

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES, CHITTOOR
(AUTONOMOUS)

Approved by AICTE, New Delhi, Affiliated to JNTUA, Ananthapuramu.



LECTURE NOTES

Subject Name: Agile Methodologies

Year/Branch: IV/CSE

Regulation: R23

Prepared By: Mr.Pandreti Praveen

1. Vision, Mission, POs, PEOs, PSOs (If any) DEPARTMENT VISION

To contribute for the society through excellence in Computer Science and Engineering with a deep passion for wisdom, culture and values.

DEPARTMENT MISSION

M1: Provide congenial academic ambience with necessary infrastructure and learning resources.

M2: Inculcate confidence to face and experience new challenges from industry and society.

M3: Ignite the students to acquire self-reliance in State-of-the-Art Technologies

M4: Foster Enterprising spirit among students.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Excel in Computer Science and Engineering program through quality studies, enabling success in computing industry.(Professional Competency).

PEO2: Surpass in one's career by critical thinking towards successful services and growth of the organization, or as an entrepreneur or in higher studies. (Successful Career Goals).

PEO3: Enhance knowledge by updating advanced technological concepts for facing the rapidly changing world and contribute to society through innovation and creativity (Continuing Education and Contribution to Society).

Program Specific Outcomes (PSOs)

PSO1: Have Ability to understand, analyze and develop computer programs in the areas like algorithms, system software, web design, big data analytics, and networking.

PSO2: Deploy the modern computer languages, environment, and platforms in creating innovative products and solutions.

1. Syllabus Contents (In-line with OBE)

UNIT- I AGILE METHODOLOGY

Lecture 9 Hrs

Theories for Agile Management – Agile Software Development – Traditional Model vs. Agile Model - Classification of Agile Methods – Agile Manifesto and Principles – Agile Project Management – Agile Team Interactions – Ethics in Agile Teams - Agility in Design, Testing – Agile Documentations – Agile Drivers, Capabilities and Values

UNIT- II AGILE PROCESSES

Lecture 8 Hrs

Lean Production - SCRUM, Crystal, Feature Driven Development- Adaptive Software Development - Extreme Programming: Method Overview – Lifecycle – Work Products, Roles and Practices.

UNIT- III AGILITY AND KNOWLEDGE MANAGEMENT

Lecture 8 Hrs

Agile Information Systems – Agile Decision Making - Earl_S Schools of KM – Institutional Knowledge Evolution Cycle – Development, Acquisition, Refinement, Distribution, Deployment, Leveraging – KM in Software Engineering – Managing Software Knowledge – Challenges of Migrating to Agile Methodologies – Agile Knowledge Sharing – Role of Story-Cards – Story-Card Maturity Model (SMM).

UNIT- IV AGILITY AND REQUIREMENTS ENGINEERING

Lecture 9 Hrs

Impact of Agile Processes in RE–Current Agile Practices – Variance – Overview of RE Using Agile Managing Unstable Requirements – Requirements Elicitation – Agile Requirements Abstraction Model – Requirements Management in Agile Environment, Agile Requirements Prioritization – Agile Requirements Modeling and Generation – Concurrency in Agile Requirements Generation.

UNIT- V AGILITY AND QUALITY ASSURANCE

Lecture 9 Hrs

Agile Product Development – Agile Metrics – Feature Driven Development (FDD) – Financial and Production Metrics in FDD – Agile Approach to Quality Assurance - Test Driven Development – Agile Approach in Global Software Development.

3. Objectives of the Subject

The main objectives of this course are to:

1. **Understand** the fundamental concepts, principles, values, and practices of Agile Software Development and compare them with traditional software development methodologies.
2. **Develop** knowledge of various Agile methodologies such as Scrum, Extreme Programming (XP), Crystal, Lean Software Development, Adaptive Software Development (ASD), and Feature Driven Development (FDD).
3. **Apply** Agile techniques for project management, requirements engineering, knowledge management, and collaborative software development in dynamic environments.
4. **Analyze** Agile requirements engineering practices, including requirements elicitation, prioritization, modeling, management, and handling changing customer requirements.
5. **Evaluate** Agile quality assurance techniques, Agile metrics, Test-Driven Development (TDD), and Agile approaches in global software development to deliver high-quality software products.
6. **Enhance** teamwork, communication, ethical values, leadership skills, and continuous learning required for successful Agile software development.

4. IntroductionPart

a) Clearly Define the Contents and Key Terminologies

What is Agile?

Agile is a modern software development methodology that emphasizes **iterative development, customer collaboration, flexibility, continuous feedback, and rapid delivery of high-quality software**. Agile enables development teams to adapt to changing requirements throughout the software development life cycle.

According to the Agile Manifesto (2001):

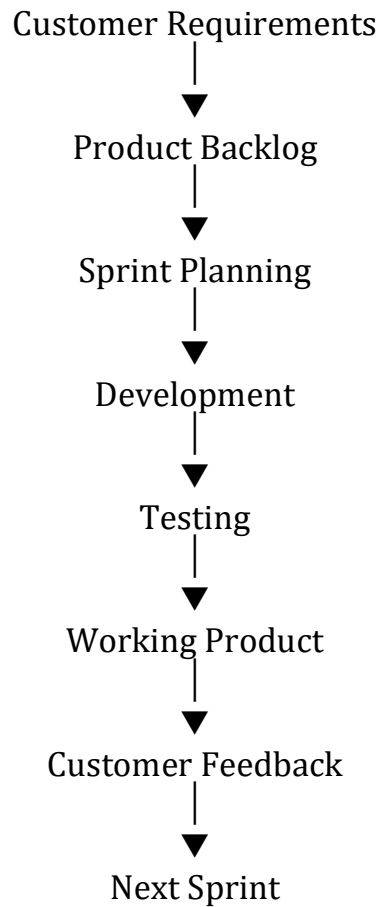
"Agile is a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams."

Key Terminologies

Term	Definition
Agile	Iterative and incremental software development approach
Sprint	A fixed development cycle (usually 2–4 weeks)
Scrum	Agile project management framework
Product Backlog	Prioritized list of project requirements
Sprint Backlog	Tasks selected for the current sprint
Product Owner	Represents customer requirements
Scrum Master	Facilitates Scrum practices
Development Team	Cross-functional software development team
Increment	Working software produced after every sprint
User Story	Short description of a feature from the user's perspective
Velocity	Amount of work completed in a sprint
Burndown Chart	Graph showing remaining work over time

b) Conceptual Depth with Suitable Illustration

Traditional software development follows a **linear approach**, whereas Agile follows an **iterative and incremental approach**.

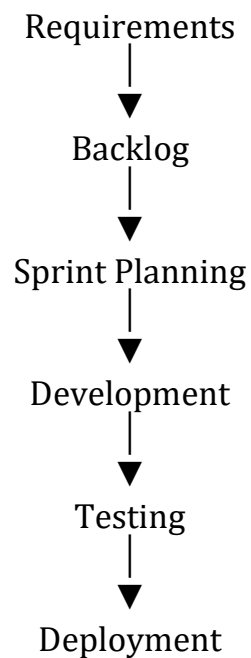


The process repeats until all requirements are completed.

c) Standard Symbols and Representation

Although Agile does not use electrical symbols or circuit diagrams, standard project representations are commonly used.

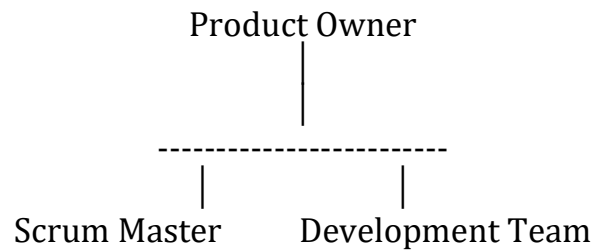
Agile Workflow





Review & Retrospective

Scrum Roles



d) Importance of Agile Methodology

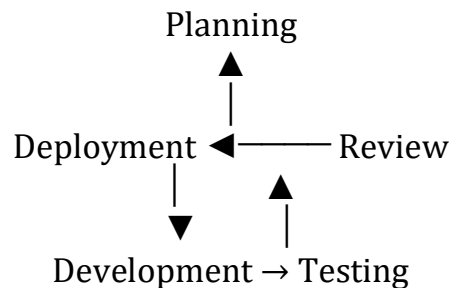
Agile has become the preferred software development methodology because it:

- Delivers software quickly
- Accepts changing requirements
- Improves customer satisfaction
- Reduces project risks
- Enhances product quality
- Encourages teamwork
- Enables continuous improvement
- Increases productivity

Today, most IT companies use Agile methodologies for software development.

e) Explanation with Suitable Figures

Agile Development Cycle



Continuous feedback ensures quality improvement.

f) Real-World Examples

Example 1: Amazon

Amazon releases software updates multiple times every day using Agile principles.

Example 2: Google

Google develops many cloud applications using Scrum and Agile methodologies.

Example 3: Microsoft

Microsoft adopted Agile practices for Windows, Azure, and Microsoft 365 development.

Example 4: Netflix

Netflix continuously deploys new features based on customer feedback using Agile methodologies.

g) Advantages and Disadvantages

Advantages

- Faster software delivery
- Better customer involvement
- Higher product quality
- Continuous improvement
- Flexible to changing requirements
- Reduced development risk
- Improved team collaboration
- Early detection of defects
- Increased customer satisfaction
- Better project visibility

Disadvantages

- Difficult for large distributed teams
- Requires experienced team members
- Customer involvement is mandatory
- Frequent requirement changes may increase cost
- Documentation is comparatively less
- Scope creep may occur
- Hard to estimate project completion initially

h) Applications and Limitations

Applications

- Software Development
- Mobile Application Development
- Web Development
- Cloud Computing
- Artificial Intelligence Projects
- Machine Learning Systems
- DevOps Projects
- Banking Software
- Healthcare Information Systems
- E-commerce Platforms
- Embedded Software
- Government Digital Services

Limitations

- Not suitable for projects with fixed requirements
- Less suitable where documentation is mandatory
- Difficult in highly regulated industries without adaptation
- Challenging for inexperienced Agile teams
- Scaling Agile across very large organizations requires additional frameworks

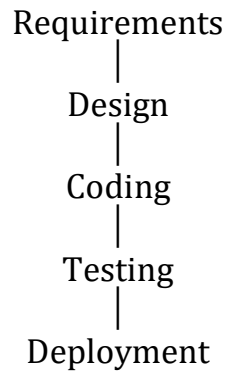
i) Comparison Table

Traditional Model vs Agile Model

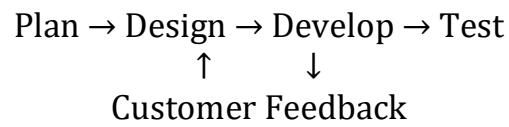
Feature	Traditional Model	Agile Model
Development Style	Sequential	Iterative
Customer Involvement	Low	High
Requirement Changes	Difficult	Easy
Delivery	End of Project	Frequent
Testing	After Development	Continuous
Documentation	Extensive	Lightweight
Risk	Higher	Lower
Team Collaboration	Moderate	High
Feedback	Limited	Continuous
Product Quality	Moderate	High

Graphical Representation

Traditional Model



Agile Model



j) Relevant Case Study

Case Study: Spotify's Agile Transformation

Spotify adopted Agile using small, autonomous teams called **Squads**, grouped into **Tribes**, supported by **Chapters** and **Guilds**. This structure improved collaboration, accelerated feature delivery, and encouraged continuous innovation while maintaining engineering quality.

Key Outcomes:

- Faster product releases
- Improved team autonomy
- Better customer satisfaction
- Continuous improvement culture

k) Safety & Standards Component

Although Agile is a software methodology, development teams should follow recognized standards and good practices.

Relevant Standards

- **ISO 9001** – Quality Management Systems
- **ISO/IEC 12207** – Software Life Cycle Processes
- **ISO/IEC 25010** – Software Product Quality Model
- **IEEE 830** – Software Requirements Specification (historical)
- **IEEE 29148** – Requirements Engineering Standard
- **IEEE 829** – Software Testing Documentation (historical)
- **ISO/IEC/IEEE 15288** – System Life Cycle Processes

Limitations

1. Cannot fully replicate human emotions
2. Depends heavily on training data
3. Requires significant computational resources
4. Limited common-sense reasoning
5. Difficult to explain some AI decisions
6. Vulnerable to adversarial attacks
7. Ethical and legal concerns

AGILE METHODOLOGIES

23CSE472A

UNIT-I

Syllabus

UNIT- I AGILE METHODOLOGY

Lecture 9 Hrs

Theories for Agile Management – Agile Software Development – Traditional Model vs. Agile Model - Classification of Agile Methods – Agile Manifesto and Principles – Agile Project Management – Agile Team Interactions – Ethics in Agile Teams - Agility in Design, Testing – Agile Documentations – Agile Drivers, Capabilities and Values

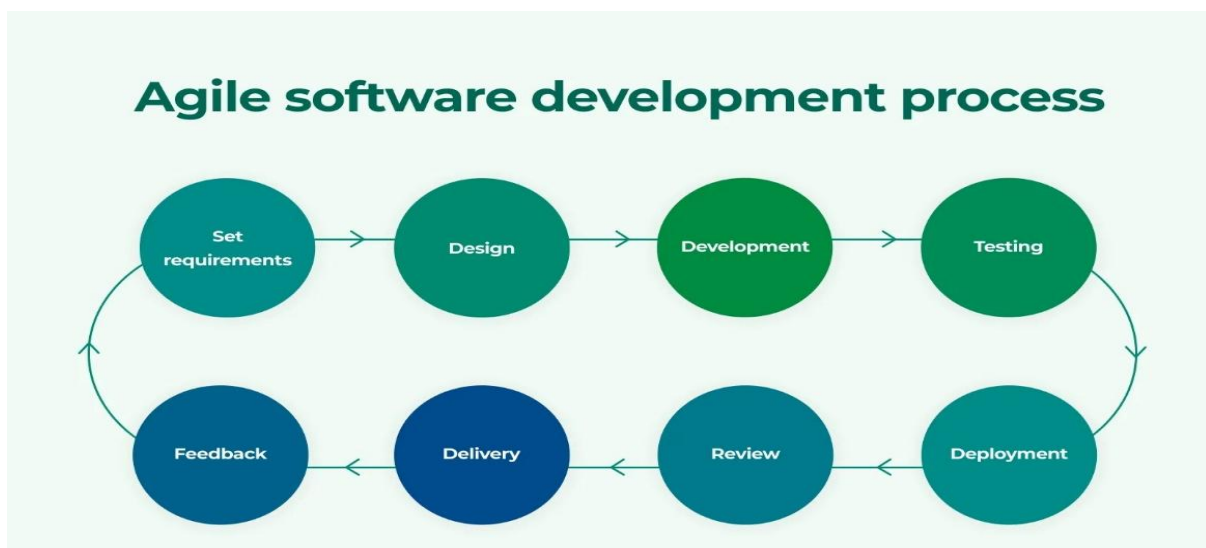
AGILE METHODOLOGIES

MODULE -1

Theories for Agile Management – Agile Software Development – Traditional Model vs Agile Model – Classification of Agile Methods – Agile Manifesto and Principles – Agile Project Management – Agile Team Interactions – Ethics in Agile Teams – Agility in Design, Testing – Agile Documentations – Agile Drivers, Capabilities and Values.

Agile Methodology

- Agile Methodology is a way to manage projects by breaking them into smaller parts. It focuses on working together and making constant improvements. Teams plan, work on the project, and then review in a repeating cycle.
- It focuses on delivering smaller pieces of work regularly instead of one big launch.
- This allows teams to adapt to changes quickly and provide customer value faster.
- Major companies like Facebook, Google, and Amazon use Agile because of its adaptability and customer-focused approach.
- Prioritize flexibility, collaboration, and customer satisfaction.



Key Principles of Agile Methodology

Agile is guided by 12 [simple principles](#) that help teams work better, adapt quickly, and deliver value to customers. These focus on collaboration, flexibility, and continuous improvement.

- **Customer Satisfaction:** Deliver valuable software early and continuously to keep customers happy.
- **Welcome Change:** Accept changing requirements even in later stages of development.
- **Frequent Delivery:** Deliver working software in short, regular intervals.
- **Collaboration:** Ensure daily cooperation between business teams and developers.
- **Motivated Teams:** Build projects around skilled, trusted, and motivated individuals.
- **Face-to-Face Communication:** Prefer direct communication for better clarity and understanding.
- **Working Software:** Measure progress based on functional software, not just documentation.
- **Sustainable Pace:** Maintain a consistent and manageable development speed.
- **Technical Excellence:** Focus on good design and high-quality coding practices.
- **Simplicity:** Minimize unnecessary work and keep solutions simple.
- **Self-Organizing Teams:** Encourage teams to take ownership and make decisions.
- **Continuous Improvement:** Regularly reflect and improve processes for better results.

Theories for Agile Management:

Agile management theories prioritize iterative development, customer collaboration, and flexibility over strict, linear planning.

Key foundational theories include

Lean manufacturing (reducing waste),

empirical process control (transparency, inspection, adaptation), and social science theories focused on self-organizing teams.

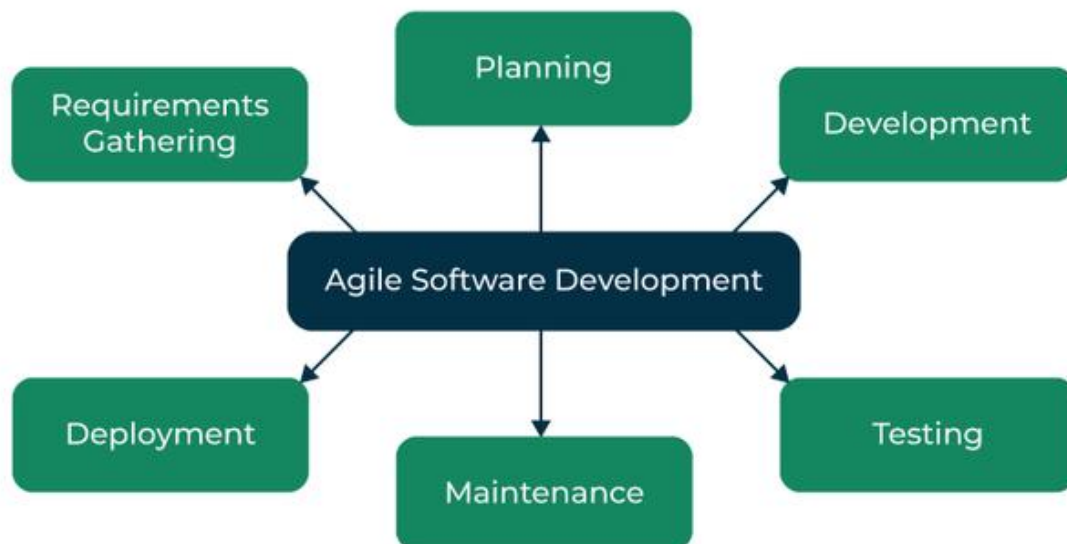
- **Empirical Process Control (Scrum Theory):** Based on three pillars: Transparency (visible work), Inspection (regular check-ins), and Adaptation (changing based on results).
- **Lean Manufacturing/Production:** Focuses on eliminating waste, amplifying learning, delivering as fast as possible, and empowering the team.
- **Cynefin Framework:** A decision-making framework that helps leaders determine whether a project is "Simple," "Complicated," "Complex," or "Chaotic" to choose the right approach (Agile fits best in complex environments).

- **Systems Theory and Cybernetics:** Views teams as complex adaptive systems, focusing on self-organization, homeostasis (maintaining stability), and responding to change (Ashby's Law).
- **Theory of Constraints (ToC):** Focuses on identifying and managing bottlenecks to speed up value delivery.

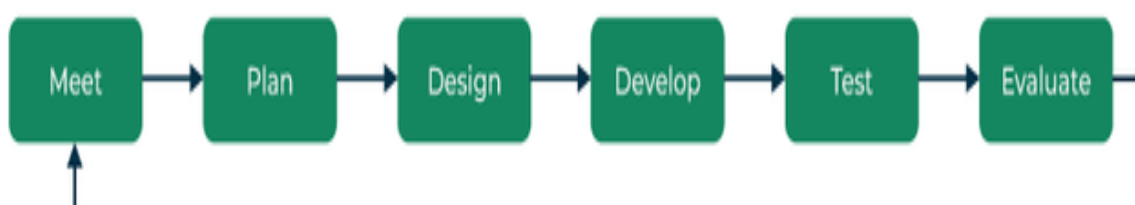
Agile Software Development

Agile Software Development is a software development methodology that values flexibility, collaboration, and customer satisfaction. It is an iterative and incremental approach to the importance of delivering a working product quickly and frequently. It involves close collaboration between the development team and the customer to ensure that the product meets their needs and expectations.

- **Agile as a Mindset:** Agile represents a shift in culture that values adaptability, collaboration, and client happiness, and it gives team members more authority.
- **Quick Response to Change:** Agile fosters a culture that allows teams to respond swiftly to constantly shifting priorities and requirements.
- **Regular Demonstrations:** Agile techniques place a strong emphasis on regular demonstrations of project progress.
- **Cross-Functional Teams:** Agile fosters self-organizing, cross-functional teams that share information effectively and communicate more effectively.



Agile Software Development Cycle

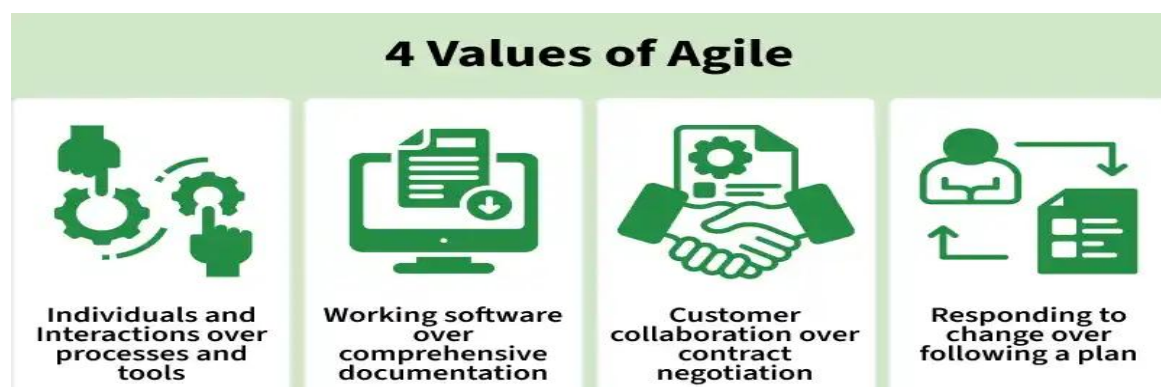


- **Meet:** Initial stage, stakeholders and team members meet to discuss the project idea, business goals, and customer requirements.
- **Plan:** planning phase, tasks are identified, and a timeline is created. The team estimates effort, allocates resources, and prepares for the upcoming iteration.
- **Design:** The system architecture and design are prepared based on the requirements.
- **Develop:** The team starts building the software according to the design. Coding is done in small increments, and features are developed step by step.
- **Test:** The software is tested to ensure it works correctly and meets the requirements. Bugs and errors are identified and fixed to maintain quality.
- **Evaluate:** the completed work is reviewed with team members and feedback is collected so that improvements are identified.

