NIST COLLEGE

SHAPING THE FUTURE

Software Engineering

CSIT- 6th Semester

Prepared by: Er. Rupak Gyawali, 2024

Unit 3: Agile Software Development

Agile Development; Plan-Driven vs. Agile Development; Agile Methods; Agile Development Techniques; Introduction to Agile Project Management

Assignment:

1. Agile Development Practices:

- Write about Dynamic System Development Method (DSDM)
- When to use DSDM, Advantages and Dis-Advantages of DSDM

2. Compare Agile vs Prototyping model

3. Difference Between XP vs Scrum vs DSDM

Rapid Software Development:

- **Rapid Software Development** is a process focused on quickly delivering software by using flexible, iterative methods. Instead of following a strict, step-by-step process, it emphasizes:
  - **Speed**: Developing and delivering software fast to meet urgent business needs.
  - **Frequent Updates**: Releasing regular, small updates so users can see progress and provide feedback.
  - **Adaptability for changing requirement**: Being flexible to make changes based on user feedback or changing requirements.
- **Examples** of rapid development methods include **Agile**, **RAD (Rapid Application Development)**, and **Prototyping**. These methods work well when requirements are not fully clear at the start and may change over time.

- **Rapid Software Development** is a way to build useful software quickly by breaking s/w component into smaller parts, or **increments**, where each part adds new features. Although there are many approaches to rapid software development, they share some fundamental characteristics:
  - **Combined Phases**: The steps of planning, designing, and coding are mixed together, rather than done one after the other. The user requirement document focus on the most important features of the systems with minimal written documentation.
  - **Multiple Versions**: The software is built in series of versions, with each version reviewed by users who give feedback and suggest changes for future versions. New requirements that should be implemented in later version are also discussed.
  - **Quick User Interface Design**: User interfaces are created fast with tools that allow easy placement of icons and design elements. The system may then generate a web-based interface for a browser or an interface for specific platform like MS-Windows.
- In **Agile methods**, which is a type of rapid development, small updates (or increments) are released every 2–3 weeks. Customers are involved throughout, giving feedback, and formal documentation is minimized in favor of simple, direct communication.

Agile Methodologies:

Background

- Over the years, different methods for analyzing and designing systems have been created. In February 2001, experts from various approaches (individuals involved in traditional waterfall development, iterative approaches, object-oriented methodologies, and others) gathered in Utah, USA.
- They agreed on common principles found in their methods, which they documented in “The Agile Manifesto.”
- The agile methodologies group argues that software development methodologies adapted from engineering generally do not fit with real-world software development (Fowler, 2003).
- In engineering disciplines, such as civil engineering, requirements tend to be well understood. Once the creative and difficult work of design is completed, construction becomes very predictable. In addition, construction may account for as much as 90 percent of the total project effort.
- For software, on the other hand, requirements are rarely well understood, and they change continually during the lifetime of the project. Construction may account for as little as 15 percent of the total project effort, with design constituting as much as 50 percent.
- Applying techniques that work well for predictable, stable projects, such as bridge building, tend not to work well for fluid, design-heavy projects such as writing software, says the agile methodology proponents.

Agile Methodology: Agile methodology is the type of methodology in which a user can see something quickly. It is the iterative as well as its incremental process. It prioritizes user collaboration, responding to change and delivering the software product in short time.

Agile Methodologies:

4 core values of Agile Development:

- **Individuals and Interactions over Processes and Tools:** Agile focuses more on the people working together and communicating effectively rather than relying solely on formal processes or specific tools.
- **Working Software over Comprehensive Documentation:** Agile values having a functional software product that works well over having extensive documentation about how the software should work.
- **Customer Collaboration over Contract Negotiation:** Agile believes in involving the customer closely throughout the development process, rather than focusing solely on negotiating contracts.
- **Responding to Change over Following a Plan:** Agile acknowledges that change is common in software development, so it prioritizes being able to adapt to change quickly rather than strictly sticking to a predetermined plan.

