programming Model**

Extreme Programming uses an **object-oriented approach** as its preferred development paradigm(model) and mixture of set of rules and practices that occurs within the context of four framework activities: planning, design, coding, and testing (Umbrella Activities) .



**Planning:** Gathering Requirement from customers in the form of user stories. Value - What sort of data used in application. Set the criteria for **user acceptance testing**. It is an iterative process i.e. from the reviews we perform next increment and follows till no reviews received from customer.

**Design:** CRC-Class relationship collaboration, identify and organize the Object oriented classes relevant to software increment.

Prototypes: Class diagrams, activity diagram, sequence diagram etc (Based on requirement we follow the diagrams) (Attributes, parameters, methods class members etc)

**Coding:** Upon clear understanding of CRC we can developthe code.

Pair programming: Working in team- 2 heads(pairs) involved to solve a task (based on user story)

Continuous Integration: any changes in the code should be integrated with actual code.

Refactor: Altering the code without effecting the actual business logic (behavior of the system).

**Test:**

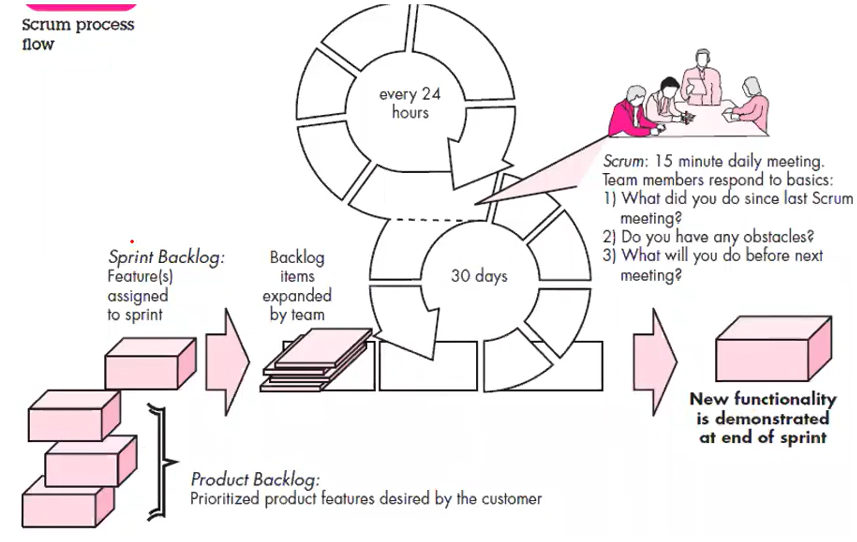
Unit Test: unit means small and for every sub module we need to perform test (meets with user story) is known as unit testing.

Acceptance Testing: weather the code developed meets the user story or not.

**Scrum Process Model**

Meets with the agile manifesto

Framework activities are Requirements, analysis, design, evolution, and delivery.



Sprint backlog(pending) is all about features. i.e. discuss previous day features(Tasks) done and what are pending and add additional features(tasks) to be completed.

Features are prioritized according to customer requirement and then these features are implemented accordingly.

They will plan to implement the prioritized features within 30 days.

As each day in 30 days sprint is 24 hours they start a scrum.

Scrum: In a day, daily 15 minutes of meeting is conducted and they discuss about

1. What did you do since last Scrum meeting?
2. Do you have any obstacles? (any problems)
3. What will you do before next meeting? (by tomorrow what will you do)

**Sprint** means develop and delivers one complete feature within 1 to 30 days.

**Transitioning to Agile processes like XP and Scrum**

**Transitioning to Agile processes like Extreme Programming (XP) and Scrum involves a significant shift in mindset, practices, and culture within an organization. Here are some steps to consider for a successful transition:**

**Educate and Train:** Ensure that everyone involved understands the principles and practices of Agile methodologies like XP and Scrum. Offer training sessions, workshops, and resources to familiarize team members with the new way of working.

**Start with Pilot Projects**: Begin the transition by selecting a few small, low-risk projects or teams to pilot Agile practices. This allows for experimentation and learning without disrupting the entire organization at once.