Values of Agile Software Development

The four core values of Agile software development, as outlined in the Agile Manifesto, focus on what truly matters for creating successful software.

- 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 down in lengthy documentation.
- 3. Customer Collaboration over Contract Negotiation:** Agile values regular collaboration with customers over sticking strictly to contracts. The idea is to involve the customer throughout the development, verifying that the product meets their needs.
- 4. Responding to Change over Following a Plan:** In Agile, change is expected, and the approach encourages flexibility. Rather than rigidly following a plan that may no longer apply, Agile teams adapt and adjust based on new information, changing market conditions.



Classification (Types) of Agile Methods

Agile is a flexible framework with several approaches, each suited for different project needs. Below are some of the most common Agile methodologies, formatted for clarity:

- **Kanban**: Uses a visual workflow and limits work-in-progress (WIP) to manage continuous tasks and improve delivery speed.
- **Scrum**: Works in short sprints with defined roles and meetings to deliver focused and incremental results.
- **Extreme Programming (XP)**: Focuses on strong coding practices, continuous feedback, and frequent releases for fast-changing requirements
- **Adaptive Project Framework (APF)**: Adapts to changing requirements, timelines, and resources during project execution.
- **Extreme Project Management (XPM)**: Designed for highly uncertain projects, using rapid iterations and constant adjustments.
- **Adaptive Software Development (ASD)**: Follows Speculate–Collaborate–Learn cycle for continuous learning and flexibility.
- **Dynamic Systems Development Method (DSDM)**: Provides a structured Agile approach covering the complete project lifecycle.
- **Feature Driven Development (FDD)**: Focuses on building and delivering small, client-valued features incrementally



Applications of the Agile Methodology

- **Unclear or Changing Requirements**: Agile is great for projects with requirements that aren't well-defined or might change.

- **Complex Projects:** It's good for big, complex projects by breaking them into smaller pieces.
- **Customer Focus:** Use Agile when making customers happy is a priority and you want to involve them throughout.

Quick Time-to-Market: If you need to get your product out fast, Agile can help.

- **Small to Medium Teams:** Agile works well for teams of a few to a few dozen people.
- **Collaboration:** Agile promotes working together and open communication.
- **Regular Updates:** If you want to check progress often and make changes as needed.
- **Continuous Improvement:** Agile fosters a culture of always getting better over time.

Characteristics of the Agile Process

1. Regular, Fixed-Length Iterations
2. Focus on Functional and Tested Features
3. Prioritizing Business Value
4. Planning at Release and Iteration Levels
5. Compact, Cross-Functional Teams
6. Continuous Improvement Culture

Advantages

- **Improved collaboration and communication:** Agile development emphasizes close interaction among team members, stakeholders, and customers.
- **Flexibility and adaptability:** It easily accommodates changes in requirements, priorities, or market conditions.
- **Better quality and reliability:** Continuous testing and improvement help ensure high-quality software.
- **Higher customer satisfaction:** Focus on delivering value ensures the product meets customer expectations.
- **Increased team morale:** A collaborative and supportive environment boosts motivation and productivity.

Limitations of Agile Methodology

- **Less Documentation:** Agile methodologies focus on less documentation; it prioritizes working on projects rather than paperwork.
- **Challenges in Large Organizations:** Busy schedule of clients can make daily meetup and face-to-face communication difficult.

- **Need for Senior Programmers:** It may require experienced programmers to make critical decisions during the development of software.
- **Limited Scope Control:** It has less rigid scope control, which may not be suitable in certain situations
- **Predictability:** Compared to more structured project management methods, it may lack predictability.

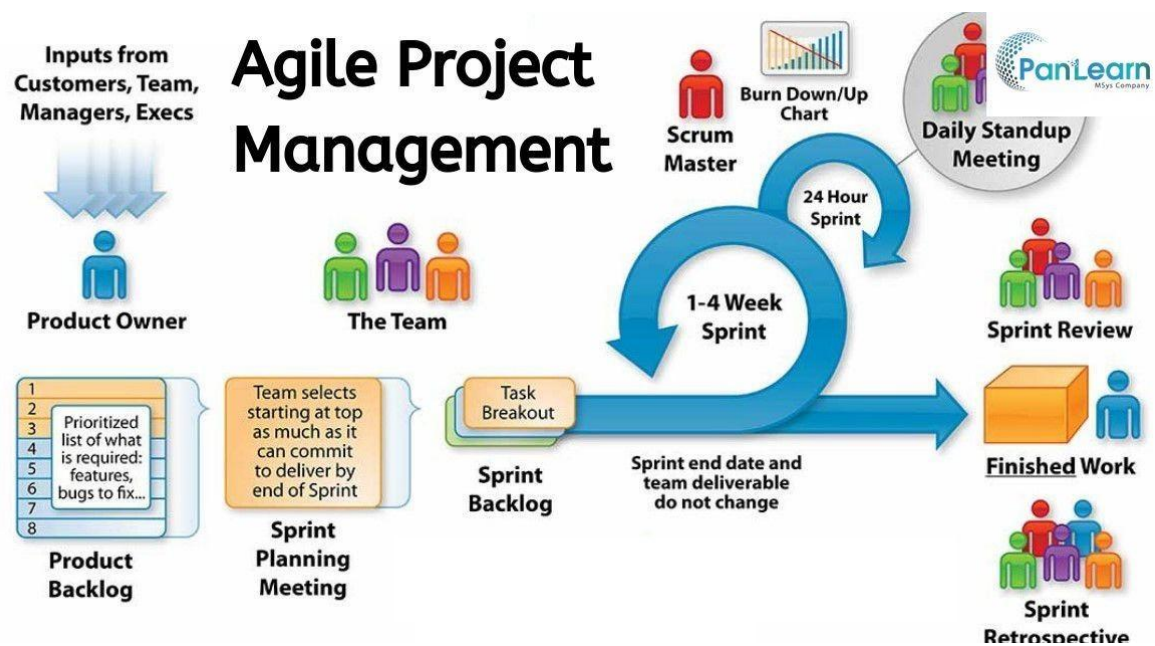
Difference Between Traditional and Agile Software Development

Traditional Software Development	Agile Software Development
It is used to develop simple software.	It is used to develop complicated software.
In this methodology, testing is done once the development phase is completed.	In this methodology, testing and development processes are performed concurrently.
It follows a linear organizational expectation structure.	It follows an iterative organizational structure.
It provides less security.	It provides high security.
Client involvement is less as compared to Agile development.	Client involvement is high as compared to traditional software development.
It provides less functionality in the software.	It provides all the functionality needed by the users.
It supports a fixed development model.	It supports a changeable development model.
It is used by freshers.	It is used by professionals.
Development cost is less using this methodology.	Development cost is high using this methodology.

Traditional Software Development	Agile Software Development
It majorly consists of five phases. It is less used by software development firms.	It consists of only three phases. It is normally used by software development firms.
The expectation is favored in the traditional model. Traditional software development approaches are formal in terms of communication with customers. For starters, typical software development approaches employ a predictive approach. There is full specification and prediction of the software development processes because the product is produced through rigorous and explicit planning. Changes are not permitted in this technique because the time and cost of project development are fixed. Examples • Office productivity suites • Data management software • Media players • Security programs Models based on Traditional Software Development- • Spiral Model • Waterfall Model • V Model • Incremental Model	Adaptability is favored in the agile methodology. Agile software development methodologies are casual. In other words, customers who work with companies that utilize Agile software development approaches are more likely to interact with them than customers who work with companies that use traditional software development methodology. Here, a flexible approach is used as the software development approaches are founded on the notion of continual design improvement and testing relies on team and client feedback. Examples • Sky • Phillips • JP Morgan Chase Models based on Agile Software Development- • Scrum • Extreme Programming (XP) • Crystal • Dynamic Systems Development Method (DSDM)

Traditional Software Development	Agile Software Development
	• Feature Driven Development (FDD) • Adaptive Software Development (ASD)

Agile Project Management



1. Inputs from Customers, Team, Managers, and Executives

At the beginning, requirements and ideas are collected from:

Customers

Development Team

Managers

Executives

These inputs help decide what features or improvements are needed in the product.

2. Product Owner

The **Product Owner** is responsible for:

- Understanding customer requirements
- Managing the product features

- Prioritizing work based on business value

The Product Owner creates and maintains the **Product Backlog**.

3. Product Backlog

The **Product Backlog** is a prioritized list of:

Features

Enhancements

Bug fixes

Customer requirements

The most important items are placed at the top.

Example:

Login feature

Payment gateway

Bug fixes

Security updates

The backlog changes continuously as new requirements arrive.

4. Sprint Planning Meeting

During the **Sprint Planning Meeting**:

The team selects tasks from the Product Backlog.

The team decides how much work can be completed during the sprint.

Responsibilities are discussed and assigned.

The selected tasks become the **Sprint Backlog**.

5. Sprint Backlog

The **Sprint Backlog** contains:

Tasks selected for the current sprint

Technical work breakdown

Daily implementation activities

This backlog remains stable during the sprint.

6. Sprint (1–4 Weeks)

A **Sprint** is a short development cycle usually lasting:

1 week - 2 weeks, Up to 4 weeks

During the sprint:

Developers code the features

Testers test the application

Team members collaborate continuously

The sprint goal must be completed within the fixed time

7. Daily Stand-up Meeting

Every day, the team conducts a short meeting called the **Daily Stand-up** (usually 15 minutes).

Each member answers:

What did I complete yesterday?

What will I do today?

Are there any obstacles?

This helps track progress and solve issues quickly.

8. Scrum Master

The **Scrum Master** acts as a facilitator.

Responsibilities include:

Guiding the Agile process

Removing obstacles

Ensuring Scrum practices are followed

Supporting the team

The Scrum Master does not manage the team directly but helps them work effectively.

9. Burn Down / Burn Up Chart

These charts are used to measure project progress.

Burn Down Chart

Shows: Remaining work versus time

Burn Up Chart

Shows: Completed work over time

They help determine whether the sprint is on schedule.

10. Finished Work (Product Increment)

At the end of the sprint, the team delivers:

Completed features ,Tested software, Working product increment

This is called the **Finished Work** or **Increment**.

11. Sprint Review

In the **Sprint Review Meeting**:

- The completed work is demonstrated to stakeholders.
- Feedback is collected from customers and management.
- New requirements may be added to the Product Backlog.

12. Sprint Retrospective

After the review, the team conducts a **Sprint Retrospective**.

The team discusses:

- What went well
- What problems occurred
- How to improve in the next sprint

This supports continuous improvement.

Overall Agile Workflow

Step-by-Step Flow

- Requirements are collected
- Product Backlog is prepared
- Sprint Planning is conducted
- Sprint Backlog is created
- Team works during Sprint
- Daily Standup meetings occur
- Product Increment is delivered
- Sprint Review is conducted
- Retrospective improves future work
- Next Sprint begins

Advantages of Agile Project Management

Faster delivery of software

- Better customer satisfaction
- Flexible to requirement changes
- Continuous feedback and improvement

- Improved teamwork and communication
- Reduced project risk

Why do we need Agile Project Management

We need [Agile Project Management](#) because traditional [project management methodologies](#) often struggle to adapt to the rapidly changing demands of modern business environments. Agile offers several benefits:

- **Flexibility:** Teams can quickly respond to changing requirements and priorities, delivering value effectively in dynamic environments.
- **Customer Satisfaction:** Frequent collaboration and short development cycles ensure products meet customer needs and expectations.
- **Efficiency:** Iterative development and continuous improvement streamline processes, reduce waste, and accelerate time-to-market.
- **Transparency:** Open communication within teams and with stakeholders builds trust and accountability.
- **Adaptability:** Teams learn from experience and adjust their approach, improving outcomes and reducing the risk of project failure.

How Agile Project Management (APM) Works

- **Iterative Development:** Large projects are divided into smaller units called iterations or sprints, typically lasting 1–4 weeks. Each iteration results in a potentially shippable product increment.
- **Collaboration:** Agile promotes teamwork among developers, designers, and stakeholders, with constant communication to ensure alignment and smooth implementation of changes.
- **Customer Feedback:** Frequent input from customers is integrated into the process, allowing teams to adjust priorities and improve the product based on real user needs.
- **Continuous Improvement:** After each iteration, teams hold retrospectives to identify successes and areas for improvement, fostering a culture of learning and adaptation.

When to Use Agile Project Management?

Agile Project Management is not suitable for every project, but it excels in situations requiring flexibility and continuous feedback. It is best used when:

- **Unclear Requirements:** The project scope or details are not fully defined at the start.
- **Continuous Customer Feedback:** Frequent input from customers is needed to shape the product.
- **Experienced Teams:** Teams are skilled, self-organizing, and capable of managing iterative workflows.

Phases of Agile Project Management

1. Envision

Define the project's goals and purpose., Create a prioritized list of features and requirements.

2. Speculate

Break the project into smaller chunks called sprints., Conduct detailed planning and select user stories from the product backlog.

3. Explore

Execute sprints, typically lasting 1–4 weeks., Hold daily Scrum meetings to discuss progress, challenges, and plans.

4. Adapt

Continuously adjust the plan based on client feedback. The Scrum Master monitors progress and helps resolve team challenges. Focus on what has been achieved and what needs improvement.

5. Close

Deploy the final product or updated version., Collect customer feedback for future improvements.

Benefits of Agile Project Management (APM)

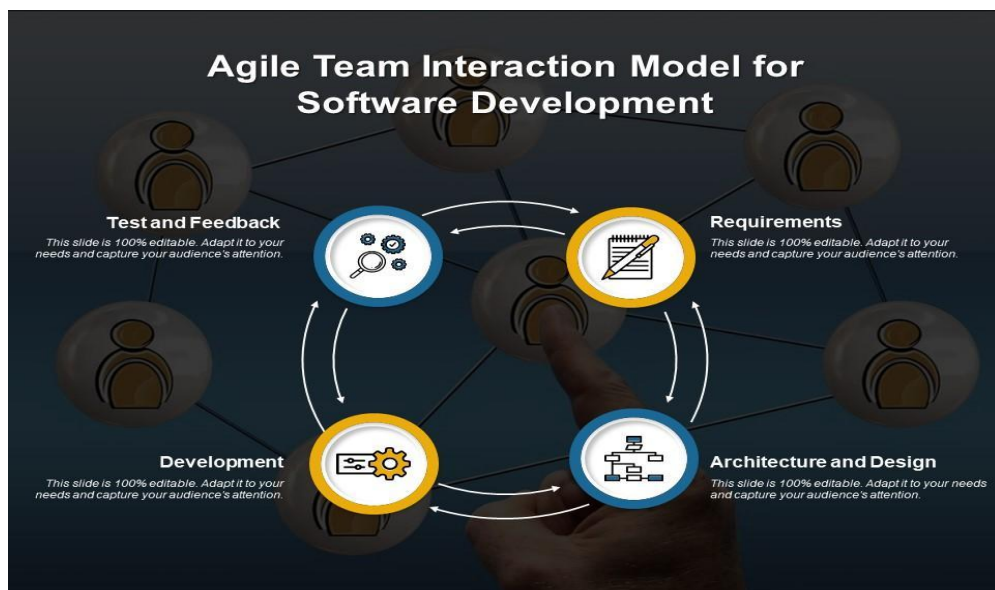
- **Rapid development of projects:** Agile accelerates development by minimizing time spent on documentation and promoting active collaboration with the client.
- **Freedom of ideas:** Encourages team members to contribute suggestions and innovative solutions.
- **Flexible and adaptable to the changes:** Unlike rigid methodologies, Agile welcomes customer feedback and adapts to changes quickly.
- **Entertains customer feedback:** Continuous collaboration with customers ensures the product aligns with their expectations.
- **Faster issue or defect detection:** Iterative cycles allow teams to identify and address defects early in the development process.
- **Continuous involvement of customers:** Maintains collaboration between the team and customers throughout the project lifecycle.
- **Efficient use of resources:** Optimizes resource allocation, reduces waste, and lowers development costs.

Limitations of Agile Project Management (APM)

- **Unavailability of customers:** Agile relies on continuous customer involvement; limited availability can delay feedback and affect progress.

- **Expertise:** Successful Agile implementation demands experienced and knowledgeable team members.
- **Resources:** Lack of required resources can hinder development and timely delivery.
- **Documentation:** Agile requires proper documentation, such as user stories, to capture project requirements accurately.
- **Poor resource planning:** Project cost, time, and scope can be difficult to predict due to ongoing feedback and changing requirements.

Agile Team Interactions



Agile team interactions prioritize intense collaboration, open communication, and self-organization over rigid processes, focusing on individuals to deliver working software in iterative cycles. Key interactions include daily stand-ups, sprint planning, and retrospectives, fostering a culture of shared responsibility and rapid adaptation to change.

There are **the three common roles** found on an agile team:

1. **Cross-functional team member** – in keeping with the non-hierarchical structure of the agile team, the cross functional team member is mentioned first, and NOT the product owner or team facilitator. They are professionals who deliver potentially releasable product on a regular basis. They need to deliver work in the shortest possible time, with higher quality, without external dependencies.
2. **The product owner** interacts with the customer and stakeholders as well as the team, and they pay attention to the highest value for the customer. They typically have a business background and have deep subject matter expertise. They create the backlog for and with the team, in a way that delivers the highest value without creating waste.
3. This role can be called various names, such as a **project manager**, scrum master, as well as a team lead, couch, or facilitator.
4. The servant leader, no matter what he or she is called, needs to focus on facilitation of the work done by the team, impediment removal, and coaching. If the servant leader

feels their internal coaching capability is not yet fully developed, then they may invite external agile coaches.

Key Characteristics of Agile Team Interactions

- **Individuals and Interactions > Processes and Tools:** Agile emphasizes direct communication, often favoring face-to-face conversations or video calls for remote members over extensive documentation.
- **Constant Collaboration:** Members work closely, sharing both successes and failures equally to maintain unity, which improves performance and goal achievement.
- **Self-Organizing and Cross-Functional:** Teams are empowered to determine how to complete their work, with members bringing diverse skills (design, dev, testing) to deliver value together.
- **Feedback Loops:** Regular interaction with stakeholders and customers, particularly during sprint reviews, ensures the product meets evolving needs.

Common Interaction Frameworks

- **Daily Stand-up:** A brief, daily meeting to discuss progress and blockers.
- **Sprint Planning:** Collaborative session to define work for the upcoming cycle.
- **Retrospectives:** Dedicated time for the team to reflect on their processes and improve, promoting a healthy, productive environment

Ethics in Agile Teams

In addition to the Agile Manifesto, two of the most popular agile approaches, Extreme Programming (XP) and Scrum, advocate specific values.

(Again, this is not to say that other software development approaches do not have values – just that XP and Scrum explicitly spell them out.)

Ken Schwaber identified the following five Scrum values in his book *Agile Software Development with Scrum*: commitment, focus, openness, respect, and courage.

Kent Beck identified these five XP values in his book *Extreme Programming Explained*: communication, simplicity, feedback, courage, and respect.

While many trainers focus on the practices to get new agile teams up and running, it is the values that will keep the team truly agile long after the trainers and coaches leave.

- **Commitment:**

It's not enough to be assigned to a team and agree to try this new agile thing, and it's not enough simply to be dedicated. One must also be committed to doing whatever is necessary to meet the goals outlined and to take the authority to do so hard. It means "to carry into action deliberately" (Merriam-Webster) or, as Yoda says, "Do, or do not. There is no try."

- **Focus:**

- Don't get sidetracked. Remember what you committed to do, and focus your energies on fulfilling that promise. Distractions are not limited to the obvious things like email and unrelated meetings. They may also be things like creating documentation because "we've always done it that way" instead of creating documentation because it truly provides the customer with something of value

Openness:

Openness is about keeping the project status highly visible to everyone all the time. Anyone who is interested should be able to look at a wall, a wiki page, or a dashboard in an agile project management tool and see how many features have been completed, what's currently being worked on, and the goals of the iteration and release. Communication: People on the team must talk to each other. This gets harder to do the more geographically separated the team is, but with tools like Skype, teams can talk to one another easily and cheaply. The point is that team members must not rely on email and documentation alone. Verbal communication is required to clarify ideas, solve complex problems, answer questions quickly, and help team members coordinate work efforts.

Kent Beck Extreme Programming Ethics:

- **Simplicity:**

Beck says the XP coach should ask the team, "What is the simplest thing that could possibly work?" Then they should do that thing. Because agile approaches develop code in increments each iteration, the thought is that it is better to develop something simple that may have to be expanded later if needed, rather than spend a large amount of time now developing a solution that's more complicated and may, in fact, not be necessary.

- **Feedback:**

As stated earlier, our number one obligation is to provide value to the customer. In order to do that, we must obtain frequent feedback from the customer or customer representative in order to make sure that the product we are building is meeting their expectations. And if it is not, we have the information we need now in order to make corrections. Beck extends this meaning of feedback to include feedback from the system itself in the form of unit, functional, and performance tests run frequently in each iteration.

- **Courage:**

In order to accept the authority and accountability for the delivery of the product, the team members all need courage. From the courage to make decisions to the courage to say no, this is a foundational value that gives rise to all the others.

- **Respect:**

Each team member must begin by agreeing that everyone deserves to be treated with respect. Many teams clarify this with working agreements that outline ways in which they choose to work together. And, as a result of working together to adhere to all the other values outlined here, team members grow to respect one another beyond their shared humanity to the ways each valued colleague contributes to the whole.

