**Mid term**

A . What is a software process? What are the differences between software process and process model? Example

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

Software Processes is a coherent set of activities for specifying, designing, implementing and testing software systems. A software process model is an abstract representation of a process that presents a description of a process from some particular perspective.

 There are many different software processes but all involve:

* Specification – defining what the system should do;
* Design and implementation – defining the organization of the system and implementing the system;
* Validation – checking that it does what the customer wants;
* Evolution – changing the system in response to changing customer needs.

## Software process is set of work that is applied to design or built a software product. The fundamental activities that is common to all software process are :

1. software specification
2. software design and implementation
3. software evaluation

A software process model is the a abstract representation of a software process. It is a structure of a software process present the description of a process . There are several process models are available

1. Waterfall model
2. Generic process model
3. Incremental model
4. Agile process model
5. Prototyping model
6. Spiral model
7. Iterative development model, etc

b. describe waterfall model.what are the advantages and disadvantages

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are −

* **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
* **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
* **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
* **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
* **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
* **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

## Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

* Simple and easy to understand and use
* Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
* Phases are processed and completed one at a time.
* Works well for smaller projects where requirements are very well understood.
* Clearly defined stages.
* Well understood milestones.
* Easy to arrange tasks.
* Process and results are well documented.

## Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

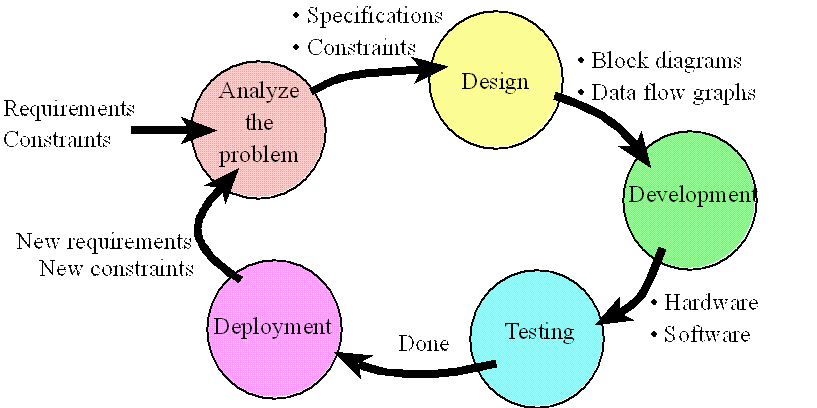
The major disadvantages of the Waterfall Model are as follows −

* No working software is produced until late during the life cycle.
* High amounts of risk and uncertainty.
* Not a good model for complex and object-oriented projects.
* Poor model for long and ongoing projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
* It is difficult to measure progress within stages.
* Cannot accommodate changing requirements.
* Adjusting scope during the life cycle can end a project.
* Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

c. Explain why incremental development is the most effective approach for developing business software systems.draw block diagram of a general model of the software design process

Incremental development is often considered an effective approach for developing business software due to several advantages it offers:

1. Faster time-to-market: Incremental development allows for the delivery of functional software in smaller iterations or increments. This means that business users can start benefiting from the system sooner, even if it is not fully complete. This approach enables faster time-to-market and allows businesses to respond more quickly to market demands and changing requirements.
2. Flexibility and adaptability: Incremental development embraces the concept of iterative and incremental improvements. It allows businesses to adapt and refine the software based on user feedback and changing needs. This flexibility ensures that the final product is better aligned with the evolving requirements of the business.
3. Early and continuous user involvement: With incremental development, users and stakeholders have the opportunity to provide feedback and review working software at regular intervals. This early and continuous involvement helps ensure that the software meets the business needs and reduces the risk of misalignment between the delivered product and user expectations.
4. Risk mitigation: Incremental development mitigates risks associated with large-scale projects. By breaking down the development process into smaller increments, businesses can identify and address potential issues and risks early on. This approach allows for course correction and adjustments, reducing the likelihood of major failures or budget overruns.
5. Enhanced transparency and communication: Incremental development encourages regular communication and collaboration between development teams and business stakeholders. As working software is delivered in iterations, it facilitates ongoing discussions, clarifications, and the resolution of any misunderstandings. This transparent and collaborative approach improves overall project understanding and reduces miscommunication.
6. Cost-effectiveness: Incremental development can be cost-effective as it allows for better resource allocation. Instead of investing a significant amount of resources into a single monolithic project, incremental development allows for smaller, manageable increments with clear goals and deliverables. This phased approach enables businesses to prioritize and allocate resources more efficiently based on immediate needs and value.
7. Quality assurance and testing: Incremental development enables continuous testing and quality assurance efforts throughout the development process. Each increment is subject to testing, ensuring that defects and issues are identified and addressed early on. This leads to a higher quality end product as compared to the traditional "big bang" approach where testing occurs towards the end of the project.



General model of the desigh process

2.5 Design-process.eps

d. describe the agile manifesto .Explain how the principles underlying agile methods lead to the development and deployment of software.