**Form Cross-Functional Teams:** Organize teams that are cross-functional, self-organizing, and empowered to make decisions. This includes representatives from different disciplines such as development, testing, design, and product management.

**Adopt Agile Practices:** Introduce specific Agile practices such as:

* **Scrum:** Implement Scrum ceremonies like sprint planning, daily stand-ups, sprint reviews, and retrospectives.
* **Extreme Programming (XP):** Incorporate XP practices such as test-driven development (TDD), pair programming, continuous integration, and collective code ownership.

**Establish Iterative Delivery:** Break down the project into smaller increments or iterations (sprints in Scrum) to deliver value early and often. Emphasize working software as the primary measure of progress.

**Encourage Collaboration and Communication:** Foster a culture of collaboration, transparency, and open communication within the team and with stakeholders. Encourage face-to-face interactions and regular feedback loops.

**Embrace Continuous Improvement:** Encourage teams to reflect on their processes and performance regularly through retrospectives. Identify areas for improvement and experiment with new practices to continually refine and enhance the Agile implementation.

**Provide Support and Resources:** Offer ongoing support, coaching, and resources to help teams overcome challenges and adopt Agile practices effectively. This may include dedicated Agile coaches, tools, and infrastructure.

**Address Organizational Challenges:** Be prepared to address organizational barriers and resistance to change. This may involve restructuring, revising policies and procedures, and aligning incentives to support Agile values and principles.

**Measure Progress and Success:** Define key metrics and indicators to track the progress and success of the Agile transition. This may include metrics related to team productivity, quality, customer satisfaction, and business value delivered.

**Celebrate Achievements:** Recognize and celebrate milestones and achievements along the Agile journey to maintain momentum and motivation within the organization.

Transitioning to Agile methodologies like XP and Scrum is not a one-time event but an ongoing process of continuous improvement and adaptation. It requires commitment, patience, and perseverance from everyone involved, from leadership to individual team members.

**Requirement Analysis**

The main purpose of the requirements analysis activity is to analyse the gathered requirements to remove all **ambiguities, incompleteness, and inconsistencies from the gathered customer requirements** and to obtain a clear understanding of the software to be developed.

During requirements analysis, the analyst needs to identify and resolve three main types of problems in the requirements:

• Anomaly

• Inconsistency

• Incompleteness

**Anomaly** means **ambiguity** in requirement which leads to the development of an incorrect system.

E.g.

While gathering the requirements for a **process control application**, the following requirement was expressed by a certain stakeholder: When the temperature becomes high, the heater should be switched off. Please note that words such as “high”, “low”, “goo d”, “bad” etc. are indications of ambiguous requirements as these lack quantification and can be subjectively interpreted. **If the threshold above which the temperature can be considered to be high is not specified, then it can be interpreted differently by different developers.**

**During the final grade computation, if any student scores a sufficiently low grade in a semester, then his parents would need to be informed.**

**Inconsistency**: Two requirements are said to be inconsistent, if one of the requirements contradicts the other

E.g.

Consider the following two requirements that were collected from two different stakeholders in a **process control application development project**.

Req1: The furnace should be switched-off when the temperature of the furnace rises above 500 C.

Req2: When the temperature of the furnace rises above 500 C, the water shower should be switched-on and the **furnace should remain on.**

The requirements expressed by the two stakeholders are clearly inconsistent

**Incompleteness:** An incomplete set of requirements is one in which some requirements have been overlooked.

E.g.

one of the clerks expressed the following—**If a student secures a** **grade point average (GPA) of less than 6, then the parents of the student must be intimated about the regrettable performance through a (postal) letter as well as through e-mail.** However, on an **examination of all requirements**, it was **found** that there is **no provision by which either the postal or e-mail address of the parents of the students can be entered into the system**. The feature that would allow entering the e-mail ids and postal addresses of the parents of the students was **missing**, thereby making the requirements incomplete.

**Use Case Approach**

The use cases represent the different ways in which a system can be used by the users.

A **use case** can be viewed as a **set of related scenarios tied together by a common goal.**

The **main line sequence** and each of the variations are **called scenarios**. Each scenario is a single path of user events and system activity.