Core Ethical Principles in Agile

- **Transparency & Honesty:** Open communication about project status, risks, and skill limitations.
- **Respect & Safety:** Creating a safe, collaborative environment where team members can raise concerns without fear.
- **Customer & User Focus:** Prioritizing user safety, privacy, and the social impact of the product.
- **Responsibility & Quality:** Delivering high-quality, secure code rather than skipping steps for speed.

Core Ethical Issues in Agile

- **AI Ethics and Governance:** As AI shapes backlogs and product strategy, teams face risks regarding algorithmic bias, lack of transparency, and AI-driven decisions that sacrifice fairness for efficiency.
- **Data Privacy by Design:** Agile teams must ensure personal data used in development is anonymized, secured, and strictly necessary, complying with regulations like GDPR.
- **Speed vs. Quality and Safety:** The pressure to deliver quickly can result in cutting corners on testing, documentation, or safety checks, leading to unintended negative outcomes.
- **Team Wellbeing and Autonomy:** Mismanagement of agile autonomy can lead to high-pressure environments, unsustainable workloads ("crunch"), and burnout.
- **Conflicts of Interest in Coaching:** Unregulated Agile coaching can lead to situations where coaches act in their own interest rather than the team's, necessitating ethical codes like those from the [Agile Alliance](#).
- **Responsibility and Accountability:** Distributed decision-making can make it unclear who is responsible when an AI-driven or quickly released product causes harm

Ethical Guardrails

To address these, teams are encouraged to implement four key guardrails:

- **Data Privacy:** Information classification.
- **Human Value Preservation:** Defining AI vs. human roles.
- **Output Validation:** Verification protocols.
- **Transparent Attribution:** Tracking contributions.

Agility in Design, Testing

Agility in design is a flexible, iterative approach that prioritizes adaptability, user feedback, and rapid prototyping over rigid, long-term planning. It allows teams to respond quickly to

changing requirements, reducing waste and improving product quality through continuous, user-centered refinement.

Agile testing is a software testing process that follows the principles of agile software development. Agile testing aligns with iterative **Development Methodology** in which requirements develop gradually from customers and testing teams. The development is aligned with customer requirements.

Agile testing is a continuous process rather than being sequential. The testing begins at the start of the project and there is ongoing integration between testing and development. The common objective of agile development and testing is to achieve a high product quality.

Core Principles of Design Agility

- **Iterative Design:** Constantly refining products based on user testing and feedback loops.
- **Rapid Prototyping:** Creating low-cost, quick prototypes to validate ideas early.
- **User-Centered Focus:** Ensuring user needs dictate design changes.
- **Cross-Functional Collaboration:** Breaking down silos to ensure developers, designers, and stakeholders work together

Benefits of Agile Design

- **Improved Product Quality:** Higher alignment with user needs.
- **Faster Time-to-Market:** Speedier development through iterative, smaller cycles.
- **Increased Efficiency:** Reduced, unnecessary documentation and overhead.
- **Enhanced Innovation:** Encourages creative, flexible solutions to complex problems

Agility vs. Traditional Design

While traditional design often follows a linear, "waterfall" approach with set milestones, agile design embraces unpredictability and welcomes changes even late in the process. It thrives on creating "good enough" designs to test, rather than perfecting a design before development begins

Key Aspects of Agility in Testing

- **Continuous Testing:** Testing is performed continuously alongside development, ensuring immediate identification of bugs rather than waiting for a separate testing phase.
- **Collaboration:** Testers work closely with developers and business stakeholders throughout the sprint, fostering shared responsibility for quality.

- **Iterative Process:** Testing occurs within short development cycles (sprints), allowing for frequent adjustments based on changing requirements and user feedback.
- **Automation:** Agile relies heavily on automated testing to keep up with fast-paced development, covering unit tests, regression tests, and integration tests.
- **Focus on Value:** Efforts are aimed at reducing unnecessary documentation and activities, focusing only on tasks that improve software quality

Benefits of Agile Testing

- **Faster Delivery:** Reduces time-to-market by integrating testing with development.
- **High Quality:** Early bug detection leads to better, more robust software.
- **Adaptability:** Enables teams to respond to changes in requirements, even late in development.
- **Better Feedback Loop:** Immediate customer feedback helps ensure the product meets expectations

Core Principles

- **Continuous testing** to provide feedback.
- **Delivering value** to the customer.
- **Reducing feedback time.**
- **Responding to change.**
- **Self-organizing teams**

Agile Documentations

Agile documentation is a lightweight, iterative approach focusing on creating "just enough" documentation to support development, rather than comprehensive, upfront paperwork.

An agile document is just barely good enough, or just barely sufficient, for the situation at hand.

Documentation is an important part of agile software development projects, but unlike traditionalists who often see documentation as a risk reduction strategy, agilists typically see documentation as a strategy which increases overall project risk and therefore strive to be as efficient as possible when it comes to documentation.

Agilists write documentation when that's the best way to achieve the relevant goals, but there proves to be better ways to achieve those goals than writing static documentation.

Best practices for increasing the agility of documentation

1. Writing

Prefer executable specifications over static documents Document stable concepts, not speculative ideas Generate system documentation

2. Simplification

Keep documentation just simple enough, but not too simple, Write the fewest documents with least overlap, Put the information in the most appropriate place, Display information publicly

3. Determining What to Document

Document with a purpose, Focus on the needs of the actual customer(s) of the document, The customer determines sufficiency

4. Determining When to Document

Iterate, iterate, iterate, Find better ways to communicate, Start with models you actually keep current

Update only when it hurts

5. General

Treat documentation like a requirement, Require people to justify documentation requests, Recognize that you need some documentation, Get someone with writing experience

Key Principles of Agile Documentation

- **Just-in-Time:** Created when needed rather than at the start, ensuring relevance.
- **Lightweight & Digital:** Favors simple formats like wiki pages, markdown files, or Agile management tools (e.g., Jira) over massive documents.
- **Living Documents:** Updated frequently, evolving along with the product during each sprint.
- **Value-Driven:** Focused on capturing essential "good to know" information (e.g., design rationale) rather than obvious details

Agile Drivers (Why Agile?)

Agile is adopted to address specific environmental pressures and business goals:

- **Rapidly Changing Requirements:** Ability to handle shifting priorities and evolving technology.
- **Speed to Market:** Customer demands for faster delivery and shorter product lifecycles.
- **Risk Reduction:** Early and continuous delivery reduces the risk of project failure.
- **Flexibility & Adaptability:** Surviving changing circumstances by adapting, similar to agile species in a volatile environment.

- **Enhanced Competitiveness:** Mass customization and increased market share.

Agile Capabilities (What Agile Provides)

These are the operational abilities that enable agility:

- **Iterative & Incremental Development:** Breaking projects into small, manageable increments (sprints).
- **Continuous Integration & Testing:** Regular, automatic testing to ensure quality and functionality.
- **Collaboration & Transparency:** Close partnership with customers and open communication across cross-functional teams.
- **Flexibility in Planning:** Re-evaluating plans based on real-time feedback rather than sticking to a rigid, upfront plan.
- **Value-Driven Delivery:** Prioritizing work that delivers the most value to stakeholders

UNIT-II

Syllabus

UNIT- II AGILE PROCESSES

Lecture 8 Hrs

Lean Production - SCRUM, Crystal, Feature Driven Development- Adaptive Software Development - Extreme Programming: Method Overview – Lifecycle – Work Products, Roles and Practices.

AGILE PROCESSES

Lean Production - SCRUM, Crystal, Feature Driven Development- Adaptive Software Development - Extreme Programming: Method Overview – Lifecycle – Work Products, Roles and Practices.

Lean Software Development (LSD):

Lean Software Development (LSD) is an agile framework that is used to streamline & optimize the software development process. It may also be referred to as Minimum Viable Product (MVP) strategy as these ways of thinking are very much alike since both intend to speed up development by focusing on new deliverables.

Advantages of LSD:

LSD has proved to improve software development in the following ways:

1. LSD removes the unnecessary process stages when designing software so that it acts as a time saver as simplifies the development process.
2. With a focus on MVP, Lean Software Development prioritizes essential functions so this removes the risk of spending time on valueless builds.
3. It increases the involvement power of your team as more and more members participate due to which the overall workflow becomes optimized and losses get reduced.

Key Principles of Lean Software Development:

There are 7 established lean principles that come with a set of tactics, practices, and processes that builds more efficient software products:

1. Eliminating the waste
2. Fast Delivery
3. Amplify Learning
4. Builds Quality
5. Respect Teamwork
6. Delay the commitment
7. Optimizing the whole system

The below figure illustrates the principles of LSD:

Eliminating the Waste: To identify and eliminate wastes e.g. unnecessary code, delay in processes, inefficient communication, the issue with quality, data duplication, more tasks in the log than completed, etc. regular meetings are held by Project Managers. Which allows team members to point out faults and suggest changes in the next turn.

Fast Delivery: Previously long time planning used to be the key success in business, but in the passage of time it is found that engineers spend too much time on building complex systems with unwanted features. So they came up with an MVP strategy which resulted in the building products quickly that included a little functionality and launch the product to market and see the reaction. Such an approach allows them to enhance the product on the basis of customer feedback

Amplify Learning: Learning is improved through ample code reviewing, meeting that is crossteam applicable. It is also ensured that particular knowledge isn't accumulated by one engineer who's writing a particular piece of code so paired programming is used.

Builds Quality: LSD is all about prevent waste, keeping an eye on not sacrificing quality. Developers often apply test-driven programming to examine the code before it is written. The quality can also be gained to get constant feedback from team members and project managers.

Respect Teamwork: LSD focuses on empowering team members, rather than controlling them. Setting up a collaborative atmosphere, keep perfect balance when there are short deadlines and immense workload. This method becomes very much important when new members join a well-established team.

Delay the Commitment: In traditional project management it often happens when you make your application and it turns out to be completely unfit for the market. LSD method recognizes this threat and makes room for improvement by postponing irreversible decisions until all experiment is done. This methodology always constructs software as flexible, so the new knowledge is available and engineers can make improvements.

Optimizing the whole system: lean's principle allows managers to break an issue into small constituent parts to optimize the team's workflow, create unity among members, and inspire a sense of shared responsibility which results in enhancing the team performance.

Weakness in LSD:

- Make it scalable as other frameworks since it strongly depends on the team involved.
- It is hard to keep pace so it is not easy for developers to work with team members as conflict may occur between them.
- It leads to a difficult decision-making process as it is mandatory for customers to clearly set their requirements for the development not to be interrupted. Lean Software Development is one of the proactive approaches that drives your body through productivity and cleanliness. It closely connects to Agile methodology, knowledge-sharing experience, fast product delivery. All processes and stages of development are accurately built to deliver the end product at minimum cost in a timely manner.

SCRUM:

What is Scrum?

It is an iterative and lightweight framework for tracking and managing product development in a dynamic manner. It provides accountability, emphasizes teamwork and iterative progress with three important pillars, i.e. inspection, transparency, and adaption. Scrum is a framework to manage knowledge work that uses agile methodology. It is like a daily short meeting between team members having three to nine members, who divide their work and complete the task within given time iterations which are called sprints. The duration of a sprint is no longer than 1 month, and then the progress is tracked in stand up a meeting called as daily scrums

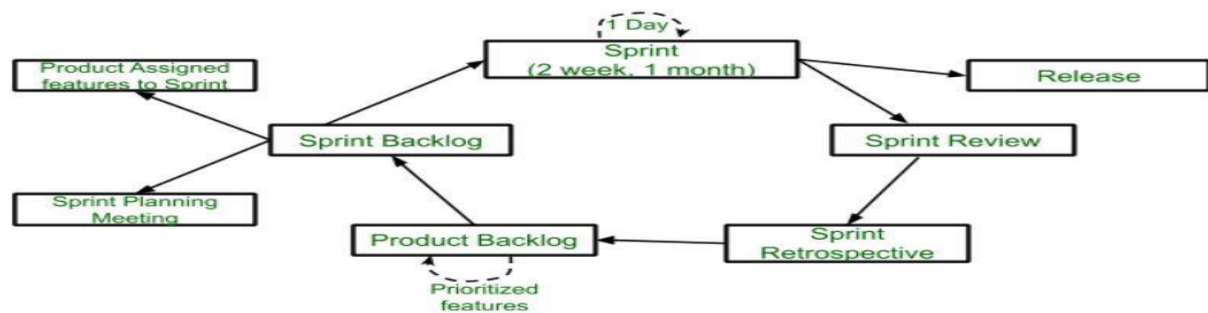


Scrum is the type of Agile framework. It is a framework within which people can address complex adaptive problem while productivity and creativity of delivering product is at highest possible values. Scrum uses Iterative process.

Silent features of Scrum are:

- Scrum is light-weighted framework
- Scrum emphasizes self-organization
- Scrum is simple to understand
- Scrum framework help the team to work together Lifecycle of Scrum:

Lifecycle of Scrum:

**Sprint:**

A Sprint is a time-box of one month or less. A new Sprint starts immediately after the completion of the previous Sprint.

Release:

When the product is completed then it goes to the Release stage.

Sprint Review:

If the product still have some non-achievable features then it will be checked in this stage and then the product is passed to the Sprint Retrospective stage.

Sprint Retrospective:

In this stage quality or status of the product is checked. Product Backlog: According to the prioritize features the product is organized.

Sprint Backlog:

Sprint Backlog is divided into two parts Product assigned features to sprint and Sprint planning meeting.

Advantage of using Scrum framework:

- Scrum framework is fast moving and money efficient.
- Scrum framework works by dividing the large product into small sub-products. It's like a divide and conquer strategy
- In Scrum customer satisfaction is very important.
- Scrum is adaptive in nature because it have short sprint.
- As Scrum framework rely on constant feedback therefore the quality of product increases in less amount of time

Disadvantage of using Scrum framework:

- Scrum framework do not allow changes into their sprint.

- Scrum framework is not fully described model. If you wanna adopt it you need to fill in the framework with your own details like Extreme Programming(XP), Kanban, DSDM.
- It can be difficult for the Scrum to plan, structure and organize a project that lacks a clear definition.
- The daily Scrum meetings and frequent reviews require substantial resource.

Crystal:

The crystal method is an agile framework that is considered a lightweight or agile methodology that focuses on individuals and their interactions. The methods are color-coded to significant risk to human life. It is mainly for short-term projects by a team of developers working out of a single workspace. Among few Agile Software Development Life Cycle (SDLC) models crystal is considered as one of the Agile SDLC models. Two core beliefs of the Crystal method:

- Find your own way and methods to optimize workflow.
- Make use of unique methods to make the project unique and dynamic.

Let's know about the history of the Crystal Method:

The crystal method was developed by an American scientist named Alistair Cockburn who worked at IBM. He decided not to focus on step-by-step developmental strategies, but to develop team collaboration and communication. Some of the traits of Cockburn's Crystal method were:

- Human-powered i.e. the project should be flexible and people involved in preferred work.
- Adaptive i.e. approaches don't have any fixed tools but can be changed anytime to meet the team's specific needs.
- Ultra-light i.e. this methodology doesn't require much documentation.

Properties of Crystal Agile Framework :

1.Frequent Delivery- It allows you regularly deliver the products and test code to real users. Without this, you might build a product that nobody needs.

2.Reflective Improvement- No matter how good you have done or how bad you have done. Since there are always areas where the product can be improved, so the teams can implement to improve their future practices.

3.Osmotic Communication- Alistair stated that having the teams in the same physical phase is very much important as it allows information to flow in between members of a team as in osmosis.

3.Personal Safety- There are no bad suggestions in a crystal team, team members should feel safe to discuss ideas openly without any fear.

4.Focus- Each member of the team knows exactly what to do, which enables them to focus their attention. This boosts team interaction and works towards the same goal.

5.Easy access to expert users- It enhances team communication with users and gets regular feedback from real users.

6. Technical tooling- It contains very specific technical tools which to be used by the software development team during testing, management, and configuration. These tools make it enable the team to identify any error within less time.

How does Crystal function?

Till now, we got to know that crystal is a family of various developmental approaches, and it is not a group of prescribed developmental tools and methods. In the beginning, the approach is set by considering the business requirements and the needs of the project. Various methodologies in the Crystal family also known as weights of the Crystal approach are represented by different colors of the spectrum.

Crystal family consists of many variants like Crystal Clear, Crystal Yellow, Crystal Red, Crystal Sapphire, Crystal Red, Crystal Orange Web, and Crystal Diamond.

1. Crystal Clear- The team consists of only 1-6 members that is suitable for short-term projects where members work out in a single workspace.

2. Crystal Yellow- It has a small team size of 7-20 members, where feedback is taken from Real Users. This variant involves automated testing which resolves bugs faster and reduces the use of too much documentation.

3. Crystal Orange- It has a team size of 21-40 members, where the team is split according to their functional skills. Here the project generally lasts for 1-2 years and the release is required every 3 to 4 months.

4. Crystal Orange Web- It has also a team size of 21-40 members were the projects that have a continually evolving code base that is being used by the public. It is also similar to Crystal Orange but here they do not deal with a single project but a series of initiatives that required programming.

5. Crystal Red- The software development is led by 40-80 members where the teams can be formed and divided according to requirements.

6. Crystal Maroon- It involves large-sized projects where the team size is 80-200 members and where methods are different and as per the requirement of the software.

7. Crystal Diamond & Sapphire- This variant is used in large projects where there is a potential risk to human life.

The below figure illustrates about crystal team



CRYSTAL FAMILY (TEAM MEMBERS)

Benefits of using the Crystal Agile Framework

- Facilitate and enhance team communication and accountability.
- The adaptive approach lets the team respond well to the demanding requirements.
- Allows team to work with the one they see as the most effective.
- Teams talk directly with each other, which reduces management overhead.

Drawbacks of using the Crystal Agile Framework :

- A lack of pre-defined plans may lead to confusion and loss of focus.
- Lack of structure may slow down inexperienced teams.
- Not clear on how a remote team can share knowledge informally.

The Crystal Method is expandable. It may be used by small teams or large teams to work on simple or complex objects. It places importance on developmental skills and interactions which in turn encourage the exchange of ideas. It is also beneficial for the clients as it delivers the most important components of the product first. But on the other hand, the Crystal Method does not plan based on the requirements of the projects.

Feature Driven Development

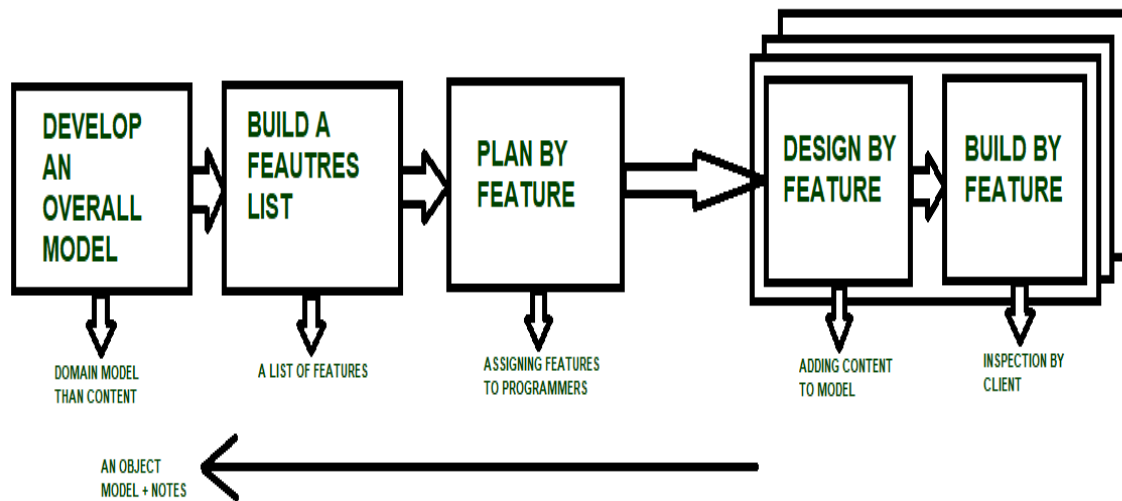
FDD stands for **Feature-Driven Development**. It is an agile iterative and incremental model that focuses on progressing the features of the developing software. The main motive of feature-driven development is to provide timely updated and working software to the client. In FDD, reporting and progress tracking is necessary at all levels.

History

FDD was first applied in the year 1997 on a real-world application by *Jeff De Luca* for large software development with specific needs of 15-month and 50 persons and published as a discussion in book *Java Modeling in Color with UML* in the year 1999.

FDD Lifecycle

- Build overall model
- Build feature list
- Plan by feature
- Design by feature
- Build by feature



Characteristics of FDD

- **Short iterative:** FDD lifecycle works in simple and short iterations to efficiently finish the work on time and gives good pace for large projects.
- **Customer focused:** This agile practice is totally based on inspection of each feature by client and then pushed to main build code.
- **Structured and feature focused:** Initial activities in lifecycle builds the domain model and features list in the beginning of timeline and more than 70% of efforts are given to last 2 activities.
- **Frequent releases:** Feature-driven development provides continuous releases of features in the software and retaining continuous success of the project.

Advantages of FDD

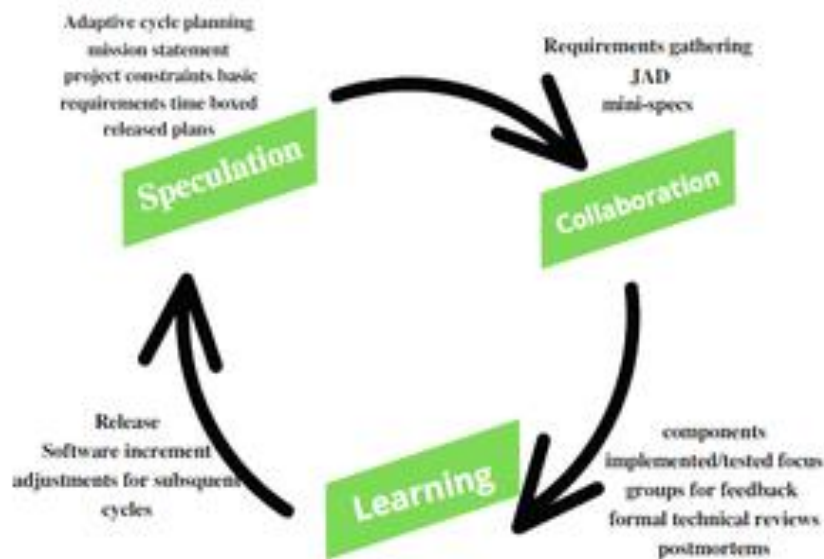
- Reporting at all levels leads to easier progress tracking.
- FDD provides continuous success for larger size of teams and projects.
- Reduction in risks is observed as whole model and design is build in smaller segments.
- FDD provides greater accuracy in cost estimation of the project due to feature segmentation.