The Agile Manifesto is a set of guiding principles that define the values and principles of agile software development. It was created by a group of software development experts in 2001 and emphasizes flexibility, collaboration, and continuous improvement in software development processes. The Agile Manifesto consists of four core values and twelve principles.

Agile Manifesto Values:

1. Individuals and Interactions over Processes and Tools: This value emphasizes the importance of effective communication, collaboration, and teamwork among individuals involved in the software development process. It recognizes that people and their interactions are the foundation of successful software development, and processes and tools should support their collaboration.
2. Working Software over Comprehensive Documentation: This value emphasizes the delivery of functional software that meets the needs of users. It acknowledges the importance of documentation but prioritizes working software as the primary measure of progress and success. The focus is on delivering value to the customer through tangible, working software.
3. Customer Collaboration over Contract Negotiation: This value emphasizes the active involvement and collaboration of customers or stakeholders throughout the development process. It promotes close collaboration, feedback, and iterative development to ensure the software meets customer needs. The goal is to work closely with customers to understand and respond to their requirements effectively.
4. Responding to Change over Following a Plan: This value recognizes the dynamic nature of software development. It acknowledges that requirements and priorities may change over time, and the development process should be flexible and responsive to accommodate those changes. Agile teams embrace change and adjust plans accordingly to maximize the value delivered.

Agile Manifesto Principles:

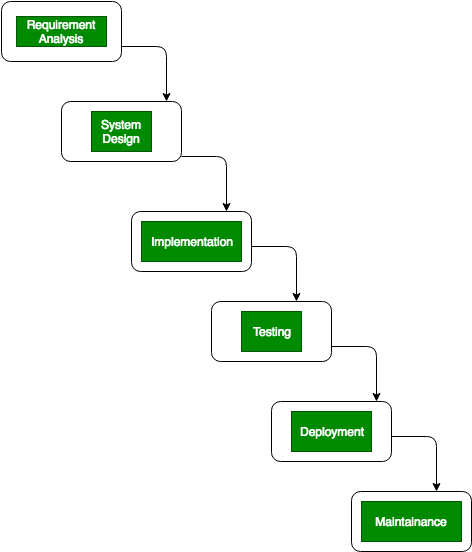
The twelve principles underlying the Agile Manifesto expand on the values and provide specific guidance for agile software development. These principles include:

1. Satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in the development process.
3. Deliver working software frequently, with a preference for shorter timescales.
4. Collaborate closely with customers and stakeholders throughout the project.
5. Build projects around motivated individuals and give them the environment and support they need.
6. Use face-to-face communication as much as possible.
7. Measure progress primarily by working software.
8. Maintain a sustainable pace for development teams.
9. Promote technical excellence and good design.
10. Keep things simple and maximize the amount of work not done.
11. Empower teams to make decisions and self-organize.
12. Regularly reflect on team effectiveness and adjust accordingly.

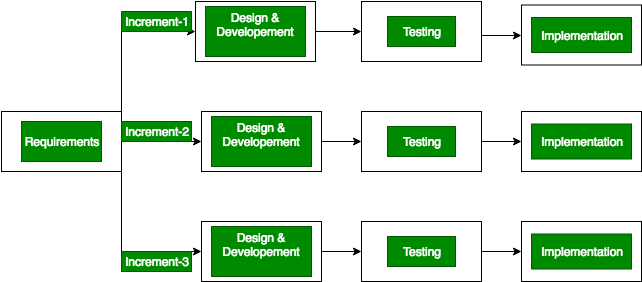
**2021 solution**

1.a. define software engineering.What are the benefits of incremental development model over waterfall model?

 is also known as **Classical/Traditional Model**. Sometimes It is referred as linear-sequential life cycle model because all phases involve in this model completed one by one in linear fashion. In this model, we get software after completion of all coding phase. This model is basically used for small projects. There exist only one cycle in waterfall model.

  
**Figure –** Waterfall Model

While in [Incremental Model](https://www.geeksforgeeks.org/software-engineering-incremental-process-model/) Multiple development cycles take place and these cycles are divided into more smaller modules. Generally a working software in incremental model is produced during first module Each subsequent release of the module adds function to the previous release. In incremental model, process continues till the complete system is achieved.

  
**Figure –** Incremental Model

Here we will see that how waterfall model is different from an incremental model in terms of time, cost, maintenance, documentation etc.