The **main line sequence** represents the **interactions between a user and the system.**

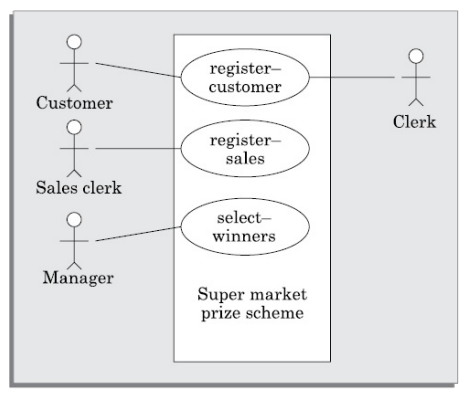
For example, the mainline sequence of **the withdraw cash by a bank ATM** would be—the user inserts the ATM card, enters password, selects the amount withdraw option, enters the amount to be withdrawn, completes the transaction, and collects the amount.

**A use case model can be documented by drawing a use case diagram** and writing an accompanying text elaborating the drawing. In the use case diagram, each **use case is represented by an ellipse with the name of the use case written inside the ellipse.** **All the ellipses (i.e. use cases) of a system are enclosed within a rectangle which represents the system boundary**. The **name of the system** being **modeled** (e.g., library information system ) **appears** **inside the rectangle**.

The **different users of the system are represented by using stick person icons**. Each **stick person icon** is **referred** to as an **actor**.

An actor is a role played by a user with respect to the system use. It is possible that the same user may play the role of multiple actors. An actor can participate in one or more use cases. The line connecting an actor and the use case is called the communication relationship

Example: The use case diagram of the **“Super market prize scheme”**



**U1: register-customer: Using this use case, the customer can register himself by providing the necessary details.**

Scenario 1: Mainline sequence

1. Customer: select register customer option

2 . System: display prompt to enter name, address, and telephone number.

3. Customer: enter the necessary values

4: System: display the generated id and the message that the customer has successfully been registered.

Scenario 2: At step 4 of mainline sequence

4 : System: displays the message that the customer has already registered.

Scenario 3: At step 4 of mainline sequence

4 : System: displays message that some input information have not been entered. The system displays a prompt to

enter the missing values.

**U2: register-sales: Using this use case, the clerk can register the details of the purchase made by a customer.**

Scenario 1: Mainline sequence

1. Clerk: selects the register sales option.

2. System: displays prompt to enter the purchase details and the id of the customer.

3. Clerk: enters the required details.

4 : System: displays a message of having successfully registered the sale.

**U3: select-winners. Using this use case, the manager can generate the winner list.**

Scenario 2: Mainline sequence

1. Manager: selects the select-winner option.

2 . System: displays the gold coin and the surprise gift winner list.

**Identifying use cases**

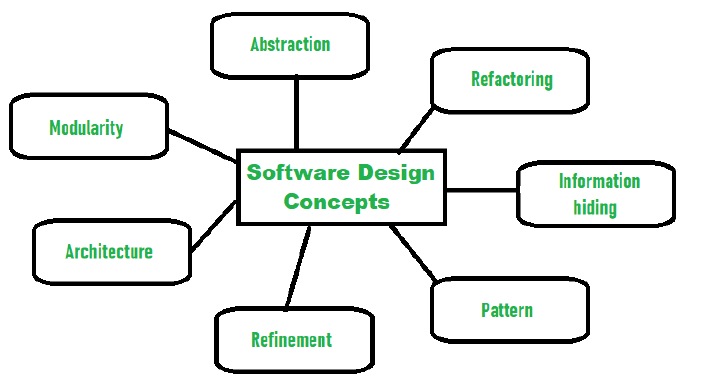
Identification of the use cases involves **brain storming and reviewing the SRS document**. Typically, the high-level **requirements** specified in the SRS document correspond to the **use cases**. In the **absence** of a **wellformulated SRS document**, a popular **method of identifying the use cases is actor-based**. This involves first identifying the different types of actors and their usage of the system. Subsequently, for each actor the different functions that they might initiate or participate are identified. For example, for a Library Automation System, the categories of users can be members, librarian, and the accountant. Each user typically focuses on a set of functionalities. For example, the member typically concerns himself with book issue, return, and renewal aspects. The librarian concerns himself with creation and deletion of the member and book records. The accountant concerns itself with the amount collected from membership fees and the expenses aspects.