Adaptive software development:

Adaptive Software Development is a method to build complex software and system. ASD focuses on human collaboration and self-organization. ASD “life cycle” incorporates three phases namely:

1. Speculation
2. Collaboration
3. Learning

These are explained as following below.



Adaptive Software Development

1.Speculation:

During this phase project is initiated and planning is conducted. The project plan uses project initiation information like project requirements, user needs, customer mission statement, etc, to define set of release cycles that the project wants.

2.Collaboration:

It is the difficult part of ASD as it needs the workers to be motivated. It collaborates communication and teamwork but emphasizes individualism as individual creativity plays a major role in creative thinking. People working together must trust each others to

- Criticize without animosity,
- Assist without resentment,
- Work as hard as possible,
- Possession of skill set,
- Communicate problems to find effective solution.

3.Learning:

The workers may have a overestimate of their own understanding of the technology which may not lead to the desired result. Learning helps the workers to increase their level of understanding over the project. Learning process is of 3 ways:

1. Focus groups
2. Technical reviews
3. Project postmortem

ASD's overall emphasis on the dynamics of self-organizing teams, interpersonal collaboration, and individual and team learning yield software project teams that have a much higher likelihood of success.

Extreme Programming (XP):

Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behaviour. The team is expected to self-organize. Extreme Programming provides specific core practices where –

- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behaviour.

Embrace Change

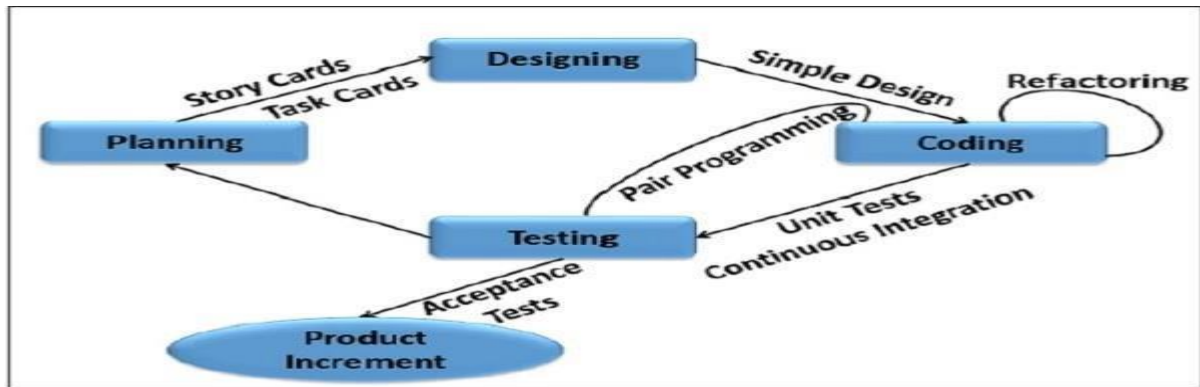
A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time. This can be achieved with –

- Emphasis on continuous feedback from the customer
- Short iterations
- Design and redesign
- Coding and testing frequently
- Eliminating defects early, thus reducing costs
- Keeping the customer involved throughout the development
- Delivering working product to the customer

Extreme Programming in a Nutshell

Extreme Programming involves –

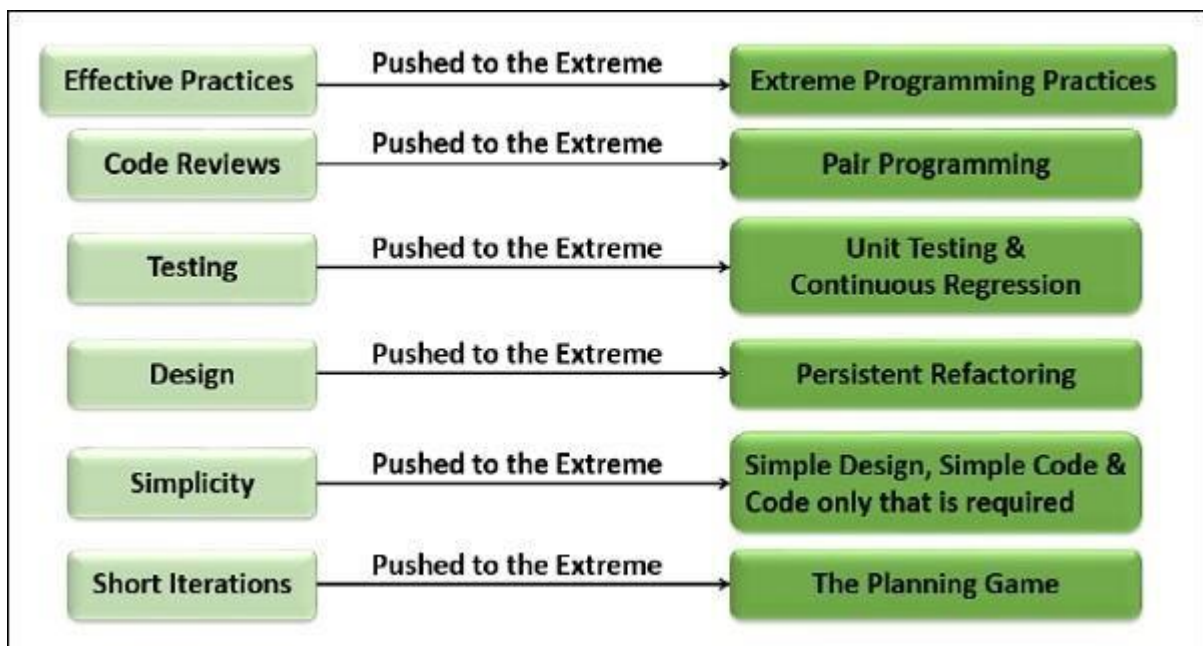
- Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminates defects early, thus reducing the costs.
- Starting with a simple design just enough to code the features at hand and redesigning when required.
- Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.
- Integrating and testing the whole system several times a day.
- Putting a minimal working system into the production quickly and upgrading it whenever required.
- Keeping the customer involved all the time and obtaining constant feedback. Iterating facilitates the accommodating changes as the software evolves with the changing requirements.



Why is it called “Extreme?”

Extreme Programming takes the effective principles and practices to extreme levels.

- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.



Extreme Programming Advantages

Extreme Programming solves the following problems often faced in the software development projects –

- **Slipped schedules** – and achievable development cycles ensure timely deliveries.
- **Cancelled projects** – Focus on continuous customer involvement ensures transparency with the customer and immediate resolution of any issues.

- **Costs incurred in changes** – Extensive and ongoing testing makes sure the changes do not break the existing functionality. A running working system always ensures sufficient time for accommodating changes such that the current operations are not affected
- **Production and post-delivery defects: Emphasis is on** – the unit tests to detect and fix the defects early.
- **Misunderstanding the business and/or domain** – Making the customer a part of the team ensures constant communication and clarifications.
- **Business changes** – Changes are considered to be inevitable and are accommodated at any point of time.
- **Staff turnover** – Intensive team collaboration ensures enthusiasm and good will. Cohesion of multi-disciplines fosters the team spirit.

Extreme Programming Values

Extreme Programming (XP) is based on the five values –

- Communication
- Simplicity
- Feedback
- Courage
- Respect

Communication

Communication plays a major role in the success of a project. Problems with projects often arise due to lack of communication. Many circumstances may lead to the breakdown in communication. Some of the common problems are –

- A developer may not tell someone else about a critical change in the design.
- A developer may not ask the customer the right questions, and so a critical domain decision is blown.
- A manager may not ask a developer the right question, and project progress is misreported.
- A developer may ignore something important conveyed by the customer.

Extreme Programming emphasizes continuous and constant communication among the team members, managers and the customer. The Extreme Programming practices, such as unit testing, pair programming, simple designs, common metaphors, collective ownership and customer feedback focus on the value of communication.

Simplicity

Extreme Programming believes in 'it is better to do a simple thing today and pay a little more tomorrow to change it' than 'to do a more complicated thing today that may never be used anyway'. • Do what is needed and asked for, but no more.

- "Do the simplest thing that could possibly work" The DTSTTCPW principle.
- Implement a new capability in the simplest possible way. Also known as the KISS principle 'Keep It Simple, Stupid!'.
- A coach may say DTSTTCPW when he sees an Extreme Programming developer doing something needlessly complicated.
- Refactor the system to be the simplest possible code with the current feature set. This will maximize the value created for the investment made till date.
- Take small simple steps to your goal and mitigate failures as they happen.
- Create something that you are proud of and maintain it for a long term for reasonable costs.
- Never implement a feature you do not need now i.e. the 'You Aren't Going to Need It' (YAGNI) principle.

Feedback

In Extreme Programming, feedback is ensured at all levels at different time scales –

- Customers tell the developers what features they are interested in so that the developers can focus only on those features.
- Unit tests tell the developers the status of the system.
- The system and the code provides feedback on the state of development to the managers, stakeholders and the customers.
- Frequent releases enable the customer to perform acceptance tests and provide feedback and developers to work based on that feedback.
- When the customers write new features/user stories, the developers estimate the time required to deliver the changes, to set the expectations with the customer and managers.

Courage

Extreme Programming provides courage to the developers in the following way –

- To focus on only what is required
- To communicate and accept feedback
- To tell the truth about progress and estimates
- To refactor the code

- To adapt to changes whenever they happen
- To throw the code away (prototypes) This is possible as no one is working alone and the coach guides the team continuously.

Respect

Respect is a deep value, one that lies below the surface of the other four values. In Extreme Programming,

- Everyone respects each other as a valued team member.
- Everyone contributes value such as enthusiasm.
- Developers respect the expertise of the customers and vice versa.
- Management respects the right of the developers to accept the responsibility and receive authority over their own work.

Combined with communication, simplicity, and concrete feedback, courage becomes extremely valuable.

- Communication supports courage because it opens the possibility for more high-risk, highreward experiments.
- Simplicity supports courage because you can afford to be much more courageous with a simple system. You are much less likely to break it unknowingly.
- Courage supports simplicity because as soon as you see the possibility of simplifying the system you try it.
- Concrete feedback supports courage because you feel much safer trying radical modifications to the code, if you can see the tests turn green at the end. If any of the tests do not turn green, you know that you can throw the code away.

Extreme Programming Principles

The values are important, but they are vague, in the sense that it may not be possible to decide if something is valuable. For example, something that is simple from someone's point of view may be complex from someone else's point of view.

Hence, in Extreme Programming, the basic principles are derived from the values so that the development practices can be checked against these principles. Each principle embodies the values and is more concrete, i.e. rapid feedback – you either, have it or you do not.

The fundamental principles of Extreme Programming are –

- Rapid feedback
- Assume simplicity
- Incremental change
- Embracing change

- Quality work

Rapid Feedback

Rapid feedback is to get the feedback, understand it, and put the learning back into the system as quickly as possible.

- The developers design, implement and test the system, and use that feedback in seconds or minutes instead of days, weeks, or months.
- The customers review the system to check how best it can contribute, and give feedback in days or weeks instead of months or years.

Assume Simplicity

To assume simplicity is to treat every problem as if it can be solved with simplicity.

Traditionally, you are told to plan for the future, to design for reuse. The result of this approach may turn into 'what is required today by the customer is not met and what is ultimately delivered may be obsolete and difficult to change.

'Assume Simplicity' means 'do a good job of solving today's job today and trust your ability to add complexity in the future where you need it.' In Extreme Programming, you are told to do a good job (tests, refactoring, and communication) focusing on what is important today.

- With good unit tests, you can easily refactor your code to do additional tests.
- Follow YAGNI (You Ain't Gonna Need It).
- Follow the DRY(Don't Repeat Yourself) principle.

For example,

- Do not have multiple copies of identical (or very similar) code.
- Do not have redundant copies of information.
- No wastage of time and resources on what may not be necessary.

Incremental Change

In any situation, big changes made all at once just do not work. Any problem is solved with a series of the smallest change that makes a difference. In Extreme Programming, Incremental Change is applied in many ways.

- The design changes a little at a time.
- The plan changes a little at a time.
- The team changes a little at a time.

Even the adoption of Extreme Programming must be taken in little steps.

Embracing Change

The best strategy is the one that preserves the most options while actually solving your most pressing problem.

Quality Work

Everyone likes doing a good job. They try to produce the quality that they are proud of. The team

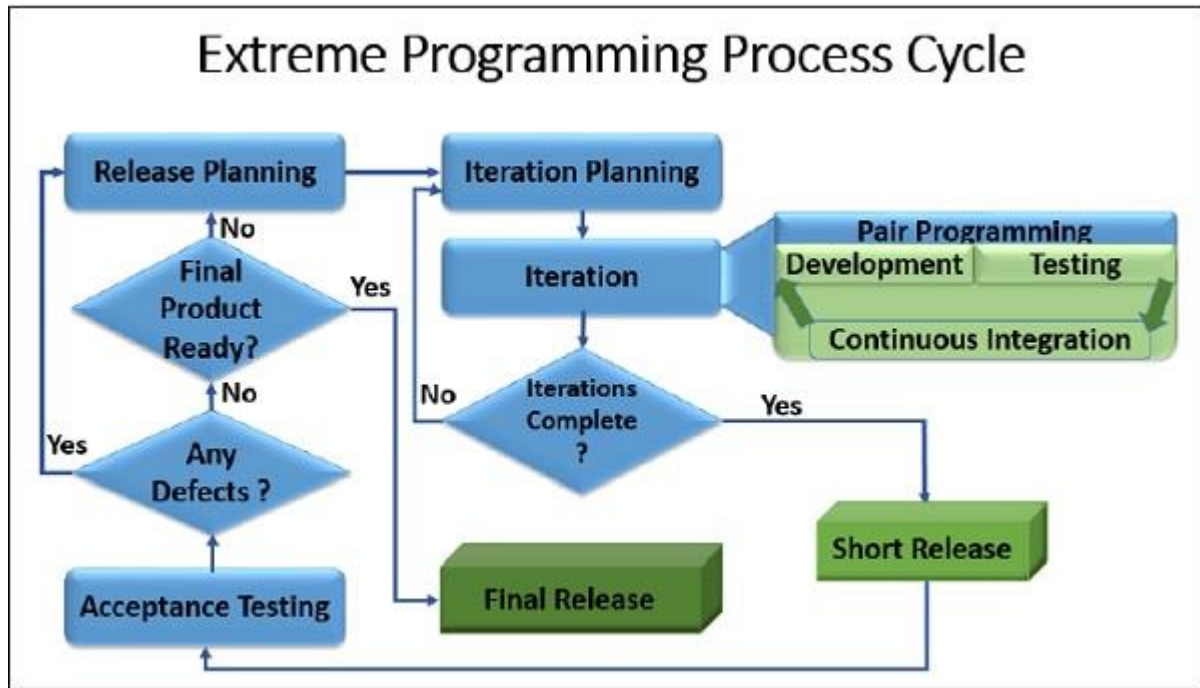
- Works well
- Enjoys the work
- Feels good in producing a product of value.

Lifecycle

Each software development methodology has its own life cycle. In some methods it is more complex; others contain just a few stages. However, it is impossible to understand a software development methodology without learning its life cycle. In this article we will tell you about the life cycle of Extreme Programming, but before doing that it is necessary to give the proper definition for XP.

Extreme Programming life cycle is the time required to perform a typical XP project. It begins at the stage of planning and ends when the final product is delivered to the customer. As we have already mentioned, XP is an Agile methodology. It means that it is based on the principle of iterative cycles. Hence, Extreme Programming life cycle model is also iterative. It repeats several times during a project until the final product is done.

Most specialists define five stages of XP software development life cycle. Let's look at them in greater detail.



Planning

This is the first stage of the Extreme Programming development life cycle. Its main task is to set goals of the entire project and certain iterative cycles. At this stage the team meets with the customer and asks him on all aspects of the future software. The customer formulates his vision of the product in user stories. The developers estimate and prioritize them in the release plan. After that the work on turning them into tasks begins.

Designing

At this stage of the project the team must define the main features of the future code. The main thing is to create a simple design, because simplicity is one of the main principles of XP methodology. Extreme Programming developers often share responsibilities at the stage of designing. Each developer is responsible for the design of a certain part of the code.

Coding

Extreme Programming developers believe that a good code must be simple. That is why they refactor it constantly. The procedure of refactoring allows them to simplify the code or its parts without affecting the functionality of the final product.

Testing

In Extreme Programming the procedure of testing is usually conducted not after the final or intermediate product is done, but along with the procedure of code writing.

Listening

At the final stage of the life cycle the XP team must get a feedback from the customer. He is the only person who estimates the final and intermediate products.

Work Products, Roles and Practices

Roles in Extreme Programming

The Roles that have been found effective in Extreme Programming are –

- Developer (also called Programmer by some teams)
- Customer
- Manager (also called tracker)
- Coach

Developer

The role of developer is the most important one in Extreme Programming. To be a developer in Extreme Programming, you need to be accustomed with the following –

Developer Rights

- You have the right to know what is needed, with clear declarations of priority.
- You have the right to produce quality work at all times.
- You have the right to ask for and receive help from peers, superiors, and customers.
- You have the right to make and update your own estimates.
- You have the right to accept your responsibilities instead of having them assigned to you.
 - Major responsibilities that you will be accountable for –
 - Estimate stories
 - Define tasks from stories
 - Estimate tasks
 - Write unit tests
 - Write code to pass the written unit tests
 - Perform unit testing
 - Refactor
 - Integrate continuously

Developer Skills

- Pair Programming
- Communication – that is necessary and to the sufficient detail
- Always use the metaphor to use the right names
- Code only what is required.
- Maintain simplicity

Collective Ownership

- If someone changes the code that you wrote, in whatever part of the system, you have to trust the changes and learn. In case the changes are wrong-headed, you are responsible for making things better, but be careful not to blame anyone.
- Be prepared to acknowledge your fears. Remember that you are part of a team and courage is required for the success of Extreme Programming.
 - Wear different hats as a developer in the team such as –

- Programmer.
- Architect and designer.
- Interface architect/UI designer.
- Database designer and DBA.
- Operations and network designer.
- Sometimes one of the developers has to wear the hat of a tester.
- Help the customer choose and write functional tests.
- Run the functional tests regularly.
- Report the test results.
- Ensure the testing tools run well.

Customer

In Extreme Programming, the customer role is as crucial as the developer role as it is the customer who should know what to program, while the developer should know how to program.

This triggers the necessity of certain skills for the customer role –

- Writing required stories to the necessary and sufficient detail.
- Developing an attitude for the success of the project.
- Influencing a project without being able to control it.
- Making decisions that are required time to time on the extent of functionality that is required.
- Willing to change decisions as the product evolves.
- Writing functional tests.

In Extreme Programming, the customer is required to be in constant communication with the team and speak as a single voice to the team. This is required since –

- Customer could be multiple stakeholders.
- Customer could be a community.
- Customer is not always the PRINCIPAL (proxies).
- Customer can be a team with the following potential members –
 - Product Managers
 - Marketing, Sales
 - Business Analysts
 - End Users, their Managers
 - Business/System Operations

The Major Responsibilities of the Customer are –

- Write user stories
- Write functional tests
- Set priorities on the stories
- Explain stories
- Decide questions about the stories

The customer is accountable for these responsibilities.

Manager

In Extreme Programming, the major responsibilities of the manager are –

- Define the rules of planning game.
- Familiarize the team and the customer on the rules of the planning game.
- Monitor the planning game, fix any deviations, modify the rules if and when required.
- Schedule and conduct release planning and iteration planning meetings.
- Participate with the team while they estimate in order to provide feedback on how reality conformed to their prior estimates, both at team level and individual level that eventually help them to come up with better estimates next time.
- Ensure that the team is working towards the next release as they progress through the iterations with committed schedule and committed functionality.
- Track functional testing defects.
- Track the actual time spent by each team member.
- Adapt an ability to get all the required information without disturbing the team's work. The Manager is Accountable for these Responsibilities.

Coach

Extreme Programming is the responsibility of everyone in the team. However, if the team is new to Extreme Programming, the role of a coach is crucial.

The responsibilities of the coach are –

- Understand, in depth, the application of Extreme Programming to the project.
- Identify the Extreme Programming practices that help in case of any problem.
- Remain calm even when everyone else is panicking.
- Observe the team silently and intervene only when a significant problem is foreseen and make the team also see the problem.
- See to that the team is self-reliant.
- Be ready to help.

Practices

There are four basic activities in Extreme Programming. They are –

- Coding
- Testing
- Listening
- Designing

These four basic activities need to be structured in the light of the Extreme Programming principles. To accomplish this, the Extreme Programming practices are defined.

These 12 Extreme Programming practices achieve the Extreme Programming objective and wherever one of the practices is weak, the strengths of the other practices will make up for it.

Kent Beck, the author of 'Extreme Programming Explained' defined 12 Extreme Programming practices as follows –

- The Planning Game

- Short Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40 hour Week
- On-site Customer
- Coding Standards

Four Areas of Extreme Programming

The Extreme Programming practices can be grouped into four areas –

- Rapid, Fine Feedback –
 - Testing
 - On-site customer
 - Pair programming
- Continuous Process –
 - Continuous Integration
 - Refactoring
 - Short Releases
- Shared Understanding –
 - The Planning Game
 - Simple Design
 - Metaphor
 - Collective Ownership
 - Coding Standards
- Developer Welfare –
 - Forty-Hour Week

In this chapter, you will understand the Extreme Programming practices in detail and the advantages of each of these practices.

Extreme Programming Practices at-a-glance

The following diagram shows how Extreme Programming is woven around the Extreme Programming practices –

- **Process** – How will the work and the team be organized? The team needs to fit the culture in which it will operate. The software must be written well rather than preserve the irrationality of an enclosing culture.
- **Detailed Scheduling** – Within a release, which stories should be done first? The developers need the freedom to schedule the riskiest segments of development first, to reduce the overall risk of the project. Within that constraint, they still tend to move business priorities earlier in the development, reducing the chance that important stories will have to be dropped towards the end of the development of a release due to time constraints.

Thus, plan is a result of collaboration between the customer, businessperson and the developers.