Some differences between them are given below:

| **S. No.** | **Waterfall Model** | **Incremental Model** |
| --- | --- | --- |
| 1. | Need for Detailed Documentation in the waterfall model is Necessary. | The need for Detailed Documentation in the incremental model is Necessary but not too much. |
| 2. | In the waterfall model, early stage planning is necessary. | In an incremental model, early-stage planning is also necessary. |
| 3. | There is a high amount of risk in the waterfall model. | There is a low amount of risk in the incremental model. |
| 4. | There is a long waiting time for running software in the waterfall model. | There is a short waiting time for running software in the incremental model. |
| 5. | The waterfall model can’t handle large projects. | The incremental model also can’t handle large projects. |
| 6. | Flexibility to change in the waterfall model is Difficult. | Flexibility to change in incremental model is Easy. |
| 7. | The cost of the Waterfall model is Low. | The cost of the incremental model is also Low. |
| 8. | Testing is done in the waterfall model after the completion of the coding phase. | Testing is done in the incremental model after every iteration of the phase. |
| 9. | Returning to the previous stage/phase in the waterfall model is not possible. | Returning to the previous stage/phase in the incremental model is possible. |
| 10. | In the waterfall model, a large team is required. | In an incremental model large team is not required. |
| 11. | In the waterfall model overlapping of phases is not possible. | In incremental model overlapping of phases is possible. |
| 12. | There is only one cycle in the waterfall model. | Multiple development cycles take place in the incremental model. |
| 13. | The customer is involved only at the beginning of development. | In incremental model, customer involvement is intermediate. |
| 14. | The linear framework type is used. | Linear with iterative framework type is used. |
| 15. | The customer is having least control over the administrator. | The customer has more control over the administrator in comparison to the waterfall model. |
| 16. | Reusability is the least possible. | Reusability is possible to some extent. |

b. describe the difference between plan driven and agile approaches with a diagram. What are the advantages and drawbacks of agile approaches?

Plan-Driven Approach: The plan-driven approach, often associated with traditional waterfall methodologies, follows a sequential and predictive process. It involves detailed planning, documentation, and a fixed scope of work. Here is a simplified diagram illustrating the plan-driven approach:

\_\_\_\_\_\_\_\_\_\_\_\_\_

| Planning |

|\_\_\_\_\_\_\_\_\_\_\_\_\_|

|

\_\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_

| Execution |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

|

\_\_\_\_\_\_/ \\_\_\_\_\_\_\_

| Testing |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

|

\_\_\_\_\_\_/ \\_\_\_\_\_\_\_

| Deployment |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Agile Approach: The agile approach, on the other hand, follows an iterative and incremental process that allows for flexibility, adaptability, and continuous improvement. Agile methodologies, such as Scrum or Kanban, involve shorter development cycles called iterations or sprints. Here is a simplified diagram illustrating the agile approach:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| Iteration/Sprint 1 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

|

\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_\_

| Iteration/Sprint 2 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

|

\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_\_

| ... |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

|

\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_\_

| Iteration/Sprint N |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Advantages of Agile Approaches:

1. Flexibility: Agile approaches allow for changing requirements and priorities throughout the development process, providing flexibility to adapt to evolving business needs.
2. Customer Collaboration: Regular customer involvement and feedback promote a deeper understanding of their needs, leading to a higher chance of delivering a product that meets their expectations.
3. Early Value Delivery: Incremental development and frequent iterations enable the delivery of functional software early, providing value to customers and stakeholders sooner.
4. Continuous Improvement: Agile methodologies emphasize regular retrospectives and process refinements, enabling continuous improvement in both the development process and the delivered product.
5. Enhanced Team Collaboration: Agile methodologies promote self-organizing, cross-functional teams that collaborate closely, fostering better communication, knowledge sharing, and problem-solving.

Drawbacks of Agile Approaches:

1. Lack of Predictability: Agile approaches are not as predictable as plan-driven methods due to changing requirements and iterative development. This may make it challenging to estimate project timelines and costs accurately.
2. Requires Active Customer Involvement: Agile methodologies rely heavily on customer collaboration, and if customers are not actively engaged or available, it can hinder the effectiveness of the approach.
3. Potential Scope Creep: The flexibility in agile approaches can lead to scope creep, where the project scope expands beyond the original intentions, potentially affecting budget and timelines if not managed effectively.
4. Learning Curve: Agile methodologies may require a learning curve for teams unfamiliar with the approach, which can initially impact productivity until the team becomes proficient.

c. write a short note on Ishikawa diagram

An Ishikawa diagram is a diagram that shows the causes of an event and is often used in manufacturing and product development to outline the different steps in a process, demonstrate where [quality control](https://www.investopedia.com/terms/q/quality-control.asp) issues might arise, and determine which resources are required at specific times.

The Ishikawa diagram was developed by Kaoru Ishikawa during the 1960s as a way of measuring quality control processes in the shipbuilding industry.

### **KEY TAKEAWAYS**

* An Ishikawa diagram is used to show the causal factors that go into some final outcome, often related to a production or design problem.
* They are named after Japanese engineering professor Kaoru Ishikawa in the 1960s, who helped apply them to manufacturing processes.
* Shaped somewhat like a fish, these charts are sometimes called fishbone or "Fishikawa" diagrams.
* Ishikawa diagrams often follow the "Six M's": manpower, machinery, methods, materials, measurement, and mother nature.

