SOFTWARE DEVELOPMENT MODELS

Extreme Programming
&
Evolutionary Model

Extreme Programming

- Agile method focused on improving software quality and adaptability.
- Emphasizes frequent releases in short development cycles.
- Continuous code review through pair programming.
- Unit testing of all code.
- Prioritizes simplicity and avoiding unnecessary features.
- Strong communication with customers and team members.

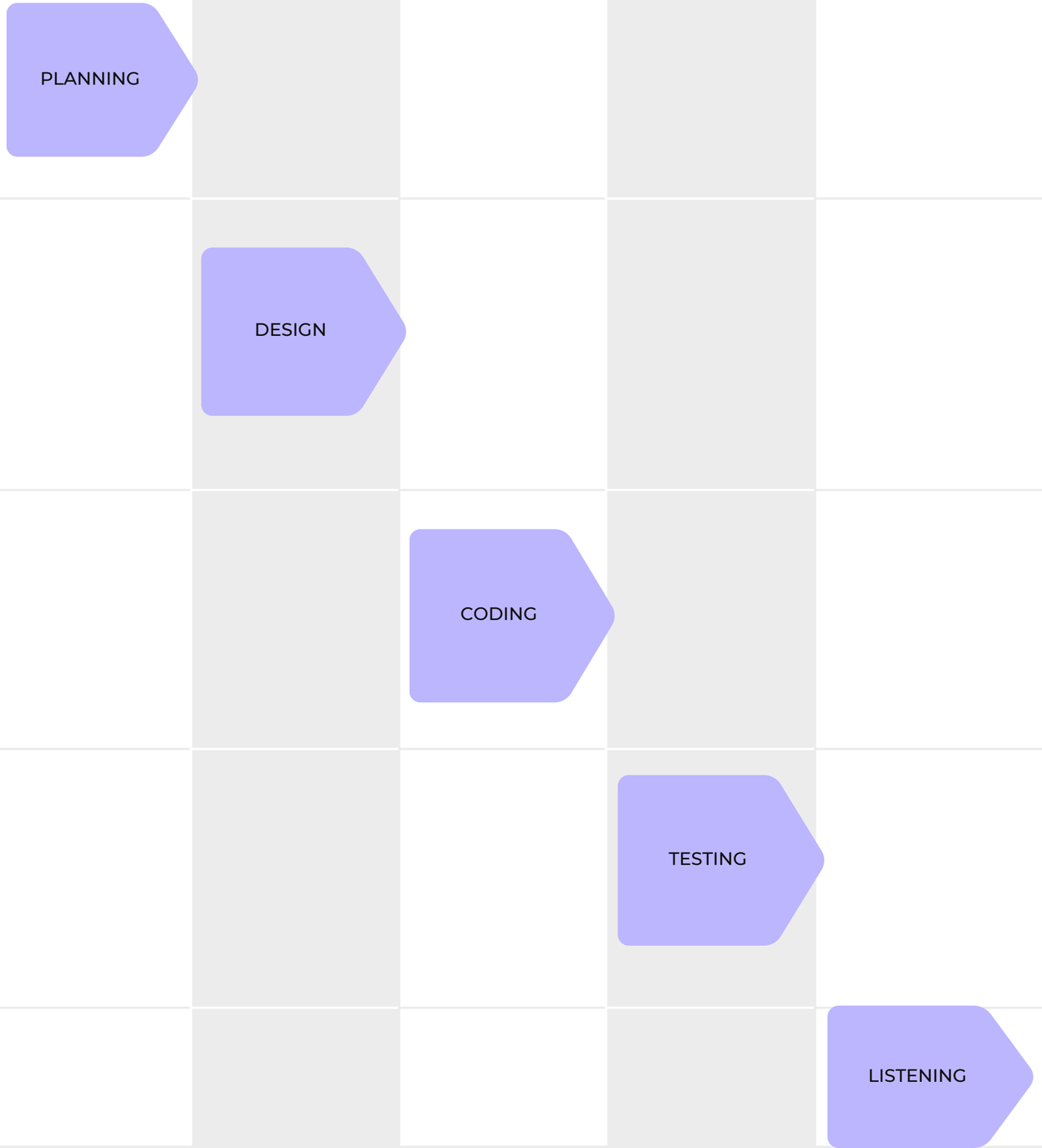
Step 1:
Defining needs based on user stories and then calculation of efforts

Step 2:
Creation of essential design is done based on the common user story to help everyone to understand the system architecture.

Step 3:
XP promotes pair programming enhancing code quality. TDD approach is used.

Step 4:
(XP) gives more importance to testing that consist of both unit tests and acceptance test.

