1. XP Model of Agile Development

1. Explain the XP model of Agile Development with the help of a neat diagram.

Definition:
Extreme Programming (XP) is an agile software development methodology that focuses on customer satisfaction, teamwork, and continuous improvement. It emphasizes short development cycles, frequent releases, and close customer collaboration.

Phases of XP:

1. **Planning:** Identify user stories and prioritize them.
2. **Design:** Create simple designs for functionality.
3. **Coding:** Developers follow coding standards and perform pair programming.
4. **Testing:** Continuous testing using unit tests.
5. **Release:** Deliver working software in small increments.
6. **Maintenance:** Handle customer feedback and improve software.

Practices of XP:

- Pair Programming
- Test Driven Development (TDD)
- Continuous Integration
- Small Releases
- Refactoring
- Simple Design

Advantages:

- Fast feedback and continuous delivery.
- Encourages teamwork and customer involvement.

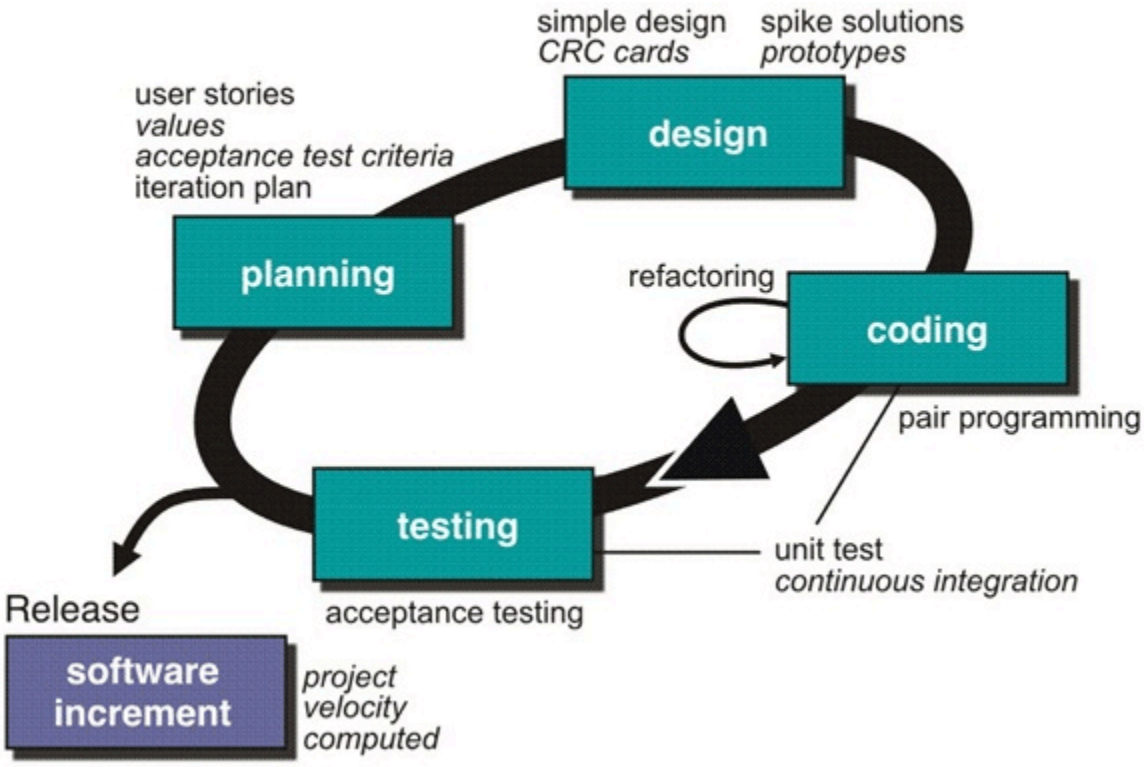
- Improves code quality and adaptability.

Disadvantages:

- Requires strong team coordination.
- Frequent customer involvement needed.
- Difficult to manage for large projects.

Diagram:

```
+-----------------------------+
|     XP Model Phases         |
+-----------------------------+
| Planning → Design → Coding  |
| → Testing → Release → Maintenance |
+-----------------------------+
```



---

2. Advantages and Disadvantages of SCRUM Process Model

2. Explain the advantages and disadvantages of SCRUM process model.

Definition:
SCRUM is an agile framework that divides the project into small, manageable iterations called *Sprints*, each lasting 2–4 weeks.