Principle of Agile Methods:

- **Customer Involvement:** Customers stay involved during development, helping set priorities and giving feedback on each version of the system.
- **Incremental Delivery:** The software is built and delivered in parts, with the customer choosing what features to add in each part.
- **People Over Process:** Developers are trusted to work in ways that suit their skills, without strict processes guiding every step.
- **Adapt to Change:** The system should be designed to handle changes since requirements may evolve.
- **Keep it Simple:** Focus on making both the software and development process as simple as possible by avoiding unnecessary complexity.

Suitability of project to use Agile :

Agile methods work well for certain types of projects:

- **Product Development:** When a software company is creating a small or medium-sized product to sell.
- **Custom Systems for Organizations:** When a company is building a system for internal use, the customer is committed to actively participating, and there aren’t many strict rules that impact the software.

Challenges of Agile Method:

Agile principles can be challenging to follow:

1. **Customer Involvement**: Agile needs a customer who can actively participate with the team. However, customers often have other responsibilities and may not have the time to fully engage in development.
2. **Team Compatibility**: Agile methods involve intense teamwork, but some developers may not work well in close collaboration due to personal work styles.
3. **Prioritizing Changes**: When many people are involved, it's hard to agree on what changes are most important, as different stakeholders often have conflicting priorities.
4. **Keeping It Simple**: Simplifying the system takes extra effort. With tight deadlines, the team may not have the time to streamline or simplify as they'd like.
5. **Culture Shift**: Many larger companies are used to structured processes. Moving to agile, with its flexible approach, can be hard for teams used to strict guidelines.

Practical Problems/ Issues in Agile Method:

- **Agile doesn't fit with legal contracts:** Agile requires flexible and changing requirements, which doesn't align well with fixed contracts. Big companies need clear, fixed contracts with specific requirements. Agile doesn't work well with this kind of strict contract.
- **Agile is for new software, not for maintenance:** Agile is great for creating new software, but most companies spend money on fixing and updating old systems, which isn't the focus of agile. Agile methods, which avoid extensive documentation, may make maintenance harder, as there may be no detailed requirements document.
- **Readable Code over Documentation:** Agile methods prioritize high-quality code, assuming good code is easier to understand than formal documents.
- **Agile works for small, local teams, not big global ones:** Agile is best for small teams working closely together in one place, but many companies now have big teams spread across different countries, which makes agile harder to use. *Problems when worked in big teams → Different Time Zone, Agile require face to face communication, Language Barrier*
- **Team Continuity:** Agile teams depend on shared, informal knowledge. If team members leave, new people may struggle without documentation.

Advantages of Agile model:

- Project is divided into short iteration.
- Quick release of 1st product version.
- It minimizes the risk of software development.
- Customer can see the result and understand whether he/she is satisfied with it or not.

Dis- advantages of Agile model:

- The development team should be highly professional and client-oriented.
- With further correction and change, there may be chances that the project will cross the expected time.
- There may be difficult to estimate the final cost of the project due to constant iteration.
- New requirement may be a conflict with the existing architecture.

When to use Agile ?

- **Dynamic Requirements:** Use Agile when project requirements are expected to change frequently or are not well-defined at the beginning. Agile's iterative nature allows for flexibility and responsiveness to evolving needs.

- **Tight Deadlines:** Agile can be effective when there is a need to deliver a working product quickly. By prioritizing features and delivering them in iterations, useful functionality can be released early and improved over time.

- **Customers Involved in the Process:** Agile methodologies prioritize close collaboration with customers throughout the development process. Projects where customers understand the Agile process and are willing to actively participate in providing feedback and direction are better suited for Agile methodologies.

- **Complex Projects:** It's good for medium, complex projects by breaking them into smaller pieces.

Agile Development Methodology:

- **Brainstorm**: The team gathers and analyzes the requirements from stakeholders, ensuring a clear understanding of what needs to be developed.
- The team collaborates to generate ideas and discuss potential solutions. This phase encourages creativity and explores how to implement the requirements effectively.

- **Design:** A design document and prototype are created. This involves defining the architecture, user interfaces, and other technical aspects to guide the development process.
- **Development:** The actual coding and implementation of the design take place in this phase. Iterations, demos, and feedback are used to continuously improve the product.
- **Quality Assurance:** Testing is conducted to identify defects and ensure the software meets quality standards. Bugs are fixed, and improvements are made as needed.
- **Deployment:** The software is deployed in a production environment where it becomes operational. Production and technical support activities help maintain the system.
- **Release to Market:** After ensuring all functionality is done and meets the requirements, the product is released to the market or delivered to the client.
- **Next Iteration:** Agile promotes iterative development. Feedback from users and stakeholders is used to identify further improvements or new features, leading to the next cycle of development.

