

Khaja Bandanawaz University

1. What is agility in software development?

"Agility" in software development is an iterative and flexible approach to building applications that delivers small, working updates to users over time rather than launching a massive product all at once

2. Define extreme programming.

Extreme Programming (XP) is one of the most popular agile software development methodologies focused on delivering high-quality software quickly and efficiently.

- Promotes close collaboration between developers, testers, and customers.
- Supports iterative development with faster delivery and continuous improvement.

3. What is the cost of change in software development?

The cost of change in software development refers to the time, effort, and financial expense required to modify software requirements, architecture, or code after development has begun.

4. Define class based modeling

Class-based modeling is a software engineering analysis technique that defines the static structure of a system by identifying the objects it will manipulate, their attributes, their operations, and how they relate or collaborate with each other.

5. List any two agile process models.

Two common agile process models are Scrum Methodology and Extreme Programming (XP)

6. State the importance of project management.

Project management in software engineering is essential to deliver quality software products on time, within budget, and aligned with stakeholder expectations

7. List any two methods used in software project management.

Two common methods used in software project management are the Waterfall Methodology and the Agile Methodology.

1. Describe the process of requirement negotiation with a suitable example.

Requirement negotiation is a structured discussion process used by project teams and stakeholders to balance competing needs, resolve conflicts over project features, and agree on a realistic set of project requirements within time and budget limits.

The Process of Requirement Negotiation

The requirement negotiation process typically follows four main stages:

- **Discussion:** Stakeholders present their individual views, highlight problematic or conflicting requirements, and share the underlying reasons for their requests.
- **Prioritization:** Disputed or competing items are ranked using scales (such as high, medium, or low urgency) to identify what is mission-critical versus what can wait.
- **Bargaining/Compromise:** Parties trade lower-priority features for higher-priority goals, making mutual concessions to find a workable solution.
- **Agreement:** The final set of agreed-upon requirements is documented, baselined, and approved by all participating stakeholders.

Practical Example: E-Commerce Mobile App

Imagine a company building a new shopping mobile app.

- **The Conflict:**
 - The Marketing Team (client stakeholder) demands a complex, real-time AI recommendation engine and custom animation effects before launch. They argue it is vital for user engagement.
 - The Development Team (implementer) reviews the timeline and budget and states that building the AI engine will delay the launch by three months and exceed the budget.

The Negotiation Process:

- Discussion: Both teams meet to explain their positions. Marketing explains that personalization drives sales; developers explain the technical constraints and high risk of delay.
- Prioritization: The team rates requirements. A basic checkout system and secure login are marked as high priority (critical for release). The AI recommendation engine is marked as medium-to-low priority for day one.
- Bargaining: They agree on a compromise (phased release).

The Final Agreement:

- Version 1.0 will launch on time with a simpler, rule-based recommendation list (low development cost).
- The advanced AI recommendation engine is officially moved to Version 2.0, scheduled for a follow-up release three months later.

2. Explain the characteristics of agile process.

- The Agile process is an iterative and incremental project management approach that focuses on adaptability, continuous delivery, and customer satisfaction.

Core Characteristics of the Agile Process

• Iterative and Incremental Delivery

- Work is broken down into small, manageable pieces.
- Projects move forward in short, fixed cycles called sprints (usually lasting 1 to 4 weeks).
- Each cycle delivers a working, tested section of the product rather than waiting until the very end.

• Adaptability and Embracing Change

- Agile welcomes shifting requirements and views late changes as a competitive advantage rather than a disruption.
- Plans evolve continuously based on new feedback, market data, and user testing.

• Customer Collaboration and Feedback

- Clients and stakeholders stay actively involved throughout the entire project lifecycle.
- Feedback is gathered after every sprint to refine and steer the next phase of development.

• Working Software as Progress

- Success is measured by delivering functional, tested features instead of relying on heavy upfront documentation.
- Tangible results let teams prove progress and catch bugs early.

• Cross-Functional and Self-Organizing Teams

- Teams typically consist of 5 to 10 members with mixed, complementary skill sets to handle work from start to finish.
- Members share accountability and manage their own daily tasks autonomously.

• Continuous Improvement (Retrospectives)

- Teams hold regular review meetings to reflect on what went well and what can be improved.
- Processes are fine-tuned continuously to boost team efficiency, product quality, and workflow

3. Explain scenario based modeling with a suitable example.

1. Scenario-based modeling in software engineering is a requirements analysis method that shows how a system behaves from the user's point of view by describing everyday tasks and user interactions.
2. It uses actors (users, roles, or external hardware) and use cases to capture functional requirements before developers write any code.

Core Components of Scenario-Based Modeling

- **Actors:** External entities that interact with the system (e.g., a customer, an administrator, or a payment gateway).
- **Use Cases:** Specific functions or goals that an actor achieves when using the software.
- **System Boundary:** The outer limit that separates what the software system does internally from the external actors.
- **Activity/Swimlane Diagrams:** Visual flowcharts that track the sequence of steps and assign responsibilities to specific actors or system components.

Example: An ATM Withdrawal System

To see how this works, consider a basic ATM (Automated Teller Machine) system.

1. Identify the Actors

- Customer: The person inserting the card and withdrawing cash.
- Bank Database: The external system that verifies account balances and pins.

2. Define the Main Use Case

- Use Case Name: Withdraw Cash
- Primary Actor: Customer
- Precondition: The ATM is active and showing the welcome screen.

3. Normal Flow of Events (The Scenario)

1. The Customer inserts an ATM card into the machine.
2. The system prompts for a PIN (Personal Identification Number).
3. The Customer enters the PIN.
4. The system validates the PIN with the Bank Database.
5. The system displays a menu with transaction options.

