Agile Software Development

Class Notes

Aradhya Chakrabarti, CSE-08

March 22, 2026

Contents

1 Unit I - Introduction to Software Development

1.1 What is Software Development?

Software development is defined as the process of designing, creating, testing, and maintaining computer programs and applications.[1]

Software development plays an important role in our daily lives. It not only empowers smart applications but also supports businesses worldwide. Software developers develop the software, which itself is a set of instructions in order to perform a specific task. Developers develop system software, programming software, and application software.[2]

1.2 Types of Software Development: Broad Classification

Software development methodologies can be broadly classified into two main paradigms:

- **Sequential (plan-driven) methodologies:** A traditional, linear process. These follow a step-by-step process, where each phase completes before the next begins. Useful when requirements are clear and unlikely to change. Often chosen when scope, timeline, and requirements are well-understood upfront, or when frequent changes are not expected.[3]
  - **Waterfall Model:** All planning, design, implementation, testing, deployment happen in a fixed linear order. Changes are hard once phases are done.
  - **V-Model:** Similar to Waterfall but emphasizes testing at every stage, pairing development phases with associated testing phases.
  - **Spiral Model:** Combines iterative development with risk analysis: project proceeds through repeated cycles (spirals), refining requirements and design iteratively especially for complex / high-risk projects.
  - **Incremental Model:** The software is built and delivered in manageable, successive pieces.
  - **Rapid Application Development (RAD) :** emphasises quick delivery via prototyping, parallel development of modules, early user involvement. Good when requirements are fairly clear and modularization is possible.
  - **Agile** and other iterative/fast-cycle methodologies : more modern, flexible process

1.3 Example: Food-Delivery App

Let's say a company wants to build a food-delivery app.

Traditional Development (Waterfall-like): They first create a huge document of all requirements: login, restaurant list, payment, offers, GPS tracking, reviews, wallet, notifications etc. Only after months of development do they release the full product.

Problems:

- By the time it launches, market trends may have changed.
- The customer might realize the initial idea was incomplete.

[1] Source: *ASD Unit 1.pptx*, Slide 2.
[2] Source: *ASD Unit 1.pptx*, Slide 2.
[3] Source: *ASD Unit 1.pptx*, Slide 3.

- If users dislike the UI, changing it becomes expensive.
- Competitors might already launch better features.

Agile Software Development (The Solution): Imagine going on a trip with friends and a solid plan for days 1-4, but a lot of problems arise upon starting, like it starts raining or the train got late. If you strictly follow the original plan, the trip becomes stressful. What do you do? You start adapting. You make small plans daily. You decide together as you go. You try something, see if it works, and then adjust.

This is exactly what Agile does in software development. The same app is built in small usable pieces:

- **Sprint 1:** Basic login + browsing restaurants
- **Sprint 2:** Add ordering + cart
- **Sprint 3:** Add payment
- **Sprint 4:** Add GPS + tracking
- **Sprint 5:** Notifications, offers, improvements

After every sprint, the team shows a working product to the client, the client gives feedback, requirements change naturally, and the next sprint adapts accordingly.

2 Basics and Fundamentals of Agile Process Methods

Agile Software Development represents a major shift from traditional plan-driven (sequential) methodologies. Instead of massive upfront planning, Agile embraces changing requirements and focuses on delivering small, workable increments of software rapidly.[4]

In simple terms, Agile Software Development is not just a methodology. It is a **mindset** and a way of building software that welcomes change, involves customers closely, and delivers value step-by-step rather than all at once.

2.1 Values of Agile (The Agile Manifesto)

At the end of the 1990s, the software world was in chaos. Projects were collapsing everywhere, companies were losing money, and big organizations were trying to build software the same way they built bridges: with long, rigid plans that couldn't survive the real world. In February 2001, frustrated by this mess, seventeen of the world's smartest software thinkers (who believed in different lightweight methods like Scrum, XP, Crystal) met at a ski resort in Snowbird, Utah. They wrote four simple lines that captured everything they felt about building good software.[5]

In early 2001, a group of industry experts met to define the core values that formed the foundation of Agile. They established four core values that prioritize dynamic development over rigid structures:[6]

1. **Individuals and Interactions over Processes and Tools:** This value stresses that the strength of the team and how well they work together is more important than the tools or processes they use.
2. **Working Software over Comprehensive Documentation:** Agile prefers delivering functional software quickly rather than getting bogged down in lengthy documentation.
3. **Customer Collaboration over Contract Negotiation:** Agile values regular collaboration with customers over sticking strictly to contracts, making adjustments based on feedback.
4. **Responding to Change over Following a Plan:** In Agile, change is expected. Rather than rigidly following a plan that may no longer apply, Agile teams adapt based on new information.

Note: While there is value in the items on the right, Agile values the items on the left more.

2.2 Principles of Agile

The manifesto is further supported by twelve underlying principles that guide Agile teams:[7]

- **1.** Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- **2.** Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- **3.** Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

[4]Source: Martin, R.C., *Agile Software Development: Principles, Patterns, and Practices*, Chapter 1, p. 3-4.
[5]Source: *ASD Unit 1.pptx*, Slide 7.
[6]Source: Martin, R.C., *Agile Software Development*, Chapter 1, p. 5.
[7]Source: Martin, R.C., *Agile Software Development*, Chapter 2, p. 8-10.

- **4.** Business people and developers must work together daily throughout the project.
- **5.** Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- **6.** The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- **7.** Working software is the primary measure of progress.
- **8.** Agile processes promote sustainable development.
- **9.** Continuous attention to technical excellence and good design enhances agility.
- **10.** Simplicity—the art of maximizing the amount of work not done—is essential.
- **11.** The best architectures, requirements, and designs emerge from self-organizing teams.
- **12.** At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

2.3 Top Agile Methodologies in Software Development

Agile teams typically use frameworks as a starting point and customize elements to suit their requirements. Some widely used ones are:

- **Scrum:** Uses short sprints (1-4 weeks) with defined roles like Product Owner, Scrum Master, and Development Team. Includes events such as Daily Scrum and Sprint Review.
- **Kanban:** Uses a visual board and WIP limits to manage a continuous flow of work. Tasks move across columns helping teams reduce bottlenecks without fixed iterations.
- **Scrumban:** Combines Scrum's sprint planning with Kanban's flow and WIP limits, making it flexible and easy to adapt.
- **Lean Software Development:** Focuses on eliminating waste, improving efficiency, and delivering only what adds value.
- **DSDM (Dynamic Systems Development Method):** Time-boxed and iterative. Aims to deliver essential features quickly, prioritizing requirements based on business value.
- **XP (Extreme Programming):** Emphasizes strong engineering practices like pair programming, TDD, and continuous integration.
- **Crystal:** Adjusts its processes based on project size and criticality. Focuses on communication, collaboration, and frequent delivery.
- **FDD (Feature-Driven Development):** Builds the system through short iterations based on small, client-valued features using a structured five-step approach.

3 Stakeholders and Roles in the Agile Lifecycle

Agile processes utilize specific roles to ensure cross-functional collaboration, technical integration, and quality assurance.

3.1 The Onion Model of Stakeholders

In any software development project, a stakeholder is any person who is affected by the product or who can affect the product. The onion model of stakeholders explains how different people involved can be arranged in layers:

- **Center (The Product):** The most important part of the project.
- **First Layer (Primary Stakeholders):** Users, support staff, and system administrators. They directly use or operate the product and influence it the most.
- **Second Layer (Secondary Stakeholders):** Technical managers, sales, purchasing, and the legal team. Involved in decisions that affect how the product is built and used in the business.
- **Outermost Layer (Tertiary Stakeholders):** Sponsors, clients, regulators, suppliers, public, and media. They impact funding, approval, compliance, and public acceptance.

Example (Order Tracking Feature): Primary: Users, Delivery Partners. Secondary: Sales Team, Legal Team. Tertiary: Sponsors, Regulators. Most involved: Product Owner, Developers, Project Manager.

3.2 Key Roles and Responsibilities

- **Product Owner (PO):** The product owner is responsible for representing the stakeholders and the end customers. They define the product vision and roadmap, prioritize the product backlog, and accept or reject the resulting work. They manage stakeholders' expectations and ensure the team is delivering maximum business value.[8]
- **Development Members / Developers:** The developers are a cross-functional group that carries out the actual work to build the product incrementally. They collaborate daily in short, high-paced meetings to analyse, design, develop, test, and implement the user stories from the product backlog. The development team are usually structured yet flexible and self-organized while carrying their task execution.[9]
- **Stakeholders:** Stakeholders are individuals or groups who are impacted by the project. They may be within or external to the organization. They provide feedback on the requirements while reviewing progress of the project and also evaluate the final product. The stakeholders are usually engaged and kept in loop to allow the project team to meet their required business needs.[10]
- **Integrator:** The integrator is responsible for technical integration of work by the development team. They ensure that the software/hardware components fit together properly. The integrator also makes sure the system meets quality standards and integrates smoothly with external systems.[11]

[8] Source: *ASD Unit 1.pptx*, Slide 26.
[9] Source: *ASD Unit 1.pptx*, Slide 26.
[10] Source: *ASD Unit 1.pptx*, Slide 26.
[11] Source: *ASD Unit 1.pptx*, Slide 26.

- **Independent Auditor and Tester:** The independent tester objectively evaluates the system to ensure the quality of the product. They audit the product for bugs and flaws from an unbiased perspective. The tester identifies defects and works with the team to get them fixed.[12]
- **Scrum Master / Agile Coach:** Though not explicitly titled in every framework, this role acts as a binding force towards reaching the conclusion of the project. They promote collaboration, remove impediments, and encourage the team to improve its practices.[13]

3.3 Challenges in Agile Adoption

Transitioning to Agile is not without its hurdles. Common challenges include:[14]

- Frequent requirement changes may cause confusion or rework.
- Requires strong customer involvement; lack of timely feedback can delay progress.
- Communication gaps can break the Agile flow, especially in distributed or remote environments.
- Self-organizing teams may struggle without skilled members or clarity in roles.
- Continuous testing and frequent delivery increase workload and pressure if not properly planned.
- Resistance to cultural change within traditional, heavily tiered organizations.
- The difficulty of maintaining robust communication without extensive documentation.
- Unpredictable final costs and timelines due to evolving requirements.

[12]Source: *ASD Unit 1.pptx*, Slide 26.
[13]Source: *ASD Unit 1.pptx*, Slide 26.
[14]Source: Martin, R.C., *Agile Software Development*, Chapter 3.

4 The Lean Approach to Software Development

Lean software development is a translation of lean manufacturing principles and practices to the software development domain. It focuses heavily on maximizing value and minimizing waste.

Think of Lean like cleaning your study table: when books, chargers, and papers are scattered, you waste time finding what you need. After you clean it, work becomes smoother. Lean means cleaning the process so work flows smoothly.

4.1 Waste Management

In Lean, anything that does not add value to the final product is considered waste (*Muda*). Lean techniques focus on removing the seven main types of waste (over-production, waiting, transport, inventory, over-processing, motion, and defects). Common software wastes include:[15]

- **Extra Features:** Building features that are never or rarely used (e.g., building a "dark mode" before the customer actually asks).
- **Delays / Waiting Time:** Waiting for approvals, testing environments, or other team hand-offs.
- **Task Switching / Multitasking:** The cognitive overhead of assigning developers to multiple projects simultaneously.
- **Defects & Rework:** Bugs that escape into production, requiring rework.
- **Over-planning & Unnecessary documentation.**

4.2 Kaizen and Kanban

- **Kaizen (Continuous Improvement):** A Japanese business philosophy ("Today better than yesterday"). Employees at all levels work together to achieve regular, incremental improvements. In Agile, this is operationalized through Sprint Retrospectives to learn from every iteration. It is often associated with the 5S Framework (Sort, Straighten, Shine, Standardize, Sustain).
- **Kanban:** Meaning "card" or "visual signal" in Japanese. It is a visual workflow management system that promotes a pull system. It uses a board with columns (e.g., To Do $\rightarrow$ In Progress $\rightarrow$ Done) to map the flow of work. Its primary directive is to **limit Work In Progress (WIP)** to prevent bottlenecks, ensure a smooth flow of value delivery, and clarify exactly who is doing what.

4.3 Adding Process and Products that Add Value

Lean dictates that every step in the software development lifecycle (SDLC) must directly contribute to the product's value. If a documentation phase or a mandatory sign-off meeting delays deployment without tangibly improving the software's quality or alignment with business goals, it must be refined or eliminated.[16]

[15] Source: Martin, R.C., *Agile Software Development*, Lean Adaptations Section.
[16] Source: Mapping to KIIT Syllabus Unit 1: Lean Approach.

5 Agile Software Development Life Cycle

The Agile development cycle breaks the project into small, usable iterations. Requirements can evolve, and the team adapts every sprint. The 8 phases of the Agile SDLC are:

1. **Pre-Planning:** Define product vision, scope, and goals. Form the team, gather initial requirements, and create the Product Backlog.
2. **Planning:** Decide sprint length, estimate development speed, and discuss major modules.
3. **Release Planning:** Break the backlog into releases (each containing multiple sprints) with a rough timeline.
4. **Iteration / Sprint Planning:** Select user stories for the sprint, break them into estimated tasks, and decide the sprint goal.
5. **Sprint Execution:** Daily Scrum meetings, followed by continuous Design → Code → Test → Integrate cycles.
6. **Sprint Review:** Demonstrate working software to the customer, collect feedback, and identify changes.
7. **Sprint Retrospective:** Discuss what worked and what didn't to improve teamwork, tools, and processes.
8. **Backlog Refinement:** Update priorities, add new requirements, and re-estimate timelines.

5.1 Quality Development in Agile

Several key engineering practices ensure quality throughout the Agile SDLC:

- **Test-Driven Development (TDD):** Write a failing test → write code to pass → refactor. Keeps design simple and reduces defects.
- **Behavior-Driven Development (BDD):** Extends TDD by writing executable specifications in natural language (Given → When → Then).
- **Pair Programming:** Two developers on one machine (driver and navigator). Leads to cleaner design and constant code review.
- **Definition of Done (DoD):** A shared checklist confirming when a task is truly complete (coded, reviewed, tested, documented).
- **Peer Review:** Team members review each other's code to strengthen quality.
- **Trunk-Based Development:** Teams work on small branches that merge back into the main trunk quickly (ideally daily) to prevent merge conflicts.
- **Build and Test Automation & Continuous Integration (CI):** Code is integrated multiple times a day. Every integration triggers automated builds and tests, keeping the system in a working state.
- **Collective Code Ownership:** The entire team owns all code, allowing anyone to improve any part.

6 Differences Between Agile and Traditional Plans

6.1 Comparative Analysis

Traditional plans attempt to predict the entire project scope upfront. Agile plans operate on empirical process control, acknowledging that initial requirements will inevitably change.[17]

Feature / Aspect	Traditional Development	Agile Development
Approach	Linear, sequential	Iterative, incremental
Requirements	Fixed at the beginning	Can change anytime
Customer Involvement	Only at start & end	Continuous & frequent
Planning	Heavy upfront planning	Adaptive planning each sprint
Delivery	Final product delivered at end	Working product delivered every iteration
Flexibility	Low — changes are costly	High — changes are welcomed
Team Structure	Hierarchical	Self-organizing & cross-functional
Documentation	Heavy documentation	Minimal but sufficient documentation
Testing	After development	Continuous during sprint
Risk Handling	High risk (issues found late)	Low risk (early detection)
Feedback	Late feedback	Continuous feedback
Quality	Improved at the end	Improved throughout
Transparency	Low	High
When to Use	Stable requirements	Dynamic requirements

Table 1: Traditional vs. Agile Methodologies

6.2 Agile Plans at Different Lifecycle Phases

Unlike a traditional Gantt chart that assumes a rigid timeline, Agile planning happens at multiple nested levels (The Agile Planning Onion):[18]

1. **Product Vision:** The overarching goal set by the stakeholders.
2. **Product Roadmap:** High-level plan of upcoming features over months.
3. **Release Plan:** Grouping of features to be deployed in the next block of sprints.
4. **Iteration (Sprint) Plan:** Detailed technical tasks planned for the immediate 2-4 week cycle.
5. **Daily Plan:** The Daily Stand-up meeting where developers plan the next 24 hours.

[17] Source: Martin, R.C., *Agile Software Development*, Chapter 2, p. 12-14.
[18] Source: General Agile SDLC Principles / Syllabus Mapping.

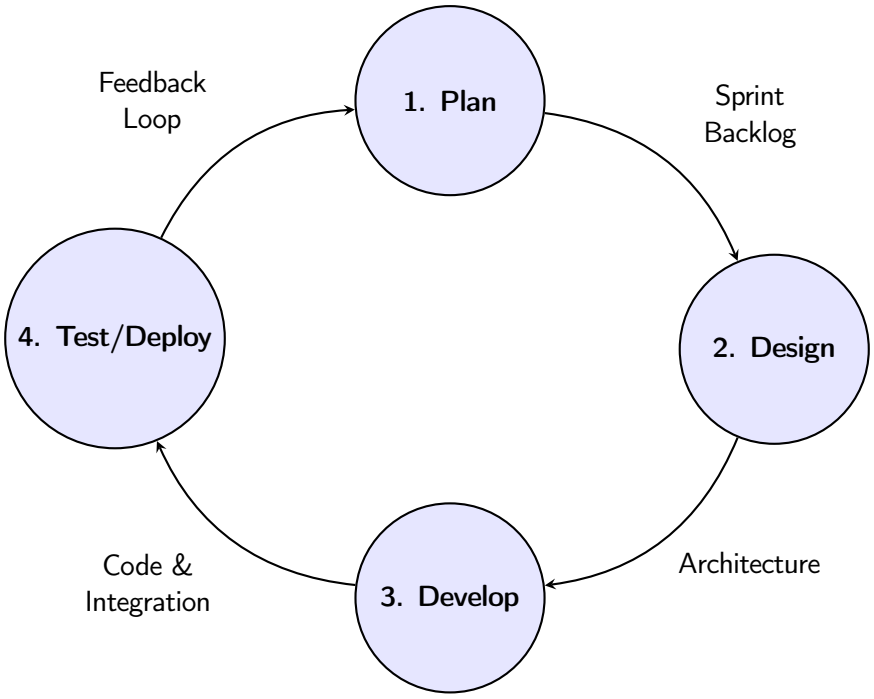

Figure 1: The Iterative Agile Lifecycle

7 Quality and Testing in Agile

7.1 How Agile Helps Build Quality

In Agile, quality is not a phase reserved for the end of the project; it is built into the product daily. Agile ensures high quality through:[19]

- **Frequent Integration:** Integrators ensure that code is merged and verified constantly, avoiding the "integration hell" common in traditional projects.
- **Test-Driven Development (TDD):** Writing automated tests before writing the actual code ensures that the codebase is always verifiable and resilient to regressions.
- **Refactoring:** The code structure is continually improved without altering its external behavior, reducing technical debt.
- **Customer Feedback:** Delivering working software frequently allows the independent auditor, tester, and stakeholders to validate the product against real-world needs, ensuring the team is building the *right* product.

7.2 Testing Plan Links Between Testing

An Agile test plan is dynamic. Instead of a static master test document, testing is linked directly to **User Stories**. Each user story must have predefined **Acceptance Criteria**. The independent tester and developers link unit tests, integration tests, and user acceptance tests directly to these criteria. This ensures a one-to-one traceability between a business requirement and the test that validates it.[20]

7.3 Understanding as a Means of Assessing Initial Status

Before Agile development begins, an "Iteration Zero" or inception phase is often used to assess the initial status of a project. This involves understanding the:

- **Business Value:** What problem are we solving?
- **Technical Feasibility:** Does the team have the architecture and tools needed?
- **Quality Benchmarks:** What constitutes a "Done" increment (Definition of Done)?

Assessing the understandability of the Agile process amongst the team is a primary assessment metric. If developers, integrators, and stakeholders do not share a unified understanding of Agile principles (like continuous integration and daily collaboration), the quality of the final product will severely degrade. Education and coaching (often by a Scrum Master) are utilized to establish this baseline for quality.[21]

[19] Source: Martin, R.C., *Agile Software Development*, Chapter 1, p. 15-20.
[20] Source: *ASD Unit 1.pptx*, Roles section - Independent Auditor and Tester, Slide 26.
[21] Source: Mapping to KIIT Syllabus Unit 1: Assessment & Quality.