Agile Development Methodologies → Scrum, Kanban, Extreme Programming, DSDM, Lean Development, Crystal Methodology etc.

Plan Driven Vs Agile Development:

- In **agile development**, the focus is mainly on **design and implementation**, they include other activities, such as requirements elicitation and testing, in design and implementation section. This approach doesn't separate these steps clearly—they all happen together as part of the design and build phases.
- In contrast, **plan-driven development** follows a **step-by-step structure**, where each stage (like requirements, design, coding, testing) has its own clear output. Each stage's output is used to guide the planning of the next stage, making the process more structured and sequential.

Figure shows the distinctions between plan-driven and agile approaches to system specification.

- In a **plan-driven approach**, changes are made only during specific steps (Req. gathering, Design, Coding, Testing) of the process, and formal documents are used to pass information from one step to the next. (Document is created after completing Design phase and shared to Coding phase). For example, the requirements can change over time, and a final requirements document is produced to guide the design and implementation phases.
- In an **agile approach**, changes happen across all activities at the same time. This means that requirements and design are developed together instead of one after other.
- A **plan-driven software process** can still allow for incremental development and delivery. This means you can break down the requirements and plan the design and development into smaller parts, or increments.
- On the other hand, an **agile process** doesn't always focus solely on coding; it can also produce some design documentation. Sometimes, the agile team might choose to create a "documentation spike," where they focus on producing documentation instead of developing a new version of the system.

- **Eg. of Plan driven models:**
Waterfall, V-Model,RUP, Incremental Model, Spiral Model (in its traditional form)

- **Eg. of Agile Models:**
Scrum, XP, Kanban

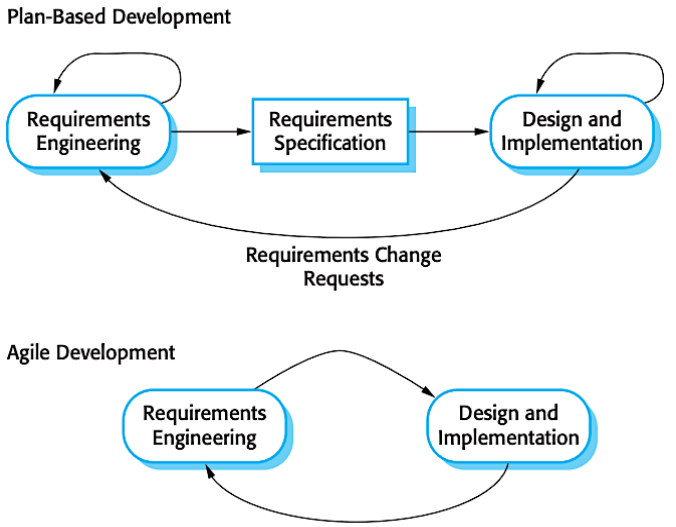

Figure: Plan-driven and agile specification

Plan Driven Vs Agile Development which one to choose?

- Do you need a very detailed specification and design before starting implementation? **If yes, go with a plan-driven approach.**
- Can you deliver software in small parts to customers and get quick feedback? **If yes, consider using agile methods.**
- How big is the system? **Agile methods work best with small, co-located teams that communicate easily. Larger systems might need a plan-driven approach.**
- What type of system are you developing? **If it requires a lot of analysis (like a real-time system), a detailed design is necessary, so a plan-driven approach may be better.**
- How long will the system be in use? **Long-lasting systems might need more design documentation to help future support teams understand the original intentions. However, agile supporters argue that documentation often becomes outdated.**
- How skilled are your designers and programmers? **Agile methods may need a higher skill level. If your team is less experienced, it might be better to have skilled people focus on design and others on coding.**

Agile Methods: Agile methods are frameworks and approaches that follow the Agile principles, focusing on flexibility, customer collaboration, and iterative development.