Advantages:

1. **Customer Involvement:** Continuous feedback from the customer.
2. **Flexibility:** Easy to adapt to changing requirements.
3. **Transparency:** Daily stand-up meetings ensure visibility.
4. **High Product Quality:** Frequent testing and reviews improve quality.
5. **Faster Delivery:** Working product after each sprint.

Disadvantages:

1. **Requires Skilled Team:** Team must be experienced in Agile.
2. **Difficult for Large Teams:** Works best with small, self-managed teams.
3. **Requires Continuous Communication:** Absence of members can slow progress.
4. **Not Suitable for Long-Term Fixed Scope Projects.**

---

3. Explain the following SOLID principles with suitable examples:
(i) The Single-Responsibility Principle
(ii) The Open-Closed Principle
(iii) The Liskov Substitution Principle
(iv) The Dependency-Inversion Principle
(v) The Interface-Segregation Principle

3. SOLID Principles

(i) Single Responsibility Principle (SRP):

A class should have only one reason to change, i.e., it should perform only one responsibility.

Example:

```
class PhoneConnection {
  void dial() {}
  void hangUp() {}
}

class MessageTransmission {
  void send() {}
  void receive() {}
}
```

👉 Here, responsibilities are separated correctly.

(ii) Open-Closed Principle (OCP):

Classes should be **open for extension but closed for modification**.

Example:

```
interface Shape {
  double getArea();
}

class Rectangle implements Shape {
  public double getArea() { return length * breadth; }
}

class Circle implements Shape {
  public double getArea() { return Math.PI * radius * radius; }
}
```

👉 AreaCalculator can handle any shape without modifying its code.

(iii) Liskov Substitution Principle (LSP):

Derived classes must be substitutable for their base class without changing behavior.

Example:

```
class Shape {}
class Rectangle extends Shape {}
class Square extends Shape {}
```

👉 Both can be used wherever Shape is expected — behavior remains consistent.

(iv) Dependency Inversion Principle (DIP):

High-level modules should not depend on low-level modules, both should depend on abstractions.

Example:

```
interface NumberGenerator {
  void generate();
}

class IsbnGenerator implements NumberGenerator {
  public void generate() {}
}

class BookService {
  NumberGenerator generator;
  BookService(NumberGenerator g) { generator = g; }
}
```

👉 BookService depends on abstraction, not concrete class.

(v) Interface Segregation Principle (ISP):

Clients should not be forced to implement interfaces they don't use.

Example:

```
interface Bird { void eat(); }
interface FlyingBird extends Bird { void fly(); }

class Penguin implements Bird { public void eat() {} }
class Dove implements FlyingBird { public void eat() {}; public void
fly() {}; }
```

---

4. Suppose you have to design a smartwatch which has features like – calling and messaging, music, GPS tracking and steps-monitoring. Give a top-down design for the smartwatch following the funnel model approach. Also draw mind-map for the same.

4. Top-Down Design for Smartwatch (Funnel Model)

Features:

- Calling & Messaging
- Music
- GPS Tracking
- Step Monitoring

Top-Down Design:

1. **Smartwatch System**
    - Communication Module
        - Calling
        - Messaging
    - Entertainment Module
        - Music Player
    - Health Module
        - Step Counter

        - Heart Rate
    - Navigation Module
        - GPS Tracker

Diagram (Funnel Model):

```
Smartwatch
|
├── Communication
|   ├── Calling
|   └── Messaging
|
├── Entertainment
|   └── Music
|
├── Health
|   └── Step Monitoring
|
└── Navigation
    └── GPS Tracking
```

Mind-Map:

```
        Smartwatch
        /    |    |    \
Calling Music GPS Steps
```

---

5. Suppose your task is to design a social networking app (eg facebook). Explain the Generative design process for this task – including Ideation, Sketching, Critiquing and Refining.

5. Generative Design Process for a Social Networking App

Phases:

1. **Ideation:**

- Brainstorm features (Profile, Chat, Post Sharing, Notifications).
- Study existing apps like Facebook, Instagram.