Step 5:
Regular Consumer Feedback is gathered.



1. **Planning: Clients describe needs as user stories; the team assesses effort and prioritizes releases.**
2. **Design: Essential design is created based on current stories, using analogies for clarity.**
3. **Coding: Promotes pair programming and Test-Driven Development (TDD), with frequent integration and automated testing.**
4. **Testing : Emphasizes both automated unit tests and customer-driven acceptance tests to ensure quality and requirements alignment.**
5. **Listening: Regular customer feedback is gathered for product adjustments and improvements.**

The Extreme model excels by using efficient architectures, enabling better scalability, higher accuracy, and faster inference in diverse tasks.

PROS AND CONS

- Improved Risk Management:

By delivering software in small, manageable increments, risks are identified and addressed early in the development process. This reduces the likelihood of major issues arising late in the project.

- User-Centric Development:

The model encourages ongoing user involvement, ensuring that the software aligns with user needs and expectations. This reduces the risk of developing a product that does not meet the user's requirements.

- Adaptability to Change:

The model’s flexibility allows for the easy incorporation of new or changing requirements, which is critical in dynamic environments where requirements may evolve over time.

- Complexity in Management:

Managing an evolutionary development process can be complex, particularly in terms of tracking progress, coordinating teams, and ensuring that all iterations align with the overall project goals.

- Higher Cost:

The iterative nature of the evolutionary model can lead to higher costs, as changes in requirements and continuous refinement may require additional resources and time.

- Dependency on User Feedback:

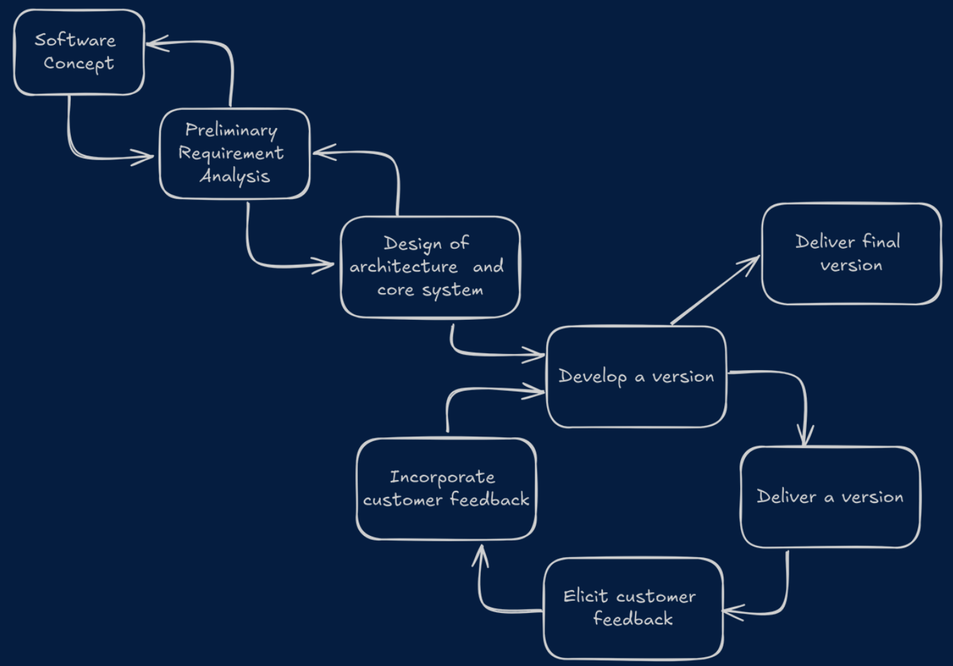
The success of the evolutionary model relies heavily on timely and effective user feedback. If user feedback is delayed or inadequate, it can lead to the development of features that do not fully meet user needs.