8 Unit II - SDLC using Agile Methodology

Agile Methodology is defined as people collaborating together to build high-quality products that meet customer needs at a sustainable pace.[22]

Key Characteristics of Agile SDLC:

- The tasks are divided into time boxes (small time frames) to deliver specific features for a release.
- Each delivery is incremental in terms of features; the final delivery holds all the features required by the customer.
- An iterative approach is taken, and working software is delivered after each iteration.

9 The Agile Manifesto and Principles

In early 2001, a group of industry experts met in Snowbird, Utah, to discuss the current state of software development. The result of this meeting was the Agile Software Development Manifesto, which outlines the core values of agile methodologies.[23]

9.1 The Four Values of the Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

1. **Individuals and interactions** over processes and tools.
2. **Working software** over comprehensive documentation.
3. **Customer collaboration** over contract negotiation.
4. **Responding to change** over following a plan.

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

9.2 The Twelve Agile Process Principles

The manifesto is further supported by twelve guiding principles that define the agile mindset:[24]

- **Principle 1:** Highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- **Principle 2:** Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- **Principle 3:** Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. (*Working software is the primary measure of progress*).
- **Principle 4:** Business people and developers must work together daily throughout the project.

[22]Source: *Unit II.pptx*, Slide 2. Introduction to SDLC using Agile.

[23]Source: Martin, Robert C., *Agile Software Development: Principles, Patterns, and Practices*, Pearson Education, pp. 3-4.

[24]Source: *Unit II.pptx*, Slide 3 & 4; Martin, Robert C., pp. 6-8.

- **Principle 5:** Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- **Principle 6:** The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- **Principle 7:** Working software is the primary measure of progress.
- **Principle 8:** Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- **Principle 9:** Continuous attention to technical excellence and good design enhances agility.
- **Principle 10:** Simplicity - the art of maximizing the amount of work not done - is essential.
- **Principle 11:** The best architectures, requirements, and designs emerge from self-organizing teams.
- **Principle 12:** At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly (Frequent Review).

10 Extreme Programming (XP) Model

Extreme Programming (XP) is one of the most prominent Agile frameworks. The name of this model reflects the fact that it promotes taking best practices that have previously worked well in projects to extremes. This model is founded on a simple philosophy: "If something is known to be beneficial, why not put it to constant use?"

10.1 Good Practices Carried to the Extreme

Based on its core principle, XP proposes several key practices that must be taken to the extreme:

- **Code review:** Code review is beneficial because it allows for the most efficient detection and correction of problems. XP recommends pair programming as a method for achieving continuous review. Pair programming involves two programmers working together to code; while one writes code, the other reviews it.
- **Testing:** Testing code helps to eliminate problems and improves reliability. XP recommends Test-Driven Development (TDD) for continuously writing and executing test cases. Test cases are created based on user stories before writing any code to implement a feature. Only when these tests pass is a piece of code considered complete.
- **Incremental development:** Incremental development is beneficial since it helps to get user input, and the number of features produced is a dependable sign of progress. XP advises that the team create fresh increments every few days.
- **Simplicity:** Simple code produces high-quality results. It simplifies testing and debugging. As a result, one should aim to write the simplest code possible while ensuring that the essential functionality works. Efficiency, reliability, and maintainability should initially be ignored in favor of simplicity; reworking can incorporate those later.

10.2 Twelve Practices of Extreme Programming (XP)

Robert C. Martin emphasizes these twelve practices as the foundation of a robust agile team:[25]

1. **The Planning Game:** Quick planning between customer and developers to decide scope and priority based on estimates.
2. **Small Releases:** Frequent releases to production on a very short cycle (e.g., every 2 weeks) to gather real customer feedback.
3. **System Metaphor:** All team members share a common, simple story or architectural vision of how the system works.
4. **Simple Design:** The system is designed as simply as possible at any given moment; design only what is required for current needs and avoid complexity.
5. **Testing (Test-Driven Development - TDD):** Write tests before writing code to ensure correctness.
6. **Refactoring:** Continuous improvement of code structure and removing duplication without changing behavior.

[25]Source: Martin, Robert C., *Agile Software Development: Principles, Patterns, and Practices*, Pearson Education, Chapter 2, pp. 13-18.

7. **Pair Programming:** Two developers work together at one workstation to improve code quality and facilitate real-time code review.
8. **Collective Code Ownership:** Any developer can change any part of the code anywhere in the system at any time.
9. **Continuous Integration:** Frequently integrate and build code (many times a day) to detect issues early.
10. **40-Hour Work Week (Sustainable Pace):** Maintain a balanced workload to avoid burnout. Overtime ruins productivity in the long run.
11. **On-Site Customer:** A customer representative is available full-time with the team to answer questions and set priorities.
12. **Coding Standards:** Common rules and style guidelines emphasizing communication and readability through the code.

10.3 Applications of Agile/XP

- **Projects involving new technology or research projects:** Ideal because requirements change rapidly and unforeseen technical problems often need to be resolved.
- **Small Projects.**

11 The Scrum Framework

Scrum is one of the Agile development models that helps teams work together. It encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.

In the Scrum model, the entire project work is divided into small work parts that can incrementally be developed and delivered over time boxes called **Sprints**. At the end of each sprint, stakeholders and team members meet to assess the developed software increment, deploy it to the client site, obtain client feedback, and suggest necessary changes.[26]

11.1 Need of Scrum

Traditional project management often struggles with changing requirements. Scrum has become essential in modern software development as organizations demand faster, flexible, and customer-centric delivery models.

Reasons Why Scrum is Needed:

- Rapidly changing business requirements.
- Need for faster product delivery and reduced time-to-market.
- Encourages teamwork and ownership.
- Enhances product quality through regular testing and reviews.
- Early detection of risks and problems.
- Increased customer satisfaction due to regular visibility.
- Better predictability and planning accuracy.
- Adaptable for complex, uncertain, and innovative projects.

When Scrum Should be Used:

- Projects with frequently changing requirements.
- Products requiring continuous upgrades and improvements.
- Complex and uncertain development environments.
- Cross-functional collaborative teams.

11.2 Scrum Values

The success of Scrum depends on the team's proficiency in living five core values:

1. **Commitment:** Dedication to achieving the goals of the Sprint and teamwork.
2. **Courage:** The team has the courage to speak openly, take risks, and work on tough challenges.
3. **Focus:** Everyone concentrates on the sprint goals and the work of the Scrum Team.
4. **Openness:** Transparent sharing of progress, problems, and ideas among stakeholders and the team.

[26] Source: *Unit II.pptx*, Slides 10-15 (Scrum Practices and Working of Scrum).

5. **Respect:** Value and support each other to be capable, independent professionals.

Importance of Scrum Values: Living these values strengthens teamwork and trust, reduces conflicts, enhances productivity and motivation, and builds a positive, high-performing workplace environment.

11.3 Working of Scrum: Roles, Artifacts, and Events

The Three Roles:

- **Product Owner:** Represents the *Business side* of the development. The Product Owner is responsible for maximizing the value of the product and is the sole person responsible for managing the Product Backlog. Responsibilities include creating the product vision, stakeholder management, scope management, and release management. In every sprint, they consult with the team to define features, decide release dates, and reprioritize features.
- **Scrum Master:** Represents the *People side* of the development. Ensures Scrum is understood and enacted; acts as a servant-leader. Responsibilities include coaching the team toward the product vision, helping the PO with backlog refinement, facilitating Sprint Planning, helping the team self-organize, and facilitating the Daily Scrum, Sprint Review, and Sprint Retrospective.
- **Development Team:** Represents the *Technology side* of the development. A self-organizing, cross-functional, empowered, and autonomous group doing the actual work of delivering a potentially shippable Increment of high-quality software. They are collectively responsible for delivery. The recommended size is 6 to 7 members, with cross-functional expertise (QA, programming, UI design, testing, etc.).

The Three Artifacts:

- **Product Backlog:** An ordered list of everything that is known to be needed in the product.
- **Sprint Backlog:** The set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment.
- **Increment:** The sum of all the Product Backlog items completed during a Sprint and the value of the increments of all previous Sprints.

The Five Events (Ceremonies): Scrum ceremonies are structured meetings that provide a framework for teams to work within the methodology.

1. **The Sprint:** A time-box of one month or less during which a "Done", useable, and potentially shippable product Increment is created.
2. **Sprint Planning:**
    - *Purpose:* Plan the work to be completed in the upcoming sprint.
    - *When:* At the beginning of each sprint.
    - *Participants:* Product Owner, Scrum Master, Development Team.
    - *Activities:* The Product Owner presents prioritized backlog items. The team discusses, estimates the effort, and commits to realistically achievable tasks.
3. **Daily Scrum (Daily Stand-up):**
    - *Purpose:* Synchronize the team's activities and plan for the next 24 hours.

- *When:* Daily 15-minute meeting, typically at the same time and place.
- *Participants:* Development Team, Scrum Master (optional), Product Owner (optional).
- *Activities:* Each member answers: What did I do yesterday? What will I do today? Are there any impediments or blockers?

4. **Sprint Review:**
   - *Purpose:* Demonstrate the work completed during the sprint and gather feedback.
   - *When:* At the end of each sprint.
   - *Participants:* Product Owner, Scrum Master, Development Team, Stakeholders.
   - *Activities:* The team demonstrates the potentially shippable increment. Stakeholders provide feedback, and the PO updates the backlog based on feedback.
5. **Sprint Retrospective:** An opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

11.4 Core Scrum Practices

Scrum relies on a set of core practices to help teams deliver value incrementally:

- **Empirical Process Control:** Based on transparency, inspection, and adaptation.
- **Timeboxing:** Fixed time duration for events and development cycles.
- **Iterative Development:** Work delivered through repeated cycles (Sprints).
- **Incremental Delivery:** Frequent delivery of functional increments.
- **Prioritized Backlog:** Work arranged by value and urgency.
- **Cross-functional Self-organizing Teams:** Teams decide how to accomplish work.
- **Daily Scrum:** Short daily meeting for alignment and issue resolution.
- **Sprint Reviews & Retrospectives:** Demonstration and process improvement.
- **Definition of Done (DoD):** Shared understanding of completion.
- **Product Increment:** Usable and potentially shippable product after every sprint.

11.5 How Scrum Works – Step-by-Step Flow

1. **Product Vision:** Defined by the Product Owner.
2. **Product Backlog Creation:** High-level requirements prioritized by value.
3. **Sprint Planning:** Team selects items (User Stories) to work on.
4. **Sprint Execution:** Development, testing, and integration activities.
5. **Daily Scrum:** 15-minute stand-up meeting for progress and issues.
6. **Sprint Review:** Demonstration of increment to stakeholders.
7. **Sprint Retrospective:** Review team performance and improvements.
8. **Product Increment Release:** Working version delivered after each sprint.

11.6 Scrum Workflow Diagram

Below is a visual representation of the Scrum lifecycle showing how artifacts and events interact.[27]

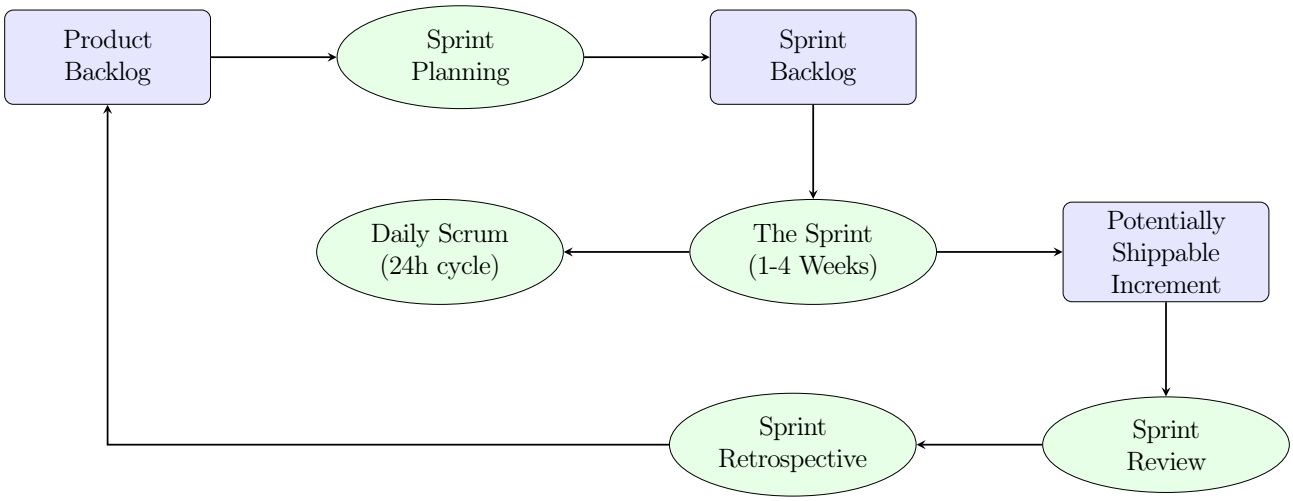

Figure 2: Working of Scrum: Lifecycle and Ceremonies

11.7 Applying Scrum and Scrum in the Organization

Scrum can be applied in project environments that require flexibility and rapid response to changes. Organizations must transition from command-and-control management styles to servant-leadership.

Steps to Apply Scrum Effectively: 1. Define Product Vision and Roadmap. 2. Form the Scrum Team. 3. Create and Prioritize Product Backlog. 4. Sprint Planning. 5. Execute Sprint. 6. Daily Scrum. 7. Sprint Review. 8. Sprint Retrospective. 9. Continuous Integration and Testing. 10. Repeat cycle until completion.

Key Success Factors: Active customer involvement, strong collaboration, leadership support, team empowerment, continuous improvement mindset, and use of Agile tools (Jira, Trello, Azure DevOps).

Advanced Practices and Scaling Methods: For large-scale projects, organizations use various scaling methods:

[27] Source: Standard Scrum Guide representation referenced in *Unit II.pptx*.

- **Scrum of Scrums (SoS):** Multiple Scrum teams work on the same product and meet to discuss integration points and cross-team dependencies.
- **Scaled Agile Framework (SAFe):** Enterprise-level scaling.
- **Large Scale Scrum (LeSS):** Simplified scaling for multiple teams working on one product.
- **Nexus Framework:** Integration management across teams.
- **Disciplined Agile Delivery (DAD):** Hybrid approach combining Agile methods.
- **Agile Release Trains (ART):** Teams working in alignment for product delivery.

Other Advanced Concepts: Release Planning & Roadmaps, DevOps integration & CI/CD pipelines, and specific Metrics (Velocity, Burndown/Burnup Charts).

Scrum and the Organization: HR policies, budgeting, and performance appraisals often need to adapt. Reward systems should shift from individual metrics to team-based outcomes.

- *Impact of Scrum:* Encourages transparency and collaboration, aligns teams with business goals, improves productivity, enables faster delivery, and supports better risk management.
- *Organizational Requirements:* Supportive leadership and culture, empowered cross-functional teams, open communication, and investment in training and tools.

12 Assessment: Impact Analysis of Scrum and its Use

To evaluate the impact of Scrum on an organization, software engineering metrics and impact analyses are utilized.[28]

An impact analysis of Scrum involves evaluating:

- **Delivery Speed (Velocity):** Tracking the number of Story Points burned down per sprint. Let S_i be the story points completed in Sprint i. The average velocity V over n sprints is given by the formula:
$$V = \frac{1}{n} \sum_{i=1}^{n} S_i$$
- **Quality Improvement:** Measured via defect density and escaped bugs. Scrum's emphasis on continuous testing usually leads to a drastic reduction in escaped bugs.
- **Customer Satisfaction:** Measured by the frequency of accepted increments during the Sprint Review.
- **Team Morale:** The Retrospective provides a qualitative measure of the team's psychological safety and engagement.

Impact Analysis Outcome: Successfully applying Scrum generally decreases the time-to-market and significantly mitigates risk since functionality is verified continuously rather than at the end of a multi-year project.

[28] Source: *Syllabus.docx* & *Unit II.pptx*, Assessment Requirements.

13 Comparison: Agile vs. Traditional Models

Understanding the differences between Agile and other traditional System Development Life Cycle (SDLC) models is crucial for determining when to apply Scrum.[29]

13.1 Agile Model vs. Iterative Waterfall Model

- The waterfall model is a structured approach that progresses through defined stages: requirements, analysis, design, coding, and testing, measuring progress through completed artifacts (documents).
- In contrast, the agile model focuses on delivering functional software frequently, assessing progress by the features developed.
- Despite their differences, agile teams often adopt a miniature waterfall cycle within each iteration. Thus, while distinct, the two methodologies can complement each other.
- **Risk Mitigation:** In contrast to the waterfall model, which yields only documents if a project is canceled midway, the agile model can provide customers with functional code that may already be operational.

13.2 Agile Model vs. RAD (Rapid Application Development) Model

The important differences between the agile and the RAD models are the following:

- While the agile model focuses on systematically developing incremental features, the RAD model prioritizes creating quick prototypes that are refined into production-quality code.
- Agile projects incrementally develop and deliver features, while the RAD approach emphasizes initially creating all application features in a rough form, followed by successive improvements over time.
- Agile teams showcase only completed, working work to customers, whereas RAD teams present simplified screen mock-ups and prototypes that often omit complex computations.

[29] Source: *Unit II.pptx*, Final Slide Snippets.

14 Unit III - Agile Product Management

14.1 Communication

Effective communication is the cornerstone of Agile methodologies.[30] Traditional software development relies heavily on extensive documentation handed off between isolated teams. In contrast, Agile promotes face-to-face communication, co-location, and continuous daily interactions.

- **Daily Stand-ups:** Brief, daily meetings to synchronize activities and create a plan for the next 24 hours.
- **Pair Programming:** Two developers work together at one workstation, providing continuous code review and immediate communication of architectural ideas.
- **On-Site Customer:** A business representative (Product Owner) is consistently available to the development team to clarify requirements and provide immediate feedback.

14.2 Planning and Estimation (Definition and Theory)

Planning in Agile is not a one-time upfront activity but an ongoing, iterative process. The purpose of planning is to find an optimal trajectory through the project, knowing that requirements and business priorities will shift.[31]

14.2.1 Theory of Estimation

Estimation in Agile avoids absolute time measurements (like exact hours or days) in favor of relative sizing.

- **User Stories:** Requirements are captured as User Stories, which are short, simple descriptions of a feature told from the perspective of the person who desires the new capability.
- **Story Points:** An abstract measure of effort required to implement a user story. It accounts for complexity, amount of work, and risk/uncertainty.
- **Velocity:** The number of story points a team can complete in a single iteration. Velocity provides a realistic, historically-backed metric to plan future iterations. By dividing the total remaining points by the team's velocity, project completion dates can be reliably forecasted.

14.2.2 Agile Planning and Estimation Levels

Planning and Estimation in Agile projects are generally done at three different stages:

- **Project Initiation Level**
- **Release Level**
- **Sprint Level** (i.e., an iteration within each Release)

[30] Source: Robert C. Martin, *Agile Software Development: Principles, Patterns, and Practices*, Pearson Education, p. 14.

[31] Source: Robert C. Martin, *Agile Software Development*, Chapter 2: Planning, pp. 15-18.

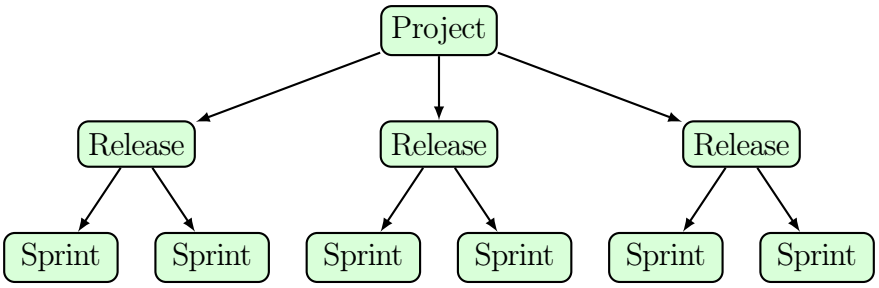

Figure 3: Agile Planning and Estimation Levels

Project Level: Planning and Estimation Project-level planning provides a big-picture view of the Agile project, including vision, roadmap, value delivery, risk coverage, and release strategy. The Product Owner plays a central role in defining goals, creating and prioritizing the Product Backlog, and guiding the team on strategic direction. This level of planning ensures predictability, stakeholder alignment, and risk mitigation through activities like backlog creation, release planning, workshops, and the use of ALM tools. High-level estimation techniques are used to forecast releases, timelines, staffing, and effort, and these estimates are continuously refined.