The Planning Game – Advantages

The Planning Game has the following advantages –

- Reduction in time, wasted on useless features
- Greater customer appreciation of the cost of a feature
- Less guesswork in planning

Short Releases

You should put a simple system into production quickly, and then release new versions in very short cycles. Every release should be as small as possible, so that it is –

- Achievable in a short cycle
- Contains the most valuable and immediate business requirements
- A working system

The duration of the short cycle may vary with the software that needs to be built. However, it needs to be ensured that the minimum possible duration is chosen.

Short Releases – Advantages

The advantages of Short Releases are –

- Frequent feedback
- Tracking
- Reduce chance of overall project slippage

Metaphor

According to Cambridge online dictionary- A Metaphor is an expression, often found in literature that describes a person or object by referring to something that is considered to have similar characteristics to that person or object. For example, ‘The mind is an ocean’ and ‘the city is a jungle’ are both Metaphors.

You should guide the entire development with a simple shared story of how the whole system works. You can think of metaphor as the architecture of the system to be built in a way that it is easily understandable by everyone involved in the development.

The metaphor consists of domain specific elements and shows their interconnectivity. The language used is the domain language. To identify the technical entities, the words used in the metaphor need to be taken consistently.

Metaphor – Advantages

The advantages of Metaphor are –

- Encourages a common set of terms for the system
- Reduction of buzz words and jargon
- A quick and easy way to explain the system

Simple Design

The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.

The right design for the software at any given time is the one that –

- Runs all the tests
- Has no duplicated logic like parallel class hierarchies
- States every intention important to the developers
- Has the fewest possible classes and methods

Simple Design – Advantages

The advantages of Simple Design are –

- Time is not wasted adding superfluous functionality
- Easier to understand what is going on
- Refactoring and collective ownership is made possible
- Helps keep the programmers on track

Testing

The developers continually write unit tests, which need to pass for the development to continue. The customers write tests to verify that the features are implemented. The tests are automated so that they become a part of the system and can be continuously run to ensure the working of the system. The result is a system that is capable of accepting change.

Testing – Advantages

The advantages of testing are –

- Unit testing promotes testing completeness
- Test-first gives developers a goal
- Automation gives a suite of regression tests

Refactoring

When implementing a feature, the developers always ask if there is a way of changing the existing code to make adding the feature simple. After they have added a feature, the developers ask if they now can see how to make the code simpler, while still running all of the tests. They restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility. This is called Refactoring.

Refactoring – Advantages

The advantages of Refactoring are –

- Prompts the developers to proactively improve the product as a whole
- Increases the developer knowledge of the system

Pair Programming

In Pair programming, the entire code is written with two developers at one machine, with one keyboard and one mouse.

There are two roles in each pair –

- The first developer (the one with the keyboard and the mouse) thinks about the best way to implement this method right here.
- The other developer is thinks more strategically
 - Is this whole approach going to work?
 - What are some other test cases that might not work yet?
 - Is there some way to simplify the whole system so the current problem just disappears?

The pairing is dynamic. It means that the two Roles A and B may exchange their places, or they may pair up with other team members. More often, anyone on the team will do as a partner. For example, if you have a responsibility for a task in an area that is unfamiliar to you, you might ask someone with recent experience to pair with you.

Pair Programming – Advantages

The advantages of Pair Programming are –

- Two heads are better than one
- Focus
- Two people are more likely to answer the following questions –
 - Is this whole approach going to work?
 - What are some test cases that may not work yet?
 - Is there a way to simplify this?

Collective Ownership

In Extreme Programming, the entire team takes responsibility for the whole of the system. Not everyone knows every part equally well, although everyone knows something about every part.

If a pair is working and they see an opportunity to improve the code, they go ahead and improve it.

Collective Ownership – Advantages

The advantages of Collective ownership are –

- Helps mitigate the loss of a team member who is leaving.
- Promotes the developers to take responsibility for the system as a whole rather than parts of the system.

Continuous Integration

Code is integrated and tested many times a day, one set of changes at a time. A simple way to do this is to have a machine dedicated to integration. A pair with code ready to integrate –

- Sits when the machine is free.
- Loads the current release.
- Loads their changes (checking for and resolving any collisions).
- Runs the tests until they pass (100% correct).

Integrating one set of changes at a time helps in knowing who should fix a test that fails. The answer- is the present pair, since the last pair left the tests at 100%. They may have to throw away what they have done and start all over, as they might not have known enough to code that feature.

Continuous Integration – Advantages

The advantages of Continuous Integration are –

- Reduces the duration, which is otherwise lengthy.
- Enables the short releases practice as the time required before release is minimal.

40-Hour Week

Extreme Programming emphasizes on the limited number of hours of work per week for every team members, based on their sustainability, to a maximum of 45 hours a week. If someone works for more time than that, it is considered as overtime. Overtime is allowed for at most one week. This practice is to ensure that every team member be fresh, creative, careful and confident.

40-Hour Week – Advantages

The advantages of 40-hour week are –

- Most developers lose effectiveness past 40 hours.
- Value is placed on the developers' well-being.
- Management is forced to find real solutions.

On-Site Customer

Include a real, live user on the team, available full-time to answer the questions, resolve disputes and set small-scale priorities. This user may not have to spend 40 hours on this role only and can focus on other work too.

On-Site Customer – Advantages

The advantages of having an onsite customer are –

- Can give quick and knowledgeable answers to the real development questions.
- Makes sure that what is developed is what is needed.
- Functionality is prioritized correctly.

Coding Standards

Developers write all code in accordance with the rules emphasizing-

- Communication through the code.
- The least amount of work possible.
- Consistent with the “once and only once” rule (no duplicate code).
- Voluntary adoption by the whole team.

These rules are necessary in Extreme Programming because all the developers –

- Can change from one part of the system to another part of the system.
- Swap partners a couple of times a day.
- Refactor each other's code constantly.

If the rules are not followed, the developers will tend to have different sets of coding practices, the code becomes inconsistent over time and it becomes impossible to say who on the team wrote what code.

Coding Standards – Advantages

The advantages of Coding Standards are –

- Reduces the amount of time developers spend reformatting other peoples' code.
- Reduces the need for internal commenting.
- Calls for clear, unambiguous code.

UNIT-III

Syllabus

UNIT- III AGILITY AND KNOWLEDGE MANAGEMENT

Lecture 8 Hrs

Agile Information Systems – Agile Decision Making - Early Schools of KM – Institutional Knowledge Evolution Cycle – Development, Acquisition, Refinement, Distribution, Deployment, Leveraging – KM in Software Engineering – Managing Software Knowledge – Challenges of Migrating to Agile Methodologies – Agile Knowledge Sharing – Role of Story-Cards – Story-Card Maturity Model (SMM).

3

Agility and Knowledge Management

Syllabus

Agile Information Systems, Agile Decision Making, Earl_S Schools of KM, Institutional Knowledge Evolution Cycle, Development, Acquisition, Refinement, Distribution, Deployment, Leveraging, KM in Software Engineering, Managing Software Knowledge, Challenges of Migrating to Agile Methodologies, Agile Knowledge Sharing, Role of Story-Cards, Story-Card Maturity Model (SMM).

Contents

- 3.1 Agile Information Systems
- 3.2 Agile Decision Making
- 3.3 Earl_S Schools of KM
- 3.4 Institutional Knowledge Evolution Cycle
- 3.5 KM in Software Engineering
- 3.6 Managing Software Knowledge
- 3.7 Challenges of Migrating to Agile Methodologies
- 3.8 Agile Knowledge Sharing
- 3.9 Role of Story-Cards
- 3.10 Story-Card Maturity Model (SMM)
- 3.11 Two Marks Questions with Answers

3.1 Agile Information Systems

Agility in general is defined as a formative construct comprised of the ability to sense a change, diagnose a change, select a response, and execute the response in realtime.

- 1) **Sense** : Ability to sense the stimuli for change (as they occur) in real-time;
- 2) **Diagnose** : Ability to interpret or analyze stimuli in real-time to determine the nature, cause, and impact of change;
- 3) **Respond** : Ability to respond to a change in real-time, further disaggregated into select and execute.
 - a) **Select** : Ability to select a response in real-time (very short planning time) needed to capitalize on the opportunity or counter the threat.
 - b) **Execute** : Ability to execute the response in real-time.

Real-time is defined as the span of time in which the correctness of the task performed not only depends upon the logical correctness of the task performed but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred.

Thus an Agile IS may be defined as one that has the ability to sense a change in real-time, diagnose the change in real-time, select a response in real-time, and execute the response in real-time.

Benefits of an Agile Information System

- 1) Reduction in time to implement changes in IS.
- 2) Increase in the robustness of the implemented changes.
- 3) Benefits from response to internal changes.
 - a) Increase in efficiency and effectiveness of the existing business processes.
 - b) Increase in efficiency and effectiveness of information processing.
 - c) Build spare capacity.
 - d) Build enhanced capabilities for future use.
 - e) Maintain the desired levels of service as per the service level agreements for the IS.
 - f) Enable fast recovery in case of outages to minimize the down time.
- 4) Benefits from response to organizational changes.
 - a) IS is aligned to business process requirements at all times.
 - b) Business processes can change rapidly (in real time).
 - c) Higher financial benefits to the organization.

5) Benefits from response to environmental changes.

- a) Avoid technology obsolescence through provision of needed latest IT/IS solutions.
- b) Allow business process innovation through incorporating the latest IS solutions.
- c) Protect the IS and the organization against unfavorable licensing terms and vendor induced situations.
- d) Allow rapid response to security threats like virus attacks, cyber-attacks, etc.

3.2 Agile Decision Making

Agile Decision Making is designed to help teams make better, faster and fully aligned decisions by working more effectively at pace. In the right context, it enables efficient, high quality, more balanced input from the whole team. It is particularly suited where there are multiple issues to be addressed or decisions to be made in constrained timeframes. It creates clear accountabilities and full alignment. It is based on the idea that, with the current rate of change and degree of uncertainty, complexity and ambiguity, ever fewer decisions can reliably anticipate the future.

Roman Voting

Roman voting is a means of saying yes, maybe, or no, using a simple hand gesture.

To use this technique, the facilitator polls the group on a particular topic and asks them to indicate where they stand, such that :

- **Thumbs Up** : I support this idea.
- **Thumbs Sideways** : I am ambivalent; I can "live with" this idea. While it may not meet all of my needs, I don't have strong reservations.
- **Thumbs Down** : I cannot live with this idea and have concerns that must be heard by the group before we move forward.

If there are no thumbs down, this is a reasonably strong indicator that the team is comfortable with moving forward.

Fist-to-Five

Fist-to-five, also known as "Fist of Five," is another approach that utilizes hand gestures to assess the extent to which there is team consensus on a topic.

To use this technique the facilitator restates a decision the group may choose to make and asks everyone to show their level of support, were they to move forward with that decision. Each person responds by showing a fist or a number of fingers that corresponds to their opinion.

As a facilitator, you can frame what each vote means any way you like. Here is one common set of definitions :

- **Fist**. I am completely opposed to the proposal.
- **1 Finger**. I have major reservations about the proposal.



- **2 Fingers.** I have some concerns about the proposal that I would like to discuss with the group.
- **3 Fingers.** I'm not in total agreement with the proposal but feel comfortable enough to move forward without further discussion.
- **4 Fingers.** I think it's a good idea/decision and agree with it.
- **5 Fingers.** It's a great idea and I fully support it.

It's up to the facilitator to work with the team to decide what the minimum number of fingers is for the team to move on without further discussion. For instance, if anyone holds up fewer than three fingers, then the working agreement for this exercise might be that they should be given the opportunity to state their objections and the team should address their concerns. Teams continue the Fist-to-Five process until they achieve consensus (e.g., a minimum of three fingers) or determine they must move on to another topic.

Dot Voting

Unlike the preceding two techniques, dot voting does not make use of hand gestures. Dot voting serves as a visual means of determining what is most important among a reasonably small set of ideas/topics to a group or team.

A common approach to facilitate dot voting is as follows :

- Write all of the ideas/topics down on a white board (whether virtual or physical) or on note cards or Post-it notes.
- Make sure there is shared understanding of what each item means.
- Inform the group they have x number of votes (three is a common number of votes), where They can place all of their votes on the same item, or they can distribute them among two or more items.
- For collocated groups, give them a writing utensil and have them mark their vote, where each vote is a dot.
- For virtual groups, ask them for their votes and make the appropriate marks using an "x" or some similar notation.
- Tally up the votes.
- If you need to break a tie, you can do another round of dot voting, or use another technique.

3.3 Earl's Schools of KM

Michael Earl mapped knowledge management into a seven schools taxonomy. The schools are categorized into three categories :

- Technocratic
- Economic
- Behavioural

Technocratic : The three schools come under this group are :

- Systems
- Cartographic
- Engineering

Systems School : This school puts emphasis on knowledge creation with no help available that is further validated by esteemed peers, codified and stored in a Knowledge management system.

Cartographic School : In this school, Knowledge management system identifies only the knower of the tacit knowledge i.e. it is a directory of experts.

Engineering School : This school consists of two components arising from business process reengineering. i.e. the description of the process plus output generated by such process. It seems to be a hybrid between systems and cartographic schools.

Economic : As the name suggests, this school seeks to market and sell knowledge as a commodity. unlike the technocratic schools where knowledge is accessed and shared to knowledge seekers, this school restricts knowledge to customers an organization considers profitable.

Behavioral : The three management schools come under this taxonomy are :

- Organizational school
- Spatial School
- Strategic school

Organizational School : This school focuses on collaborating knowledge in a learning community by strengthening the ties between different knowledge holders. It is a Community of Practice (CoP).

Spatial School : This school fronts its argument that modern management practices - such as hierarchical structures -, commercial buildings and technology are an antithesis to a conducive environment for learning, knowledge sharing, discussions and bonding. It suggests that space must be set aside for exciting new ideas to emerge.

Strategic School : The name says it all. A word of caution though : it doesn't mean that all the other schools cannot support an organization's competitive strategy. The difference of thought with this school compared to others is that in the other schools, not all knowledge is immediately strategic. Also, this school looks at, in addition to the internal improvements and problem solving, the bigger corporate picture.

3.4 Institutional Knowledge Evolution Cycle

The Institutional Knowledge Evolution Cycle considers five stages in the pathway :

- 1) **Knowledge Development** : Knowledge is developed through learning, innovation, creativity, and importation from outside;
- 2) **Knowledge Acquisition** : Knowledge is captured and retained for use and further treatment;
- 3) **Knowledge Refinement** : Knowledge is organized, transformed, or included in written material, knowledge bases, and so on to make it available to be useful;
- 4) **Knowledge Distribution and Development** : Knowledge is distributed to Points-of-Action (PoAs) through education, training programs, automated knowledge based systems, expert networks, to name a few- to people, practices, embedded in technology and procedures, etc.; and
- 5) **Knowledge Leveraging** : Knowledge is applied or otherwise leveraged. By using (applying) knowledge, it becomes the basis for further learning and innovation as explained by other mechanisms.

3.4.1 Development

Knowledge Development can be defined as “ a process carried out at strategic, operational and tactical levels of command, to provide the decision makers with a comprehensive understanding of complex environments. This will include relationships and interactions between systems and actors within the engagement space”.

Knowledge development is particularly challenging in certain types of organizations. Project-Based Organizations (PBOs) is one context where it is especially difficult to organize for successful knowledge development.

Project-based organizations operate mainly on two levels - the project level and the organizational level (strategy, top management, cross-project coordination, etc.). Success hinges upon the interlinkages between these levels.

Projects are excellent mechanisms for integrating knowledge and for innovation, however, projects are often becoming rather independent, even autonomous, within the PBO structure, resulting in difficulties in achieving accumulative knowledge development in the PBO.

Project-based organizations with leaders who believed in the individuals' ability and willingness to grow and prosper, who stimulated many different knowledge activities and thus clearly stated that knowledge development is vital for their organization and who focused mainly on socialization control of knowledge (yet a bit on the performance), they were the most successful ones.

3.4.2 Acquisition

Knowledge acquisition is the process to extract structure and organize the knowledge from various sources of human experts and is also consider as the process of adding new knowledge and

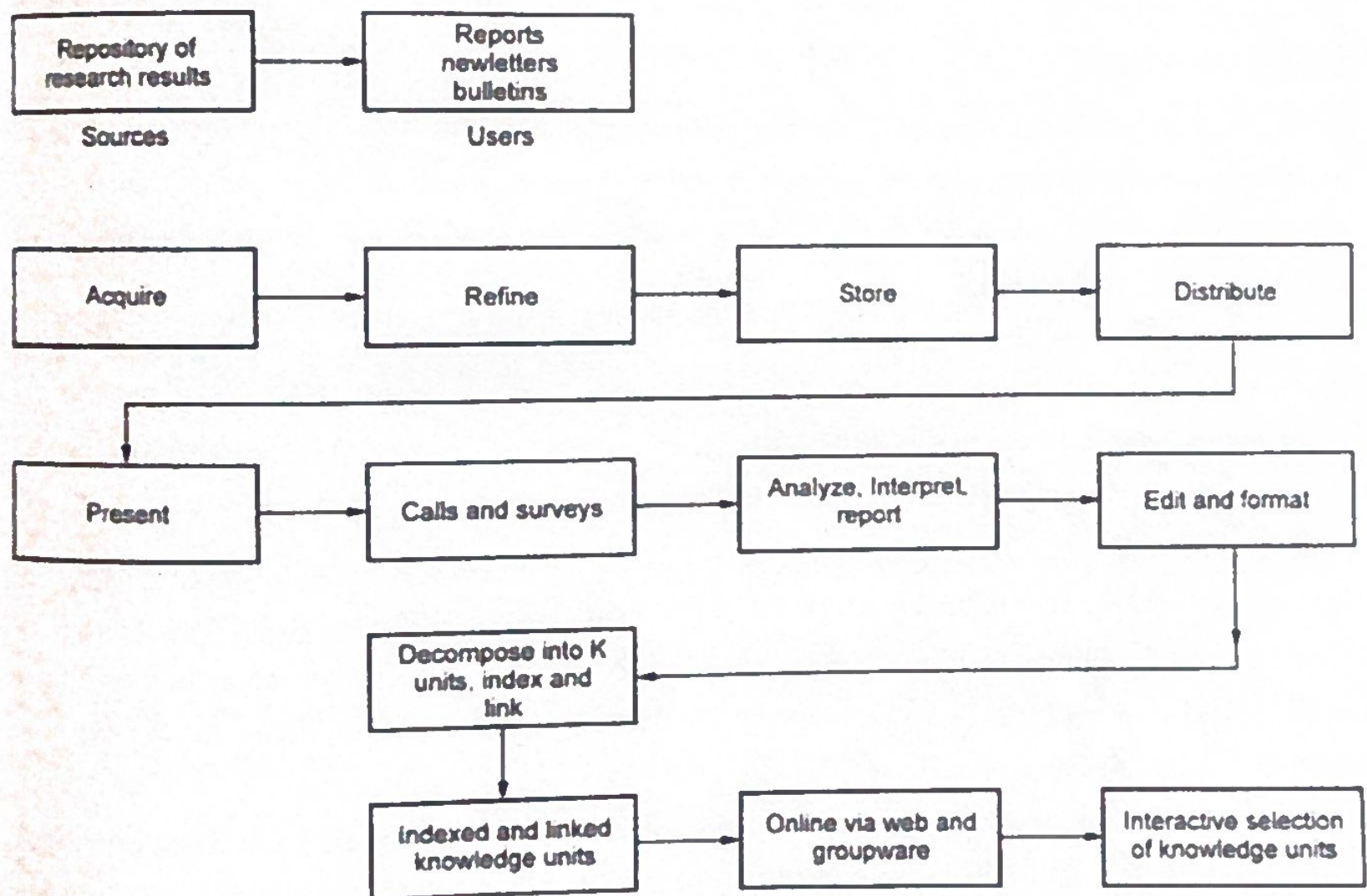
to change which was anonymously acquired to the knowledge base. It is mainly used in the system development. The purpose of knowledge acquisition is to elaborate the capability of the system to improve the specific task of the performance consisting of facts, rules, concepts, procedures, heuristics, formulas, relationships or other useful information

Knowledge acquisition process facilitates the assimilation of knowledge and experiences of different specialties. For example, an agricultural diagnostic expert system requires the integration of specialists in various fields such as nutrition, plant pathology, entomology, breeding, and production. When problem occurs, the system can help the user more efficiently in identifying the cause of the problem. It then helps in consulting a document that handles a specific problem

The existing knowledge and newly acquired knowledge should be incorporated in a significant way to draw inferences from resultant body of knowledge which will be effective and many of the important conclusions for which the systems can be intended. The knowledge should be accurate, non-redundant, consistent, non-contradictory and fairly complete in the sense that it will be possible to rely reasonably.

3.4.3 Refinement

Refinement may be physical (like migrating from one medium to another) or logical (like restructuring, relabeling, indexing, and integrating.)



Refining also defines cleaning up (like sanitizing content so as to ensure complete anonymity of sources and key players involved) or standardizing (like conforming to templates of a best practice or lessons learned as used within that particular organization). This stage also adds up to the value by creating more readily usable knowledge objects and by storing the content more flexibly for future use.

3.4.4 Distribution

Knowledge from users for users. The term describes knowledge that is collected from a large number of individuals and can be used by each person.

In the age of globalization and Internet, gathering and distributing content such as "common knowledge" or "collective knowledge" is important. It helps promote the general knowledge and education of the user based on the knowledge of others. This concept is also used in work groups.

Knowledge is distributed to Points-of-Action through education, training programs, automated knowledge-based systems, expert networks, to name a few; to people, practices, embedded in technology and procedures, etc.

Distribution defines how the product is to be delivered to the end-user (like fax, print, email) and encloses not only the medium of delivery but also its timing, frequency, form, language, and so on.

3.4.5 Deployment

The set of repeatable, quantifiable and measurable activities that are performed within a specific operating Environment by Resources and Systems to successfully construct, deliver, and prepare a versioned Knowledge Item or Entity to or for that Environment, for use by its targeted End Users.

It is crucial for designers to be able to use effectively and efficiently design knowledge. The designers must be provided with in the form ready for effective and efficient applications. This is the knowledge deployment that is an activity from a meta-level point of view to understand and to develop a mechanism for systematizing, structuring, and integrating design knowledge.

The deployment of a knowledge base, while often the catalyst and focus for renewed focus on KM, is only a framework to enable a new set of practices.

Knowledge deployment should mean, selecting from a variety of knowledge, to find and to make knowledge ready for a right combination that can generate added-value in an integrated manner.