**Design concepts**

The software design concept **means** the **principle behind the design**.

It describes how you plan to **solve the problem of designing software**. The software design concept provides a supporting and essential structure or model for developing the right software.

**There are many concepts of software design:**



**Abstraction: (hide irrelevant data)**

o Abstraction means to **hide the details to reduce complexity and increases efficiency or quality.**

o Different levels of abstraction are necessary and musts be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution.

o The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

**Refinement:**

o It is a top-down design strategy.

o Refinement is a **process of elaboration**. We begin with a statement of function (or description of information) that is defined at a high level of abstraction **i.e. the statement describes function or information conceptually but provides no information about the internal workings of the function or the internal structure of the information.**

o Refinement causes the designer to elaborate on the original statement, providing more and more detail as each successive refinement (elaboration) occurs.

Abstraction and refinement are complementary concepts.

* **Abstraction** ***enables*** *a* ***designer*** to **specify procedure and data and yet suppress low-level details.**
* **Refinement** ***helps the designer*** to **reveal low-level details as design progresses**.
* Both concepts aid the designer in creating a complete design model as the design evolves.

**Modularity: (subdivide the system)**

o **Modularity** **means** dividing the system or project into smaller parts to reduce the complexity of the system or project.

o Modularity in design means **subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions.**

o Modularity is the single attribute of software that allows a program to be intellectually manageable.

o Monolithic Software 🡪 a large program composed of a single module, cannot be easily grasped by a reader. The number of control paths, span of reference, number of variables and over all complexity would make understanding close to impossible.

o So modularity in design reduces complexity, facilitates change and results in easier implementation by **encouraging parallel development of different parts of a system.**

**Architecture/ Software Architecture:**

**o** Architecture is the **hierarchical structure of program components**, the manner in which these components interact and the structure of data that are used by the components.

o One goal of software design is to derive an architectural rendering of a system. This rendering serves as a framework from which more detailed design activities are conducted.

**Information Hiding:**

o Information hiding is achieved by **designing the modules in a manner that the information gathered or contained in one module is hidden and can’t be accessed by any other modules**.

o The use of information hiding as a design criterion for modular systems provides the greatest benefits when modifications are required during testing and later, during software maintenance. Because most data and procedure are hidden from other parts of the software, inadvertent errors introduced during modification are less likely to propagate to other locations within the software.

**Software Testing**

Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not.

According to ANSI/IEEE 1059 standard, Testing can be defined as:

“A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.”

**Who does testing?**

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in the context of the given requirements. Moreover,

developers also conduct testing which is called Unit Testing.

In most cases, following professionals are involved in testing of a system within their respective capacities:

Software Tester

Software Developer

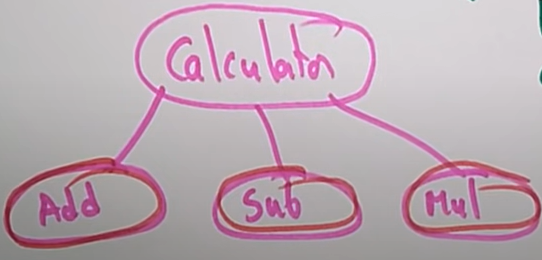
Project Lead/ Manager

End User

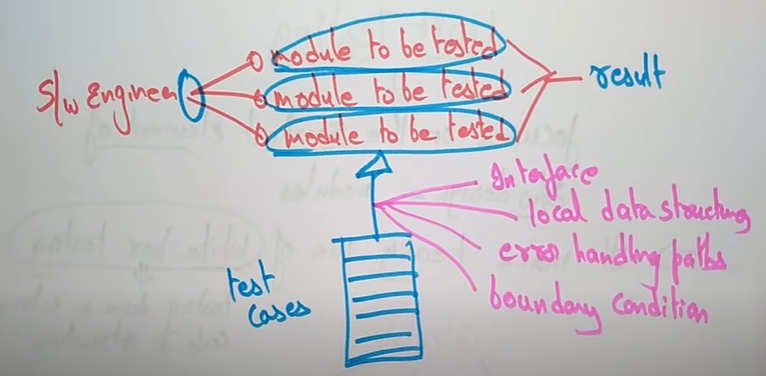
Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer and QA Analyst etc.