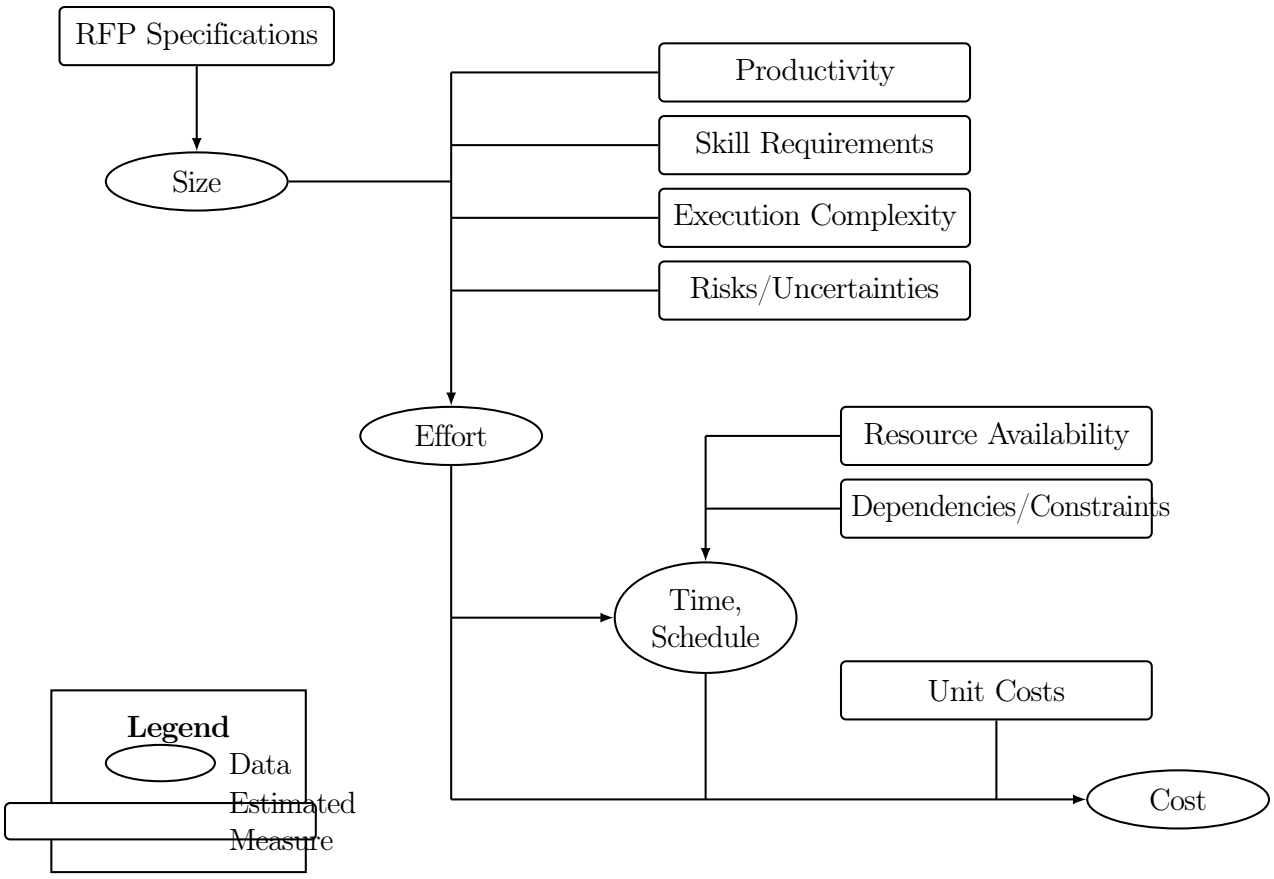

Figure 4: Project Level Estimation Architecture

Release Level: Planning and Estimation Release planning translates the product vision into time-boxed releases consisting of multiple sprints. It focuses on selecting, estimating, and prioritizing user stories from the Product Backlog to meet release objectives. Release planning enables incremental

delivery, early feedback, and alignment with business goals while allowing scope adjustments as the project evolves.

- Estimation is done using User Stories from the Product Backlog.
- User stories are estimated using Story Points.
- **Inputs considered:** Business priority, Team capacity, Previous velocity (if available).
- **Helps decide:** Total story points per release, Number of releases required.
- **Estimation Techniques Used:** Planning Poker (Fibonacci series), Wideband Delphi (consensus-based), Estimation by Analogy.

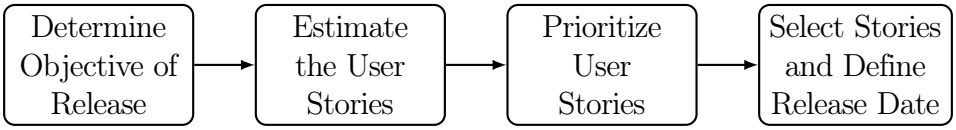

Figure 5: Release Level Planning Sequence

Sprint-Level: Planning and Estimation Conducted at the start of each Sprint. Defines the Sprint goal and User stories selected for the sprint. The team decides what can be delivered based on Team capacity and Velocity. User stories are broken down into tasks, forming the Sprint Backlog. Involves Product Owner, Scrum Master, and entire Scrum Team.

- Uses bottom-up estimation.
- Tasks are estimated in hours or days.
- Estimation is done by the task owner, based on the Definition of Done (DoD).
- Remaining task effort is updated daily and used to generate Sprint Burndown Charts.
- Ensures realistic sprint commitment and better tracking.

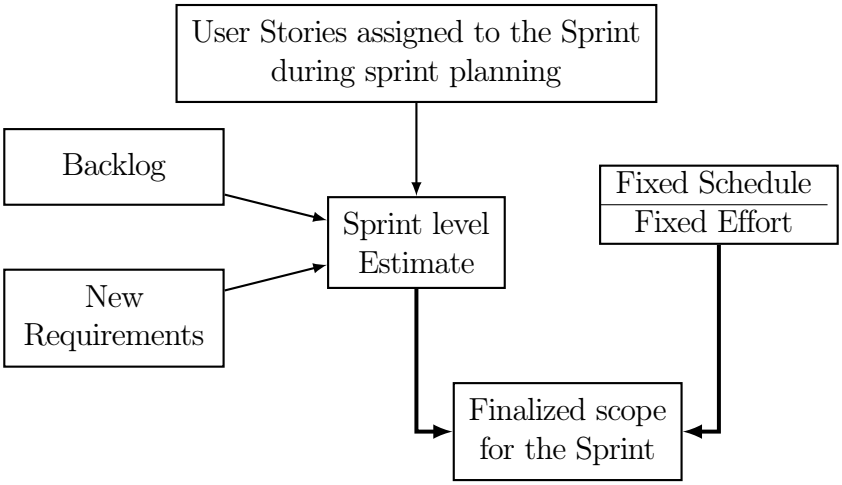

Figure 6: Sprint Level Estimation Process

15 Managing the Agile Approach

15.1 Monitoring Progress

Agile teams monitor progress transparently and continuously.[32] Progress is measured by the delivery of working software rather than the completion of phase deliverables.

- **Information Radiators:** Large, highly visible displays (like Kanban boards or physical burndown charts) used to track tasks and progress.
- **Iteration/Sprint Reviews:** At the end of every iteration, the team demonstrates the working software to stakeholders to measure actual progress against the planned goal.

15.2 Targeting and Motivating the Team

Agile builds projects around motivated individuals. Management's role shifts from assigning tasks to providing the environment and support the team needs.[33]

- **Self-Organization:** Teams are trusted to figure out the best way to accomplish their work.
- **Sustainable Pace:** Agile avoids crunch time and overtime. Teams work at a pace they can sustain indefinitely, reducing burnout and maintaining high quality.

15.3 Managing Business Involvement and Escalating Issues

The business (stakeholders, clients) must be continuously involved. The Product Owner manages this involvement by prioritizing the product backlog based on business value and ROI.[34]

- **Issue Escalation:** When impediments block the development team (identified during Daily Stand-ups), the Scrum Master or Agile Coach escalates and resolves them promptly, shielding the team from external distractions.

[32]Source: Unit III.pptx, Slide 3.
[33]Source: Robert C. Martin, *Agile Software Development*, p. 21.
[34]Source: Unit III.pptx, Agile Product Management Overview.

16 Quality, Risk, Metrics and Measurements

16.1 Overview of Agile Metrics

Measuring progress is essential in Agile project management, as metrics provide objective insight into team performance and areas for improvement.[35]

Traditional plan-driven metrics are often inadequate, so Agile relies on leading indicators that reflect team behavior, velocity, quality, and value delivery. These metrics help validate Agile practices, forecast progress, optimize planning, monitor quality, and measure customer value.

Effective agile metrics share some key traits:

- Lightweight and Actionable
- Customer-focused
- Trend-based
- Simple to measure
- Automated tracking and Real-time visibility

16.2 Common Agile Metrics

Agile metrics are broadly categorized into three levels:[36]

Category	Specific Metrics
Team-level Metrics	Sprint Burndown, Velocity, Cycle Time, Throughput, Escape Rate, Team Happiness / Morale
Product-level Metrics	Time-to-Market, Release Frequency, Defects, Technical Debt, Customer Satisfaction
Process-level Metrics	Value Delivered, Return on Investment (ROI), Cost per Iteration, Lead Time, Flow Efficiency

Table 2: Classification of Agile Metrics

16.3 Detailed Team-Level Metrics

- **Sprint Burndown:** Tracks completion of sprint backlog items over the sprint timeline. Useful for projecting delivery.
- **Velocity:** Number of backlog items (story points) a team completes per sprint on average. Helps estimate workload.
- **Cycle time:** The time taken from starting work on a backlog item to completion. Faster cycle time increases agility.
- **Throughput:** Total number of backlog items completed by a team in a given period (sprint, release, etc). Increased throughput drives delivery.

[35]Source: Unit III.pptx, Slide 4 (Exact Transcription).
[36]Source: Unit III.pptx, Slide 5.

- **Escape rate:** Percentage of committed backlog items not completed in a sprint. Lower escape rates improve predictability.
- **Happiness/Morale:** Subjective but important metric on team engagement and cohesiveness. Gather feedback frequently.

16.4 Detailed Product-Level Metrics

- **Time-to-market:** The elapsed time from initiating a new product or feature until it is released and adopted by customers. Faster time-to-market provides a competitive advantage.
- **Release frequency:** How often new versions of a product are released to customers. Shorter release cycles get value to customers faster through incremental deliveries.
- **Defects:** Count of defects reported on release versions. Defects impact user experience. Target zero production defects.
- **Technical debt:** Accumulated backlog of non-functional work needed to keep a product maintainable. Too much debt slows progress.
- **Customer satisfaction:** Subjective but essential rating of how happy customers are with a product. Gather continuous customer feedback.

16.5 Detailed Process-Level Metrics

- **Value delivered:** An estimate of the overall business value provided by agile initiatives through product capabilities released. A higher value increases ROI.
- **Return on investment (ROI):** Measures net benefit of agile projects by comparing value delivered to cost. Helps demonstrate success.
- **Cost per iteration:** The summed cost of resources, tools, and overhead needed to complete an iteration. Lower is better for scaling agile.
- **Lead time:** The total elapsed time for a backlog item to go from request to release. Faster lead time increases responsiveness.
- **Flow efficiency:** Assesses how smoothly work moves through agile process stages without stoppage or delay. Improving flow reduces waste.

17 Assessment: Measuring Risk and Quality (Numericals)

The following sections detail mathematical metrics used to frame software plans and assess quality and risk, complete with solved examples.[37]

17.1 Velocity

$$\text{Velocity} = \frac{\text{Total Story Points Completed}}{\text{Number of Sprints}} \tag{1}$$

Elements:

- **Total Story Points Completed:** Sum of story points finished
- **Number of Sprints:** Sprints considered for calculation

Solved Example:
Total = 18+22+20=60. Sprints = 3.

$$\text{Velocity} = \frac{60}{3} = 20 \text{ story points per sprint} \tag{2}$$

Practice Question:
A team completes 15, 25, 30, and 20 story points in four sprints. Find the average velocity.
Solution:
Total = 15+25+30+20=90. Sprints = 4.
Velocity = 90/4=22.5 story points per sprint

17.2 Sprint Burndown (Remaining Work)

$$\text{Remaining Work} = \text{Total Planned Work} - \text{Completed Work} \tag{3}$$

Elements:

- **Total Planned Work:** Story points committed
- **Completed Work:** Story points finished

Solved Example:
A sprint starts with 50 story points. After one week, 32 points are completed. Find the remaining work.

$$50 - 32 = 18 \text{ story points} \tag{4}$$

Practice Question:
A sprint starts with 40 story points. 27 are completed. Calculate the remaining work.
Solution:
40−27=13 story points

[37] Source: Unit III.pptx, Slides 12-15 (Assessments and Metrics).

17.3 Escape Rate

$$\text{Escape Rate} = \left(\frac{\text{Incomplete Committed Items}}{\text{Total Committed Items}} \right) \times 100 \tag{5}$$

Elements:

- **Incomplete Committed Items:** Planned but not completed
- **Total Committed Items:** Initially committed items

Solved Example:
A team commits to 16 user stories but completes only 12. Calculate the escape rate.
Incomplete $= 16 - 12 = 4$

$$\frac{4}{16} \times 100 = 25\% \tag{6}$$

Practice Question:
A team commits to 20 stories but completes 17. Find the escape rate.
Solution:
Incomplete $= 20 - 17 = 3$
$(3/20) \times 100 = 15\%$

17.4 Cycle Time

$$\text{Cycle Time} = \text{Completion Date} - \text{Start Date} \tag{7}$$

Elements:

- **Start Date:** When work begins
- **Completion Date:** When work ends

Solved Example:
A task starts on Day 3 and finishes on Day 9. Find the cycle time.

$$9 - 3 = 6 \text{ days} \tag{8}$$

Practice Question:
A task starts on Day 5 and finishes on Day 11. Calculate the cycle time.
Solution:
$11 - 5 = 6$ days

17.5 Throughput

$$\text{Throughput} = \text{Number of Items Completed per Time Period} \tag{9}$$

Elements:

- **Items Completed:** Stories/tasks finished
- **Time Period:** Sprint/week

Solved Example:
A team completes 14 user stories in a sprint. Calculate the throughput.
Throughput = 14 stories per sprint

Practice Question:
A team completes 18 backlog items in a sprint. Find the throughput.
Solution:
Throughput = 18 items per sprint

17.6 Time-to-Market

Time-to-Market measures the total duration taken from the inception of the development effort until the product is released to the customers.

$$\text{Time-to-Market} = \text{Release Date} - \text{Start Date} \tag{10}$$

Elements:

- **Start Date:** Beginning of development
- **Release Date:** Feature release date

Solved Example:
Scenario: Development started on 1st February and release happened on 2nd April. Calculate the time-to-market.

$$\text{Time-to-Market} = 60 \text{ days} \tag{11}$$

Practice Question:
Development starts on 10th January and release is on 25th February. Find the time-to-market.
Solution:
Days in January (from 10th to 31st inclusive) = 22 days
Days in February (up to 25th) = 25 days
Total Time-to-Market = 22+25 = 47 days.

17.7 Defect Density

Defect density is a measure of product quality, identifying the number of confirmed defects in a software module during a specific period of operation or development.

$$\text{Defect Density} = \frac{\text{Number of Defects}}{\text{Size of Software}} \tag{12}$$

Elements:

- **Number of Defects:** Bugs found
- **Size of Software:** Modules / KLOC (Kilo Lines of Code) / FP (Function Points)

Solved Example:
A software system has 12 defects across 4 modules. Calculate the defect density.

$$\text{Defect Density} = \frac{12}{4} = 3 \text{ defects per module} \tag{13}$$

Practice Question:
A system has 20 defects in 5 modules. Calculate defect density.

$$\text{Defect Density} = \frac{20}{5} = 4 \text{ defects per module} \tag{14}$$

17.8 Value Delivered

This metric evaluates the actual business value generated by the completed work items (user stories) during an iteration.[38]

$$\text{Value Delivered} = \sum \text{Business Value of Completed Stories} \tag{15}$$

Elements:

- **Business Value:** Assigned arbitrary or monetary value to each story by the Product Owner.
- **Completed Stories:** Finished work items that meet the Definition of Done.

Solved Example:
A sprint completes stories with business values 15, 25, and 10. Calculate the value delivered.

$$\text{Value Delivered} = 15+25+10 = 50 \tag{16}$$

Practice Question:
Stories with values 20, 30, and 40 are completed. Find the value delivered.

$$\text{Value Delivered} = 20+30+40 = 90 \tag{17}$$

17.9 Return on Investment (ROI)

ROI measures the profitability and financial risk of the project by comparing the value delivered to the cost incurred.

$$\text{ROI} = \left(\frac{\text{Value Delivered} - \text{Cost}}{\text{Cost}} \right) \times 100 \tag{18}$$

Elements:

- **Value Delivered:** Total business benefit (often in monetary terms).
- **Cost:** Total project or sprint cost.

Solved Example:
A project costs \$1,50,000 and delivers value worth \$2,00,000. Calculate the ROI.

$$\text{ROI} = \left(\frac{2{,}00{,}000 - 1{,}50{,}000}{1{,}50{,}000} \right) \times 100 = 33.33\% \tag{19}$$

Practice Question:
A project costs \$1,00,000 and delivers value \$1,40,000. Find the ROI.[39]

$$\text{ROI} = \left(\frac{1{,}40{,}000 - 1{,}00{,}000}{1{,}00{,}000} \right) \times 100 = \left(\frac{40{,}000}{1{,}00{,}000} \right) \times 100 = 40\% \tag{20}$$

[38] Source: Unit III.pptx, Slide 14.
[39] Source: Unit III.pptx, Slide 15.

17.10 Lead Time

$$\text{Lead Time} = \text{Delivery Date} - \text{Request Date} \tag{21}$$

Elements:

- **Request Date:** When work is requested
- **Delivery Date:** When delivered

Solved Example:
A feature request is raised on Day 4 and delivered on Day 16. Calculate the lead time.

$$16 - 4 = 12 \text{ days} \tag{22}$$

Practice Question:
A request is made on Day 7 and delivered on Day 21. Find the lead time.
Solution:
$21 - 7 = 14$ days

18 Unit IV - Requirements and Testing

18.1 Agile Requirements

In Agile software development, requirements are not treated as static documents to be finalized before coding begins. Instead, they are dynamic, evolving representations of user needs.

18.1.1 User Stories

User stories are a key component of agile software development. They are short, simple descriptions of a feature or functionality from the perspective of a user. User stories are used to capture requirements in an agile project and help the development team understand the needs and expectations of the users.[40]

Here are some key characteristics of user stories:

- **User-centric:** User stories focus on the needs of the user and what they want to achieve with the software.
- **Simple:** User stories are short and simple descriptions of a feature or functionality.
- **Independent:** User stories can stand on their own and do not rely on other user stories.
- **Negotiable:** User stories are open to discussion and can be refined and modified based on feedback from stakeholders.
- **Valuable:** User stories provide value to the user and the business.
- **Estimable:** User stories can be estimated in terms of time and effort required for implementation.
- **Testable:** User stories can be tested to ensure they meet the needs of the user.
- **Prioritized:** User stories are prioritized based on their importance to the user and the business goals.
- **Iterative:** User stories are developed iteratively, allowing for feedback and changes throughout the development process.
- **Consistent:** User stories follow a consistent format, making them easy to understand and work with.
- **Contextual:** User stories are written in a way that provides context to the development team, helping them understand the user's needs and goals.
- **Acceptance criteria:** User stories have clear and specific acceptance criteria that define when the story is considered "done" and ready for release.
- **Role-based:** User stories are written from the perspective of a specific user role, helping to ensure that the development team is building features that are relevant and useful to that user.
- **Traceable:** User stories are tracked and linked to specific features and functionality in the software, making it easy to trace back to the original user need.

In agile software development, user stories are typically written on index cards or in a digital format, and are used to drive the development process. The development team uses user stories to plan and prioritize work, estimate the effort required for implementation, and track progress towards

[40] Source: *Unit-iv.pdf*, Page 2.

completing the user stories. By using user stories in agile software development, teams can ensure that they are building software that meets the needs of the users and delivers value to the business.

Pattern of User Story: User stories are completely from the end-user perspective which follows the Role-Feature-Benefit pattern: *"As a [type of user], I want [an action], so that [some reason]"*

> **For example:** As the project manager of a construction team, I want our team-messaging app to include file sharing and information update so that my team can collaborate and communicate with each other in real-time as a result the construction project development and completion will be fast.