Some of the Agile Methods/ Frameworks/ Agile Development Methodology:

1. **Scrum**

- Breaks the work into short cycles called **sprints** (2–4 weeks).
- Teams plan what to do, work on it, and review progress regularly.
- Key roles:
  - **Product Owner**: Decides what to build.
  - **Scrum Master**: Ensures the process runs smoothly.
  - **Team**: Does the work.
- Focuses on small, manageable goals to deliver value quickly.
- Event: Daily standups, sprint planning, reviews

2. **Kanban**:

- A visual workflow management method.
- Work is represented on a board (e.g., To Do, In Progress, Done).
- Focuses on limiting work-in-progress (WIP) to improve efficiency.

Some of the Agile Methods/ Frameworks:

3. Extreme Programming (XP)

- Focuses on writing **high-quality code**.
- Encourages practices like:
  - **Pair programming**: Two developers work together.
  - **Test-driven development (TDD)**: Write tests before the code.
  - **Continuous feedback**: Improve based on user feedback.

4. Lean Development

- Inspired by manufacturing.
- Focuses on **reducing waste (overproduction, Delays in dev. Process, Errors)**, delivering value faster, and improving efficiency.
- Avoids unnecessary steps and makes the process as simple as possible.

5. Crystal Methodologies:

- Crystal is an **Agile method** that focuses on **people and teamwork** instead of strict rules or processes.
- It believes that **every project is unique**, so the way a team works should depend on the **size of the team** and how critical the project is.

Agile Methods/ Frameworks

a) Extreme Programming (XP):

- Extreme Programming (XP) is one of the most popular Agile methods.
- It was named "Extreme" by its creator, Kent Beck, because it takes well-known best practices, like developing in small cycles and frequent testing, to the extreme.
- In XP, for instance, programmers create and test several new versions of the software in just one day.
- This approach focuses on delivering high-quality software quickly through continuous collaboration and improvement.

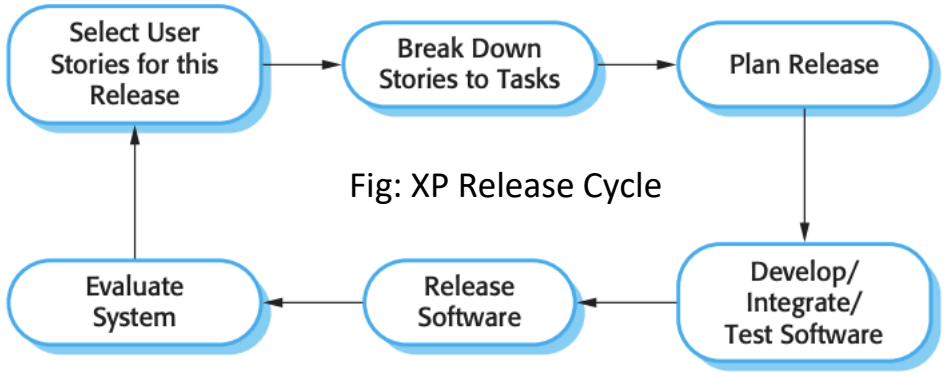

Fig: XP Release Cycle

a) Extreme Programming (XP Release Cycle):

- **Select User Stories for this Release:** In extreme programming, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks.
- **Break Down Stories to Tasks:** Each user story is broken down into tasks means turning big ideas into smaller, manageable steps.
- **Plan Release:** Create a plan for completing these selected user stories, setting goals and timelines.
- **Develop/Integrate/Test Software:** Programmers work in pairs to code the tasks, write tests, and integrate the software, ensuring that all tests pass with each update.
- **Release Software:** Make the new version of the software available to users.
- **Evaluate System:** Gather feedback from users on the latest release to improve and guide future updates.

Extreme Programming (XP) is an agile software development methodology characterized by short development cycles, frequent releases, and continuous customer involvement, aiming to deliver high-quality software that meets evolving requirements through close collaboration and adaptability.