**Unit Testing :**

Coding is undertaken once the design phase is complete and the design documents have been successfully reviewed.Focuses on testing the smallest element of software design i.e. modules.



So, testing that is done on every individual module is known as unit testing.



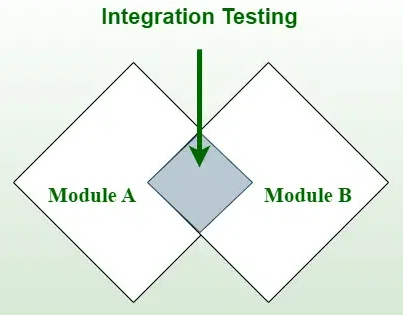
Take the software system and divide the system into modules/sub-modules then prepare test cases that will be tested on each module/sub-module. Test cases can be an interfaces, a local data structures, an error handling paths and a boundary condition.

**Integration Testing**

Purpose: To expose faults in the interaction between integrated units.

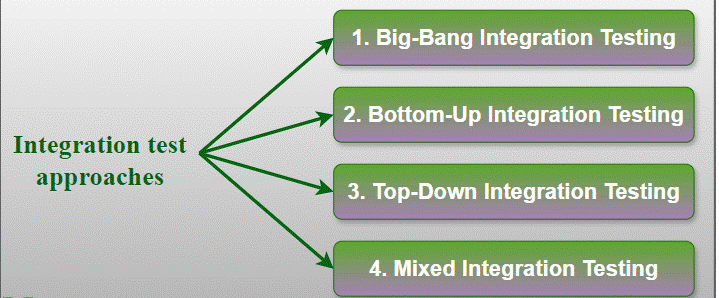
i.e. it is a process of testing the interface between two software units or modules.

While we integrate two or more modules which are having relationship with each other is known as integration testing.



**Integration test approaches**

There are four types of integration testing approaches. Those approaches are the following:



**1. Big-Bang Integration Testing**

Combining all the modules once and verifying the functionality after completion of individual module testing.

**i.e. combining all the modules after the completion of unit testing into a single module and now perform testing on the single module this is known as Big-bang integration testing.**

Single Module

**Perform Testing on the integrated module**

**Unit Testing completed Unit Testing completed Unit Testing completed**

**2. Top-Down Integration Testing (Approach)**

In this approach testing takes place from top to bottom.

High level modules are tested first and then low level modules are tested and finally integrating the lower level modules to the higher level modules to ensures that the system is working as intended.

Stubs are used as a temporary module if the module is not ready for integration testing.

A

B

G

Unit Testing not done so replaced with dummy modules named as stubs

F

Unit Testing not done so replaced with dummy modules named as stubs

C

E

Unit Testing not done so replaced with dummy modules named as stubs

D

Note: stubs are replaced with actual components once at a time in depth first manner.

**3. Bottom- Up Integration Testing (approach)**

Here testing takes place from bottom to up approach.

Lowest level modules are tested first and then high level modules are tested and finally integrating the high level modules to low level modules to ensure the system is working as intended.

Drivers are used as a temporary module for integration testing

A

B

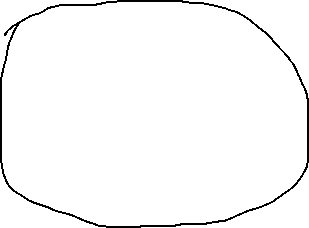
G

Unit Testing not done so replaced with dummy modules named as drivers

F

Unit Testing not done so replaced with dummy modules named as drivers

C



E

D

cluster



Drivers are replaced once unit testing is completed in depth first search order.

Worker modules(100% working modules) are grouped together known as clusters.

**4. Sandwich (Mixed) Integration Testing:**

Combination of Top down and bottom up testing approaches.

In top-down approach, testing can start only after the top-level module have been coded and unit tested. In bottom-up approach, testing can start only after the bottom level modules are ready. stubs and drivers are used  in mixed integration testing.

**Acceptance testing**

It is formal testing according to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users, customers, or other authorized entities to determine whether to accept the system or not.