2. a. Scrum,JRP,Unit testing

**Scrum** is the type of **Agile framework**. It is a framework within which people can address complex adaptive problem while productivity and creativity of delivering product is at highest possible values. Scrum uses **Iterative process**. **Silent features of Scrum are:**

* Scrum is light-weighted framework
* Scrum emphasizes self-organization
* Scrum is simple to understand
* Scrum framework help the team to work together

JRP, or Joint Requirements Planning, is a collaborative approach used in software engineering to gather and analyze requirements for a software project. It involves bringing together stakeholders, including business analysts, developers, users, and other relevant parties, to work together and define the requirements for the software system.

The JRP process typically includes the following stages:

1. Pre-planning: Identify the purpose and objectives of the JRP, select the appropriate participants, and establish a schedule and agenda for the process.
2. Discovery: Gather information about the existing system, user needs, and other requirements for the new software system.
3. Analysis: Analyze the information collected to identify requirements and prioritize them based on their importance and feasibility.
4. Specification: Define the requirements in detail, including functional and non-functional requirements, as well as any constraints and assumptions.
5. Validation: Review the requirements with stakeholders to ensure they are complete, accurate, and acceptable.
6. Management: Track and manage changes to the requirements throughout the software development lifecycle.

JRP has several benefits for software development projects, including:

1. Collaboration: JRP promotes collaboration and communication among stakeholders, leading to a shared understanding of the requirements and greater buy-in from all parties.
2. Efficiency: JRP allows for a more efficient and streamlined requirements gathering process, reducing the likelihood of missed or misunderstood requirements.
3. Clarity: JRP facilitates the clear and concise documentation of requirements, making them easier to understand and interpret for all parties involved.
4. Flexibility: JRP allows for the rapid adaptation of requirements in response to changing project needs or stakeholder feedback.

**Unit Testing** is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

b. describe factory design pattern with proper examples.

The Factory Design Pattern is a creational design pattern that provides an interface for creating objects without specifying their concrete classes. It encapsulates the object creation logic and allows the client code to use the created objects through a common interface. The Factory Design Pattern promotes loose coupling by separating object creation from the client code.

Here's an example to illustrate the Factory Design Pattern:

Let's say we have an application that deals with different types of documents, such as PDF documents, Word documents, and Excel documents. We want to abstract the document creation process using the Factory Design Pattern.

1. **Product Interface**: First, we define an interface called **Document** that declares the operations common to all types of documents. It could include methods like **open()**, **close()**, or **save()**.

public interface Document {

void open();

void close();

void save();

}

1. **Concrete Product Classes**: Next, we create concrete classes that implement the **Document** interface for each specific document type.

public class PDFDocument implements Document {

// Implement PDF document operations

// ...

}

public class WordDocument implements Document {

// Implement Word document operations

// ...

}

public class ExcelDocument implements Document {

// Implement Excel document operations

// ...

}

1. **Factory Class**: We create a factory class called **DocumentFactory** that encapsulates the creation logic. It provides a method, such as **createDocument()**, which returns a **Document** object based on the specified type.

public class DocumentFactory {

public Document createDocument(String type) {

if (type.equals("PDF")) {

return new PDFDocument();

} else if (type.equals("Word")) {

return new WordDocument();

} else if (type.equals("Excel")) {

return new ExcelDocument();

}

throw new IllegalArgumentException("Invalid document type: " + type);

}

}

1. **Client Code**: Finally, the client code can use the **DocumentFactory** to create documents without knowing the concrete document classes.

public class Client {

public static void main(String[] args) {

DocumentFactory documentFactory = new DocumentFactory();

Document pdfDocument = documentFactory.createDocument("PDF");

pdfDocument.open();

// Use PDF document

Document wordDocument = documentFactory.createDocument("Word");

wordDocument.open();

// Use Word document

Document excelDocument = documentFactory.createDocument("Excel");

excelDocument.open();

// Use Excel document

}

}

In this example, the **DocumentFactory** acts as the factory, and it creates instances of **PDFDocument**, **WordDocument**, or **ExcelDocument** based on the type specified by the client code. The client code doesn't need to know the concrete classes; it can work with the created **Document** objects through the common **Document** interface.

c. what are the attributes of a good software?

A good software typically exhibits certain attributes that contribute to its overall quality, usability, maintainability, and performance. Here are some key attributes of good software:

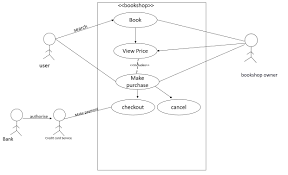
1. **Functionality**: Good software fulfills its intended purpose and meets the specified requirements. It performs the tasks it was designed for accurately and efficiently, providing the expected functionality to its users.
2. **Reliability**: Reliable software operates consistently and predictably, producing correct results even in different environments and under varying conditions. It minimizes errors, exceptions, crashes, and unexpected behaviors.
3. **Usability**: Usability refers to how easily and efficiently users can interact with the software. Good software has an intuitive and user-friendly interface, clear instructions, and well-designed features. It focuses on the user experience and minimizes the learning curve.
4. **Performance**: Performance relates to the speed, responsiveness, and efficiency of the software. Good software is optimized to execute tasks quickly, handle large data volumes, and utilize system resources effectively. It minimizes latency, processing time, and memory usage.
5. **Scalability**: Scalable software is designed to handle increased workloads and accommodate growing user bases or data volumes. It can scale horizontally (by adding more servers) or vertically (by utilizing more resources on a single server) to maintain performance as demand increases.
6. **Security**: Security is crucial for protecting the software and its users' data from unauthorized access, breaches, or malicious activities. Good software implements appropriate security measures, encryption, access controls, and follows secure coding practices.
7. **Maintainability**: Maintainable software is easy to modify, enhance, and fix. It follows clean code principles, modular design, and separation of concerns. It uses meaningful variable names, comments, and documentation to facilitate understanding and future maintenance by developers.
8. **Testability**: Testable software is designed to facilitate effective testing. It has a modular and loosely coupled architecture, provides interfaces for testing or mocking, and includes automated test suites. Good software aims for high test coverage to ensure correctness and catch regressions.
9. **Portability**: Portable software can run on different platforms, operating systems, or environments without requiring major modifications. It is designed to be independent of specific hardware or software dependencies, facilitating deployment and use across various systems.
10. **Compliance**: Compliance refers to adherence to legal, industry, or regulatory standards applicable to the software. Good software follows applicable guidelines, regulations, and standards, such as data protection laws, accessibility standards, or coding conventions.

3.a. briefly describe the fact finding methods.

Fact-finding methods are techniques used in the early stages of system development or problem-solving to gather information and understand the requirements, constraints, and objectives of a project. These methods involve interacting with stakeholders, observing processes, and analyzing existing documentation to collect relevant data. Here are some commonly used fact-finding methods:

1. **Interviews**: Interviews involve direct one-on-one or group discussions with stakeholders, subject matter experts, and end-users. The purpose is to gather information about their perspectives, requirements, expectations, and concerns. Structured or semi-structured interviews help elicit specific information, while open-ended interviews encourage a broader discussion.
2. **Questionnaires**: Questionnaires are written surveys distributed to stakeholders or end-users. They allow for collecting information from a larger group of people. Questionnaires can be closed-ended (with predefined answer choices) or open-ended (allowing respondents to provide detailed responses). They provide quantitative or qualitative data that can be analyzed for patterns and insights.
3. **Observation**: Observation involves watching and studying people, processes, or activities in their natural environment. By directly observing how tasks are performed, interactions occur, or workflows are executed, valuable insights can be gained. Observations can be passive (non-participatory) or active (engaging with participants), depending on the desired level of involvement.
4. **Document Analysis**: Document analysis involves reviewing existing documents such as reports, manuals, requirements specifications, policies, or user guides. By examining these materials, valuable information about the current system, business processes, rules, and constraints can be extracted. It helps in understanding the existing context and identifying areas for improvement.
5. **Prototyping**: Prototyping involves building partial or simplified versions of the system to gather feedback and refine requirements. By creating prototypes, stakeholders can visualize and interact with a representation of the intended system, providing feedback and clarifying their needs. Prototyping helps validate requirements and refine the design early in the development process.
6. **Workshops and Focus Groups**: Workshops and focus groups bring together stakeholders, users, and subject matter experts in a facilitated setting. These interactive sessions encourage collaboration, brainstorming, and discussion to generate ideas, explore requirements, and resolve conflicts. They are particularly useful for gathering diverse perspectives and reaching consensus on key decisions.
7. **Surveys**: Surveys involve collecting data from a large number of participants to gain insights into their preferences, opinions, or experiences. Surveys can be conducted online, via email, or through web forms. They provide statistical data that can be analyzed to identify trends, patterns, or correlations.
8. **Benchmarking**: Benchmarking involves comparing a system or process to best practices or similar solutions in the industry. It helps identify performance gaps, areas for improvement, and potential solutions. Benchmarking can be done through research, interviews, or direct comparisons with competitor systems.

3.b . A book is written by an author, published by a publisher ,sold by a bookstore, and read by a reader .Moreover, for a reader to read a book, she must but it from a book store that is selling it .Draw a use case diagram for this scenario, showing relationships between different use cases



+------------+

| Author |

+------------+

| 1

|

v

+------------+

| Write Book |

+------------+

|

| 1

v

+------------+

| Publish |

+------------+

| |

v 1 v 1

+------------+ +-------------+

| Bookstore | | Publisher |

+------------+ +-------------+

| 1 | 1

v v

+------------+ +-------------+

| Sell | | Publish |

+------------+ +-------------+

| 1 |

v |

+------------+ |

|  |
| --- |
|  |

| Read |<----+

Explanation:

1. **Author**: Represents the author of the book who writes the book.
   * Use Case: "Write Book" - The author writes the book.
2. **Publish**: Represents the publishing process that involves the publisher and the book.
   * Use Case: "Publish" - The author submits the book to the publisher, who then publishes it.
3. **Bookstore**: Represents the bookstore that sells books to readers.
   * Use Case: "Sell" - The bookstore sells books to readers.
4. **Publisher**: Represents the publisher who publishes the book written by the author.
   * Use Case: "Publish" - The publisher publishes the book.
5. **Sell**: Represents the process of selling books to readers by the bookstore.
   * Use Case: "Sell" - The bookstore sells the book to the reader.
6. **Read**: Represents the reader's action of reading the book after buying it from the bookstore.
   * Use Case: "Read" - The reader reads the book.