Phases in XP:

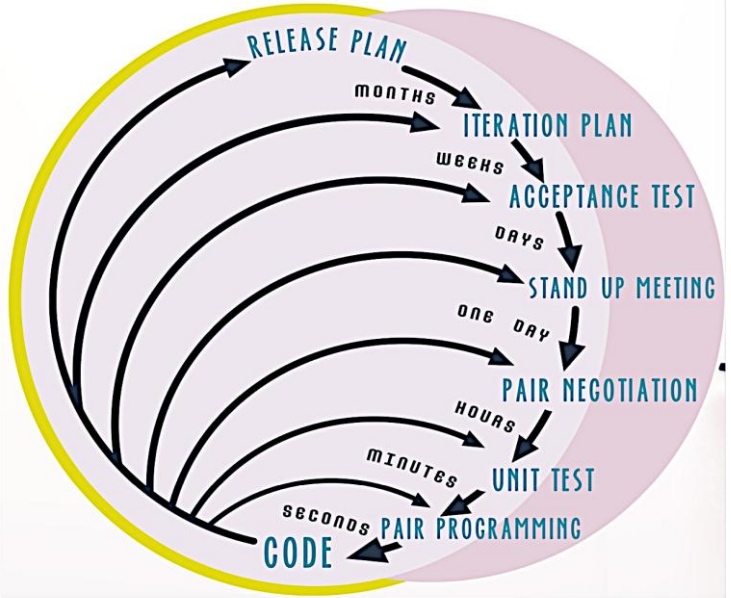

Fig: eXTREME Programming

- **Release Plan:** In release planning, customers share what they want with the programmers, whereas the programmers estimate the difficulties of achieving the desired features. They figure out the costs and, together with the customer, make a project plan. This plan outlines what will be done first, second, and so on, but it might change as the project goes on.
- **Iteration Plan:** Iteration planning happens every couple of weeks whereby the XP teams typically take two weeks to build the software, which is called iteration. Once they complete the build in each iteration, they deliver the software at the end of each iteration.

a) Extreme Programming:

Phases in XP:

- **Acceptance Test:** Tests are written to make sure the software does what it's supposed to do, based on what the customer wants.
- **Standup Meeting:** The team meets every day to talk about what they're doing and any problems they're having. It's a quick meeting where everyone stands up.
- **Pair Negotiation:** Programmers work together to figure out how to build each feature, making sure they agree on the best way to do it.
- **Unit Test:** Each piece of code is tested to make sure it works on its own, catching any mistakes early on.
- **Pair Programming:** Two programmers work together on the same piece of code, helping each other and catching mistakes as they go.
- **Code:** The team writes the actual code, following the plans they've made and making sure it does what it's supposed to.

Note: release can happen in months, iterations of that release happen in weeks, acceptance testing happens in days, and unit testing happens in minutes.

XP Principles and Practices:

- **Incremental Planning**: Requirements are listed on Story Cards. The team picks the most important stories that can fit in the release timeline and splits them into smaller tasks.
- **Small Releases**: Start with a basic version that adds real value, then release updates regularly to add more features.
- **Simple Design**: Design only what's needed to fulfill current requirements—no extra.
- **Test-First Development**: Write tests for each feature before building it, ensuring it works as expected from the start.
- **Refactoring**: Developers continuously improve code, keeping it clean and easy to work with.
- **Pair Programming**: Developers work in pairs to check each other's work, improving quality and sharing knowledge.
- **Collective Ownership**: All developers work across the entire codebase, so everyone understands and takes responsibility for it.
- **Continuous Integration**: After finishing a task, it's integrated into the system, and all tests are run to make sure everything works together.
- **Sustainable Pace**: Avoid excessive overtime, as it often lowers the quality of code and productivity over time.

Agile Development Techniques OR

Some important agile practices introduced by XP

i) **User Stories:** In agile methods, software requirements are not fixed and can change over time. Instead of having a separate process for gathering requirements, agile integrates it with the development process.

- One way to gather requirements is by using **user stories**. These are short, simple descriptions of how a user will use the system. They help the team understand what the user wants and make changes easily during development.
- The customer works closely with the development team to discuss their needs. Together, they create a "story card" that describes a user scenario, such as prescribing medication in the Healthcare system. The development team then works to implement this scenario in a future software release.
- As needs change, new story cards may be created, or existing ones might be discarded.
- User stories are useful because they are easy to understand, unlike traditional documents. However, it's hard to make sure all important details are included, and sometimes users forget key parts when describing their tasks.