Writing User Stories: User stories are from a user perspective. So when user stories are written, users are given more importance during the process. Some points outlined which are taken into consideration during writing user stories like:

- Requirements
- Tasks and their subtasks
- Actual user
- Importance to user words/feedback
- Breaking user stories for larger requirements

With this also some other principles which are given importance during creating user stories are discussed below.

INVEST Principle of User story: A good user story should be based on INVEST principle which expresses the quality of the user story because in base a good software product is completely dependent upon a good user story. In 2003 INVEST checklist was introduced by Bill Wake in an article.

- **I**ndependent - Not dependent on other.
- **N**egotiable - Includes the important avoid contract.
- **V**aluable - Provide value to customer.
- **E**stimable - It should be estimated.
- **S**mall - It should be simple and small not complex.
- **T**estable - It should be evaluated by pre-written acceptance criteria.

3 C's in User Stories:

1. **Card** - Write stories on cards, prioritize, estimate and schedule it accordingly.
2. **Conversation** - Conduct conversations, Specify the requirements and bring clarity.
3. **Confirmation** - Meet the acceptance criteria of the software.

Working with User Stories: Below points represents working with user stories:

- Once the user story is fully written then it undergoes review and verification.
- In project workflow meetings it is reviewed and verified then added to the actual workflow.
- Actual requirements and functionality are decided based on the stories.
- User stories are scored based on their complexity and the team starts work based on user stories.

Advantages of using User Stories in Agile Software Development:

- User stories help to keep the focus on the user's needs and expectations, which leads to better customer satisfaction.
- User stories are easy to understand and can be created quickly, which speeds up the requirements gathering process.
- User stories are flexible and can be refined and modified easily as requirements change.
- User stories are independent, which makes it easier to prioritize and plan the work.
- User stories are small and manageable, which makes it easier to estimate effort and track progress.
- User stories promote collaboration between stakeholders, which leads to better communication and understanding.
- User stories help to reduce scope creep and feature bloat, as they focus on the essential needs of the user.

Disadvantages of using User Stories in Agile Software Development:

- User stories may not provide enough detail or clarity to fully capture the requirements.
- User stories may not be sufficient for complex or large-scale projects.
- User stories may be subjective and influenced by the biases of the stakeholders.
- User stories may not capture all of the requirements, which can lead to gaps in the software.
- User stories may not be suitable for projects with a high degree of technical complexity.
- User stories may not capture non-functional requirements, such as performance, scalability, or security, which are critical to the success of the software.
- User stories may not be appropriate for projects with a high degree of interdependence between features or components, as they are designed to be independent.
- User stories may not be suitable for teams that lack experience with Agile methodologies or have difficulty working collaboratively.

18.1.2 Backlog Management

Backlog management in Agile is the continuous process of creating, refining, and prioritizing a dynamic list of tasks-mainly user stories-to ensure the team focuses on high-value features. Managed primarily by the Product Owner, it involves regular grooming (refinement) sessions to break down complex tasks, estimate efforts, and align with stakeholder needs. This keeps the product backlog lean and actionable.

Key Components and Practices:

- **Product Backlog:** A master list of all desired features, improvements, and fixes.
- **Sprint Backlog:** A subset of top-priority items selected for the current development iteration, owned by the development team.
- **Backlog Refinement (Grooming):** Regularly updating the backlog to remove obsolete items, add new ones, and refine existing stories with clear acceptance criteria.

- **Prioritization Techniques:** Using methods like MOSCOW (Must-have, Should-have, Could-have, Won't-have) or RICE (Reach, Impact, Confidence, Effort) to rank items by business value.
- **Tools:** Using tools like Jira or Trello to maintain visibility.

Benefits of Effective Backlog Management:

- **Improved Efficiency:** Keeps the team focused on the most valuable tasks, reducing wasted effort.
- **Better Adaptability:** Allows teams to quickly pivot based on changing requirements or customer feedback.
- **Enhanced Quality:** Regular refinement ensures technical requirements are understood before development begins, improving product quality.
- **Reduced Scope Creep:** A well-managed, prioritized list prevents the backlog from becoming unmanageably large.

18.2 Agile Architecture: Feature Driven Development (FDD)

An Agile methodology for developing software, Feature-Driven Development (FDD) is customer-centric, iterative, and incremental, with the goal of delivering tangible software results often and efficiently. FDD in Agile encourages status reporting at all levels, which helps to track progress and results.

FDD allows teams to update the project regularly and identify errors quickly. Plus, clients can be provided with information and substantial results at any time. FDD is a favorite method among development teams because it helps reduce two known morale-killers in the development world: Confusion and rework.

First applied in 1997 during a project for a Singapore bank, FDD was developed and refined by Jeff De Luca, Peter Coad and others. The original project took 15 months with 50 people, and it worked; it was followed by a second, 18-month long, 250-person project.

How is FDD Different from Scrum? FDD is related to Scrum, but as its name implies, it's a feature-focused method (as opposed to a delivery-focused method). Features are a foundational piece of FDD; they're to FDD what user stories are to Scrum: Small functions that are, most importantly, client-valued.

"During FDD, a feature should be delivered every 2-10 days – which differs from Scrum, in which sprints typically last two, but sometimes four, weeks."

FDD values documentation more than other methods (Scrum and XP included), which also creates differences in the roles of meetings. In Scrum, teams typically meet on a daily basis; in FDD, teams rely on documentation to communicate important information, and thus don't usually meet as frequently. Another key difference is the end user; in FDD, the actual user is viewed as the end user, whereas in Scrum it's typically the Product Owner who is seen as the end user.

How Does FDD Work? Typically used in large-scale development projects, five basic activities exist during FDD:

1. Develop overall model
2. Build feature list
3. Plan by feature

4. Design by feature
5. Build by feature

An overall model shape is formed during the first two steps, while the final three are repeated for each feature. The majority (roughly 75%) of effort during FDD will be spent on the fourth and fifth steps - Design by Feature and Build by Feature.

How is a Feature Defined? In FDD, each feature is useful and important to the client and results in something tangible to showcase. And because businesses appreciate quick results, the methodology depends on its two-week cycle.

Stages of Feature-Driven Development:

- **Stage 0: Gather Data:** As with all Agile methodologies, the first step in FDD is to gain an accurate understanding of content and context of the project, and to develop a clear, shared understanding of the target audience and their needs. This data-gathering can be thought of as stage 0, but one that cannot be skipped. To compare product development with writing a research paper, this is the research and thesis development step.
- **Stage 1: Develop an overall model:** Continuing the research paper metaphor, this stage is when the outline is drafted. Using the "thesis" (aka primary goal) as a guide, the team will develop detailed domain models, which will then be merged into one overall model that acts as a rough outline of the system.
- **Stage 2: Build a features list:** Use the information assembled in the first step to create a list of the required features. Remember, a feature is a client-valued output. Make a list of features (that can be completed in two weeks' time).
- **Stage 3: Plan by Feature:** Enter: Tasks. Analyze the complexity of each feature and plan tasks that are related for team members to accomplish. This stage should also identify class owners, individual developers who are assigned to classes. Because every class of the developing feature belongs to a specific developer, someone is responsible for the conceptual principles of that class.
- **Stage 4: Design by Feature:** A chief programmer will determine the feature that will be designed and build. He or she will also determine the class owners and feature teams involved. By the end of the design stage, a design review is completed by the whole team before moving forward.
- **Stage 5: Build by Feature:** This step implements all the necessary items that will support the design. User interfaces are built, components detailed, and a feature prototype is created. The completed feature can be promoted to the main build.

Conclusion & Best Practices: Feature-Driven Development is a practical Agile approach suited for long-term, complex projects. It is a suitable choice for development teams seeking a simple but structured Agile method that is scalable and delivers predictable results.

- **Best Practices:** FDD emphasizes domain object modeling, developing by feature, individual class ownership, feature teams, inspections, regular builds, and configuration management.
- **Key Roles:** FDD uses specialized roles including a Project Manager, Chief Architect, Development Manager, Chief Programmers, and Class Owners.

- **Benefits:** It is highly scalable for large teams, providing improved visibility through status reporting at all levels, resulting in reduced confusion and faster, high-quality delivery.

18.3 Agile Risk Management

Agile risk management is a proactive, iterative process integrated into daily and sprint-level activities to identify, assess, and mitigate threats to project scope, schedule, and quality. By emphasizing continuous communication, frequent client feedback, and small, incremental deliveries, teams can quickly address risks, reducing the likelihood of failure.

Risk Defined: "Anything that could prevent or marginalize project success if it comes to fruition." Related, an Issue is when a Risk materializes.

Frequent Risks:

- The stakeholder requirements could be in conflict with each other.
- The estimate is not based on historical throughput.
- Team members are allocated to multiple projects.
- The project was approved without team buy-in.
- A 3rd party may not deliver their part.
- Example - Hardware - bad weather in Asia could delay equipment.

Traditional Risk Management Steps:

1. Identify
2. Quantify Impact
3. Quantify Probability
4. Create contingencies for high impact - high probability risks
5. Manage highest scoring risks

When do we identify risks? In our PMLC - usually once to Initiate the project, then revisit and update the risks after Planning. We can call this BURP - Big Upfront Risk Planning. Whereas Agile looks for risk throughout the lifecycle.

What Do We Identify? Risk ID Number, Risk Categories, Risk Description, Severity of Impact (1-5), Likelihood of Occurring (in %), and Risk Rating. Ultimately, we create Contingency Plans for Risks with a High Rating.

RISK CATEGORY	DEFINITION	SAMPLE QUESTIONS
Business Continuity	Includes risk associated with the duration, or impact, of an interruption of critical business processes and their associated people, vendors, systems, technology,	Will the introduction of a new product or service cause an interruption to existing business processes? Is there a Business Continuity Plan?
Compliance	Includes risks introduced to the company either during the project, or as a result of the project, associated with failure to meet regulatory requirements.	Is this a new process, product or business model that the Company has not had significant experience in implementing?
E-Commerce Risk	Risks associated with Internet interfaces	Is web site privacy adequately protected? Is the website and session security appropriately handled?
Financial	Includes operational risks associated with theft or misuse of Company or customer assets or information,	Is the forecast of the financial performance of the project adequate?
Fraud/Theft	Includes risks introduced to the company either during the project, or as a result of the project, associated with theft or misuse of Company or customer assets or information,	

Table 3: Risk Categories and Definitions

Risk Response Approach	Definition
Risk Avoidance	Eliminating the threat of a risk by eliminating the cause.
Mitigation (Controlling)	Reducing the consequences of a risk by reducing its severity of impact or likelihood of occurring.
Acceptance	Accepting the risk if it occurs.
Share or Transfer	Assigning the risk to another party by purchasing insurance or subcontracting.

Table 4: Risk Response Approaches

Agile Principles Address Risk:

- **Transparency** - Expose everything we are doing so we can see risks early.
- **Collaborative planning** - Harness the knowledge of the entire team and see more risks.
- **Customer involvement** - Mitigate customer risk by involving them throughout the lifecycle.

Project Envisioning Practices:

- Envisioning the product with the customer - The team and customer are synchronized on the need. Less risk of delivering the wrong product.
- Quantifying the value with the customer - Less risk of the team not supporting the project.

Project Planning Practices:

- Estimation based on history - Risk of estimate inaccuracy reduced since constants are involved in estimation.
- Work reviewed at the feature level for more detailed risk evaluation - Less chance of missing a risk since features are examined separately for technical risk.

Development/Implementation Practices:

- Daily Standup Meetings - Agile teams meet every 24 hours. Which mean risks are exposed every 24 hours.
- Product Demos Every 2 to 4 weeks - Constant exposure to the customer. Minimizes the risk of building to the spec but not to the need.

Project Tracking Risk Practices: Don't Manage Based on % of Plan Complete. Percentages are misleading. There is a risk that 1% takes as long as 99%. Instead manage by binary attributes: Complete or not complete. Less risk of overrun on construction tasks.

Key Aspects of Agile Risk Management:

- **Iterative Process:** Risk management is not a one-time, up-front activity but a recursive cycle (identify, analyze, plan, implement, monitor) that occurs during every sprint.
- **Risk Identification & Prioritization:** Risks are identified through team meetings (daily stand-ups), retrospectives, and backlog refinement. They are prioritized using a formula of probability times impact.
- **Key Techniques:** Sprint Planning & Retrospectives, Backlog Refinement, Continuous Delivery, Risk Register (A document maintained to monitor known risks throughout the project).

Common Agile Risks:

- **Scope Creep:** Constantly changing requirements.
- **Technical Debt:** Poorly maintained code causing future issues.
- **Resource Constraints:** Lack of team availability or expertise.
- **Unclear Requirements:** Misalignment with stakeholder expectations.

18.3.1 Implementing Best Practices for Quality Assurance in Agile

When it comes to software development, quality is everything. Quality Assurance (QA) is a conventional method that guarantees an effective product and service.

What Is An Agile QA Process? Most businesses have started the shift from the classical waterfall development methodology to an Agile method. Agile testing adds QA into the project as soon as feasible to anticipate problems, write and perform test cases, and reveal any gaps in time. With the project divided into iterative steps, QA engineers are ready to add focus to the development process and give rapid, continuous feedback. Sprints help the customer by delivering working software at

the beginning rather than later, expecting innovation, providing more reliable assessments in less time, and providing for course changes instead of completely wrecking the project.

6 Best Practices for an Agile QA Process:

1. **Risk Analysis:** An essential focus of any QA method is risk analysis. Risk analysis is interpreted as the means of classifying and evaluating potential risks and their result. Knowing all the potential outcomes of a project enables your team to discover precautionary measures that decrease the likelihood of occurrence.
2. **Test First also Test Often:** The Agile model strives to include QA at each step of the project's life cycle. Within each sprint, QA technicians test and retest the product with every new feature added. Testing first and often leads to the saving of time and resources.
3. **White-box Vs Black-box:** Black-box testing implies no knowledge of how a method does what it does. White-box testing allows the QA engineer to generate a more profound knowledge of the system's internals, enabling QA units to predict potential failure conditions and produce better test situations.
4. **Automate If Feasible:** Automation can help to maximize the effectiveness of your QA team better. Since regression examination can use a large portion of the QA team's time, self-regulation provides a way to secure previous deliverables. Implementing automation costs more upfront, but protects money in the long run.
5. **Know your Audience:** Knowing your target audience will help to develop the QA process that is more relevant. Tailoring the expansion and QA process around your user's requirements will allow your team to create value-driving applications.
6. **Teamwork Makes the Dream Work:** Behind each high-quality product, there is a unit of professionals. Managing the Agile QA process, engineers are the super-sleuths who root out obstacles and support the team to deliver high-quality results.

18.4 Agile Tools

Agile tools for software development facilitate iterative planning, team collaboration, and progress tracking. These tools support Kanban and Scrum frameworks through backlog management, sprint planning, and real-time reporting.

- **Jira:** The market leader for Scrum and Kanban, offering in-depth reporting (burndown charts, velocity) and issue tracking.
- **Trello:** A user-friendly, Kanban-based tool ideal for simple task management and visualization.
- **ClickUp:** Known for high customization, offering all-in-one productivity with built-in docs and goal tracking.
- **Asana:** Excellent for team collaboration, task tracking, and mapping out workflows.
- **Azure DevOps:** A comprehensive tool for planning, tracking, and CI/CD pipelines.
- **Confluence:** A collaboration tool for documentation that integrates directly with Jira.
- **Slack:** Widely used for real-time team communication and integrating with other tools.

Common Agile Frameworks Supported:

- **Scrum:** Focused on iterative, time-boxed sprints.
- **Kanban:** Focused on visualizing work and optimizing flow.
- **Extreme Programming (XP):** Emphasizes technical practices like Test-Driven Development (TDD) and pair programming.

18.5 Agile Testing

Agile Testing is a Type of Software Testing that follows the principles of agile software development to test the software application. All members of the project team, along with the special experts and testers, are involved in agile testing. Agile testing is not a separate phase, and it is carried out with all the development phases, which are requirements, design, coding, and test case generation.

It is an informal process that is specified as a dynamic type of testing. Customer satisfaction is the primary concern for agile test engineers.

Agile Testing Principles:

- **Shortening feedback iteration:** Continuous feedback minimizes the feedback response time, and the fixing cost is also reduced.
- **Testing is performed alongside:** Agile testing is not a different phase. It ensures that the features implemented during that iteration are actually done. Testing is not kept pending.
- **Involvement of all members:** It includes various developers, experts, and testing team members.
- **Documentation is weightless:** Uses reusable checklists to suggest tests and focus on the essence of the test rather than the incidental details. Lightweight documentation tools are used.
- **Clean code:** The defects that are detected are fixed within the same iteration.
- **Constant response:** Helps to deliver responses or feedback on an ongoing basis.
- **Customer satisfaction:** Customers are exposed to the product throughout the development process and can modify the requirements.
- **Test-driven:** Testing needs to be conducted alongside the development process to shorten the development time.

18.5.1 Agile Testing Methodologies

- **Test-Driven Development (TDD):** Tests are written before writing the actual code. It involves three steps: writing a unit test, coding to pass the test, and then refactoring the code.
- **Behavior Driven Development (BDD):** BDD is all about understanding and testing how users interact with the application. Focuses on creating features based on user behavior.
- **Exploratory Testing:** Testers are free to explore the software as they see fit, without predefined test scripts to uncover unknown risks.
- **Acceptance Test-Driven Development (ATDD):** The customer, developers, and testers work together to define the requirements and potential challenges before coding begins.
- **Extreme Programming (XP):** Focused on delivering high-quality software that meets customer needs through simplicity and frequent releases.

- **Session-Based Testing:** Involves structured, time-limited testing sessions (usually 45 to 90 minutes), where testers document their findings in a charter document.
- **Dynamic Software Development Method (DSDM):** An Agile framework providing principles for developers, users, and testers to collaborate.
- **Crystal Methodologies:** Focuses on the people involved in a project rather than on processes or tools, emphasizing communication and simplicity.

18.5.2 Agile Testing Strategies

1. **Iteration 0 (Initiate the Project):** The first stage where the initial setup is performed. This involves finding people for testing, preparing the usability testing lab, preparing resources. Important requirements and use cases are summarized, initial cost valuation planned, risks identified, and candidate designs outlined.
2. **Construction Iteration:** The major phase divided into:
    - *Confirmatory testing:* Verifies that the system meets stakeholder requirements (includes Developer testing like unit/integration tests and Agile acceptance testing).
    - *Investigative testing:* Detects the problems that are skipped during confirmatory testing. Focuses on load testing, security testing, and stress testing.
3. **Release (End Game):** Also known as the transition phase. This phase includes full system testing, acceptance testing, training end-users, support people, marketing of the product release, back-up and restoration, and finalization of the system documentation.
4. **Production:** The product is finalized in this stage after the removal of all defects and issues raised.

18.5.3 Agile Testing Quadrants

- **Quadrant 1 (Automated):** Focuses on the internal quality of code which contains test cases executed by engineers. Technology-driven (Unit testing, Component testing).
- **Quadrant 2 (Manual and Automated):** Focuses on customer requirements. Business-driven (Pair testing, Testing scenarios and workflow, User Stories and Prototypes).
- **Quadrant 3 (Manual):** Provides feedback to the first and second quadrant. (Usability testing, Collaborative testing, User Acceptance Testing (UAT), Pair testing with customers).
- **Quadrant 4 (Tools):** Focuses on the non-functional requirements of the product like performance, security, and stability. (Security testing, Scalability testing, Infrastructure testing, Data migration testing, non-functional stress/load testing).

Types of Security Testing: Vulnerability Scanning, Security Scanning, Penetration Testing, Risk Assessment, Security Auditing, Ethical Hacking, Posture Assessment, Application Security Testing, Network Security Testing, Social Engineering Testing.

Performance Testing Definitions:

- *Scalability Testing:* Tests capability to scale up or scale down the number of user request load. It ensures a software product can manage the scheduled increase in user traffic and data volume.

- *Stress Testing:* Verifies the stability and reliability under the burden of some load conditions. It tests beyond the normal operating point to ensure the system does not crash under crunch situations (Torture Testing).
- *Load Testing:* Determines the performance of a system under real-life-based normal and extreme load conditions when multiple users use it at the same time.