Relationships:

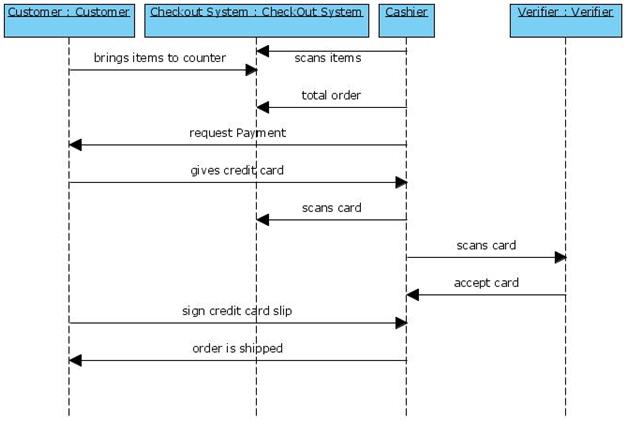
* The **Author** "Write Book" use case is associated with the **Publish** "Publish" use case since the author submits the book to the publisher for publication.
* The **Publish** "Publish" use case is associated with the **Bookstore** "Sell" use case since the publisher's published book is sold by the bookstore.
* The **Bookstore** "Sell" use case is associated with the **Reader** "Read" use case since the reader can read the book after buying it from the bookstore.

3.c. describe multiplicity notations in class diagram.

Multiplicity (**Cardinality**) Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for just one company.

4.a.

to buy a book electronically from chapters .com, a customer needs to select the book from a list provided by chapters ecommerse system, provide credit card information to the system , then the system gets authorization from the bank for the payment , and if positive confirms the sale. the order is then sent to the orders department and when the book becomes available, it is shipped to the customer. also, the order department charges the customers credit card by informing the bank of the amount.draw a sequence diagram that models this process.Make sure to model all relevant actors and the interactions between them .do show explicitly the time interval when different actors actively participate in the process you are modeling.



Customer Chapters.com Bank Orders Department

| | | |

| Selects a book | | |

|----------------->| | |

| | | |

| Provides credit | | |

| card information| | |

|-----------------> | | |

| | | |

| | Sends | |

| | authorization | |

| |--------------->| |

| | | |

| | | Processes |

| | | authorization |

| | |------------------->|

| | | |

| | | Confirms sale |

| | |<-------------------|

| | | |

| | | |

| | | Notifies |

| | | order department |

| | |------------------->|

| | | |

| | | Charges credit |

| | | card for the book |

| | |<-------------------|

| | | |

| | | |

| | | |

| | Notifies | |

| | customer | |

| |<--------------| |

| | | |

| Receives the | | |

| book | | |

|<-----------------| | |

| | | |

In this sequence diagram:

1. The customer selects a book from the Chapters.com list.
2. The customer provides credit card information to the Chapters.com e-commerce system.
3. The Chapters.com system sends authorization request to the bank.
4. The bank processes the authorization and responds with the result (positive or negative).
5. If the authorization is positive, the Chapters.com system confirms the sale.
6. The Chapters.com system notifies the orders department about the sale.
7. The orders department charges the customer's credit card for the book.
8. The orders department notifies the customer about the successful transaction and shipment of the book.
9. The customer receives the book.

4.b.

in terms of use case modeling ,define the extends relationship with example

The use case diagram displays three kinds of entities: the system, the actors, and the use cases. The **system** is represented by a box encompassing the use cases. The **actors** are users or systems that interact with the main system and are represented by figures. The **use cases** are functions performed by the actors interacting with the system. Ellipses represent use cases.

To represent complex relationships between different use cases, we can use the extend and include relationships.

* **Extend relationship:** The use case is optional and comes after the base use case. It is represented by a dashed arrow in the direction of the base use case with the notation <<extend>>.
* **Include relationship:** The use case is mandatory and part of the base use case. It is represented by a dashed arrow in the direction of the included use case with the notation <<include>>.

Use case diagram

In the above use case diagram, the use case “Read book” includes the use case “Open book”. If a reader reads the book, she must open it too, as it is mandatory for the base use case (read book). The use case “read book” extends to the use case “turn page”, which means that turning the page while reading the book is optional. The base use case in this scenario (read book) is complete without the extended use case.