EVOLUTIONARY MODEL

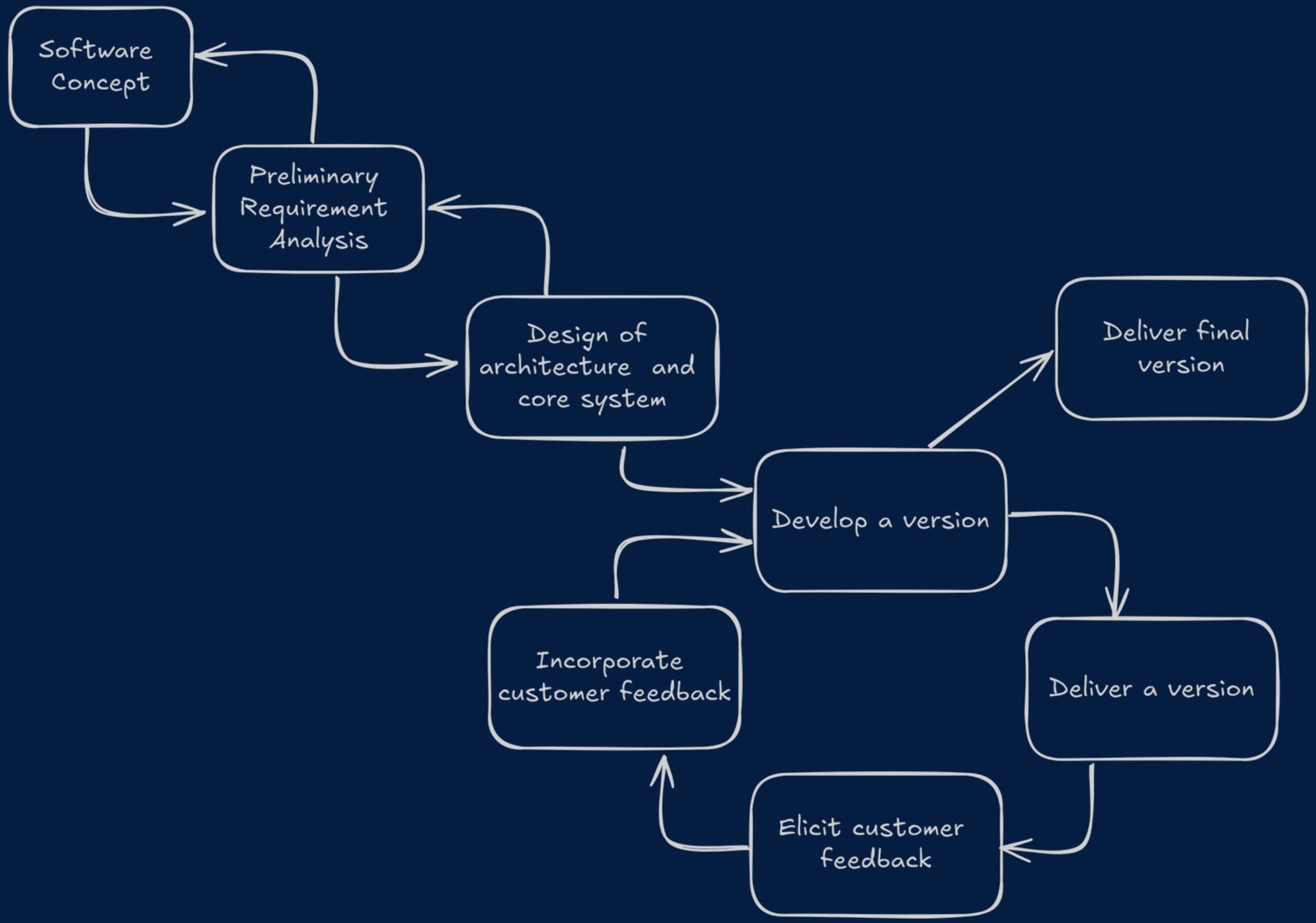
Definition: The Evolutionary Model is an approach to software development that emphasizes continuous refinement of a system through iterative cycles. Unlike traditional models that follow a linear process, the Evolutionary Model allows the software to evolve over time based on user feedback and changing requirements.

Key Characteristics:

- Iterative Development
- Feedback-driven:
- Flexibility
- User Involvement



EVOLUTIONARY = ITERATIVE + INCREMENTAL



PROS AND CONS

- Flexibility in Requirements: The evolutionary model allows for changes in requirements at any stage, making it suitable for projects with evolving needs.
- User Feedback: Early versions of the software are delivered quickly, allowing users to provide feedback, which can guide further development.
- Risk Management: Frequent iterations help identify and mitigate risks early in the process, reducing the chance of major issues later on.
- Better Resource Utilization: Resources can be allocated based on priorities identified during each iteration, allowing for efficient use of time and effort.
- Improved Quality: Continuous testing and refinement in each iteration lead to gradual improvements in the software's quality.

- Project Complexity: Frequent changes and iterations can lead to increased project complexity, making it harder to manage.
- Potential for Scope Creep: The model's flexibility can lead to unplanned feature additions, resulting in scope creep and delays.
- High Customer Involvement: The model requires constant customer feedback, which may be difficult to obtain consistently.
- Documentation Challenges: Changes and iterations may lead to incomplete or outdated documentation, making future maintenance harder.
- Costly for Large Projects: Due to multiple iterations and testing cycles, the model can become expensive, especially for large-scale projects.

Thank you