18.5.4 Agile Testing Life Cycle

1. **Impact Assessment:** Also known as the feedback phase where inputs and responses are collected from users and stakeholders.
2. **Agile Testing Planning:** Developers, customers, test engineers, and stakeholders team up to plan testing process schedules and deliverables.
3. **Release Readiness:** Test engineers review the fully created features and test if they are ready to go live or need to be sent back.
4. **Daily Scrums:** Daily morning meetings to check on testing and determine objectives for the day.
5. **Testing Agile Review:** Weekly meetings with stakeholders to evaluate and assess the progress against the goals.

Agile Test Plan: Written and updated for every release. It includes: Test Scope, Testing instruments, Data and settings, Approaches/strategies, Skills required, New functionalities, Levels/Types of testing based on complexity, Resourcing, Deliverables, Infrastructure Consideration, Load/Performance Testing, Mitigation or Risks Plan.

Benefits of Agile Testing: Saves time and money, Reduces documentation, Enhances software productivity, Higher efficiency (by dividing work into small parts), Improve product quality through regular user feedback.

Limitations of Agile Testing: Project failure (if key members leave), Limited documentation (makes it difficult to predict expected results), Introduce new bugs (due to repeated modifications), Poor planning (challenging to predict costs from day one), No finite end (easy to get sidetracked without a clear vision).

Challenges During Agile Testing: Changing requirements, Inadequate test coverage (due to continuous integration), Tester's availability (lacking skills for API/Integration testing), Less Documentation, Performance Bottlenecks (developer ignores end-user requirements), Early detection of defects, Skipping essential tests due to time constraints.

Risks During Agile Testing: Automated UI slow to execute, Use a mix of testing types required for quality, Poor Automation test plan, Lack of expertise, Unreliable/brittle tests leading to false positives.

18.6 User Acceptance Testing (UAT)

User Acceptance Testing (UAT) serves the purpose of ensuring that the software meets the business requirements and is ready for deployment by validating its functionality in a real-world environment. It allows end-users to test the software to ensure it meets their needs and operates as expected, helping to identify and fix any issues before the final release.

Who performs UAT? It is typically performed by end-users or clients who will ultimately use the software in their daily operations.

What is the purpose of UAT? To identify bugs in software, systems, and networks that may cause problems for users. Users are allowed to interact with the software before its official release to see if any features were overlooked.

Basic guidelines of UAT: When doing UAT, it is important to test with users in different locations, not just on different devices. Using email is often more reliable than social media for communication.

Acceptance criteria attributes: Completeness, Accuracy, User-friendliness, Performance, Reliability, Security, Scalability, Compatibility.

Types of User Acceptance Testing:

- **Beta Testing:** The application is tested in a natural environment by external users to gather feedback, reducing risks and failures.
- **Black Box Testing:** End-users or testers evaluate specific functionalities without knowing the internal workings or code structure, checking inputs against expected outcomes.
- **Operational Acceptance Testing (OAT):** Evaluates operational readiness before release (e.g., backup plans, client preparation, support cycles, security checks).
- **Contract Acceptance Testing:** Testing developed software against predefined and agreed-upon criteria and specifications defined in the project contract.
- **Regulation Acceptance Testing:** Ensures compliance with rules set by governing associations (Compliance AT).
- **Alpha Testing:** Tested internally before users get a hold of the product to validate feature maturity and ensure it passes acceptance tests before beta.

How to perform User Acceptance Testing (UAT):

1. **Requirements Analysis:** Analyzes business requirements (Business Use Cases, BRD, SRS, Process Flow Diagrams).
2. **UAT Test Plan Creation:** Outline the test strategy, entry criteria, exit criteria, and test scenarios.
3. **Identify Test Scenarios:** Identify scenarios with respect to business requirements.
4. **Create UAT Test Cases:** Create cases covering most scenarios based on business use cases.
5. **Prepare Test Data:** Best practice is to use live data; testers should be familiar with database flow.
6. **Test Run:** Execute cases and report bugs. Re-test once fixed.
7. **Confirm Business Objectives:** UAT testers sign off the mail to ensure the product is good to go for production.

Challenges of User Acceptance Testing: Misreporting Activities by potential users, Proper Example to demonstrate performance aspects, Proper Evaluation of handled information, Usability demonstration, Proper Balance between user input and costs constraints, Limitations of Actions Performed by User (e.g., expiration dates to prevent wasting data).

Tools for User Acceptance Testing:

- **Marker.io:** Report visual bugs straightforwardly into your devices without leaving your site.
- **FullStory:** Enables clients to track and screen every client action; visualizes user acceptance.
- **Hotjar:** Uncovers internet-based conduct, providing a '10,000-foot view' via an online database of people viewing your website.
- **CrazyEgg:** A web-based device that screens individual pages, breaking down where guests clicked.
- **Qualaroo:** A library that allows users to easily test Web Apps.
- **Sentry:** A web interface allowing users to write acceptance tests on their own.

Exit Criteria for User Acceptance Testing:

- **Confidence:** High level of confidence that the proposed user has enough knowledge and skill set to perform a task effectively.
- **Proper Execution:** Tests show users can contribute fully to existing tasks successfully.
- **Lesser Defects:** If there are more errors than expected but they show the program is easy to learn, it can be a positive sign. Fewer errors indicate successful early security measures.
- **No Critical Defects:** Project managers must ensure all critical defects found during testing are resolved within one month after launch before releasing the final copy.

19 Agile Review

19.1 Agile Metrics and Measurements

Agile metrics focus on the delivery of value and team sustainability rather than purely tracking hours spent.

Table 5: Agile Metrics in Practice

Practice	**Aim**	**Based on**	**Beneficiary**	**When**
Software size	Estimation of size/effort	User stories	Project Manager	At start of each iteration
Velocity	Productivity of team	User stories	Project Manager	At end of each iteration
Burndown	Progress monitoring	User stories	Team	At end of each iteration
Cumulative Flow	Lead time and WIP queue depth	Work in progress	Top managers/customers	At end of each iteration
Responding to change	Ability of team to handover quality	Defects fixing cost	Project Manager	At end of/after project
Earned business value	Monitoring bv delivered	Business value	Top managers/customers	As each feature is delivered
Total estimation effort	Planning and budgeting	User stories and reworks	Top managers/customers	At beginning of project
Story estimation	Time spent on story estimation	User stories	Team	Beginning of iteration
Requirements ambiguity	Misinterpreted requirements	User stories	Team	End of iteration
Unfinished stories	Identify problems with stories	User stories	Team	End of iteration
Number of impediments	Identify impediments team faced	Impediments in retrospectives	Team	During retrospectives
Work-in-progress	Items being actively worked on	User stories on kanban board	Team	During iteration

Continued on next page

Table 5 – continued from previous page

Practice	**Aim**	**Based on**	**Beneficiary**	**When**
Unplanned changes	Changes able to process in increment	Changes to user stories	Team	During increment
Employee satisfaction	Individual satisfaction	Questionnaires/Surveys	Individuals	End of iteration

Table 6: Software Metrics and Tools

Software Metrics	Usage in ASD	Measurement Tools
Customer satisfaction	Measures the outcome of customer satisfaction	Customer feedback, Net promoter score
Business value delivered	Helps the product managers to link their products with their business value	Customer feedback
On-time delivery	Helps in measuring customer expectations regarding the timely delivery of the product	Burn-down charts, Burn-up charts
Quality	Measures the conformance of the project with the quality standards	Number of defects
Productivity	Helps managers to understand how much the team is delivering	Burn-up charts
Predictability	Measures the amount of work the team has to perform to complete all the planned work	Velocity trend charts
Process improvement	Helps the project managers understand how the proposed adjustments are impacting productivity	Cumulative flow charts
Project visibility	Helps in sharing project updates and progress with all stakeholders	Dependency charts
Project scope	Helps the project manager and their teams to set and accomplish goals over a specific period	Burn-down charts, Kanban boards
Budget versus actual cost	Helps compare the budget with the actual cost of the project and forecast revenues and expenses	Cumulative flow charts, Dependency charts

Table 7: Software Metrics and Organizational Levels

Software Metrics	Description	Organizational Level
Story point estimation	Measures the outcome of customer satisfaction.	Team
Velocity	Helps the product managers to link their products with their business value.	Team
Sprint burn-down	Helps in measuring customer expectations regarding the timely delivery of the product.	Program

Continued on next page

Table 7 – continued from previous page

Software Metrics	Description	Organizational Level
Defects in production	Measures the conformance of the project with the quality standards	Team
Planned velocity	Helps managers to understand how much the team is delivering.	Program
Program velocity per sprint	Measures the amount of work the team has to perform to complete all the planned work	Program
Number of features per PI	Helps the project managers understand how the proposed adjustments are impacting productivity.	Program
Planned program velocity per sprint	Helps in sharing project updates and progress with all stakeholders.	Program
Release burn-down	Helps the project manager and their teams to set and accomplish goals over a specific period.	Program
Test Coverage	Helps the project manager to compare the budget with the actual cost of the project	Program

Software Quality Attribute	**Quality Metrics & Description**
Maintainability	Measured by MTTR (Mean time to repair) and MTTF. MTTR is the average time to repair a system technically or mechanically.
Availability	Measured as a relation of MTTF and MTTR.
Reliability	Inverse of MTTF.
Portability	Ratio of successful ports to total ports.
Testability	Measured by observability, simplicity, modularity, and stability.
Reusability	Weighted summation of Maintainability, Adaptability, Documentation, Complexity, and Availability.

Table 8: Quality Attributes and Metrics

19.1.1 The Agile Approach to Estimating and Project Variables

Estimation: Schedule, Scope, Cost, and Effort are the four major variables that typically control Software projects. Any changes in any of these variables can have an effect on a project. Estimation is a process to forecast these variables to develop or maintain software based on the information specified by the client.

There are three main challenges faced during estimation i.e., Uncertainty, Self-knowledge, and Consistency of Method used for Estimation. Usage of standardized and scientific estimation methods for estimating size, effort, and schedule, helps towards maintaining minimal variance between the planned estimates and actual values thereby achieving maximum estimation accuracy. This provides

a better client experience. All estimation needs for a project cannot be determined by a single method. It is important to have different methods of estimation for different stages.

Planning and Estimation in Agile projects bring a lot of focus on preparation and forecasting. Both these activities are done keeping business context in mind and measurable value delivery is committed to the client. Therefore, it is recommended to have required planning and estimation in Agile from the start of the project, in order to ensure better risk coverage and higher predictability.

During the Project Initiation stage, functional requirements are at a high level, or the details of requirements are not well-formulated and documented. In the Product Backlog, many requirements are still at the Epic or Feature level. At this point, estimation is required as it will help in the prioritization and planning of the Releases. Different estimation techniques can be used at this stage. Some of them are:

- QFPA (Quick Function Point Analysis)
- Use Case Points
- Complexity based estimation
- COCOMO (Constructive Cost Model)
- COSMIC FP

During the Project Initiation stage, estimation should be done as a group activity, where results should be compared and disagreements are resolved. All the project assumptions and risks should be highlighted clearly. Typically, the team for performing this activity will include:

- Product Owner
- Clients & Business Stakeholders
- Project Manager
- Developers and Designers
- Testers

19.2 Assessment: Estimating testing plans for Agile S/W

19.2.1 Unit IV Numericals

Estimation using Function Point Analysis -

Question 1 -

A project has the following components:

- External Inputs (EI) = 12 (Average complexity, weight = 4)
- External Outputs (EO) = 8 (Average complexity, weight = 5)
- External Inquiries (EQ) = 6 (Average complexity, weight = 4)
- Internal Logical Files (ILF) = 5 (Average complexity, weight = 10)
- External Interface Files (EIF) = 3 (Average complexity, weight = 7)

If the Value Adjustment Factor (VAF) = 1.05, calculate the Adjusted Function Points (AFP).

Solution:

Step 1: Calculate Unadjusted Function Points (UFP)

$$UFP = (12 \times 4) + (8 \times 5) + (6 \times 4) + (5 \times 10) + (3 \times 7)$$

$$UFP = 48 + 40 + 24 + 50 + 21$$

$$UFP = 183$$

Step 2: Calculate Adjusted Function Points

$$AFP = UFP \times VAF$$

$$AFP = 183 \times 1.05$$

$$AFP = 192.15 \approx 192 \text{ FP}$$

Question 2 -

A small Agile project includes:

- 15 EI (Low complexity, weight = 3)
- 10 EO (Low complexity, weight = 4)
- 4 ILF (Low complexity, weight = 7)
- 2 EIF (Low complexity, weight = 5)
- 6 EQ (Low complexity, weight = 3)

If VAF = 0.95 calculate AFP.

Solution:

Step 1: Calculate Unadjusted Function Points (UFP)

$$UFP = (15 \times 3) + (10 \times 4) + (4 \times 7) + (2 \times 5) + (6 \times 3)$$

$$UFP = 45 + 40 + 28 + 10 + 18$$

$$UFP = 141$$

Step 2: Calculate Adjusted Function Points

$$AFP = UFP \times VAF$$

$$AFP = 141 \times 0.95$$

$$AFP = 133.95 \approx 134 \text{ FP}$$

Question 3 - Capacity Assessment

Given a sprint parameter set where Required Effort is calculated as 240 hours and the Team Capacity is 240 hours. Evaluate sprint completion context.

Conclusion

Required effort = 240 hours
Team capacity = 240 hours

Yes, the sprint can be completed exactly within capacity.

Question 4 -

A project has:

- 20 Simple
- 15 Medium
- 10 Complex

Productivity = 4 hours per point

a) Calculate total effort
b) If 1 person works 160 hours/month, estimate number of persons required to finish in 2 months.

Solution -

Step 1: Complexity Points

$$=(20\times1)+(15\times3)+(10\times5)$$

$$=20+45+50$$

$$=115 \text{ points}$$

Step 2: Effort

$$=115\times4$$

$$=460 \text{ hours}$$

Step 3: Capacity in 2 Months 1 person capacity in 2 months $=160\times2$

$$=320 \text{ hours}$$

Persons required:

$$=460/320$$

$$=1.44$$

$$\approx2 \text{ persons required}$$

Question 5 -

An Agile release includes:

- 25 Simple
- 12 Medium
- 8 Complex

Productivity = 5 hours per point
Team velocity = 50 complexity points per sprint

a) Calculate total points
b) Number of sprints required

Solution -

Step 1: Total Points

$$=(25\times1)+(12\times3)+(8\times5)$$

$$=25+36+40$$

$$=101 \text{ points}$$

Step 2: Number of Sprints

$$=101/50$$

$$=2.02$$

$$\approx 3 \text{ sprints required}$$

19.3 Unit IV Numericals - Pt. 2

19.3.1 Estimation using Function Point Analysis (FPA)

Question 1
A project has the following components:

- External Inputs (EI) = 12 (Average complexity, weight = 4)
- External Outputs (EO) = 8 (Average complexity, weight = 5)
- External Inquiries (EQ) = 6 (Average complexity, weight = 4)
- Internal Logical Files (ILF) = 5 (Average complexity, weight = 10)
- External Interface Files (EIF) = 3 (Average complexity, weight = 7)

If the Value Adjustment Factor (VAF) = 1.05, calculate the Adjusted Function Points (AFP).

Solution:
Step 1: Calculate Unadjusted Function Points (UFP)

$$UFP=(12\times4)+(8\times5)+(6\times4)+(5\times10)+(3\times7)$$

$$UFP=48+40+24+50+21=183$$

Step 2: Calculate Adjusted Function Points (AFP)

$$AFP=UFP\times VAF$$

$$AFP=183\times1.05=192.15\approx192 \text{ FP}$$

Question 2
A small Agile project includes:

- 15 EI (Low, weight = 3), 10 EO (Low, weight = 4), 4 ILF (Low, weight = 7), 2 EIF (Low, weight = 5), 6 EQ (Low, weight = 3)

If VAF = 0.95, calculate AFP.

Solution:
Step 1: UFP

$$UFP=(15\times3)+(10\times4)+(4\times7)+(2\times5)+(6\times3)=141$$

Step 2: AFP

$$AFP = 141 \times 0.95 = 133.95 \approx 134 \text{ FP}$$

Question 3

An Agile team estimates 250 Function Points for a project. If productivity rate = 5 FP per person-month, estimate the required effort.

Solution:

$$Effort = \frac{\text{Total FP}}{\text{Productivity}} = \frac{250}{5} = 50 \text{ person-months}$$

Question 4

A project has UFP = 300. The total Degree of Influence (DI) from 14 General System Characteristics is 40. Given formula: $VAF = 0.65 + (0.01 \times DI)$. Find AFP.

Solution:

Step 1: Calculate VAF

$$VAF = 0.65 + (0.01 \times 40) = 1.05$$

Step 2: AFP

$$AFP = 300 \times 1.05 = 315 \text{ FP}$$

Question 5

An Agile product backlog contains: 20 EI (Avg, 4), 12 EO (Complex, 7), 10 EQ (Avg, 4), 8 ILF (Complex, 15), 5 EIF (Avg, 7). If VAF = 1.08, compute UFP, AFP, and number of Sprints if 1 Sprint handles 40 FP.

Solution:

Step 1: UFP

$$UFP = (20 \times 4) + (12 \times 7) + (10 \times 4) + (8 \times 15) + (5 \times 7) = 359$$

Step 2: AFP

$$AFP = 359 \times 1.08 = 387.72 \approx 388 \text{ FP}$$

Step 3: Number of Sprints

$$\text{Sprints} = 388/40 = 9.7 \approx 10 \text{ Sprints}$$

19.3.2 Use Case Points (UCP) Estimation

Weightage Tables:

- **Unadjusted Actor Weight (UAW):** Simple = 1, Average = 2, Complex = 3.
- **Unadjusted Use Case Weight (UUCW):** Simple ($\leq$3 transactions) = 5, Average (4–7) = 10, Complex ($\geq$8) = 15.

Formulas:

- $UUCP = UAW + UUCW$
- $UCP = UUCP \times TCF \times ECF$
- $Effort = UCP \times$ Productivity Factor

Question 1
A project has: 3 Simple Actors, 2 Average Actors, 1 Complex Actor; 4 Simple Use Cases, 3 Average Use Cases, 2 Complex Use Cases. TCF = 1.1, ECF = 0.9, Productivity Factor = 20. Calculate total effort.

Solution:
Step 1: UAW $= (3\times1)+(2\times2)+(1\times3) = 10$
Step 2: UUCW $= (4\times5)+(3\times10)+(2\times15) = 80$
Step 3: UUCP $= 10+80 = 90$
Step 4: UCP $= 90\times1.1\times0.9 = 89.1 \approx 89$
Step 5: Effort $= 89\times20 = 1780$ person-hours

Question 2
System includes: 5 Average Actors, 2 Complex Actors; 6 Average Use Cases, 4 Complex Use Cases. TCF = 1.05, ECF = 1.0, Productivity = 18. Find effort.

Solution:
UAW $= (5\times2)+(2\times3) = 16$; **UUCW** $= (6\times10)+(4\times15) = 120$; **UUCP** $= 136$;
UCP $= 136\times1.05\times1 = 142.8 \approx 143$; **Effort** $= 143\times18 = 2574$ person-hours

Question 3
Given: UAW = 12, UUCW = 150, TCF = 0.95, ECF = 1.1, Productivity = 22. Find effort.

Solution:
UUCP $= 162$; **UCP** $= 162\times0.95\times1.1 = 169.29 \approx 169$; **Effort** $= 169\times22 = 3718$ person-hours

Question 4
Project has: 4 Simple Actors, 3 Average Actors; 5 Simple, 5 Average, 3 Complex Use Cases. TCF = 1, ECF = 1, Productivity = 20. Find effort.

Solution:
UAW $= 10$; **UUCW** $= 120$; **UUCP** $= 130$; **UCP** $= 130$; **Effort** $= 130\times20 = 2600$ person-hours

Question 5
UAW = 20, UUCW = 200, TCF = 1.08, ECF = 0.92, Productivity = 18. If team capacity/sprint = 600 hours, find sprints.

Solution:
UUCP $=220$; **UCP** $=220\times1.08\times0.92\approx219$; **Effort** $=219\times18=3942$ hours.
Sprints $=3942/600=6.57\approx7$ Sprints

19.3.3 Complexity-Based Estimation

Weights: Simple (S) = 1, Medium (M) = 3, Complex (C) = 5.
Formula: Total Complexity Points $=(S\times1)+(M\times3)+(C\times5)$.

Question 1
Backlog: 12 S, 8 M, 5 C. Productivity = 6 hours/point. Find effort.