6. The Customer selects "Withdraw Cash" and enters an amount (e.g., ₹2,000).
7. The system checks if the account has enough balance via the Bank Database.
8. The system dispenses the cash and prints a receipt.
9. The Customer takes the cash and card.

4. Exception Flow (When things go wrong)

- What if the PIN is wrong? The system displays an error message, ejects the card or allows a retry (up to 3 attempts), and ends the session.
- What if there is insufficient balance? The system shows an "Insufficient Funds" warning and returns to the main menu

1. Discuss the various requirement elicitation techniques and explain how they help in gathering software requirements.

Requirement elicitation is the methodical process of gathering explicit and implicit needs, expectations, and constraints from stakeholders to define a software system.

Key Requirement Elicitation Techniques

- **Interviews:** Direct, one-on-one or group conversations (structured with fixed questions or unstructured with open-ended dialogue) that allow analysts to clarify individual stakeholder expectations.
- **Surveys and Questionnaires:** Written sets of questions distributed to a large audience to collect quantitative and qualitative feedback quickly and efficiently.
- **Requirements Workshops / Focus Groups:** Collaborative meetings bringing together key stakeholders, developers, and users to build consensus, map workflows, and uncover hidden system needs.
- **Observation / Shadowing:** Watching users perform their daily tasks in their natural work environment to catch unspoken workflow steps, practical bottlenecks, and real-world usage patterns.
- **Document Analysis:** Reviewing existing corporate manuals, business plans, legacy system specs, or competing software to establish a baseline of operational rules and regulations.
- **Prototyping:** Building early mockups, wireframes, or working models visualize the final product and provide concrete feedback.
- **Brainstorming:** Creative group sessions encouraging participants to freely pitch a wide range of features and ideas to solve specific user problems.

- **Use Cases and User Stories:** Scenario-based narratives that capture functional interactions (e.g., "As a user, I want...") to make system requirements relatable and actionable for developers.

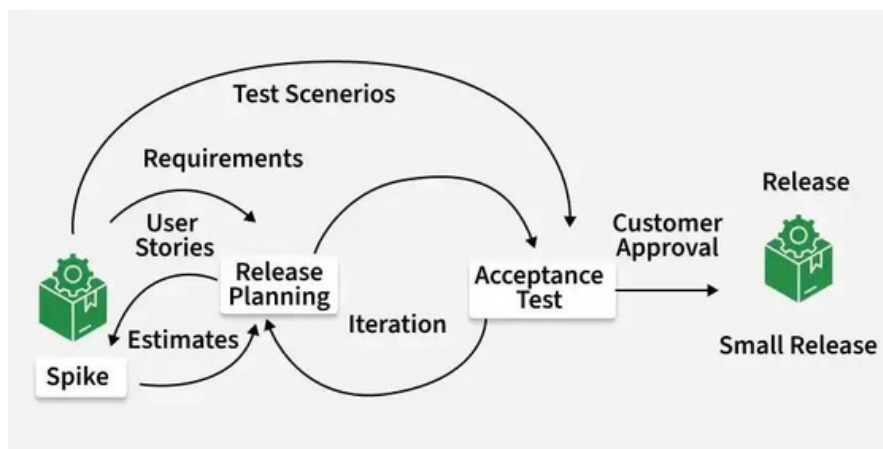
Advantages:

- Requirement Elicitation helps stakeholders and the development team gain a deeper and shared understanding of project requirements.
- Requirement elicitation results in more accurate and complete requirements, leading to a higher-quality final product.
- Requirement Elicitation allows the early identification of potential issues and conflicts in requirements.

2. Describe the extreme programming(XP) process model with a neat diagram, also explain its major practices and advantages.

Extreme Programming (XP) is one of the most popular Agile software development methodologies focused on delivering high-quality software quickly and efficiently.

- Promotes close collaboration between developers, testers, and customers.
- Supports iterative development with faster delivery and continuous improvement.



Extreme Programming (XP) Workflow Diagram

The above diagram represents the workflow of Extreme Programming (XP) and explains how customer requirements move through planning, iterations, testing, and final software release. It highlights the continuous feedback cycle followed in XP to improve software quality and customer satisfaction.

- Requirements are collected from customers in the form of user stories.
- Release planning is performed to estimate effort and organize development tasks.
- Development takes place through short iterative cycles with continuous coding and testing.
- Acceptance testing ensures the developed software meets customer expectations.

- Customer approval is taken before delivering small and frequent releases.
- Continuous feedback and iteration help improve the software throughout the development process.

- **Good Practices in Extreme Programming**

Code Review: Helps identify and fix errors quickly. XP promotes pair programming, where two developers work together and regularly switch roles.

Testing: Improves reliability by removing errors. XP follows Test-Driven Development (TDD), where tests are written before the actual code.

Incremental Development: Software is built in small parts, allowing frequent feedback and continuous improvement after each iteration.

Simplicity: Keeping the design simple makes the code easier to develop, test, and maintain.

Design: Continuous attention to design ensures better quality software.

Integration Testing: Frequent integration and testing help detect issues early. XP supports continuous integration by testing multiple times a day.

Advantages

Timely delivery: Short and manageable development cycles help maintain project schedules.

Better understanding of requirements: Continuous customer involvement reduces misunderstandings about the business or domain.

Reduced project failure: Regular communication and feedback help identify and solve issues early.

Improved team collaboration: Strong teamwork and a supportive environment boost morale and handle staff changes effectively.

Lower cost of changes: Continuous testing ensures that changes can be made without affecting system performance.

Adaptability to business changes: Changes are accepted at any stage, making the process flexible.

Fewer defects: Regular unit testing helps detect and fix bugs early, improving software quality