4.c.

suppose you are writing a stack class .design some test cases to verify your code

ChatGPT

When designing test cases for a stack class, it is important to consider various scenarios and edge cases to ensure that the code handles different situations correctly. Here are some test cases you can use to verify the functionality of your stack class:

1. Test Case: Empty Stack
   * Actions:
     + Create an empty stack.
     + Verify that the stack is empty using the **isEmpty()** method.
     + Try to **pop()** an element from the stack and ensure that it throws an exception or returns an appropriate error code.
2. Test Case: Pushing Elements
   * Actions:
     + Create an empty stack.
     + Push several elements onto the stack using the **push()** method.
   * Expected Results:
     + Verify that the **isEmpty()** method returns **false**.
     + Verify that the **size()** method returns the correct number of elements in the stack.
     + Verify that the top element of the stack matches the last element pushed.
3. Test Case: Popping Elements
   * Actions:
     + Create a stack and push some elements onto it.
     + Pop elements from the stack using the **pop()** method.
   * Expected Results:
     + Verify that the **isEmpty()** method returns **true** after popping all elements.
     + Verify that the **size()** method returns 0 after popping all elements.
     + Verify that the elements are popped in the reverse order in which they were pushed.
4. Test Case: Peek at Top Element
   * Actions:
     + Create a stack and push some elements onto it.
     + Use the **peek()** method to retrieve the top element without removing it.
   * Expected Results:
     + Verify that the **peek()** method returns the correct top element.
     + Verify that the **size()** and **isEmpty()** methods are not affected by the peek operation.
5. Test Case: Clearing the Stack
   * Actions:
     + Create a stack and push some elements onto it.
     + Call the **clear()** method to remove all elements from the stack.
   * Expected Results:
     + Verify that the **isEmpty()** method returns **true** after clearing the stack.
     + Verify that the **size()** method returns 0 after clearing the stack.
6. Test Case: Handling Stack Overflow (if applicable)
   * Actions:
     + Create a stack with a limited capacity (e.g., 5 elements).
     + Push more elements than the capacity of the stack.
   * Expected Results:
     + Verify that the stack throws an exception or returns an appropriate error code when trying to push beyond its capacity.
7. Test Case: Handling Stack Underflow
   * Actions:
     + Create an empty stack.
     + Try to **pop()** or **peek()** an element from the stack.
   * Expected Results:
     + Verify that the stack throws an exception or returns an appropriate error code when trying to pop or peek an element from an empty stack.

These test cases cover various aspects of stack functionality, such as pushing, popping, peeking, checking the stack's state, and handling different boundary conditions. It's important to consider additional test cases based on the specific implementation details and requirements of your stack class.

5.a.

draw an activity diagram that represents the making of a cup of tea. the initial three activities are fill kettle with water ,find cup and find tea bag and they may be performed in parallel.when the find cup and find tea bag are cimpleted the activity place tea bag in cup can start .the kettle must have boiled an the tea bag must have been placed in the cup before the activity andd water to cup can begin .if milk is required then activity add milk should be performed.

+------------------------+

| Start the Process |

+------------------------+

| | |

v v v

+------------------------+

| Fill Kettle with Water |

+------------------------+

|

v

+------------------------+

| Find Cup and Tea Bag |

+------------------------+

| AND

v

+------------------------+

| Place Tea Bag in Cup |

+------------------------+

| AND

v

+------------------------+

| Boil Kettle |

+------------------------+

| AND

v

+------------------------+

| Add Water to Cup |

+------------------------+

| AND

v

+------------------------+

| Optional: Add Milk |

+------------------------+

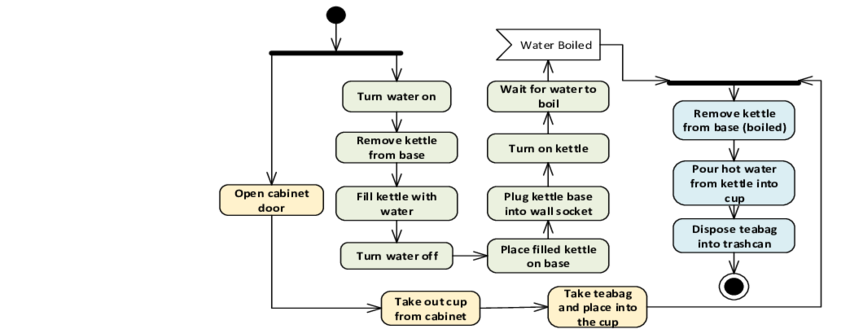
|

v

+------------------------+

| End the Process |

+------------------------+



Explanation:

1. The process starts with the "Start the Process" activity.
2. The activities "Fill Kettle with Water," "Find Cup and Tea Bag," and "Find Tea Bag" can be performed in parallel.
3. When the activities "Find Cup and Tea Bag" and "Find Tea Bag" are completed, the activity "Place Tea Bag in Cup" can start.
4. The "Kettle must have boiled" condition and the "Tea Bag must have been placed in the Cup" condition are shown as converging "AND" branches before starting the "Add Water to Cup" activity.
5. After the "Boil Kettle" and "Place Tea Bag in Cup" activities are completed, the "Add Water to Cup" activity can begin.
6. After the "Add Water to Cup" activity, there is an optional activity "Add Milk" that may or may not be performed, depending on whether milk is required.
7. Finally, the process ends with the "End the Process" activity.