2. **Sketching:**
   - Draw rough layouts of login, feed, and chat screens.
3. **Critiquing:**
   - Review sketches with team and users for feedback.
4. **Refining:**
   - Improve design based on feedback.
   - Add consistent color, fonts, and navigation.

Diagram:

```
Ideation → Sketching → Critiquing → Refining → Final Design
```

---

6. Compare Empirical UX Evaluation with Analytical UX Evaluation.

6. Empirical UX Evaluation vs Analytical UX Evaluation

Basis	Empirical UX Evaluation	Analytical UX Evaluation
Definition	Involves real users testing the system.	Involves experts analyzing design without users.
Methods	Usability testing, field studies.	Heuristic evaluation, cognitive walkthrough.
Data Type	Collects **empirical (observed)** data.	Collects **theoretical or predicted** data.
Cost	High (needs users, equipment).	Low (only experts needed).
Accuracy	Realistic feedback from users.	Depends on expert experience.

---

7. Explain the purpose of Formative and Summative UX evaluation.

7. Formative and Summative UX Evaluation

Formative Evaluation:

- Conducted **during design and development**.
- Purpose: To **improve** design through feedback.
- Example: Testing prototypes with users.

Summative Evaluation:

- Conducted **after the product is completed.**
- Purpose: To **measure overall effectiveness and satisfaction.**
- Example: Final usability testing before release.

Comparison:

Aspect	Formative	Summative
Timing	During development	After development
Goal	Improve design	Evaluate performance
Output	Qualitative feedback	Quantitative results

---

8. Assume you have to design an online ticket generation and distribution system for Garba Pass. Users have to upload their personal details like Name, DOB, Aadhar card and photo and a unique pass will be generated for them. Design the UX evaluation metrics to measure subjective User Experience for this task.

8. UX Evaluation Metrics for Garba Pass System

System Description:
Online Garba Pass generation where users upload name, DOB, Aadhar card, and photo.

Subjective UX Metrics:

1. **Usability:**

- Ease of filling details.
- Clarity of instructions.

2. **Satisfaction:**
    - User comfort and visual appeal.
3. **Efficiency:**
    - Time required to generate pass.
4. **Error Rate:**
    - Number of input errors faced by users.
5. **Learnability:**
    - How easily a new user understands the process.
6. **Feedback:**
    - Promptness and clarity of system messages.

Diagram:

```
UX Metrics → Usability | Efficiency | Satisfaction | Learnability | Feedback
```

---

9. Explain the following terms:
(i)storyboard
(ii)moodboard
(iii)wireframe model
(iv)design catalyst
(v)sprint

9. Short Notes

(i) Storyboard:
A visual sequence of drawings showing how users interact step by step with the system.

(ii) Moodboard:
A collection of colors, images, and styles representing the emotional tone of the design.

(iii) Wireframe Model:
A basic structural layout of an interface showing placement of buttons, text boxes, and menus.

(iv) Design Catalyst:
A tool, person, or idea that stimulates creative thinking and innovation in design.

(v) Sprint:
A short, fixed-time iteration (usually 2 weeks) where the team completes a set of planned tasks.

---

10. How UX and SE Work Together to Improve Customer Satisfaction

10. With the help of a diagram explain how UX and SE can work together to improve customer satisfaction.

Explanation:
User Experience (UX) and Software Engineering (SE) complement each other.

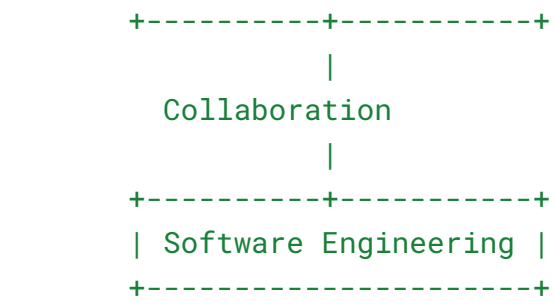
- UX focuses on **usability, design, and interaction.**
- SE focuses on **reliability, performance, and maintainability.**
  Together, they ensure software is both **technically strong** and **user-friendly.**

Steps for Collaboration:

1. Joint planning of requirements.
2. UX designers create prototypes.
3. SE team develops based on prototypes.
4. Continuous feedback and testing.
5. Refine design and functionality together.

Diagram:

```
+----------------------+
 |  User Experience (UX)|
```



Result → Improved Customer Satisfaction