Solution:
Points $=(12\times1)+(8\times3)+(5\times5)=61$.
Effort $=61\times6=366$ hours.

Question 2
Sprint: 10 S, 6 M, 4 C. Capacity = 240 hours. Productivity = 5 hours/point. Can team complete?

Solution:
Points $=(10\times1)+(6\times3)+(4\times5)=48$.
Required Effort $=48\times5=240$ hours.
Conclusion: Yes, completed exactly within capacity.

Question 3
20 S, 15 M, 10 C. Productivity = 4. 1 person = 160 hours/month. Persons required for 2 months?

Solution:
Points $=115$; **Effort** $=115\times4=460$ hours.
Capacity (1 person, 2 months) $=160\times2=320$.
Persons $=460/320=1.44\approx2$ persons.

Question 4
25 S, 12 M, 8 C. Productivity = 5. Team velocity = 50 points/sprint. Find sprints.

Solution:
Points $=101$. **Sprints** $=101/50=2.02\approx3$ Sprints.

Question 5
30 S, 20 M, 15 C. Productivity = 3. Cost/hour = \$800. Find effort and cost.

Solution:
Points $=(30\times1)+(20\times3)+(15\times5)=165$.
Effort $=165\times3=495$ hours.
Cost $=495\times\$800=\$396{,}000$.

20 Unit V - Measurement

20.1 Agile Measurement

Measurement in Agile shifts away from traditional lines-of-code (LOC) or heavy documentation metrics towards value-driven and team-centric metrics. The goal is to measure working software and team sustainability.[41]

Key Agile Measurement Metrics include:

- **Velocity:** The amount of work (usually in story points) a team can successfully complete during a single Sprint.
- **Lead Time:** The time from the moment a request is made by a client until it is completely delivered.
- **Cycle Time:** The time it takes for the team to complete work items once they actively start working on them.
- **Escaped Defects:** The number of bugs or issues discovered by the user after the release in production.

20.2 Agile Control & Control Parameters

Control in Agile is empirical, meaning it is based on observation and adaptation rather than upfront predictive planning.

The Agile Triangle of Control Parameters: Unlike the traditional iron triangle (Fixed Scope, Estimated Time, Estimated Cost), the Agile triangle operates on:

1. **Fixed Time (Schedule):** Sprints/Iterations are timeboxed (e.g., exactly 2 weeks).
2. **Fixed Cost (Resources):** Team sizes are kept constant.
3. **Variable Scope:** Scope is continually prioritized and adjusted based on what can fit into the timebox.

Let V be Velocity, E be Total Estimation of Backlog, and N be the number of Sprints required. The control equation is simply:

$$N = \lceil \frac{E}{V} \rceil$$

20.3 Agile Approach to Risk

Agile mitigates risk inherently through its lifecycle. Traditional development defers integration and testing to the end (high risk of failure). Agile reduces risk via:

- **Early and Continuous Delivery:** Delivers the riskiest features first to fail fast.
- **Feedback Loops:** Daily stand-ups, Sprint Reviews, and Retrospectives actively identify and mitigate risks continually.
- **Transparency:** Information radiators (like Kanban boards) make bottlenecks highly visible.[42]

[41] Source: Reference Textbook - *Agile Software Development: Principles, Patterns, and Practices* by Robert C. Martin (Overview of Agile practices, Value delivery).

[42] Source: Industry Standard Agile Methodologies and PMI-ACP Agile Risk Management framework.

21 Configuration Management and DSDM Atern

21.1 Agile Approach to Configuration Management (CM)

In Agile, Configuration Management is critical because the code base changes rapidly and collaboratively.

- **Version Control:** Everything must be in a repository (Git, SVN). Not just code, but database schemas, configuration files, and test scripts.
- **Branching Strategies:** Agile favors trunk-based development or short-lived feature branches to avoid "merge hell" later.
- **Branch by Abstraction:** A technique for making large-scale changes to a system in a gradual way, allowing the team to continue releasing software while the change is in progress.[43]

21.2 The Atern Philosophy and Principles

Atern (now known as DSDM - Dynamic Systems Development Method) is an Agile project delivery framework primarily used in enterprise environments where governance and strict lifecycle management are required.

Atern Philosophy:
"Best business value emerges when projects are aligned to clear business goals, deliver frequently, and involve the collaboration of motivated and empowered people."

The Eight Atern Principles:

1. **Focus on the business need:** Every decision must be justified by its business value.
2. **Deliver on time:** Timeboxing is non-negotiable.
3. **Collaborate:** Teams must be cross-functional and collaborative.
4. **Never compromise quality:** Quality levels are agreed upon early and not compromised to meet deadlines.
5. **Build incrementally from firm foundations:** Do enough design upfront to understand the architecture, but no more.
6. **Develop iteratively:** Build, feedback, adjust.
7. **Communicate continuously and clearly:** Daily stand-ups, workshops, and rich communication.
8. **Demonstrate control:** Project managers must proactively track progress using objective metrics.[44]

Rationale for using Atern: Atern provides the rigorous structure and documentation that heavily regulated industries (like banking or healthcare) require, while still maintaining Agile flexibility and iterative delivery.

[43]Source: CM principles are aligned with Extreme Programming (XP) practices outlined in Robert C. Martin's text (Continuous Integration chapter).
[44]Source: DSDM Consortium - *The DSDM Agile Project Framework* (Formally Atern).

22 Refactoring and Continuous Integration

Note: These concepts are foundational to Extreme Programming (XP) and are thoroughly detailed by Robert C. Martin.

22.1 Refactoring

Refactoring is the process of restructuring existing computer code-changing the factoring-without changing its external behavior.

- **Purpose:** To improve the design, structure, and implementation of the software, making it more readable and less complex (reducing Technical Debt).
- **Rule of Thumb:** "Leave the campground cleaner than you found it." Every time a developer touches a module, they should aim to improve its structure.[45]

Common Refactoring Techniques:

- *Extract Method:* Taking a long, complex function and breaking it into smaller, descriptive methods.
- *Rename Variable/Method:* Changing names to reveal intent clearly.
- *Replace Conditional with Polymorphism:* Using Object-Oriented principles to eliminate massive `switch` statements.

22.2 Continuous Integration (CI) & Automated Build Tools

Continuous Integration is a development practice where developers integrate code into a shared repository frequently, preferably several times a day.

The CI Process:

1. A developer commits code to the version control repository.
2. The CI server monitors the repository and detects the change.
3. The CI server automatically pulls the code, compiles it, and runs unit tests.
4. If the build fails, the team is notified immediately, and fixing the build becomes the highest priority.

Automated Build Tools: Tools that automate the compilation, testing, and deployment processes are essential.

- **Build Managers:** Maven, Ant, Gradle (Java), Webpack (JS).
- **CI/CD Servers:** Jenkins, Travis CI, GitHub Actions, GitLab CI.

Benefit: "Integration is no longer a major event at the end of a project; it is a non-event that happens constantly."[46]

[45]Source: Robert C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, Section on Extreme Programming Practices.

[46]Source: Robert C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, Chapter 2: Continuous Integration.

23 Scaling Agile and Scrum Management

23.1 Scaling Agile: Scrum of Scrums

When a project is too large for a single Agile team (usually 7±2 members), multiple teams are formed. To coordinate these teams, a technique called **Scrum of Scrums** is utilized.

- **Mechanism:** Each individual team holds their daily stand-up. Following this, a designated representative (usually the Scrum Master or Technical Lead) from each team attends the Scrum of Scrums meeting.
- **Agenda:**
    1. What did your team do since we last met?
    2. What will your team do before we meet again?
    3. Is anything slowing your team down?
    4. *Crucial addition:* Are you about to put something in another team's way?

23.2 Estimating and Tracking a Scrum Project

Estimation Techniques: Scrum avoids estimating in absolute hours initially due to the cone of uncertainty. Instead, relative estimation is used.

- **Story Points:** A unit of measure expressing the overall effort, complexity, and risk of a user story.
- **Planning Poker:** A consensus-based estimation technique where team members use cards with Fibonacci numbers (1, 2, 3, 5, 8, 13...) to estimate effort privately, then reveal simultaneously to prevent anchoring bias.[47]

Tracking Tools:

- **Sprint Burndown Chart:** A graphical representation of work left to do versus time. The ideal line goes diagonally from total effort down to zero by the end of the sprint.
- **Release Burnup Chart:** Tracks completed work upward toward a target scope line, making scope changes highly visible.

23.3 Best Practices to Manage Scrum

To successfully manage a Scrum environment, organizations should adhere to the following:

[47] Source: *Succeeding with Agile* by Mike Cohn, and standard Scrum guides referenced in Syllabus unit V.

Practice	Description
Protect the Team	The Scrum Master must shield the development team from external interruptions and scope creep during a Sprint.
Refine the Backlog	Devote 5-10% of Sprint capacity to backlog grooming to ensure upcoming stories are "Ready" for the next sprint.
Maintain Sustainable Pace	Avoid forced overtime. A tired team writes buggy code. Track velocity to plan realistically.
Actionable Retrospectives	Retrospectives must result in 1 or 2 concrete action items that are assigned owners and tracked in the next sprint.

Table 9: Scrum Management Best Practices

24 Assignment I

Q1. Define Agile Software Development.

Agile Software Development is a modern software development approach that focuses on delivering high-quality software through iterative and incremental development. It emphasizes customer collaboration, continuous feedback, and adaptability to change rather than rigid planning. In Agile, software is developed in small, manageable iterations, allowing teams to respond quickly to changing requirements and deliver working software frequently, ensuring better alignment with user needs and improved project success.

To elaborate further, software development is fundamentally the process of designing, creating, testing, and maintaining computer programs. While traditional "plan-driven" models (like Waterfall) treat these phases sequentially, Agile shifts the paradigm.

- **Collaboration & Pace:** Agile involves people collaborating together to build high-quality products that meet customer needs at a *sustainable pace*.
- **Time-boxing:** Tasks are divided into time boxes (small time frames, typically 1-4 weeks) to deliver specific functional features for a release.
- **Incremental Delivery:** Each delivery is incremental in terms of features. Instead of delivering everything at the end, functional code is provided to the customer continuously, which may already be operational.

Q2. List the four values of the Agile Manifesto.

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

The Agile Manifesto was created as a reaction to heavyweight, document-driven software development processes. It is vital to note the underlying caveat of these values: *"That is, while there is value in the items on the right, we value the items on the left more."*

- **Individuals and interactions:** Emphasizes face-to-face communication (the most efficient method of conveying information) and building projects around motivated individuals.
- **Working software:** It serves as the primary measure of progress. Rather than presenting complex mock-ups or exhaustive requirement documents, Agile teams showcase completed, functional work.
- **Customer collaboration:** Business people and developers must work together daily throughout the project to minimize misunderstandings.
- **Responding to change:** Agile processes harness change for the customer's competitive advantage, welcoming changing requirements even late in development.

Q3. What is a sprint in Scrum?

A sprint in Scrum is a time-boxed iteration, usually lasting from one to four weeks, during which a Scrum team works to complete a selected set of tasks from the product backlog. The goal of a sprint is to deliver a potentially shippable product increment that meets the defined sprint objectives. Each sprint begins with sprint planning and ends with a sprint review and retrospective, ensuring continuous feedback, improvement, and progress toward the final product.

In the broader ecosystem of Agile Project Management, a Sprint acts as the heartbeat of the Scrum framework.

- **Metrics Tracking:** During a sprint, progress is highly visible. Teams use a **Sprint Burndown Chart**, which is a graphical representation of work left to do versus time. The ideal line goes diagonally from total effort down to zero by the end of the sprint.
- **Velocity:** The average amount of work (usually measured in story points) a team completes during a sprint is their "Velocity," which is a crucial process-level metric for future estimation.
- **Best Practices:** The Scrum Master must protect the team during a sprint by shielding them from external interruptions and scope creep, ensuring a sustainable pace without forced overtime.

Q4. Name any four Agile Methodologies.

1. **Scrum:** Scrum is an agile framework that uses time-boxed iterations called sprints to deliver working software incrementally. It emphasizes defined roles, ceremonies, and continuous feedback.
2. **Kanban:** Kanban is a visual workflow management method that uses a Kanban board to track tasks. It focuses on continuous delivery and limiting work in progress to improve efficiency.
3. **Extreme Programming (XP):** It is an Agile methodology that emphasizes high-quality code and frequent releases through practices such as pair-programming, test-driven development, and continuous integration.
4. **Lean Software Development:** Lean Software Development focuses on eliminating waste, improving efficiency, and delivering maximum value to the customer by optimizing processes and encouraging continuous improvement.

Agile itself is not a specific methodology but an umbrella term for these frameworks that adhere to the Agile Manifesto. While Scrum focuses heavily on management and project flow, Extreme Programming (XP) focuses intensely on the technical engineering practices to ensure software quality. Lean, originally derived from Toyota's manufacturing system, provides the overarching philosophy of maximizing customer value while minimizing waste (Muda).

Q5. What is Kanban?

Kanban is an Agile methodology that focuses on visualizing work and managing workflow efficiently. It uses a Kanban board with columns such as To Do, In Progress, and Done to represent different stages of work. By limiting work in progress and promoting continuous delivery, Kanban helps teams identify bottlenecks, improves productivity, and delivers tasks smoothly without overloading the team.

Kanban, translating to "visual signal" or "card" in Japanese, is intrinsically linked to Lean principles. Unlike Scrum, which operates in fixed-length timeboxes (sprints), Kanban is a continuous flow system. A core principle of Kanban is establishing strict **WIP (Work In Progress) limits** for each column to prevent bottlenecks and context-switching, thereby optimizing the *Flow Efficiency* and reducing *Lead Time* from request to delivery.

Q6. List the key Scrum roles.

- **Product Owner:** Defines and prioritizes the product backlog to maximize business value.
- **Scrum Master:** Facilitates Scrum practices and removes obstacles for the team.
- **Development Team:** Builds, tests, and delivers the product increment.
- **Stakeholders:** Provide requirements, feedback, and overall direction for the project.
- **Customers/End Users:** Use the product and provide feedback based on their experience.

While Stakeholders and Customers are vital to the project's ecosystem, the official "Scrum Team" comprises solely the Product Owner, Scrum Master, and the cross-functional Development Team.

- **Cross-functional Nature:** The Development Team must contain all the skills necessary to turn Product Backlog items into a potentially shippable increment (e.g., developers, designers, **integrators**, and **independent testers**).

Q7. What is meant by incremental development?

Incremental development is an approach in Agile Software Development where the software is built and delivered in small, manageable parts called increments. Each increment adds new functionality to the existing system and is fully developed, tested, and usable. This approach allows early delivery of working software, continuous user feedback, reduced risk, and gradual improvement of the product over time.

It is helpful to distinguish "Incremental" from "Iterative."

- **Iterative** means revisiting and refining existing work (like polishing a rough sketch).
- **Incremental** means adding entirely new, finished pieces (like building a house room by room). Agile combines both: it iteratively refines the design while incrementally adding functional software pieces, ultimately resulting in a Potentially Shippable Product Increment (PSPI) at the end of each timebox.

Q8. Define stakeholder in Agile projects.

In Agile projects, a stakeholder is any individual or group that has an interest in the project or is affected by its outcome. Stakeholders may include customers, end users, product owners, sponsors, managers, and business representatives. They actively contribute by providing requirements, feedback, and insights throughout the development process, helping ensure that the product delivers real value and meets user expectation.

Beyond merely providing requirements, Stakeholders actively "evaluate the final product" incrementally. Their ongoing feedback loop directly influences metrics like **Value Delivered** and **Return on Investment (ROI)**, ensuring the team does not spend time developing features that do not align with current business realities.

Q9. Why is Agile considered a flexible development approach?

- Requirements can change at any time.
- Development is done in short iterations.
- Continuous customer feedback is encouraged.
- Planning is adaptive, not fixed.
- Faster response to market and user needs.

This flexibility is codified in the Agile Manifesto Principle: *"Welcome changing requirements, even late in development."* Traditional models treat late changes as costly errors to be avoided (via Change Control Boards). Agile, through adaptive planning and short iterations, views change as a competitive advantage for the customer.

Q10. Purpose of Scrum ceremonies.

The purpose of Scrum ceremonies is to ensure effective communication, transparency, and continuous improvement throughout the developmental process. These ceremonies are:

- **Sprint Planning:** The team defines the sprint goal and selects items from the product backlog to work on during the sprint.
- **Daily Scrum:** A short daily meeting where team members discuss progress, plans, and obstacles.
- **Sprint Review:** The team demonstrates completed work to stakeholders and collects feedback.
- **Sprint Retrospective:** The team reflects on the sprint to identify improvements for future sprints.

This helps the team plan improvements, thereby enabling the delivery of high-quality software in a structured and Agile manner.

These ceremonies create a structured cadence for empirical process control. They provide regular, time-boxed opportunities to **Inspect** (evaluate the current state of the product or process) and **Adapt** (make adjustments based on the inspection), which are core pillars of the Scrum framework.

Q11. Role of Scrum Master in Agile success.

- Facilitates Scrum ceremonies.
- Removes obstacles faced by the team.
- Ensures Scrum practices are followed.

- Acts as a servant leader.
- Improves team productivity and collaboration.

A crucial, often overlooked role of the Scrum Master is to **Protect the Team**. They must actively shield the development team from external interruptions, organizational politics, and scope creep mid-sprint, thereby ensuring the team can maintain a sustainable pace without forced overtime.

Q12. How Agile handles changing requirements?

Agile handles changing requirements by welcoming and adapting to change at any stage of development. Instead of fixing all requirements at the beginning, Agile stores them in a product backlog, where they can be added, modified, or reprioritized based on customer feedback and business needs. These changes are then implemented in upcoming iterations, ensuring the product remains relevant, flexible, and aligned with user expectations.

Technically, Agile handles changes through robust engineering practices and principles like the **Open-Closed Principle (OCP)**. The system's architecture is designed to be extensible. Furthermore, new requirements are captured dynamically as **User Stories** which are constantly re-estimated and prioritized by the Product Owner rather than requiring massive rewrites of requirements specification documents.

Q13. Lean Software Development with example.

Example: Removing unused features in an app to save time and improving the performance.

- Lean Software Development focuses on delivering maximum value by eliminating unnecessary work.
- It emphasizes reducing waste such as extra features and delays.
- It encourages continuous improvement and fast delivery.
- Quality is built into every stage of development.
- Continuously improving the process based on team feedback.

Lean focuses explicitly on **Waste Management** (identifying and eliminating non-value-adding activities) and **Kaizen** (continuous, incremental improvement). In software, "waste" can be partially done work, extra unrequested features, task switching, waiting (delays), and defects. By building quality in (via automated testing) and deferring commitments until the last responsible moment, Lean maximizes efficiency.

Q14. Incremental development of a food-delivery app.

- **First Increment:** User login and registration feature is developed to allow users to create and access accounts.
- **Second Increment:** Restaurant listing module is added so users can browse restaurants and menus.

- **Third Increment:** Order placement functionality is implemented to enable users to select food and place orders.
- **Fourth Increment:** Payment integration is developed to support secure online transactions.
- **Fifth Increment:** Delivery tracking and feedback features are added to allow real-time order tracking and user reviews.

This represents a highly effective feature-breakdown into Sprints. Notice that after Increment 3, even without integrated payments (they could pay Cash on Delivery), the app is technically functional and provides business value, demonstrating the "Potentially Shippable" nature of Agile increments.

Q15. Kanban in a student project.

- **Step 1:** List all student project tasks on a Kanban board, such as topic selection, literature review, coding, testing, and report writing.
- **Step 2:** Create columns like To Do, In Progress, and Done to organize the work stages.
- **Step 3:** When a student starts working on a task, move it from To Do to In Progress.
- **Step 4:** Limit the number of tasks in In Progress so the student focuses on one or two activities at a time, such as only coding and documentation.
- **Step 5:** Track task movement visually on the board to clearly see project progress and pending work.
- **Step 6:** After completing a task, move it to Done, helping improve time management.

The critical element in Step 4 is the implementation of **WIP (Work In Progress) Limits**. In Agile metrics, this practice dramatically reduces *Cycle Time* (the time it takes for a task to move from In Progress to Done) by eliminating context-switching penalties and preventing team burnout.

Q16. Scrum ceremonies during a sprint.