"Deploying Knowledge Management" means launching or updating one's *KM Program*, which includes tools, content, culture and governance. Acting holistically, looking at these facets as

interdependent parts of a whole, will help assure the greatest possible impact of KM in driving user efficiency and effectiveness - as well as the overall success of the initiative as an evolving organizational competency.

3.4.6 Leveraging

Leverage is the ability to influence a system, or an environment, in a way that multiplies the outcome of one's efforts without a corresponding increase in the consumption of resources.

In other words, leverage is the advantageous condition of having a relatively small amount of cost to yield a relatively high level of returns.

Leveraging Knowledge - Effectively managing existing knowledge and transferring the knowledge between various stakeholders for improving the capabilities of an organization that allow them to differentiate their goods and services from those of their competitors.

Levering knowledge can be done through knowledge management systems.

A learning organization has to ensure that it not only generates knowledge but puts in place mechanisms to ensure that such knowledge is transferred and leveraged for competitive advantage.

Often, it is the case that organizations have pockets of individuals possessing knowledge as well as specific teams that are better than the others in innovation and knowledge building. While this is good for those teams and individuals in their personal and group capacities, the organization as a whole does not benefit since such distributed and disparate as well as discrete pockets of knowledge cannot be transferred to the whole organization as well as leveraged by those seeking synergies for organizational benefit.

Organizational knowledge evolution life cycle :

- 1. Originate/create knowledge :** Members of an organization develop knowledge through learning, problem solving, innovation, creativity, and importation from outside sources.
- 2. Capture/acquire knowledge :** Members acquire and capture information about knowledge in explicit forms.
- 3. Transform/organize knowledge :** Organizations organize, transform, or include knowledge in written material and knowledge bases.
- 4. Deploy/access knowledge :** Organizations distribute knowledge through education, training programs, automated knowledge-based systems, or expert networks.
- 5. Apply knowledge :** The organization's ultimate goal is applying the knowledge.

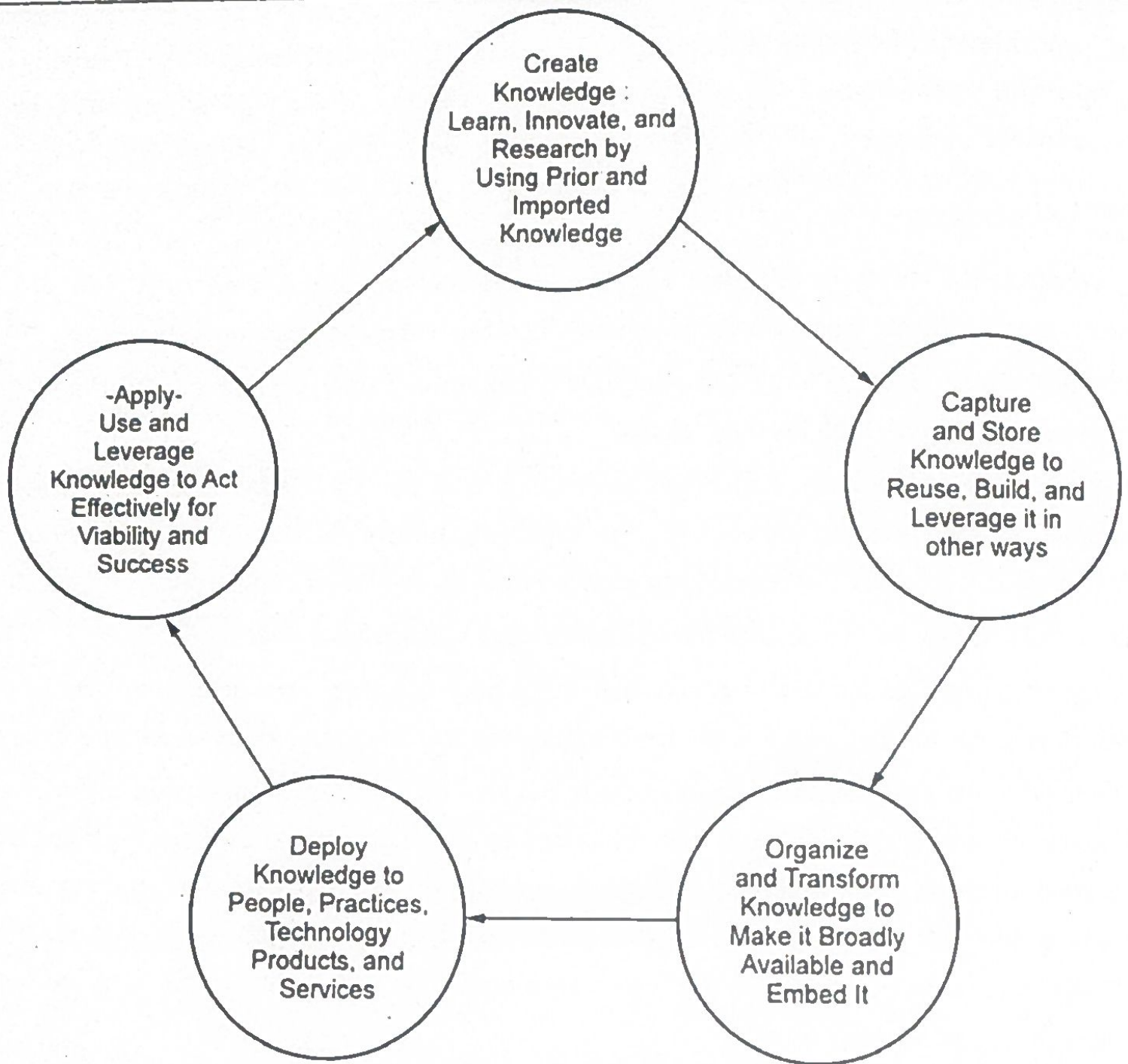


Fig. 3.4.1 The organizational knowledge evolution cycle

Personal knowledge evolution life cycle :

The Personal Knowledge Evolution Cycle also has five stages that depict how knowledge, as it becomes better established in an individual's mind, migrates from barely perceived notions to be better understood and useful. The five stages of this cycle are as follows :

- 1) **Tacit Subliminal Knowledge** : This knowledge is mostly nonconscious and is not well understood. It is often the first glimpse we have of a new concept.
- 2) **Idealistic Vision and Paradigm Knowledge** : Part of this knowledge is well known to us and explicit - we work consciously with it. Much of it-our visions and mental models - is not well known, it is tacit, and only accessible nonconsciously

3) Systematic Schema and Reference Methodology Knowledge : Our knowledge of underlying systems, general principle, and problem-solving strategies is, to a large extent, explicit and mostly well known to us.

4) Pragmatic Decision : Making and Factual Knowledge. Decision-making knowledge is practical and mostly explicit. It supports everyday work and decisions, is well known, and is used consciously.

5) Automatic Routine Working Knowledge : We know this knowledge so well that we have automated it. Most has become tacit - we use it to perform tasks automatically - without conscious reasoning.

3.5 KM in software Engineering

Knowledge management is an expanding and promising discipline that has drawn considerable attention. Recent developments in computer and IT technologies enable sharing documented knowledge independently of time and location.

The KM concept emerged in the mid-1980s from the need to derive knowledge from the "deluge of information" and was mainly used as a "business world" term. In the 1990s, many industries adopted the term KM in connection with commercial computer technologies, facilitated by development in areas such as the Internet, group support systems, search engines, portals, data and knowledge warehouses, and the application of statistical analysis and AI techniques. KM implementation and use has rapidly increased since the 1990s : 80 percent of the largest global corporations now have KM projects. Over 40% of Fortune 1000 companies now have a chief knowledge officer, a senior-level executive who creates an infrastructure and cultural environment for knowledge sharing.

Need of Knowledge Management in Software Engineering

Software development is a quickly changing, knowledge-intensive business involving many people working in different phases and activities. The available resources are not increasing along with the increasing needs; therefore, software organizations expect a rise in productivity. Knowledge in software engineering is diverse and its proportions immense and steadily growing. Organizations have problems identifying the content, location, and use of the knowledge. An improved use of this knowledge is the basic motivation and driver for KM in software engineering and deserves deeper analysis.

Benefits of Knowledge Management to Business Organizations

- **Decreasing time and cost and increasing quality :** Organizations constantly need to decrease software projects' development time and costs. Avoiding mistakes reduces rework; repeating

successful processes increases productivity and the likelihood of further success. So, organizations need to apply process knowledge gained in previous projects to future projects. Unfortunately, the reality is that development teams do not benefit from existing experience and they repeat mistakes even though some individuals in the organization know how to avoid them. Project team members acquire valuable individual experience with each project - the organization and individuals could gain much more if they could share this knowledge.

- **Making better decisions :** In software development, every person involved constantly makes technical or managerial decisions. Most of the time, team members make decisions based on personal knowledge and experience or knowledge gained using informal contacts. This is feasible in small organizations, but as organizations grow and handle a larger volume of information, this process becomes inefficient. Large organizations cannot rely on informal sharing of employees' personal knowledge. Individual knowledge must be shared and leveraged at project and organization levels. Organizations need to define processes for sharing knowledge so that employees throughout the organization can make correct decisions.

Benefits of Knowledge Management to Software Organizations

Software organizations have vast amounts of knowledge in various areas, such as knowledge that is critical to achieve business goals. Let us look at some of these knowledge areas and organizations' related needs.

- **Acquiring knowledge about new technologies :** The emergence of new technologies makes software development more efficient, but at the same time, they can be a project manager's worst nightmare. It is difficult for developers to become proficient with a new technology and managers to understand its impact and estimate a project's cost when using it. When developers or project managers use a technology that a project's team members are unfamiliar with, engineers often resort to the "learning by doing" approach, which can result in serious delays. So, organizations must quickly acquire knowledge about new technologies and master them.
- **Accessing domain knowledge :** Software development requires access to knowledge not only about its domain and new technologies but also about the domain for which software is being developed. An organization must acquire new domain knowledge either by training or by hiring knowledgeable employees and spreading it throughout the team.
- **Sharing knowledge about local policies and practices :** Every organization has its own policies, practices, and culture, which are not only technical but also managerial and administrative. New developers in an organization need knowledge about the existing software base and local programming conventions. Unfortunately, such knowledge typically exists as organizational folklore. Experienced developers often disseminate it to inexperienced

developers through ad hoc informal meetings; consequently, not everyone has access to the knowledge they need. Passing knowledge informally is an important aspect of a knowledge-sharing culture that should be encouraged. Nonetheless, formal knowledge capturing and sharing ensures that all employees can access it. So, organizations must formalize knowledge sharing while continuing informal knowledge sharing.

- **Capturing knowledge and knowing who knows what :** Software organizations depend heavily on knowledgeable employees because they are key to the project's success. However, accessing these people can be difficult. Software developers apply just as much effort and attention determining whom to contact in an organization as they do getting the job done. These knowledgeable people are also very mobile. When a person with critical knowledge suddenly leaves an organization, it creates severe knowledge gaps - but probably no one in the organization is even aware of what knowledge they lost. Knowing what employees know is necessary for organizations to create a strategy for preventing valuable knowledge from disappearing. Knowing who has what knowledge is also a requirement for efficiently staffing projects, identifying training needs, and matching employees with training offers.
- **Collaborating and sharing knowledge :** Software development is a group activity. Group members are often geographically scattered and work in different time zones. Nonetheless, they must communicate, collaborate, and coordinate. Communication in software engineering is often related to knowledge transfer. Collaboration is related to mutual sharing of knowledge. Group members can coordinate independently of time and space if they can easily access their work artifacts. So, group members need a way to collaborate and share knowledge independently of time and space.

3.6 Managing Software Knowledge

In the modern world everything is moved toward new management and knowledge management is not an exception. New management approaches mentioned a four-step cycle including knowledge recognition and achievement, knowledge sharing, knowledge utilization and creation for knowledge management.

According to this cycle knowledge and its resources (including the individuals' explicit and implicit knowledge, information banks, documents, etc) are recognized and stored appropriately. Then, in order to create more value for the related awareness and more possibility of the regeneration of the knowledge, the achieved knowledge should be shared between the individuals. Then, it is necessary for the achieved knowledge to be utilized for the purposes and creation of the knowledge finally happens based on the entrance and utilization of the new information.

According to the new management approach, software development cycle consists of steps including needs, analysis, planning, implementation, examination, and maintenance. This cycle

consists of engineering activities which are designed and programmed with the purpose of managing the life cycle of a software product. These are generally divided into three parts including programming, implementing, and maintaining.

As XP methodology is one of the Agile software development methodologies which consists of the new management activities of the Agile software development cycle, it seems that factors existing in the XP life cycle have the capacity to be adapted to the elements existing in the knowledge management cycle.

Knowledge recognition and achievement

Knowledge recognition and achievement means that knowledge is achieved and adapted to the related needs and is stored appropriately in order to be used in necessary time, place, and circumstances. The first phase in the XP life cycle is exploration phase. This phase which consists of the Spike architecture is based on the definitions proposed by customers, which are called user stories.

User stories are considered as major motivations in the XP methodology. These stories provide high-level needs of the system and are considered as definite inputs in planning process. Thus, it can be said that in this phase customer has the role of knowledge producer.

Requirements related to the release planning as well as system metaphors related to the basic architecture are made available to the developers through the knowledge obtained from user stories. In this step, developers have the role of knowledge consumers.

Knowledge sharing

Knowledge sharing means that knowledge should always be continuing and it should be avoided to be stagnant, because the only dynamic flow is knowledge which can keep its value. The second phase in the XP life cycle is planning phase. In this phase, developers use knowledge created by customers in order to reach an agreement on prioritized user stories. This phase is related to the agreement on the user's prioritized stories with features which the system should have from the viewpoint of the developers. In this step two types of estimation including uncertain and confident estimations derived from the achieved and recognized knowledge are done.

Knowledge utilization

Knowledge utilization happens after knowledge recognition and achievement, sharing and dividing in order to use knowledge. In the next steps on the XP life cycle there is the iteration and release step. In this step, a lot of iterations are done many times and after some little iterations an acceptance test is done by customer. If the test scenario is confirmed by the customer, additional tests are done in the production phase in order to check performance of the system. If

not, the process remains in the same iteration in order for the errors to be totally removed. Due to this reason these iterations and releases lead to decreasing the danger of misleading development.

Thus, it can be generally said that these steps help with utilization of the knowledge created by the acceptance test in order to remove the explored faults and they lead to utilization of this knowledge in order to pattern, test, and consolidate the related system.

Knowledge creation

Creation of knowledge means that the organization should be able to recognize its needed knowledge appropriately and if necessary add some points to it or achieve it from the external resources of knowledge. The last step of the XP life cycle is maintenance phase. This phase encompasses planning, iteration, and distribution phases. It needs an attempt for doing the duties related to customer support. This phase leads to knowledge creation from customer through encompassing planning, iteration, distribution, and production phases. Knowledge created in this phase has to be recognized again and if needed, some points should be added to it, because development of the system stops in this phase until the time when all of the customer's needs are satisfied in all of the related aspects.

3.7 Challenges of Migrating to Agile Methodologies

Management and organizational issues :

- Organizational Culture
- Management Style
- Organizational Form
- Management of Software Development Knowledge
- Reward Systems

Agile methodologies require a shift from command-and control management to leadership-and-collaboration. The organizational form that facilitates this shift needs the right blend of autonomy and cooperation to achieve the advantages of synergy while providing flexibility and responsiveness. The project manager's traditional role of planner and controller must be altered to that of a facilitator who directs and coordinates the collaborative efforts of those involved in the development, thus ensuring that the creative ideas of all participants are reflected in the final decision.

The biggest challenge here is to get the project manager to relinquish the authority he/she previously enjoyed. Knowledge management is of vital importance to organizations. Traditional development approaches create much documentation. Such records serve as useful artifacts for communication and traceability of design.

Agile methodologies, on the other hand, encourage lean thinking and cutting down on overhead, particularly documentation. Much of the knowledge in agile development is tacit and resides in the heads of the development team members. This can make the organization heavily dependent on the development teams and can potentially shift the balance of power from the management to the development teams. Such a situation may not be acceptable to many organizations. This impasse can be resolved by determining which knowledge should be codified and what may remain tacit.

Agile development relies on teamwork, as opposed to individual role assignment that characterizes traditional development. Performance measurement and reward systems, therefore, must be suitably designed for successful adoption of agile methodologies.

People related issues :

- Working effectively in a team
- High level of competence
- Customer relationships - commitment, knowledge, proximity, trust, respect

A cooperative social process characterized by communication and collaboration between a community of members who value and trust each other is critical for the success of agile methodologies. For programmers accustomed to solitary activities or working with relatively homogeneous groups of analysts and designers, the ideas of shared learning, reflection workshops, pair programming, and collaborative decision making may be overwhelming. At the present time, there is little evidence to suggest that agile principles will work in the absence of competent and above-average people. This can pose serious problems related to staffing and morale.

First, it will be difficult to find enough personnel to staff software development teams that use agile methodologies. Second, it will create a culture of elitism within the systems development group that may affect the morale of non-agile developers. In an agile environment, the development team comprising software developers and the customer makes most of the decisions. This creates a pluralist decision-making environment due to the diverse backgrounds, attitudes, goals, and cognitive dispositions of the team members.

Decision making in this environment is more difficult compared to the traditional approach where the project manager is responsible for most decisions. It may take an organization enormous effort, time, and patience to build a culture of trust and respect among its employees to facilitate such collaborative decision making. The success of agile development hinges on finding customers who will actively participate in the development process. Further, the customers are expected to be "Collaborative, Representative, Authorized, Committed, and Knowledgeable". It is not an easy task to find such persons, especially for complex systems.

Process Related Issues :

- Change from process-centric to a feature-driven, people-centric approach
- Short, iterative, test-driven development that emphasizes adaptability
- Managing large, scalable projects
- Selecting an appropriate agile method

The problem of changing attitudes and practices from process-centric to people centric is acute. Organizations that have for years attempted to achieve higher levels of CMM are particularly susceptible to this problem. The idea of changing a process to fit the capabilities and competencies of people and the characteristics of the project, rather than using a rigid process encompassing standardized activities may be sound, but can be achieved only through significant investment of time, effort, and capital.

Traditional processes are compliance-driven and activities- and measurement-based, aimed at providing assurance. Agile methodologies rely on speculation, or planning with the understanding that everything is uncertain, to guide the rapid development of flexible and adaptive systems of high value. They stress the importance of assessing as opposed to measuring, and are highly tolerant of change. One of the biggest barriers to migration is the change in a process model from a life cycle model to one that supports feature based development using evolutionary and iterative development. Such a change entails major alterations to work procedures, tools and techniques, communication channels, problem-solving strategies, and roles of people.

Agile methodologies place a premium on testing, urging developers to develop the test code upfront. The notion of Test-Driven Development (TDD) is becoming increasingly popular within the agile community. TDD is motivated by the fact that thinking about and writing tests prior to coding will make the code more understandable and maintainable. TDD also facilitates continuous integration of new code and/or changes without adversely affecting the existing code base. However, the firmly entrenched tradition of writing code prior to testing must be overcome to institutionalize the practice of early and frequent testing. TDD also redefines the role of the quality assurance function in systems development.

Technological issues :

- Appropriateness of existing technology and tools
- New skill sets - refactoring, configuration management, JUnits

An organization's existing technology can impact the efforts to migrate to agile methodologies. Companies that rely solely on mainframe technologies may find it difficult to assimilate agile methods compared to those that use OO development techniques. Tools play a critical role in successful implementation of a software development methodology. Organizations planning to

adopt agile methodologies must invest in tools that support and facilitate rapid iterative development, versioning/configuration management, JUnits, refactoring, and other agile techniques. Of course, tools alone cannot make software development successful. People must be trained to use them correctly.

3.8 Agile Knowledge Sharing

Unlike traditional project management where a project manager bears most of the responsibilities, agile project management distributes all responsibilities of the project among team members. The workforce itself shoulders project management roles when defining the best strategies to attain the final objectives. Group members collectively select which individual would accomplish what task, which methodical practices are essential to attain specific objectives, and so on.

Here are four ways you can manage and share information in an agile workforce :

1. Creating a learning environment

Agile organizations generate corporate principles that inspire collaboration and learning from one another. Such companies have interactive communication at their core, that's mirrored as "persons and communications over procedures and tools" in an agile strategy. Businesses using agile approach develop their work practices to incorporate team meetings and continuous knowledge sharing between designers. All this aids in transmitting tacit knowledge from one employee to another.

2. Pair programming

Pair programming is a popular software development technique in which two programmers work together on a single workstation. This technique can also be applied to the agile project teams with regards to knowledge management. This way, employees who have faced a particular design issue in the past can educate their companions on how to best tackle it and resolve it in the existing setting.

Moreover, by means of pair alternation the whole team can share similar information, as associates will handover experience among one another every time they alternate. Consequently, it surges corporate learning and retains the business information as now if any employee chooses to leave the business, the knowledge would be sustained with the other employees.

This way, agile practices can reduce the obstacle of work-related procedures being detached from information management practices, as information management occurs concurrently with development in agile companies.

3. Transforming tacit knowledge into explicit knowledge

Transmitting tacit information between group associates may prove valuable. Nonetheless, to accurately retain all the information within the company, organizations would want to convert the tacit information of employees into explicit information accessible to everyone working in the company (such as, by publishing documents into relevant repositories).

Owing to their force and speed, agile practices do not offer much room for maintainable information generation and recovery. For instance, after some initial meetings are over, coding process starts straight away. Hence, there is not much space for examining earlier work to realize arrangements or methods that could be utilized in the existing task.

Agile teams focus at plummeting their product's time-to-market and do not have much capacity to concentrate on their learning during their job hours. Moreover, once the project is over and designers find time for assessment, the contextual nature of the product they produced does not permit them to appropriately share ideas and knowledge for prospective usage. Consequently, they would either have to adapt these into abstract forms appropriate for several contexts, which would be time-consuming, or share it in its existing form, which may not be relevant to impending ventures.

4. Using knowledge management tools

Agile organizations share information well within small groups positioned in the conjoint workplace. However, in case of distributed teams, they allow for feebleness in information transfer. For example, when one business has a small number of agile groups employed in discrete workplaces dispersed worldwide. Due to the absence of mutual repository for information, employees are only capable of transferring both explicit and tacit information between employees in their office and only explicit information (such as the version control types) with employees in dispersed locations.