Acceptance Testing is the last phase of software testing.

**Types of Acceptance Testing**

1. User Acceptance Testing (UAT)
2. Business Acceptance Testing (BAT)
3. Contract Acceptance Testing (CAT)
4. Regulations Acceptance Testing (RAT)
5. Operational Acceptance Testing (OAT)
6. Alpha Testing
7. Beta Testing

**1. User Acceptance Testing (UAT)**

* User acceptance testing is used to determine whether the product is working for the user correctly.
* Specific requirements which are quite often used by the customers are primarily picked for testing purposes. This is also termed as End-User Testing.

**2. Business Acceptance Testing (BAT)**

* BAT is used to determine whether the product meets the business goals and purposes or not.

**3. Contract Acceptance Testing (CAT)**

* Here is a contract termed a Service Level Agreement (SLA), which includes the terms where the payment will be made only if the Product services are in-line with all the requirements, which means the contract is fulfilled.

**4. Regulations Acceptance Testing (RAT)**

* RAT is used to determine whether the product violates the rules and regulations that are defined by the government of the country where it is being released.
* This may be unintentional but will impact negatively on the business. Generally, the product or application that is to be released in the market, has to go under RAT, as different countries or regions have different rules and regulations defined by its governing bodies.
* If any rules and regulations are violated for any country then that country or the specific region then the product will not be released in that country or region.
* If the product is released even though there is a violation then only the vendors of the product will be directly responsible.

**5. Operational Acceptance Testing (OAT)**

* OAT is used to determine the operational readiness of the product and is non-functional testing.
* It mainly includes testing of recovery, compatibility, maintainability, reliability, etc. OAT assures the stability of the product before it is released to production.

**6. Alpha Testing**

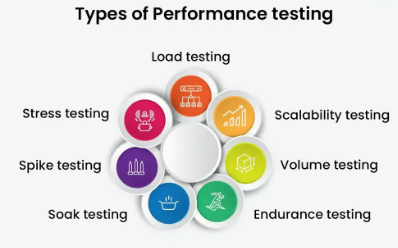
Alpha testing is the first end-to-end testing of a product to ensure it meets the business requirements and functions correctly. It is typically performed by internal employees and conducted in a lab/stage environment.

**7. Beta Testing**

* Beta testing is used to assess the product by exposing it to the real end-users, typically called beta testers in their environment.
* Feedback is collected from the users and the defects are fixed. Also, this helps in enhancing the product to give a rich user experience.

**Performance Testing**

Performance Testing is a type of software testing that ensures software applications perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity, and stability under a particular workload.



**1. Load testing**

Load testing simulates a real-world load on the system to see how it performs under stress. It helps identify bottlenecks and determine the maximum number of users or transactions the system can handle. It checks the product’s ability to perform under anticipated user loads. The objective is to identify performance congestion before the software product is launched in the market.

**2. Stress testing**

Stress testing is a type of **load testing** that tests the system’s ability to handle a high load above normal usage levels. It helps identify the breaking point of the system and any potential issues that may occur under heavy load conditions. It involves testing a product under extreme workloads to see whether it handles high traffic or not. The objective is to identify the breaking point of a software product.

**3. Spike testing**

Spike testing is a type of load testing that tests the system’s ability **to handle sudden spikes in traffic**. It helps identify any issues that may occur when the system is suddenly hit with a high number of requests. It tests the product’s reaction to sudden large spikes in the load generated by users.

**4. Soak testing**

Soak testing is a type of load testing that tests the system’s ability to handle a **sustained load over a prolonged period**. It helps identify any issues that may occur after prolonged usage of the system.

**5. Endurance testing**

Endurance testing is similar to soak testing, but it focuses on **the long-term behavior of the system under a constant load.** It is performed to ensure the software can handle the expected load over a long period.

**6. Volume testing**

In Volume testing, a large number of data is saved in a database and the overall software system’s behavior is observed. The objective is to **check the product’s performance under varying database volumes.**

**7. Scalability testing**

In Scalability testing, **the software application’s effectiveness is determined by scaling up to support an increase in user load**. It helps in planning capacity additions to your software system.