- **Sprint Planning** is conducted to define the sprint goal and select items for the sprint backlog.
- **Daily Scrum** is held every day to track progress, discuss plans, and identify blockers.
- **Development work** continues throughout the sprint based on the sprint backlog.
- **Sprint Review** is conducted at the end of the sprint to demonstrate completed work and gather feedback.
- **Sprint Retrospective** is held to analyze performance and identify improvements for future sprints.

An additional continuous practice that happens throughout the sprint is **Backlog Refinement** (or Grooming). As per Agile best practices, the team devotes 5-10% of their Sprint capacity to clarify, estimate, and prepare User Stories for the *upcoming* sprints, ensuring a smooth transition into the next Sprint Planning.

Q17. TDD applied to login feature.

- Write a test for login validation.
- Test fails initially.
- Write minimum code to pass the test.
- Refactor the code.
- Repeat for other login scenarios.

This process is formally known in Extreme Programming (XP) as the **Red-Green-Refactor Cycle**.

- **Red:** Write a failing automated test case.
- **Green:** Write the simplest, quickest code possible to make the test pass.
- **Refactor:** Clean up the code, apply design patterns, and ensure quality without breaking the passed test.

This guarantees high **Quality and Testing Metrics** and low *Defect Density* as the codebase grows.

Q18. Agile Development life-cycle (real-world example)?

- **Step 1: Concept:** The idea of developing an online shopping application is identified.
- **Step 2: Planning:** Requirements are collected and prioritized in the product backlog.
- **Step 3: Iteration:** Features such as product listing and cart are developed in short sprints.
- **Step 4: Testing:** Each sprint includes continuous testing to ensure quality.
- **Step 5: Release:** Working features are released frequently to users.
- **Step 6: Feedback:** User reviews are analyzed, and improvements are made in the next iteration.

Unlike the linear, sequential progression of traditional lifecycles, Steps 2 through 6 operate in a continuous loop. Through the use of **Continuous Integration/Continuous Deployment (CI/CD)**, steps like testing and releasing happen continuously rather than acting as isolated phases.

Q19. Difference between Agile and Waterfall model?

Aspect	**Waterfall Model (Sequential/Plan-driven)**	**Agile Model (Iterative/Incremental)**
Development Approach	Linear and Sequential development in fixed phases (Requirements → Design → Coding → Testing).	Iterative and Incremental development in small cycles (Sprints).
Flexibility	Rigid; changes are difficult and costly once a phase is completed.	Highly flexible; changes can be made at any stage based on feedback.
Requirements	Fixed at the beginning upfront. Scope is well-understood before starting.	Evolve throughout the project lifecycle via the Product Backlog.
Customer Involvement	Minimal after the initial requirement gathering phase.	Continuous customer and stakeholder involvement daily or per iteration.
Testing	Done only after the development (coding) phase is entirely completed.	Continuous and integrated with development (Test-driven development).
Delivery	Software is delivered only at the very end of the project lifecycle.	Working software is delivered frequently (every couple of weeks).
Artifacts (Enriched)	*Yields extensive documents if canceled midway.*	*Yields functional, potentially operational code if canceled midway.*

Table 10: Comparative Analysis of Waterfall vs. Agile

Q20. Stakeholders involvement in Agile success.

- **Step 1:** Stakeholders share business goals and initial requirements at the start of the project.
- **Step 2:** They participate in sprint reviews to evaluate completed features.
- **Step 3:** Stakeholders provide regular feedback based on product usage.
- **Step 4:** Feedback helps the Product Owner reprioritize the backlog.
- **Step 5:** Continuous involvement reduces misunderstanding and risks.
- **Step 6:** Active stakeholder participation ensures higher product quality and customer satisfaction.

Stakeholders are individuals or groups who are impacted by the project, existing either internally or externally to the organization. In an Agile environment, their continuous loop of feedback is paramount.

- **Evaluating the Final Product:** They do not just provide requirements; they actively evaluate the working increments.

- **Business Value Delivery:** Stakeholder involvement directly impacts *Value Delivered* (a key Agile metric). By keeping stakeholders in the loop, the project team ensures the software aligns with the highest priority business needs first, optimizing the Return on Investment (ROI).
- **Interaction with Roles:** They work closely with the **Product Owner**, who translates their business goals into manageable **User Stories** (simple descriptions of features from the perspective of the user) and manages the Product Backlog.

25 Assignment II

Section A: 1 Mark Questions

Q1. Define Agile Product Management.

Agile Product Management is a continuous, value-driven approach to overseeing software development that prioritizes adaptability and customer feedback over rigid, upfront planning. It is characterized by several core functions:

- **Visionary Leadership:** The Product Owner defines the product vision and roadmap, ensuring alignment with business goals and stakeholder expectations.
- **Backlog Ownership:** It involves the continuous management and prioritization of the Product Backlog to maximize Business Value and Return on Investment (ROI).
- **Empirical Control:** Progress is monitored through working software and iterative planning levels (Vision, Roadmap, Release, and Sprint) rather than static documentation phases.

Q2. What is estimation in Agile planning?

Estimation in Agile is the process of forecasting the effort required to implement user stories by using abstract metrics rather than absolute time. Key characteristics include:

- **Relative Sizing:** Teams utilize Story Points to account for the complexity, volume of work, and inherent risks or uncertainties of a task.
- **Consensus-Based Techniques:** Methods such as Planning Poker (utilizing the Fibonacci sequence) are used to prevent anchoring bias and leverage collective team knowledge.
- **Predictability:** These estimates, combined with the team's historical Velocity, allow for reliable forecasting of release dates and project completion timelines.

Q3. What is backlog management?

Backlog management is the continuous process of creating, refining, and prioritizing the Product Backlog, which serves as the single source of requirements for a product. It involves:

- **Refinement (Grooming):** Regular sessions where the team breaks down epics into smaller user stories, clarifies requirements, and identifies acceptance criteria.
- **Prioritization Frameworks:** Utilizing techniques like MoSCoW (Must-have, Should-have, Could-have, Won't-have) to ensure the team delivers the highest business value first.
- **Dynamic Maintenance:** The backlog is emergent, meaning it is updated frequently based on customer feedback and changing market conditions to prevent scope creep.

Q4. Define user stories in Agile development.

User stories are concise, informal descriptions of a software feature written explicitly from the perspective of the end-user. They serve as mnemonic tokens for ongoing conversations and follow these principles:

- **INVEST Criteria:** A quality user story must be Independent, Negotiable, Valuable, Estimable, Small, and Testable.

- **The 3 C's:** They involve the Card (the physical or digital token), Conversation (dialogue between developers and stakeholders), and Confirmation (acceptance criteria).
- **Standard Format:** Typically follows the pattern: "As a [type of user], I want [an action] so that [some reason/benefit]."

Q5. What is Test-Driven Development (TDD)?

Test-Driven Development (TDD) is a fundamental Agile engineering practice where automated unit tests are written before the actual production code. It follows a strict iterative cycle:

- **Red Phase:** Write a failing unit test for a specific functionality that does not yet exist.
- **Green Phase:** Write the minimal code necessary to make the failing test pass.
- **Refactor Phase:** Clean up the internal structure of the code while maintaining its external behavior, ensuring technical excellence and reducing technical debt.
- **Outcome:** This practice ensures continuous verifiability and leads to a highly modular, decoupled software architecture.

Q6. What is Agile risk management?

Agile risk management is a proactive and iterative process integrated directly into the daily development cycle to identify and mitigate threats to project success. It focuses on:

- **Early Exposure:** Risks are surfaced every 24 hours during Daily Scrums and evaluated during Sprint Retrospectives.
- **Failure Mitigation:** By delivering functional software frequently (failing fast), the team validates technical feasibility and business value early in the lifecycle.
- **Transparency:** Information radiators like Kanban boards make bottlenecks and integration risks highly visible, allowing for immediate corrective actions rather than deferring them to the project's end.

Q7. What is Agile measurement?

Agile measurement evaluates project health by focusing on tangible value delivery and team sustainability rather than traditional metrics like lines of code. Key aspects include:

- **Team-Level Metrics:** Indicators such as Velocity (story points per sprint), Sprint Burndown, and Cycle Time help track internal efficiency.
- **Product and Process Metrics:** These include Time-to-Market, Release Frequency, Escaped Defects, and Return on Investment (ROI).
- **Characteristics:** Effective Agile metrics are lightweight, trend-based, actionable, and customer-focused, providing real-time visibility into the flow of work and overall quality.

Q8. Define continuous integration.

Continuous Integration (CI) is an Agile practice where developers frequently merge their code changes into a central repository, typically multiple times per day. The process ensures:

- **Automated Verification:** Every commit triggers an automated build and a suite of unit tests to detect integration issues early.
- **Baseline Stability:** It prevents "integration hell" by identifying conflicts immediately, keeping the software in a verified and deployable state at all times.
- **Feedback Loop:** CI provides instant feedback to developers, making the fixing of a broken build the team's highest priority to maintain development momentum.

Q9. What is refactoring in Agile?

Refactoring is the process of restructuring existing code to improve its internal design without changing its external, observable behavior. It is essential for:

- **Reducing Technical Debt:** Continually cleaning the codebase to prevent complexity from accumulating over iterations.
- **Improving Readability:** Utilizing techniques like Extract Method or Renaming Variables to reveal developer intent and enhance maintainability.
- **Collective Ownership:** It allows the team to improve any part of the system safely, relying on automated test suites to ensure that no existing functionality is broken during the restructuring process.

Q10. What is Scrum of Scrums?

Scrum of Scrums is an Agile scaling technique used to coordinate multiple Scrum teams working on a single, large-scale product. It operates through:

- **Ambassador Representation:** A designated representative (often the Scrum Master) from each team attends a higher-level coordination meeting.
- **Dependency Management:** The meeting focuses on cross-team dependencies, integration points, and potential impediments that affect multiple groups.
- **Lightweight Governance:** It provides a communication framework to resolve technical conflicts and align architectural decisions without the need for heavy corporate bureaucracy or extensive documentation.

Section B: 2 Mark Questions

Q1. Explain the role of communication in Agile product management.

Effective communication is the absolute cornerstone of Agile methodologies. Unlike traditional software development which relies heavily on extensive documentation handed off between isolated teams, Agile promotes direct communication, co-location, and continuous daily interactions to ensure project success.

- **Daily Stand-ups:** These are brief, daily meetings used to synchronize activities, track progress, and create a concrete plan for the next 24 hours.
- **Pair Programming:** Two developers work together at a single workstation. This provides continuous code review, immediate communication of architectural ideas, and shared code ownership.
- **On-Site Customer:** A business representative or Product Owner is consistently available to the development team to clarify requirements and provide immediate feedback on features.

Q2. Write short notes on Agile estimation techniques.

Estimation in Agile avoids absolute time measurements like exact hours or days in favor of relative sizing. It focuses on preparation and forecasting to ensure better risk coverage and project predictability.

- **User Stories and Story Points:** Requirements are captured as User Stories and estimated using Story Points. Story points are an abstract measure of effort that accounts for complexity, amount of work, and uncertainty.
- **Velocity:** This is the number of story points a team successfully completes in a single iteration. It provides a realistic metric to reliably forecast project completion dates.
- **Planning Poker:** A consensus based estimation technique where team members use cards with Fibonacci numbers to estimate effort privately, preventing anchoring bias.
- **Other Techniques:** Methods used during project initiation include Quick Function Point Analysis, Use Case Points, and Complexity based estimation.

Q3. What is the importance of monitoring progress in Agile projects?

Monitoring progress is an essential management activity in Agile projects that provides objective insight into team performance and highlights areas for improvement. Progress is measured continuously by the delivery of working software rather than the completion of phase deliverables.

- **Transparency:** Agile teams monitor progress transparently using leading indicators that reflect team behavior, quality, and velocity.
- **Information Radiators:** Large, highly visible displays, such as Kanban boards or physical burndown charts, are utilized to track tasks and progress effectively within the workspace.
- **Sprint Reviews:** At the end of every iteration, the team demonstrates the working software to stakeholders to measure actual progress against the planned goal.
- **Agile Metrics:** Metrics like Sprint Burndown, Cycle Time, and Throughput help validate Agile practices and forecast future progress.

Q4. Explain backlog management with an example.

The Product Backlog is an ordered list of everything that is known to be needed in the product. It acts as the single source of requirements for any changes to be made to the product. Backlog management involves continuously updating, organizing, and refining this list.

- **Backlog Refinement:** The ongoing process of reviewing items, prioritizing them, and sizing them. Teams usually devote five to ten percent of their Sprint capacity to backlog grooming.
- **Prioritization:** Techniques like MOSCOW (Must have, Should have, Could have, Won't have) are used to ensure the team delivers the highest business value first.
- **Example:** For a food delivery application, initial requirements like user login, restaurant listing, and order placement are collected and prioritized in the backlog. If user feedback demands a new feature like 'delivery tracking', it is added to the backlog and prioritized for upcoming iterations.

Q5. Describe the concept of Agile architecture.

Agile architecture emphasizes that the best architectures, requirements, and designs emerge from self-organizing teams. Unlike traditional models, Agile teams do not attempt to build a massive, all-encompassing architecture upfront. Instead, the architecture evolves iteratively based on the simplest design that meets current requirements.

- **Simplicity:** The system is designed as simply as possible at any given moment. Extra complexity is removed as soon as it is found through constant refactoring.
- **Feature Driven Development:** An iterative Agile architectural process driven by delivering tangible, working software. It involves developing an overall model, building a features list, and then designing and building by feature.
- **Flexibility:** Agile architecture remains highly flexible to accommodate changing requirements without causing severe degradation or rigid constraints.

Q6. What are Agile testing techniques?

Testing in Agile is not a separate, final phase but an integral and continuous part of the software development lifecycle. Agile embeds quality assurance into the sprint, with testers working alongside developers to validate user stories immediately.

- **Exploratory Testing:** Testers actively explore the application without pre-scripted test cases to find edge-case defects and vulnerabilities.
- **Automated Regression Testing:** This ensures that new code modifications do not break existing functionality, maintaining system stability across iterations.
- **Test Driven Development:** A core practice where automated unit tests are written before the actual production code, following a strict Red, Green, and Refactor cycle.
- **User Acceptance Testing:** The final testing phase where end-users validate the software against initial User Stories in a production-like environment.

Q7. Explain User Acceptance Testing (UAT) in Agile.

User Acceptance Testing is the final phase of Agile testing where the end-users, stakeholders, or the product owner validate the software against the initial User Stories.

- **Business Value Validation:** The primary goal of UAT is to ensure that the software provides the intended business value and meets the practical, day-to-day needs of the customer.
- **Production Environment:** The software is tested in a production-like environment to simulate real-world usage and verify system behavior accurately.
- **Traceability:** Acceptance tests are linked directly to User Stories and their predefined Acceptance Criteria, ensuring a one-to-one traceability between a business requirement and the test that validates it.
- **Executable Documentation:** Once acceptance tests pass, they serve as compileable and executable documentation of the system's features.

Q8. What are Agile metrics and measurements?

Agile metrics shift away from traditional, heavy documentation or lines of code metrics towards value-driven and team-centric measurements. They provide objective insight into team performance, velocity, quality, and overall value delivery.

- **Characteristics:** Effective Agile metrics are lightweight, actionable, customer-focused, trend-based, simple to measure, and provide real-time visibility.
- **Team Level Metrics:** These include Sprint Burndown tracking, Velocity (average story points completed per sprint), and Cycle Time.
- **Product Level Metrics:** These focus on Time-to-Market, Release Frequency, Defect Density, and Customer Satisfaction indices.
- **Process Level Metrics:** These measure Value Delivered, Return on Investment, Lead Time, and Flow Efficiency to validate overall Agile practices and optimize planning.

Q9. Explain the purpose of configuration management in Agile.

In Agile, Configuration Management is critical because the code base changes rapidly, collaboratively, and continuously through multiple iterations. It ensures stability, organization, and traceability amidst frequent updates.

- **Version Control:** All project artifacts must be maintained in a shared repository like Git or SVN. This includes not just source code, but also database schemas, configuration files, and automated test scripts.
- **Branching Strategies:** Agile favors trunk-based development or short-lived feature branches to prevent complex integration issues and avoid merge conflicts during later stages.
- **Branch by Abstraction:** A technique utilized for making large-scale changes to a system gradually, allowing the team to continue releasing software reliably while the architectural change is still in progress.

Q10. Write the advantages of automated build tools.

Automated build tools are essential software utilities that automate the compilation, testing, and deployment processes in an Agile environment. Standard examples include Maven, Gradle, Jenkins, and GitLab CI.

- **Frequent Integration:** They facilitate Continuous Integration by automatically pulling code, compiling it, and running unit tests several times a day whenever developers commit changes.
- **Immediate Feedback:** If a build fails or tests do not pass, the tools notify the development team immediately, making fixing the build the absolute highest priority.
- **Reduced Risk:** Automation ensures that integration is no longer a major, risky event at the end of a project, but rather a routine, non-event that happens constantly.
- **Deployable State:** They ensure that the software is always in a verified and deployable state, significantly reducing integration risks and time to market.

Section C: 5 Mark Questions

Q1. Explain Agile product management and its role in planning and estimation.

Agile product management is a continuous, value-driven approach to overseeing software development that prioritizes adaptability and customer feedback over rigid, upfront planning. Unlike traditional project management, which relies on fixed scopes and extensive documentation, Agile product management focuses on delivering functional increments of the product in short iterations. This approach ensures that the development team is always working on the highest priority features that deliver the most business value, while remaining flexible enough to accommodate changing market requirements.

Role in Planning:

- **Iterative Approach:** Planning occurs at multiple levels, including product vision, release planning, and sprint planning, allowing for continuous refinement.
- **Adaptive Scope:** The scope of the project is kept flexible. Plans are regularly updated based on empirical feedback gathered at the end of each iteration.
- **Prioritization:** Features are planned based on their immediate business value and risk, ensuring early delivery of critical functionality.

Role in Estimation:

- **Relative Sizing:** Agile estimation uses techniques like story points rather than absolute time metrics to gauge the complexity and effort required for user stories.
- **Collaborative Estimation:** Techniques such as Planning Poker involve the entire cross-functional team, ensuring that estimates account for testing, development, and design perspectives.
- **Velocity Tracking:** Teams measure their capacity by tracking the number of story points completed per sprint, using this historical data to make accurate future estimates.

Q2. Discuss how Agile teams monitor progress and manage business involvement.

Monitoring progress in Agile environments relies heavily on empirical data and complete transparency rather than static milestone reports. Progress is measured by the delivery of working software rather than the completion of documentation phases. Concurrently, business involvement is managed as a continuous, active partnership. Stakeholders and business representatives do not merely hand over requirements at the beginning; they are integral to the daily development process, ensuring the product continuously aligns with the overarching business strategy.

Monitoring Progress:

- **Daily Stand-ups:** Short, daily synchronization meetings where team members discuss what they completed, what they will do next, and any blockers they face.
- **Information Radiators:** Visual tools such as Kanban boards and sprint burndown charts provide an immediate, real-time view of sprint progress and remaining work.
- **Sprint Reviews:** At the end of an iteration, the team demonstrates the working increment to stakeholders to evaluate progress and gather immediate feedback.

Managing Business Involvement:

- **The Product Owner Role:** A dedicated individual acts as the voice of the business, constantly available to clarify requirements and prioritize the backlog.
- **Continuous Feedback Loops:** Stakeholders interact with the working software frequently, allowing them to pivot project direction without costly rework.
- **Collaborative Grooming:** Business representatives actively participate in backlog refinement sessions to define acceptance criteria and clarify user goals.

Q3. Explain user stories and backlog management with suitable examples.

User stories are concise, informal descriptions of a software feature told from the perspective of the end user or customer. They serve as placeholders for future conversations rather than exhaustive requirement documents. Backlog management, often called backlog refinement or grooming, is the continuous process of prioritizing, estimating, and detailing these user stories to ensure the team has a steady flow of actionable work for upcoming iterations.

User Stories Overview:

- **Standard Format:** User stories typically follow the structure: "As a [role], I want [feature] so that [benefit]."
- **Example:** "As an online shopper, I want to filter products by price range so that I can easily find items within my budget."
- **Acceptance Criteria:** Each story is accompanied by specific conditions that must be met for the story to be considered complete, forming the boundary of the feature.

Backlog Management Techniques:

- **Prioritization:** The Product Owner ranks stories based on business value, technical risk, and market dependencies, ensuring high-value items sit at the top.
- **DEEP Sizing:** The backlog should be Detailed appropriately, Emergent, Estimated, and Prioritized. Top items are broken down into small, actionable pieces, while items lower down remain large epics.
- **Refinement Meetings:** Routine sessions where the development team and the Product Owner review upcoming stories, clarify ambiguities, and assign preliminary estimates.