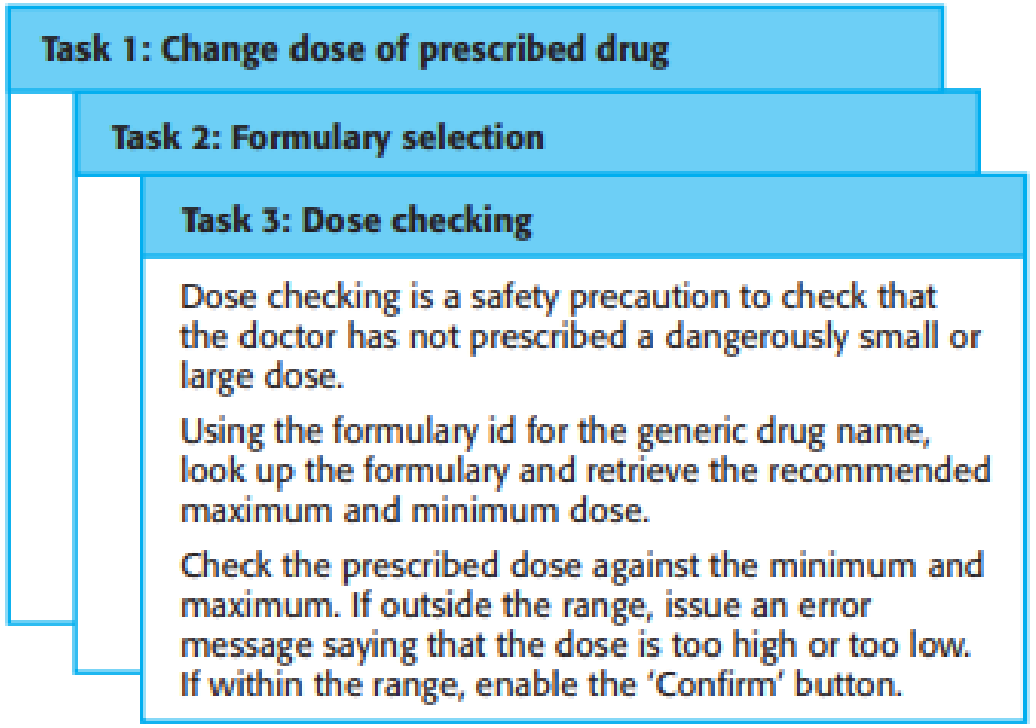

Figure: Examples of task cards for prescribing medication

Agile Development Techniques OR

Some important agile practices introduced by XP

ii) **Refactoring:** Traditional software engineering advises designing software with future changes in mind, making it easier to update later. However, extreme programming disagrees with this idea, arguing that it's often a waste of time. The changes we expect may never happen, or the changes that do come might be completely different from what we planned for.

- In software development, changes are always needed. To make these changes easier, extreme programming (XP) recommends constantly improving the code, a process called refactoring.
- Refactoring means that developers fix and improve the code right away, even if it's not urgently needed.
- As software is updated, the code can become messy, making future changes harder. Refactoring helps keep the code clean and organized, so it's easier to work with later.
- Refactoring means cleaning up the code, like removing duplicates, renaming things for clarity, and using reusable methods. Tools can help make this easier.

iii) **Test-first Development:** Test-first development is a key part of Extreme Programming (XP). It means writing tests before writing any code. This way, developers can check if the code works as it's being written, helping catch problems early.

- In XP, testing is automated, so tests run automatically as the code is developed. The main steps of testing in XP are:
  - Writing tests before coding (test-first).
  - Creating tests based on user stories.
  - Involving users in creating and checking tests.
  - Using tools to automate the testing process.
- Test-first development helps make requirements clearer and avoids confusion. It also stops the issue of developers working faster than testers (Test-lags). Each task has tests to make sure the code works properly.
- However, there are challenges with test-first development:
  - Programmers might skip writing complete tests or take shortcuts.
  - Some parts of the system, like complex user interfaces, are hard to test fully.
- Although there may be many tests, it's hard to know if they've covered everything. If tests aren't reviewed or updated, bugs might still slip through.

Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

Agile Development Techniques OR
Some important agile practices introduced by XP

iv) **Pair programming:** Pair Programming is a software development technique where two programmers work together at one workstation. Here's a simple breakdown of how it works:

- **Roles**: One programmer (the "driver") writes the code, while the other (the "observer" or "navigator") reviews each line of code as it's written. The observer can also suggest improvements or ask questions.
- **Collaboration**: This method encourages collaboration and knowledge sharing, allowing both programmers to discuss ideas, catch mistakes in real-time, and ensure high code quality.
- **Benefits**:
  - **Improved Code Quality**: With two sets of eyes on the code, bugs are often caught early.
  - **Knowledge Sharing**: Team members learn from each other, leading to better overall team skills.
  - **Faster Problem Solving**: Complex problems can be tackled more efficiently as two minds work on them.
  - **Collective Ownership:** The whole team shares responsibility for the code, following the idea of "egoless programming." This means problems are tackled as a team rather than blaming individuals.
  - **Support for Refactoring:** Refactoring (improving existing code) is easier because the benefits are shared among the team, encouraging everyone to improve the code without worrying about being judged as less productive.
- **Continuous Switching**: The roles can be switched frequently (e.g., every 15 minutes), allowing both programmers to engage fully in both driving and navigating.

a) Extreme Programming:

Advantages of XP:

- Quick prototype production.
- Works well in project with undefined or changing requirements.
- Frequent feedback from user.
- Increased user satisfaction.
- Most beneficial for teams with less than 10 team members.

Dis- advantages of XP:

- Works best in small projects.
- Requires much user inputs.
- Over focus on early product.
- Requires experienced developers.
- Less cost effective; Pair programming.

Agile Project Management:

- Agile Project Management focuses on delivering software on time and within budget, but it requires a different approach compared to traditional plan-driven methods. Agile Project Management framework is Scrum.

a) Scrum:

Scrum is a flexible Agile method that focuses on managing the process of iterative development without specifying technical details or coding practices. It doesn't tell you how to code, like using pair programming or test-first methods. Instead, it works alongside technical methods like Extreme Programming (XP) to help organize and manage projects better.

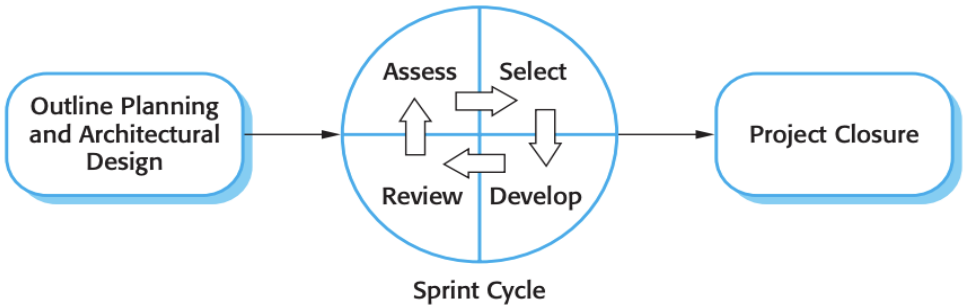

Fig: The Scrum Process

2-4 weeks → 1 Sprint

Scrum has three main phases:

1. **Outline Planning Phase**: This is where the project's general goals are set and the software architecture is designed.
2. **Sprint Cycles**: The heart of Scrum, these are short work periods (typically 2-4 weeks) where the team focuses on developing specific features from a prioritized list called the product backlog. Here's how it works:

Product backlog → list of work to be done on the project

   - **Scrum Planning**: The team reviews the product backlog with input from the customer to decide what to work on next.
   - **Development**: The team works together to build the selected features, holding daily meetings to track progress and adjust priorities as needed.
   - **Scrum Master Role**: The Scrum Master helps keep the team focused and shields them from outside distractions.
3. **Project Closure Phase**: After several sprints, this phase wraps up the project, finalizes documentation (like user manuals), and reflects on what was learned during the project.

The Scrum master is like a coach for the team. They organize daily meetings where everyone stands up to keep things quick and focused. In these meetings, team members share updates on what they've done, any issues they're facing, and what they plan to work on next. This way, everyone stays informed, and if problems come up, the team can adjust their plans together instead of just following orders from the Scrum master.