In such a situation, using a dedicated knowledge management software that allows distributed teams to access information from any location, at any time and from any device is the best solution. Such software allows employees to share ideas and experiences not only by sharing files and documents but by also communicating with one another via discussion forums and chats. Equipped with robust reporting and analytics features, these corporate knowledge sharing tools make it easier to keep track of who is contributing what information and by rewarding the employees who contribute the most, information sharing practices can be highly motivated.

3.9 Role of Story-Cards

What is a user story ?

User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template :

As a <type of user>, I want <some goal> so that <some reason>.

User stories are often written on index cards or sticky notes, stored in a shoe box, and arranged on walls or tables to facilitate planning and discussion. As such, they strongly shift the focus from writing about features to discussing them. In fact, these discussions are more important than whatever text is written.

One of the benefits of agile user stories is that they can be written at varying levels of detail. We can write a user story to cover large amounts of functionality. These large user stories are generally known as epics. Here is an epic agile user story example from a desktop backup product :

As a user, I can backup my entire hard drive.

Because an epic is generally too large for an agile team to complete in one iteration, it is split into multiple smaller user stories before it is worked on. The epic above could be split into dozens (or possibly hundreds), including these two :

- As a power user, I can specify files or folders to backup based on file size, date created and date modified.
- As a user, I can indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved.

In XP, Development starts with planning game where customer writes user stories on story cards. Those cards are estimated by the developer, based on those estimation customer priorities them depends on their needs to establish a timebox of an iteration. Developers develop those story cards through pair programming and test driven development. At last customer provides acceptance test to accept the developed functionality. In between they consider all of the XP practices in mind to improve the quality of the software.

Story cards are one of the important aspects of the XP. They are playing vital role in XP. It describes functionality of system or software to be build that will be valuable to either purchaser or user of software. User stories are composed of three aspects :

- A written description of the story used for planning and as a reminder
- Conversation about the story that serves to flush out the details of the story
- Tests that convey and document details and that can be used to determine when a story is complete.

Story cards are written by the customer in XP to articulate their business needs. According to Cohn story cards must be testable, estimable, valuable to the customer, small and independent. These story cards must be written by the customer because they know their business need very well compared to developer.

XP methodology highly relies on the onsite-customer interaction with the developer to identify or to tell which features to implement in a next release. XP builds software systems based on customer's domain knowledge and his/her expertise. In traditional XP or agile software development methodology story cards are written by the customers perhaps with the help of developer. Therefore each user story or story card must be split into unique and independent requirement.

Traditionally user story is written on the 2" X 3" Cards. Usually any size is acceptable to write user story on story cards.

Story Name		ID:	
As a			
I want to			
so that			
Estimate		Actual	

Fig. 3.9.1 User story card template

Characteristics of a user story :

- All user stories should be developed with the expectation that once completed, the functionality defined in the user story will add value to the final product. If it does not add value to the finished product then it should be avoided.
- User Stories should be developed and written collaboratively by the project team.
- The user stories represent a tool to facilitate team communication and collaboration instead of a formal specification or requirement.

The Three C'S : Card, Conversation, Confirmation"; this formula (from Ron Jeffries) captures the components of a User Story :

- **Card** : A written description of the story used for planning and estimation.
- **Conversation** : A discussion about the story to flesh out details. Happens 'just in time'.
- **Confirmation** : Tests that confirms a story's completion.

INVEST : INVEST is an acronym that defines a simple set of rules used in creating well-formed user stories.

Independent : Each User Story should represent a distinct and independent set of business value such that, were it released on its own, it would deliver incremental value over the previous state

Negotiable : While the end-goal may be clearly described, the methods by which that goal is achieved should be negotiable - between the Product Owner and the Development Team, the Product Owner and the Customer, or any other involved stakeholders - so as to prevent unrealistic constraints on the feature or functionality.

Valuable : The business value of any User Story should be readily recognizable by reading the story, and each story should represent some sort of value to a specific user type.

Estimable : We must have enough information that we can properly size a story so that we may properly plan and commit to our work. (But no more!)

Small : User Stories should be small enough that they are able to be completed within a sprint.

Testable : All members of the team need a clear and precise way to verify whether or not a User Story has been completed.

3.10 Story - Card Maturity Model (SMM).

The SMM helps organizations streamline process improvement, encouraging a productive, efficient culture that decreases risks in software, product and service development.

The SMM model breaks down organizational maturity into five levels. For businesses that embrace SMM, the goal is to raise the organization up to Level 5, the "optimizing" maturity level. Once businesses reach this level, they aren't done with the SMM. Instead, they focus on maintenance and regular improvements.

SMM five Maturity Levels are :

- **Initial** : Processes are viewed as unpredictable and reactive. At this stage, "work gets completed but it's often delayed and over budget." This is the worst stage a business can find itself in - an unpredictable environment that increases risk and inefficiency.
- **Managed** : There's a level of project management achieved. Projects are "planned, performed, measured and controlled" at this level, but there are still a lot of issues to address.
- **Defined** : At this stage, organizations are more proactive than reactive. There's a set of "organization-wide standards" to "provide guidance across projects, programs and portfolios."

Businesses understand their shortcomings, how to address them and what the goal is for improvement.

- **Quantitatively managed** : This stage is more measured and controlled. The organization is working off quantitative data to determine predictable processes that align with stakeholder needs. The business is ahead of risks, with more data-driven insight into process deficiencies.
- **Optimizing** : Here, an organization's processes are stable and flexible. At this final stage, an organization will be in constant state of improving and responding to changes or other opportunities. The organization is stable, which allows for more "agility and innovation," in a predictable environment.

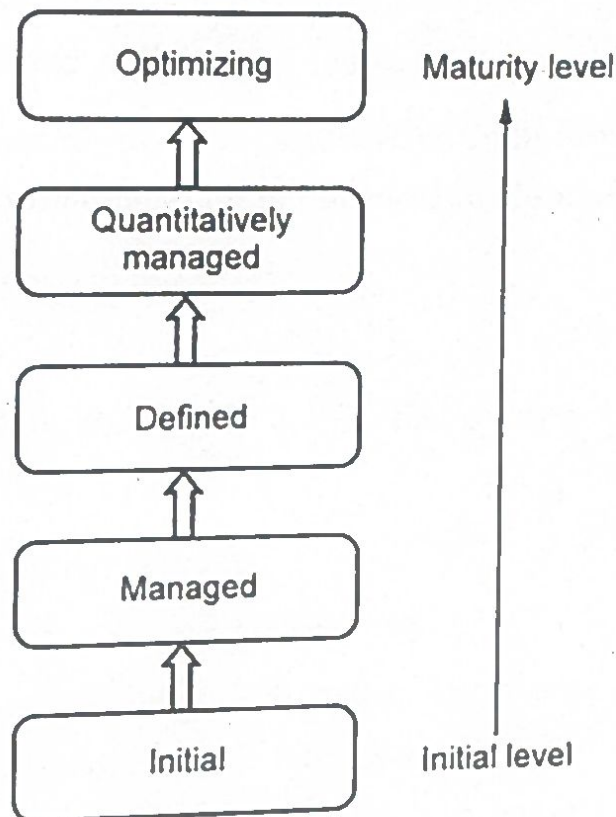


Fig. 3.10.1 SMM for agile software development

SMM Business benefits :

- Increased customer satisfaction
- Improved success with landing and retaining new clients
- Better productivity and efficiency, which will create more profits
- Decreased risk.

3.11 Two Marks Questions with Answers

Q.1 Define Agile Information System.

Ans. : Agile IS may be defined as one that has the ability to sense a change in real-time, diagnose the change in real-time, select a response in real-time, and execute the response in real-time.

Q.2 What are the benefits of Agile Information System ?**Ans. :**

- 1) Reduction in time to implement changes in IS.
- 2) Increase in the robustness of the implemented changes.
- 3) Benefits from response to internal changes.
- 4) Benefits from response to organizational changes.
- 5) Benefits from response to environmental changes.

Q.3 Define Agile Decision Making.

Ans. : Agile Decision Making is designed to help teams make better, faster and fully aligned decisions by working more effectively at pace. In the right context, it enables efficient, high quality, more balanced input from the whole team.

Q.4 Classify the Earl_S Schools of Knowledge management (KM).**Ans. :**

- Technocratic
 - Systems
 - Cartographic
 - Engineering
- Economic
- Behavioral
 - Organizational
 - Spatial
 - Strategic

Q.5 List the five phases of Institutional knowledge evolution cycle.

Ans. : The Institutional Knowledge Evolution Cycle considers five stages in the pathway :

- 1) **Knowledge Development** : Knowledge is developed through learning, innovation, creativity, and importation from outside;
- 2) **Knowledge Acquisition** : Knowledge is captured and retained for use and further treatment;
- 3) **Knowledge Refinement** : Knowledge is organized, transformed, or included in written material, knowledge bases, and so on to make it available to be useful;
- 4) **Knowledge Distribution and Development** : Knowledge is distributed to Points-of-Action (PoAs) through education, training programs, automated knowledgebased systems, expert networks, to name a few- to people, practices, embedded in technology and procedures, etc.;

- 5) **Knowledge Leveraging :** Knowledge is applied or otherwise leveraged. By using (applying) knowledge, it becomes the basis for further learning and innovation as explained by other mechanisms.

Q.6 List out the types of organizational evolution cycle.

Ans. :

1. Originate/create knowledge
2. Capture/acquire knowledge
3. Transform/organize knowledge
4. Deploy/access knowledge
5. Apply knowledge.

Q.7 List out the five stages of personal knowledge evolution life cycle.

Ans. :

- 1) Tacit subliminal knowledge
- 2) Idealistic Vision and Paradigm Knowledge
- 3) Systematic Schema and Reference Methodology Knowledge
- 4) Pragmatic Decision
- 5) Automatic Routine Working Knowledge.

Q.8 State the need of KM in software engineering.

Ans. : Software development is a quickly changing, knowledge-intensive business involving many people working in different phases and activities. The available resources are not increasing along with the increasing needs; therefore, software organizations expect a rise in productivity. Knowledge in software engineering is diverse and its proportions immense and steadily growing. Organizations have problems identifying the content, location, and use of the knowledge. An improved use of this knowledge is the basic motivation and driver for KM in software engineering and deserves deeper analysis.

Q.9 How to manage software knowledge ?

Ans. :

- Knowledge recognition and achievement
- Knowledge sharing
- Knowledge utilization
- Knowledge creation

Q.10 What are the challenges of migrating to Agile methodologies ?

Ans. :

- Management and organizational issues.
- People related issues
- Process related issues
- Technological issues.

Q.11 What are the four ways you can manage and share information in an agile workforce ?

Ans. :

1. Creating a learning environment
2. Pair programming
3. Transforming tacit knowledge into explicit knowledge
4. Using knowledge management tools.

Q.12 What is a user story ?

Ans. : User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:

As a < type of user >, I want < some goal > so that < some reason >.

Q.13 Define the characteristics of a user story.

Ans. :

- All user stories should be developed with the expectation that once completed, the functionality defined in the user story will add value to the final product. If it does not add value to the finished product then it should be avoided.
- User Stories should be developed and written collaboratively by the project team.
- The user stories represent a tool to facilitate team communication and collaboration instead of a formal specification or requirement.

Q.14 What are the three c's in story cards ?

Ans. :

- **Card** : A written description of the story used for planning and estimation.
- **Conversation** : A discussion about the story to flesh out details. Happens 'just in time'.
- **Confirmation** : Tests that confirms a story's completion.

Q.15 Define INVEST.

Ans. : INVEST is an acronym that defines a simple set of rules used in creating well-formed user stories.

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

Q.16 Define Story -Card Maturity Model.

Ans. : The SMM model is based on the agile requirements engineering values, practices and principles.

The SMM model is designed to improve and enhance the agile software development methodology and boost up the agile requirements principles and objectives like the lower cost, customer satisfaction, requirements quality, etc.

Q.17 Define the goals of SMM.

Ans. : The ultimate goal of the SMM (Story cards maturity model) is as :

- Customer satisfaction
- Maintain story cards (requirements) change
- Solution to Vague requirements
- Obtain an understanding of the user story on the story card
- Obtain commitment to the user story (user Requirements)
- Maintain bidirectional traceability of requirements
- Identify inconsistency between project work and user story
- Identify and involve stakeholders instead of single on-site customer
- Manage Requirements Stories, on-site customer, daily meeting,
- Establish Estimates of story cards and define acceptance tests with story cards
- Develop a project plan based on the story cards.

Q.18 List the four levels of Story- card maturity Model.

Ans. :

Level1 - Initial

Level2 - Explored

Level 3 - Defined

Level4 - Improved

Q.19 List the various advantages of SMM.

Ans. :

- To define a generic process model for Story cards based requirements engineering process improvement that is suitable for RE at agile software development environments.
- To design and implement an automated tool that support to apply the proposed model in order to help facilitate process improvement.
- Identify and define story cards based requirements engineering practices
- Recognise story cards based requirement engineering practices problems
- Access and agree story cards based RE practices improvement priorities
- Relate story cards based RE practices problem to RE practices improvement goals
- Contains guidelines for many story cards related requirements engineering activities
- Is designed to be tailored to focus on specific process areas
- Goal focused
- Maturity structure to help with process prioritization.

Q.20 Define the role of story cards in XP.

Ans. : Story cards are one of the important aspects of the XP. They are playing vital role in XP. It describes functionality of system or software to be build that will be valuable to either purchaser or user of software. In XP, Development starts with planning game where customer writes user stories on story cards. Those cards are estimated by the developer, based on those estimation customer priorities them depends on their needs to establish a timebox of an iteration. Developers develop those story cards through pair programming and test driven development. At last customer provides acceptance test to accept the developed functionality. In between they consider all of the XP practices in mind to improve the quality of the software.

□□□

UNIT-IV

Syllabus

UNIT- IV AGILITY AND REQUIREMENTS ENGINEERING Lecture 9 Hrs

Impact of Agile Processes in RE–Current Agile Practices – Variance – Overview of RE Using Agile Managing Unstable Requirements – Requirements Elicitation – Agile Requirements Abstraction Model – Requirements Management in Agile Environment, Agile Requirements Prioritization – Agile Requirements Modeling and Generation – Concurrency in Agile Requirements Generation.

AGILITY AND REQUIREMENTS ENGINEERING

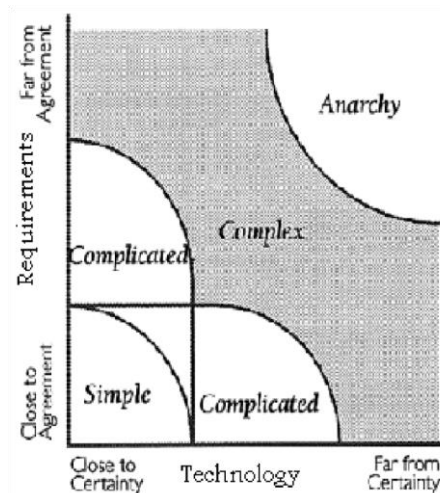
Impact of Agile Processes in RE–Current Agile Practices – Variance – Overview of RE Using Agile – Managing Unstable Requirements – Requirements Elicitation – Agile Requirements Abstraction Model – Requirements Management in Agile Environment, Agile Requirements Prioritization – Agile Requirements Modeling and Generation – Concurrency in Agile Requirements Generation.

Impact of Agile Processes in RE-current agile practices:

- Current Agile Practices
- Agile Experiences
- Variance Between Formal Requirements Engineering and Agile Processes

1.Current Agile Practices

Building advanced technology, competitive systems is complicated. As shown in the requirements/technology figure1 , the degree of complexity increases as the requirements are less known and the technology is less certain



1.Iterative development – frequent iterations generate increments of work that can be inspected to determine the state of the project and serve as a basis for adaptation.

2. Increments of work - composed of working system functionality rather than artifacts. These increments create a 1:1 relationship between progress and product delivery, and provide a mechanism for user feedback to real product rather than arcane internal artifacts.

3. Collaboration – the customers and engineers form teams that work together.

4. Daily meetings – provide a daily picture of the internal status of a project.

5. Adaptation – the teams of developers selforganize daily based on the daily meetings and the developers and customers self-organize at the end of every increment to guide the project to create the greatest value.

6. Emergence² – the architecture, team structure, and requirements emerge during the course of the project rather than being determined at its outset.

2. Agile Experiences

At a recent Agile Workshop³, the subject of emergence was discussed. The participants indicated that they had successfully applied emergence in literally hundreds of projects over the last seven years. These projects ranged from relatively straightforward systems to n-tier, complex systems applying cutting edge technology to poorly defined business problems. Business, embedded, and life critical systems had been successfully developed.

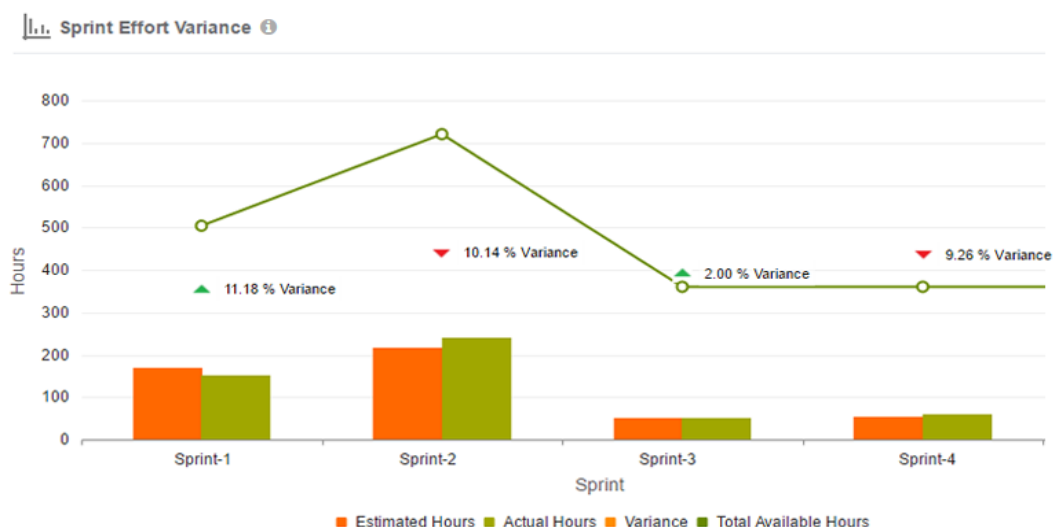
The participants felt that the use of refactoring, or periodic cleaning up of business and technical architecture, was required. Otherwise, in all likelihood, the resulting system would be difficult to maintain and sustain. It would contain unrectified designs and code that wasn't orthogonal. However, none of the participants was able to explain why emergence worked.

3. Variance Between Formal Requirements Engineering and Agile Processes

The fundamental principle underlying requirements engineering is the assumption that a system should be clearly specified before its design and implementation can start. Ideally, a specification should be unambiguous, consistent, concise, traceable, implementation independent, etc., and signed off by all stakeholders involved before any further development is done. Failing to do so has in the past caused tremendous cost.

Variance:

- **Variance** represents the comparison of the effort that was planned (Estimated Hours) amongst the actual efforts (Spent Hours) within a sprint.
- **X Axis** – shows last 15 sprints.
-



1. If the Spent Hours is less than the Estimated Hours, then Effort Variance will be positive.
2. If the Spent Hours is more than the Estimated Hours, then Effort Variance will be negative. If the Variance is positive then the indicator will be in green and spiked up. If the variance is negative then the indicator will be in red and spiked down.
3. Positive Variance represents that the team member has spent less time (in Hours) in comparison of their Estimation (in Hours) within a sprint.
4. Negative Variance represents that the team member has spent more time (in Hours) in comparison of their Estimation (in Hours) within a sprint.
5. **Orange Bar** – Total Hours of work estimated by the Team members (in Hours) within a sprint.
6. **Green Bar** – Total Hours of time spent by a Team members (in Hours) within a sprint.
7. **Green Trend line** – It will show total available hours of all the sprints within a sprint.
8. **Variance** is based on total hours that are spent more/ less in comparison of your estimated hours.
(Sprint Effort Variance = Estimation Hours – Spent Hours).

Overview of RE using agile managing unstable requirements:

Agile development is a better way of managing the changing requirements that many software development projects face. There's been a lot of thought around Agile change management. Here are five ways Agile helps manage changing requirements:

- Customer input happens throughout the development process
- Product backlog sets development priorities
- Daily meetings promote communications
- Task boards make developer tasks and details visible
- User stories and sprints orchestrate change

1. Customer input happens throughout the development process

Back when I got my start as a technical writer, I worked on software development projects that lasted months. The software wasn't completed until the "end of the project."

- Gains customer and stakeholder feedback on features sooner rather than later
- Improves scope control because stakeholders can add new requirements, shift priorities, or rethink requirements on a feature or architectural level
- Gives project teams the room to take risks and innovate based on customer feedback without sacrificing too much time or budget because agile teams can pivot on requirements as needed

2. Product backlog sets development priorities

The one element that took me a while to warm up to was the product backlog. It's like the scrum team's prioritized to-do list of features to build.

Managing or grooming the product backlog can be an art unto itself. In some organizations, the scrum master manages the backlog. Other organizations might choose to have product managers or cross-functional team leaders involved in managing it.

3. Daily meetings promote communications

Holding daily meetings, or stand-ups, are another tool for managing changing requirements. These meetings take place at the same time each day and give team members a chance to talk about the tasks they've completed and any obstacles standing in their way.

A properly managed daily meeting lets developers, team leads and stakeholders (if invited) to share information. Some of that information could be issues and feedback about product requirements that might arise during the implementation process.

4. Task boards make developer tasks and details visible

Product requirements documents are too often read once and left in an email inbox for the duration of the project. Agile development uses the concept of a task board to divide up tasks into multiple columns and make them visible 24/7. These boards parcel out projects into the following stages:

- To do
- In progress
- In testing
- Done

Tasks boards help manage changing requirements because of the visibility they offer, which include:

- Project requirement status is visible to every team member.
- Dependencies of project requirements impacted by changing requirements are clear.
- Shows threaded comments about the changing requirements before and during sprints from the developer and other team members.

5. User stories and sprints orchestrate change

I spent some of my formative years in the IT industry writing requirement documents. I became accustomed to documenting every feature that was being built. When I was first learning about Agile, I came to marvel at how, if you orchestrated user stories and sprints appropriately, you can manage change quite effectively.

For example, a product owner creates a story. Developers can build a new application feature based on the story. During or after the sprint where that feature is built, a salesperson delivers feedback from a customer that shows the feature is missing critical functionality. The product owner can create a new story to build out the feature with the missing functionality during the next sprint.