Q4. Discuss Agile testing methodologies including Test-Driven Development.

Agile testing methodologies integrate quality assurance directly into the daily development cycle, contrasting sharply with traditional models where testing is an isolated, sequential phase. The core philosophy is that quality must be built into the product from the very beginning. Test-Driven Development is one of the foundational Extreme Programming practices that heavily supports this philosophy by requiring tests to be written before the actual implementation code.

Test-Driven Development (TDD):

- **Red-Green-Refactor Cycle:** Developers first write a failing automated unit test (Red), then write the minimum code required to pass the test (Green), and finally restructure the code for optimization and readability (Refactor).

- **Design Improvement:** TDD forces developers to consider the application's design and architecture from the perspective of its usage, leading to highly modular and decoupled code.

Other Agile Testing Methodologies:

- **Behavior-Driven Development (BDD):** Extends TDD by writing test cases in natural language that business stakeholders can understand, focusing on system behavior rather than technical implementation.
- **Continuous Automation:** Agile relies heavily on automated regression test suites that run continuously during integration to prevent new code from breaking existing features.
- **Exploratory Testing:** Testers use their domain knowledge to manually explore the application without pre-scripted steps, finding complex edge cases that automated tests might miss.

Q5. Explain Agile risk management and quality assurance techniques.

Risk management in Agile is embedded fundamentally into its iterative structure rather than existing as a separate planning document. By delivering small increments of functional software frequently, Agile naturally mitigates schedule, technical, and market risks early in the lifecycle. Quality Assurance similarly shifts left, becoming a proactive team responsibility rather than a reactive phase at the end of development.

Agile Risk Management:

- **Iterative Delivery:** By releasing functional software every few weeks, teams validate assumptions early, dramatically reducing the risk of building the wrong product.
- **Technical Debt Tracking:** Teams actively monitor and address technical debt during sprints to prevent systemic architectural failures later in the project.
- **Fail Fast Philosophy:** Agile encourages rapid experimentation; if a technical approach is fundamentally flawed, the team discovers it within days rather than months.

Quality Assurance Techniques:

- **Definition of Done (DoD):** A strict, shared checklist that every user story must meet before being considered complete, ensuring uniform quality standards across the product.
- **Pair Programming:** Two developers collaborate on a single workstation. This serves as a continuous, real-time code review, reducing defect rates significantly.
- **Continuous Code Review:** Utilizing pull requests and peer reviews to ensure adherence to coding standards and architectural guidelines before code is merged into the main branch.

Q6. Discuss Agile metrics and measurement techniques used in project evaluation.

Agile metrics are diagnostic tools used to evaluate the health of the project, the efficiency of the team, and the quality of the software increment. Crucially, these measurements are designed to empower the team to self-organize and optimize their workflows, rather than being used by management as punitive productivity trackers. Effective metrics focus on outcomes and value delivery rather than merely output and hours worked.

Key Measurement Techniques:

- **Velocity:** Calculates the total number of story points successfully completed by the team within a single sprint. It provides a historical average used to accurately forecast future delivery capacity.
- **Burndown Charts:** A visual representation tracking the amount of outstanding work on the vertical axis against time on the horizontal axis, providing a daily measure of the likelihood of achieving the sprint goal.
- **Lead Time and Cycle Time:** Lead time measures the period from when a request is made until it is delivered. Cycle time measures the active development time taken to complete the task once work begins.
- **Escaped Defects:** Tracks the number of bugs discovered by users in production after a release. A rising number indicates a failure in the team's internal quality assurance processes.

Q7. Explain Agile control parameters and their importance in project management.

In software project management, control parameters form the boundary conditions that guide decision-making and project constraints. In traditional project management, the scope is often fixed upfront, leading to fluctuating costs and missed deadlines when complexities arise. Agile fundamentally inverts this model, utilizing distinct control parameters to ensure predictable, sustainable delivery while accommodating inevitable changes in requirements.

The Agile Control Parameters:

- **Fixed Time:** Agile utilizes strict timeboxing, meaning iterations and meetings have absolute maximum durations. A sprint ends when the time expires, regardless of whether all work is finished, enforcing discipline and cadence.
- **Fixed Cost:** Because team sizes are stable and sprint durations are constant, the cost per iteration is fixed and highly predictable for stakeholders.
- **Variable Scope:** Scope is the primary adjustable parameter. If a team cannot complete all planned work within the fixed timebox, lower-priority scope is removed or deferred to a later iteration.
- **Importance:** This inversion guarantees that the most valuable features are delivered within the available budget and timeline, eliminating the common failure mode of delivering everything late and over budget.

Q8. Describe the Atern principles and philosophy in Agile development.

Atern, originally known as the Dynamic Systems Development Method (DSDM), is an Agile framework heavily focused on corporate project delivery and strict alignment with overarching business strategy. The philosophy of Atern is built upon the premise that any project must be aligned to clearly defined strategic goals and focus upon the early delivery of tangible benefits to the business. It emphasizes governance and rigorous foundations while retaining Agile flexibility.

Core Atern Principles:

- **Focus on Business Need:** Every decision made during development must directly contribute to achieving the core business objectives of the project.
- **Deliver on Time:** Delivery deadlines are absolute. To achieve this, Atern aggressively prioritizes scope using the MoSCoW method (Must have, Should have, Could have, Won't have).

- **Never Compromise Quality:** Quality standards are fixed at the beginning of the project and must not become a variable used to meet delivery timelines.
- **Build Incrementally from Firm Foundations:** Atern requires a solid architectural baseline and high-level requirement understanding before iterative development begins, mitigating severe structural risks early.
- **Demonstrate Control:** Proactive management and transparent tracking are mandatory to assure stakeholders that the project is healthy and progressing correctly.

Q9. Explain continuous integration and automated build tools in Agile projects.

Continuous Integration is a vital Agile software engineering practice wherein developers frequently merge their code changes into a central repository, typically multiple times a day. This practice is facilitated by automated build tools, which instantly compile the integrated code and run comprehensive test suites. This immediate feedback loop prevents the severe integration conflicts traditionally known as "integration hell" that occur when developers work in isolation for long periods.

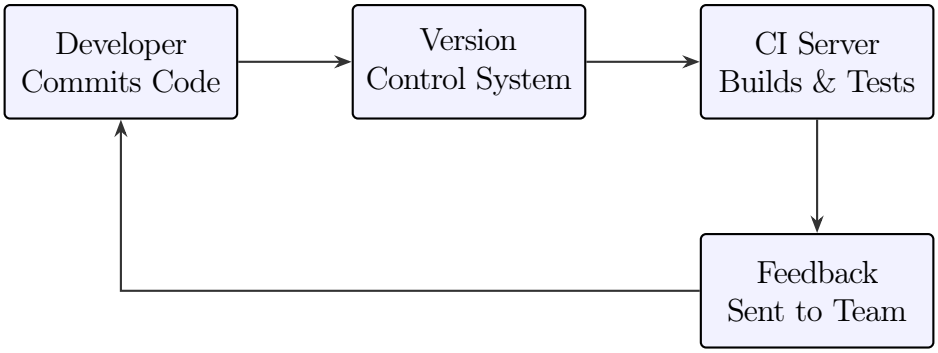

Figure 7: The Continuous Integration Cycle

Automated Build Tools and Practices:

- **The CI Server:** Tools like Jenkins, GitLab CI, or GitHub Actions monitor the repository. Upon detecting a change, they fetch the source code and automatically initiate the build script.
- **Automated Testing:** As part of the build, the tool executes unit and integration tests. If a test fails, the build is marked as broken, and the team is notified instantly.
- **Importance:** It drastically reduces debugging time, ensures the code base is always in a deployable state, and builds confidence within the team that their recent changes have not broken existing functionality.

Q10. Discuss how Scrum of Scrums helps in scaling Agile for large projects.

Scrum of Scrums is a structured scaling technique designed to connect multiple individual Scrum teams working simultaneously on a large, complex product. While a single Scrum team operates optimally with fewer than ten members, enterprise projects often require hundreds of developers. Scrum of Scrums provides a lightweight communication framework to manage dependencies, resolve cross-team blockers, and align architectural decisions without introducing heavy corporate bureaucracy.

Mechanics and Benefits:

- **Representation:** Each individual team elects an "Ambassador" (frequently the Scrum Master or a lead developer) to represent them at the higher-level Scrum of Scrums meeting.
- **Meeting Structure:** The meeting follows a format similar to a daily stand-up but focuses on team-level interactions: What has our team done that impacts other teams? What are we planning to do that impacts others? What external blockers are impeding our progress?
- **Dependency Management:** It explicitly identifies overlapping work and technical dependencies, allowing teams to coordinate API changes or shared database structures proactively.
- **Problem Resolution:** The forum provides a fast-track mechanism to escalate severe impediments that a single team cannot resolve independently, fostering a cohesive organizational workflow.

Section D: Case Study Questions

Q1. A software company is developing a mobile banking application using Agile methodology. The team is facing communication gaps between developers and business stakeholders. Explain how Agile product management practices can improve communication and project progress.

In Agile methodologies, effective communication is the cornerstone of product management. Unlike traditional plan driven models that rely on isolated teams and extensive documentation handoffs, Agile emphasizes face to face communication and continuous daily interactions. Implementing Agile product management practices can directly bridge the communication gaps between developers and business stakeholders, ensuring the mobile banking application aligns with business goals.

- **On-Site Customer Collaboration:** A business representative, typically the Product Owner, must be consistently available to the development team. This role clarifies requirements in real time and provides immediate feedback on developed features.
- **Daily Stand-up Meetings:** Conducting brief daily synchronization meetings ensures that developers and stakeholders are aware of the progress. This 15 minute event highlights plans for the next 24 hours and immediately surfaces any blocking issues.
- **Iteration Reviews:** At the end of every sprint, the team demonstrates the working software to stakeholders. This tangible demonstration measures actual progress and allows stakeholders to evaluate the product and provide constructive feedback.
- **Information Radiators:** Utilizing large visual displays, such as physical Kanban boards or digital sprint boards, provides transparent, real time visibility of the project status to all stakeholders without requiring formal reports.
- **Use of User Stories:** Requirements should be captured as User Stories. These are simple descriptions of features told from the user perspective. They act as mnemonic tokens that encourage ongoing conversations between the developers and the business stakeholders rather than serving as rigid technical specifications.

Q2. A Scrum team is working on an e-commerce platform. During sprint review, many user stories are found incomplete due to poor backlog management. Analyze the situation and suggest how proper backlog management and user stories can improve the development process.

The scenario indicates a failure in backlog refinement and estimation, leading to tasks that are either too large or ill defined to be completed within a single sprint. Proper backlog management and well structured user stories are critical for maintaining a predictable velocity and ensuring that the team delivers a potentially shippable product increment at the end of every iteration.

- **Backlog Refinement:** The team must devote five to ten percent of their sprint capacity to backlog grooming. This ongoing process involves reviewing upcoming items, prioritizing them, and sizing them so they are completely ready before the next sprint planning meeting.
- **Splitting User Stories:** Stories that are too large to estimate accurately or complete in one sprint must be split. Breaking down large features into smaller, independent user stories ensures they can be finished and tested within a single timebox.

- **Clear Acceptance Criteria:** Every user story must possess predefined acceptance criteria. This defines the boundaries of the requirement and provides a clear Definition of Done, preventing incomplete work from being presented at the review.
- **Relative Estimation Techniques:** The team should estimate effort using story points and consensus based techniques like Planning Poker. This accounts for complexity and risk, allowing the team to select a realistic amount of work based on their historical velocity.
- **Value Based Prioritization:** The Product Owner must prioritize the backlog using techniques like MoSCoW (Must have, Should have, Could have, Won't have). This ensures that the development team focuses exclusively on delivering the highest business value items first.

Q3. A development team introduces Test-Driven Development (TDD) in their Agile workflow. Discuss how TDD can improve software quality and reduce defects in the system.

Test-Driven Development is an Extreme Programming engineering practice where automated test cases are written before the actual production code. Introducing this methodology into a workflow fundamentally shifts the development process from verification to active design. This approach significantly improves overall code quality, ensures high verifiability, and minimizes the number of escaped defects that reach the end user.

- **The Red Green Refactor Cycle:** Developers first write a failing unit test for a new feature, then write the simplest code necessary to pass the test, and finally clean up the code. This strict cycle ensures that every single function is covered by verification tests.
- **Built-in Regression Testing:** TDD naturally produces a comprehensive suite of automated tests. This suite acts as a safety net, ensuring that new code modifications do not accidentally break existing functionality, thereby reducing the defect density over time.
- **Architectural Decoupling:** Writing tests first forces developers to view the program from the perspective of the caller. To make the code testable, modules must be isolated from their surroundings, often using Mock Objects, which leads to a highly decoupled and flexible architecture.
- **Executable Documentation:** The unit tests serve as compileable, reliable documentation. They provide a suite of accurate examples that help other programmers understand how to call functions and interact with the code without relying on outdated external documents.
- **Safe Refactoring:** Code tends to degrade as new features are added. The comprehensive test suite provided by TDD gives developers the confidence to continuously restructure and improve the internal code structure without fear of altering its external behavior.

Q4. A large project has multiple Scrum teams working together. The organization decides to implement Scrum of Scrums. Explain how Scrum of Scrums improves coordination and communication among teams.

When a software project scales beyond a single Agile team of optimal size (usually seven plus or minus two members), managing technical boundaries and coordination becomes highly challenging. Scrum of Scrums is an advanced Agile scaling framework designed to manage cross team dependencies,

eliminate integration bottlenecks, and maintain architectural alignment across multiple teams working concurrently on the same product.

- **Designated Representation:** Each individual team holds their own daily stand up. Following this, a designated representative (usually the Scrum Master or Technical Lead) from each team attends the higher level Scrum of Scrums meeting to synchronize efforts.
- **Cross Team Dependency Identification:** The meeting follows a specific agenda focused on overlapping work. Representatives discuss what their team has done, what they will do, and crucially, whether they are about to put an obstacle in another team's way.
- **Impediment Resolution:** The framework allows for rapid escalation of blocking issues. If one team is waiting on an API or database schema from another team, the Scrum of Scrums provides a formal venue to negotiate priorities and resolve the blocker immediately.
- **Synchronization of Integration:** Large projects face high risks during code integration. This coordination meeting allows teams to plan their automated builds and integration points effectively, avoiding the severe delays associated with integration hell.
- **Preserving Small Team Dynamics:** Instead of creating one massive, unmanageable team with high communication overhead, Scrum of Scrums allows the organization to scale while preserving the flexibility, self organization, and sustainable pace of the individual Agile teams.

Q5. A project manager notices frequent build failures and integration issues in an Agile project. Explain how continuous integration and automated build tools can help solve this problem.

Frequent build failures and severe integration issues, often referred to as integration hell, typically occur when developers isolate their code on local branches for extended periods. Continuous Integration (CI) is a core Agile practice where developers integrate their code into a shared repository frequently. Combined with automated build tools, this practice ensures that the software remains in a verifiable and deployable state at all times.

- **Frequent Code Commits:** Developers are required to check in their code and integrate it several times a day. This minimizes the size of each merge, making conflicts trivial to resolve compared to massive integrations at the end of a sprint.
- **Automated Build Triggers:** Automated CI servers (such as Jenkins or GitLab CI) monitor the version control repository. The moment a developer commits new code, the CI server automatically pulls the code and compiles the entire system from end to end.
- **Automated Unit Testing:** Immediately after compilation, the CI tool runs the entire suite of unit and acceptance tests. This ensures that the newly integrated code satisfies all requirements and has not broken any existing functionality.
- **Immediate Feedback Loop:** If a build or test fails, the automated tool alerts the team instantly. In Agile, fixing a broken build becomes the highest priority, preventing the accumulation of errors and ensuring the baseline is always stable.
- **Mitigation of Merge Conflicts:** By utilizing non-blocking source control and trunk based development strategies, developers continuously merge with the latest code. This completely eliminates the severe risk of late stage integration failures that plague traditional development lifecycles.

Section E: Numerical / Analytical Problems

Q1. A Scrum team completes 20 story points in Sprint 1, 25 story points in Sprint 2, and 30 story points in Sprint 3. Calculate the average velocity of the team.

Velocity is an Agile metric used to measure the amount of work a development team can successfully complete within a single sprint or iteration.

Given:

- Story points completed in Sprint 1 (S_1) = 20
- Story points completed in Sprint 2 (S_2) = 25
- Story points completed in Sprint 3 (S_3) = 30
- Number of sprints (n) = 3

Formula: The average velocity V over n sprints is given by the formula:

$$V = \frac{1}{n} \sum_{i=1}^{n} S_i$$

Calculation:

$$V = \frac{20+25+30}{3}$$
$$V = \frac{75}{3}$$
$$V = 25 \text{ story points}$$

Conclusion: The average velocity of the Scrum team is 25 story points per sprint. This figure should be used as the team's capacity limit for forecasting the workload of Sprint 4.

Q2. A product backlog contains 120 story points. If the team's average sprint velocity is 20 story points per sprint, estimate the number of sprints required to complete the backlog.

Given:

- Total Product Backlog Estimation (E) = 120 story points
- Average Team Velocity (V) = 20 story points per sprint

Formula: The required number of sprints (N) is formulated as the ceiling of the total effort divided by velocity:

$$N = \left\lceil \frac{E}{V} \right\rceil$$

Calculation:

$$N = \left\lceil \frac{120}{20} \right\rceil$$
$$N = \lceil 6 \rceil$$
$$N = 6 \text{ sprints}$$

Conclusion: The team will require exactly 6 sprints to complete the current product backlog. If the standard sprint duration is two weeks, the estimated time-to-market would be approximately 12 weeks.

Q3. A project has 5 developers working 6 hours per day for 10 days in a sprint. Calculate the total development effort available in person-hours.

During iteration planning, the Scrum Master and development team calculate their total capacity by multiplying the number of team members by their daily available working hours and the total number of working days in the sprint, factoring in constraints like planned time off or organizational meetings.

Given:

- Number of developers (D) = 5
- Working hours per developer per day (H) = 6 hours
- Duration of the sprint in working days (T) = 10 days

Formula:

$$\text{Total Capacity (Person-Hours)} = D \times H \times T$$

Calculation:

$$\text{Total Capacity} = 5 \text{ developers} \times 6 \text{ hours/day} \times 10 \text{ days}$$

$$\text{Total Capacity} = 30 \text{ person-hours/day} \times 10 \text{ days}$$

$$\text{Total Capacity} = 300 \text{ person-hours}$$

Conclusion: The development team has a total working capacity of 300 person-hours for this sprint. When breaking down user stories into technical tasks, the total estimated hours for all tasks combined must not exceed this 300-hour threshold to ensure a realistic and sustainable sprint commitment.

Q4. In an Agile project, the planned story points for a sprint were 40, but only 32 story points were completed. Calculate the sprint completion percentage.

The sprint completion percentage is used to assess the accuracy of the team's sprint planning and their overall commitment reliability.

Given:

- Planned Story Points (P) = 40
- Completed Story Points (C) = 32

Formula:

$$\text{Sprint Completion Percentage} = \left(\frac{C}{P}\right) \times 100$$

Calculation:

$$\text{Sprint Completion Percentage} = \left(\frac{32}{40}\right) \times 100$$

$$\text{Sprint Completion Percentage} = 0.80 \times 100$$

$$\text{Sprint Completion Percentage} = 80\%$$

Conclusion: The team successfully completed 80% of their planned workload for the sprint. The remaining 20% (8 story points) represents incomplete work that must be re-evaluated, re-prioritized by the Product Owner, and likely moved to the subsequent sprint's backlog.

Q5. A team has an average velocity of 18 story points per sprint. If the remaining backlog contains 90 story points, estimate the number of sprints required to finish the project.

Given:

- Remaining Product Backlog Estimation (E) = 90 story points
- Average Team Velocity (V) = 18 story points per sprint

Formula: The required number of remaining sprints (N) is formulated as the ceiling of the remaining effort divided by the average velocity:

$$N = \left\lceil \frac{E}{V} \right\rceil$$

Calculation:

$$N = \left\lceil \frac{90}{18} \right\rceil$$

$$N = \lceil 5 \rceil$$

$$N = 5 \text{ sprints}$$

Conclusion: Based on the team's current velocity, an estimated 5 additional sprints are required to completely clear the remaining 90 story points in the backlog.