5.b.

represent a librry management system using layered architectural and client server architectural patterns with necessay justifications

ChatGPT

Sure! Here's an architectural representation of a Library Management System using both the Layered Architecture and Client-Server Architecture patterns:

**Layered Architecture:**

In the Layered Architecture pattern, the system is divided into multiple layers, each responsible for specific functionality. This helps to achieve separation of concerns and maintain a modular and scalable system. Here are the layers commonly found in a Library Management System:

1. Presentation Layer:
   * Responsible for handling user interactions and displaying information.
   * Provides a user interface for library staff and patrons to access the system.
   * Justification: Separating the presentation layer allows for easy customization of the user interface and improves user experience.
2. Application Layer:
   * Implements business logic and coordinates the flow of data between the presentation and domain layers.
   * Handles tasks such as user authentication, request processing, and validation.
   * Justification: Separating the application layer enables easier maintenance, testing, and reusability of business logic.
3. Domain Layer:
   * Contains the core business entities, rules, and operations specific to the library domain.
   * Manages data persistence and retrieval.
   * Justification: Separating the domain layer ensures that the business logic is decoupled from the underlying infrastructure and can be reused across different applications or services.
4. Infrastructure Layer:
   * Provides interfaces and implementations for data access, external services, and other infrastructure-related concerns.
   * Includes database access, file management, network communication, and external API integrations.
   * Justification: Separating the infrastructure layer allows for easy swapping of technologies, improves maintainability, and promotes testability.

**Client-Server Architecture:**

In the Client-Server Architecture pattern, the system is divided into two main components: the client, which makes requests for resources or services, and the server, which provides the requested resources or services. Here's how it can be applied to a Library Management System:

1. Client:
   * Represents the user interface or application that interacts with the library system.
   * Sends requests to the server for operations such as searching for books, borrowing, returning, etc.
   * Receives and displays responses from the server.
   * Justification: Separating the client allows for different types of clients to access the library system, such as web-based interfaces, mobile applications, or desktop software.
2. Server:
   * Provides the core functionality of the Library Management System.
   * Receives requests from clients, processes them, and returns the appropriate responses.
   * Implements the business logic and data management operations.
   * Justification: Separating the server allows for centralized control, scalability, and better management of system resources. It also enables concurrent access from multiple clients.

The communication between the client and server follows a request-response model, where the client initiates a request and the server responds with the requested data or performs the necessary operations.

By combining the Layered Architecture and Client-Server Architecture patterns, the Library Management System achieves modular design, separation of concerns, scalability, and flexibility in terms of the user interface and the platform on which it runs. It also enables efficient communication between clients and the server, ensuring reliable access to library resources and services.

5.c.

Describe agile manifesto

# **Agile Manifesto**

In February 2001, at the Snowbird resort in Utah, a team of 17 software developers met to discuss lightweight development methods. The result of their meeting was the following Agile Manifesto for software development:-

We are uncovering the better ways of developing software by doing it and helping others to do it. Through this meeting, we have come to value -

* Individuals and interactions over Processes and tools.
* Working software over comprehensive documentation.
* Customers are collaboration over contact negotiation.
* Responding to change over following a plan.

So that, while there is value in the items on the right, we value the items on the left more.

## The Twelve Principle of Agile Manifesto

1. **Customer Satisfaction:** Manifesto provides high priority to satisfy the costumer's requirements. This is done through early and continuous delivery of valuable software.
2. **Welcome Change:** Making changes during software development is common and inevitable. Every changingrequirement should be welcome, evenin the late development phase. Agile process works to increase the customers' competitive advantage.
3. **Deliver the Working Software:** Deliver the working software frequently, ranging from a few weeks to a few months with considering the shortest timeperiod.
4. **Collaboration:** Business people (Scrum Master and Project Owner) and developers must work together during the entire life of a project development phase.
5. **Motivation:** Projects should be build around motivated team members. Provide such environment that supportsindividual team members and trust them. It makes them feel responsible for gettingthe job donethoroughly.
6. **Face-to-face Conversation:** Face-to-face conversation betweenScrum Master anddevelopment team and between the Scrum Master and customers for the most efficient and effective method of conveying information to and within a development team.
7. **Measure the Progress as per the Working Software:** The working software is the key and primary measure of the progress.
8. **Maintain Constant Pace:** The aim of agile development is sustainable development. All the businesses and users should be able to maintain a constant pace with the project.
9. **Monitoring:** Pay regular attention to technical excellence and good design to maximize agility.
10. **Simplicity:** Keep things simple and use simple terms to measure the work that is not completed.
11. **Self-organized Teams:** The Agile team should be self-organized. They should not be depending heavily on other teams because the best architectures, requirements, and designs emerge from self-organized teams.
12. **Review the Work Regularly:** The work should be reviewed at regular intervals, so that the team canreflect on how to become more productive and adjust its behavior accordingly.