Requirements Elicitation:

Requirements elicitation is the process of gathering and defining the requirements for a software system. The goal of requirements elicitation is to ensure that the software development process is based on a clear and comprehensive understanding of the customer needs and requirements. Requirements elicitation involves the identification, collection, analysis, and refinement of the requirements for a software system. It is a critical part of the software development life cycle and is typically performed at the beginning of the project. Requirements elicitation involves stakeholders from different areas of the organization, including business owners, end-users, and technical experts. The output of the requirements elicitation process is a set of clear, concise, and well-defined requirements that serve as the basis for the design and development of the software system.

Requirements elicitation is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

Requirements elicitation Activities:

Requirements elicitation includes the subsequent activities. Few of them are listed below –

- Knowledge of the overall area where the systems is applied.
- The details of the precise customer problem where the system are going to be applied must be understood.
- Interaction of system with external requirements.
- Detailed investigation of user needs.
- Define the constraints for system development.

Requirements elicitation Methods:

There are a number of requirements elicitation methods. Few of them are listed below –

1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. Use Case Approach

The success of an elicitation technique used depends on the maturity of the analyst, developers, users, and the customer involved.

1.Interviews:

Objective of conducting an interview is to understand the customer's expectations from the software.

It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.

Interviews maybe be open-ended or structured.

1. In open-ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
2. In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

2. Brainstorming Sessions:

- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

3. Facilitated Application Specification Technique:

It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.

A team oriented approach is developed for requirements gathering.

Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

4. Quality Function Deployment:

In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.

3 types of requirements are identified

- **Normal requirements:** In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc
- **Expected requirements:** These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.
- **Exciting requirements:** It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes.

The major steps involved in this procedure are –

1. Identify all the stakeholders, eg. Users, developers, customers etc
2. List out all requirements from customer.
3. A value indicating degree of importance is assigned to each requirement.
4. In the end the final list of requirements is categorized as –
 - It is possible to achieve
 - It should be deferred and the reason for it
 - It is impossible to achieve and should be dropped off

5. Use Case Approach:

This technique combines text and pictures to provide a better understanding of the requirements.

The use cases describe the 'what', of a system and not 'how'. Hence, they only give a functional view of the system.

The components of the use case design includes three major things – Actor, Use cases, use case diagram.

1. **Actor**

It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.

- Primary actors – It requires assistance from the system to achieve a goal.
- Secondary actor – It is an actor from which the system needs assistance.

2. **Use cases:**

They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.

3. **Use case diagram:**

A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.

- A stick figure is used to represent an actor.
- An oval is used to represent a use case.
- A line is used to represent a relationship between an actor and a use case.

ADVANTAGES OR DISADVANTAGES:

Advantages of Requirements Elicitation:

- Helps to clarify and refine the customer requirements.
- Improves communication and collaboration between stakeholders.
- Increases the chances of developing a software system that meets the customer needs.

Disadvantages of Requirements Elicitation:

- Can be time-consuming and expensive.
- Requires specialized skills and expertise.
- May be impacted by changing business needs and requirements.

Agile Requirements Abstraction Model:

This article presents an investigation of the possibility to integrate traditional requirements engineering in an agile software development method. One reason for integrating a traditional with an agile method was that the case company experienced difficulties with the agile requirements engineering process. The results are based on a qualitative case study of one company that uses scrum for its software development. An agile model, with four abstraction levels was developed in close collaboration with industry

Abstraction levels:

Requirements engineering (RE), on the other hand is a traditional method with the goal to identify, analyze, document and validate requirements. Is it possible to bridge the traditional requirements engineering process with agile methods, and still stay agile? According to, requirements engineering and agile methods are seen as incompatible. One reason is that RE often relies on documentation while agile methods are focusing on face-to-face communication between customers and developers. However, several studies have looked into bridging traditional and agile methods. In the study shows that it is possible to successfully integrate a traditional project management approach with XP projects. Furthermore, Boehm and Turner concluded that agile concepts will continue to integrate into traditional organizations and vice versa. In addition, Nawrocki et al. proposed modifications to the XP methodology to embrace RE practices. One of the modifications was to have a RE phase in the beginning of the project that provides a wider perspective, and still manage to stay as lightweight as the original XP.

ARAM STRUCTURE

This section describes the ARAM's structure. The version of ARAM presented in this paper is based on the development of ARAM at Tactel and is an example of how ARAM can look like. The concepts of the ARAM model as described in this paper, makes it possible for any organization to adapt and tailor ARAM to fit their specific organization. The aim of ARAM is to bridge the traditional approaches and agile methods by allowing project plans and contacts, and still keep time-boxed iterations where the development team can work on their sprint backlog items. ARAM does not interfere with the scrum sprint, rather focus on making the pre-project requirements engineering efficient.

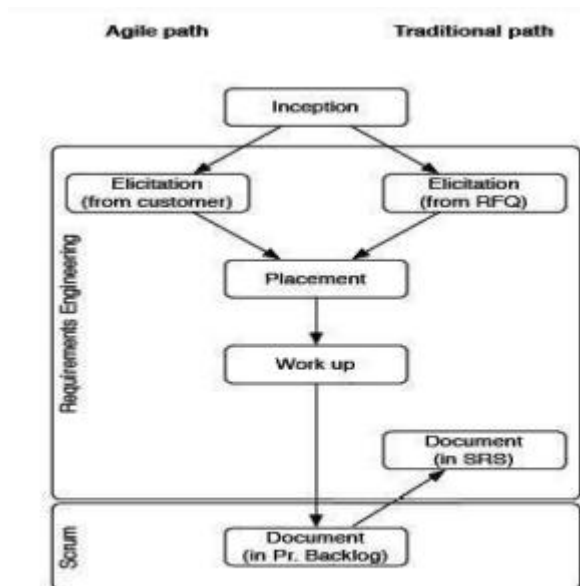


Figure 1. ARAM structure

Project Inception Phase

In a contract-driven business scenario, the customer contacts the software vendor and requests for product information. The response to the customer at this stage is an overview of the product containing the product's feature and what value the product offers. The agile information should include an overview of scrum, target-cost contracts and how the agile path works. The contract and requirements processes will ultimately end up in the product backlog, but there are two different paths to reach that goal. If market-driven development process is preferred, there is no direct customer to contact. In this case, the project inception may come from the product manager, marketing department, or from an innovation meeting at the producing company where high level goals are derived. In the agile path, a target-cost contract with a variable scope is selected. The delivery date and the budget are known in advanced while requirements will be implemented based on their priorities.

Scrum and ARAM

Requirements in the product backlog need to be prioritized according to most value for the customer. One main advantage by using ARAM is that all requirements can be compared. A requirement on feature level should only be compared with other requirements on the feature level. In other words, requirements should only be compared with requirements on the same abstraction level. After prioritizing all requirements in the product backlog, they are analyzed at the next sprint planning meeting. The high priority requirements are selected for the coming sprint. When estimating the effort for each story, requirements need to be broken down to at least the functional abstraction level to be able to make as accurate estimations as possible. An agile practice and basic rule of scrum, is to develop a common language for all team members which was pointed out as one problem in the case study . Therefore, it is desirable to have common names for requirements at each abstraction level, as is illustrated in figure.

To get structure from this set of requirements RAM specifies a number of abstraction levels. In the study by Gorschek the number of abstraction levels are four, but they state that this is tailored for their specific case study and can be variable. When implementing for a specific organisation, more or fewer levels can be used. The four level model can be seen in figure.

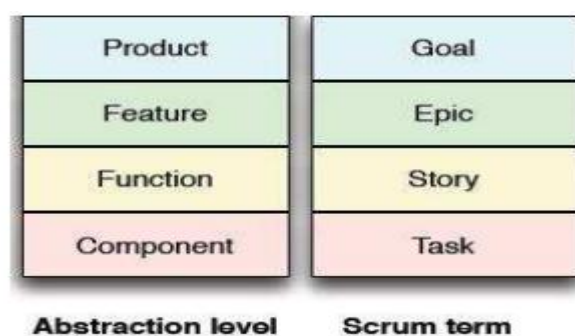
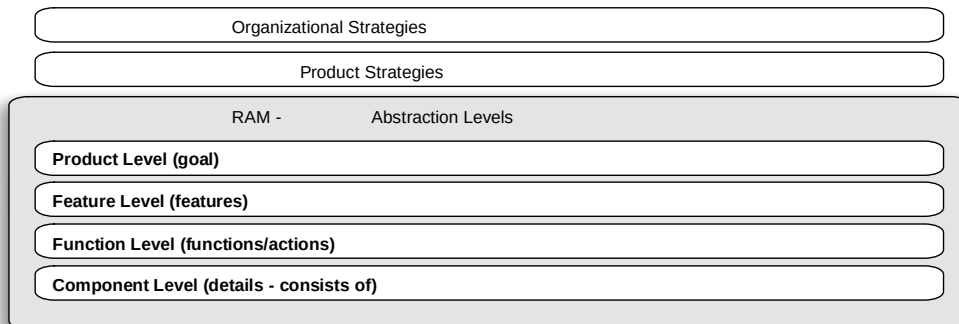
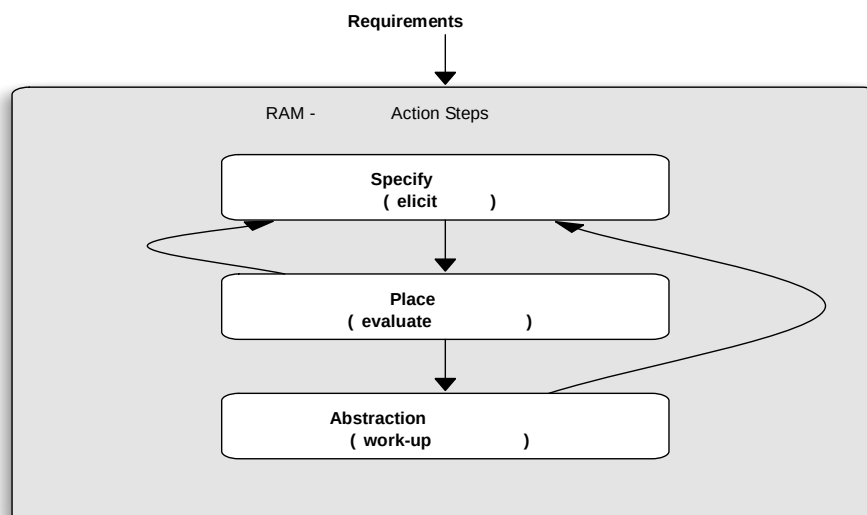


Figure 6. Scrum terms and abstraction levels

The Product level represents the most abstract requirements.



Action steps : RAM has three specified action steps (figure 11) to achieve the right requirements specification based on the abstraction levels. These steps are iterated for all requirements in the system. First of all, the requirement should be specified (elicited). This is the first sketch of an idea about what should be included in the system. Here the four most general attributes to the requirement - including [32] description, reason/benefit/rationale, restrictions/risks and title - are specified. After this specification the requirement is placed (evaluated) according to the heart of RAM model



Attributes

In the process of specifying, placing and working-up the requirements, additional attributes are added. These include the two groups traceability and process attributes for fitting into RAM. The traceability attributes suggested by RAM are *Requirement source*, *Requirement owner*, *Requirements manager* and *Relation/dependency*. The process attributes include *State*, *Reject reason (for rejected requirements only)*, *Due date*, *Version*, *Date of creation* and *Last changed*. Many of the attributes are the same or almost the same as in traditional RE with some anomalies

Requirements management in agile environment

In many ways, the manner of capturing requirements in an Agile project management environment is similar to a “*waterfall*,” or traditional project management environment - numerous meetings with subject matter experts, end users, walkthrough / documenting the current business workflow, creating mockups, etc. However, Agile and traditional project management approaches contrast in how requirements are managed over time.

Some fundamental differences in managing requirements include:

Traditional Requirements Management	Agile Requirements Management
In a “waterfall” project management environment, the approach is to capture and define all of the end-state requirements upfront.	In an Agile project management environment, while high-level requirements are also captured upfront, it is understood that requirements may evolve over the course of the effort.
Requirements are documented in a business requirements document (BRD) or business specifications document (BSD) for the purpose of designing the end state of a product.	The effort begins with a high-level scope agreement where initial business requirements are translated into user stories. Those user stories specify the needs of the product based on the information at the time given.
The goal is to understand the “as-is” state of the existing product or the business gaps that define the lack, so that the “to-be” state of the desired product can be defined.	The goal is no longer focused on eliciting the “as-is” in order to define the “to-be,” but to clarify and ensure understanding of the business need for all users.
By defining the end-state of the application from the beginning, there is little room for change once development begins.	As more information becomes known over time, the team is better able to adjust and make changes accordingly.

Managing requirements in an Agile project management environment is to think through the full life cycle of the requirement; to consider the full user experience and even beyond the defined stakeholders

Agile Requirement Prioritization:

Prioritization in literary terms means the decision of arranging things in order of their importance. Prioritization in agile is the act of deciding in what order the agile team will work on the requirements in a project

Here is the list of popular prioritization techniques:

- MoSCoW prioritization
- Kano model
- Relative weighting method
- Opportunity Scoring
- Stack Ranking
- Priority Poker
- Cost of Delay
- 100 Dollar Test

Agile Prioritization Factors

- The financial value of the requirements is a major factor to be considered in prioritizing requirements. The value could be expressed as new revenue, incremental revenue, or as operational efficiency.
- The cost of developing the requirements is another essential factor to be considered by the product owner. Value and cost together indicate the RoI for the requirements.
- The next factor to be considered in prioritization is the amount and significance of knowledge and capabilities that the team will gain while working on the requirements.
- Understanding the level of risks involved in introducing the new features is essential in the process of prioritization.

Popular Prioritization Techniques

- MoSCoW prioritization – popularized by the DSDM methodology.
- Kano model – introduced by Prof. Noriaki Kano
- The relative weighting method – by Karl Wiegers

8 Popular Prioritization Techniques

1. MoSCoW prioritization – popularized by the DSDM methodology.
2. Kano model – introduced by Prof. Noriaki Kano
3. The relative weighting method – by Karl Wiegers

4. Opportunity Scoring
5. Stack Ranking
6. Priority Poker
7. Cost of Delay
8. 100 Dollar Test

MoSCoW Agile Prioritization Technique

MoSCoW Prioritization in Agile: In the DSDM methodology, the priorities are expressed as per the MoSCoW model:

- Must– The must requirements is given the topmost priority
- Should– Next priority is given to the requirements that are highly desirable, though not mandatory
- Could– The next priority is given to the requirement that is nice to have
- Won't– And the final consideration is given to the requirements which will not work in the process at that point of time.

Kano Model of Prioritization in Agile

Professor Noriaki Kano propagated Kano Model of Prioritization. This prioritization technique involves three levels that include considering customer satisfaction from disappointment to not happy to immediate happiness to get delighted. Two important factors that create an impact on the satisfaction level during this prioritization are the existence of features and the degree of implementation.

Relative Weighting Prioritization Technique

The relative weighting scheme is a simple model where prioritization is done based upon all the factors mentioned above. The major factors considered in relative weighing prioritization technique are:

- The value of a feature and the negative impact that might be caused by the absence of the feature
- Based on the expert judgment made by the product owner and supported by the agile team in ranking the score of features in the following way (a scoreboard from 1 to 9 is usually used)

- Benefit from having the feature
- Penalty for not having the feature
- Cost of producing the feature
- The risk incurred in producing the feature

Opportunity Scoring

Opportunity Scoring is a beneficial prioritization method used by organizations to develop agile products. This prioritization model uses data from market research to help determine what the users expect from your product or service. It allows organizations to create the schedule according to their target audience's wants and needs.

Stack Ranking

Stack Ranking is one of the most popular forms of prioritization techniques that is currently used by a lot of software companies. It is also one of the easiest techniques that allow prioritization based on the user story.

Priority Poker

This agile priority technique is based on similar rules as actual poker played with cards. When playing poker, prioritization is done in a calculative manner, with big wins being the ultimate goal. Similarly, in agile priority poker, items that will yield the highest results in specific target markets are given priority.

Cost of Delay

The objective of this prioritization technique is to understand how much money would the organization lose if a certain feature is not available. This prioritization focuses on monetary loss to understand which features are the most important and the list is created accordingly. It is a proactive approach to ensure the manager fight fires and deal with emergencies that can result in losses.

100 Dollar Test

This technique is also known as Cumulative Voting and is a straightforward process. It is similar to the poker technique but each stakeholder is given 100 points or dollars to assign to each feature or task.

The stakeholders divide their 100 dollars by assigning a spending amount to each feature. Once all the 100 dollars are spent, the moderator then tallies all the points and the feature with the most dollars assigned is given the highest priority, followed by tasks with the next highest amounts.

Agile Requirements Modeling and Generation

The basic idea of agile requirements modeling is to create a foundation for your project's specifications for higher-level understanding. This is done in the beginning stages with details being added as you need them on a just-in-time basis. To fully understand what this model is, it's important to look at agile software development too.

The best way to understand this process is to look at its main goals. Agile requirements modeling is designed to support the goals of software development and aims to achieve the following objectives:

- Establish the best practices for an effective model
- Outline the ways to put those practices into place
- Display alternatives to improve the modeling approach

Types of Requirements:

There are two categories that agile requirements modeling can be divided into. The distinction between the two is important to establish as they distinguish the technicalities of a project from the user interactions.

These two categories are:

1. Behavioral: This form of requirement refers to any user interface issues, usage, and business rules. This can also be seen as the functional requirements of a project.
2. Non-behavioral: On the technical side, non-behavioral looks at a system's features relating to availability, security, performance, interoperability, dependability, and reliability. In other words, these are the non-functional requirements.

Applying Agile Requirements Modeling Principles

The future of your agile project relies heavily on several factors. As any developer already knows, anything could happen, from client changes to user-need adaptations, that can turn the project's success into failure.

Without a clear plan or project outline, a system can quickly become confused by varying stories and goals. However, agile requirements modeling prevents this from happening and ensures that the same goal is communicated at all times.

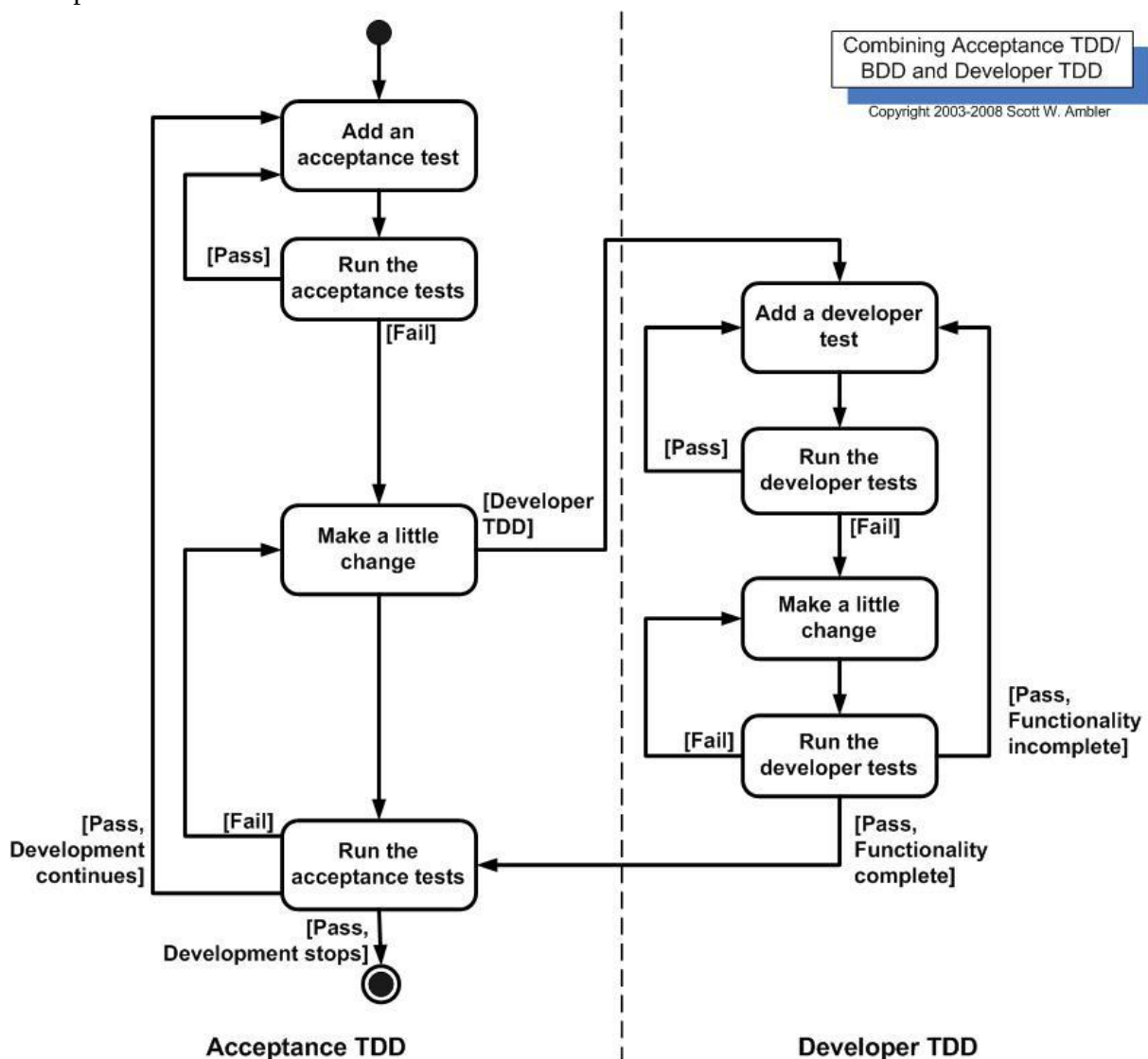
With this in mind, when it comes to agile modeling, getting the key principles right is crucial. Although this isn't a rigid procedure, keeping these practices at the foundation of your work will steer your project towards a more streamlined process.

These principles include:

- Outlining how the software is your primary goal
- Establishing the next effort as your secondary goal
- Remaining agile by focusing only on the models required
- Assuming simplicity
- Embracing change
- Modeling with purpose
- Creating multiple models
- Producing quality work
- Maximizing stakeholder investment

The key benefits of this modeling form are based on the corporation of the above principles. Agile modeling is a synergistic approach to development that requires all of these practices to be utilized. Leaving one out can open up the project to becoming less effective.

The reason for this is that software is complex and it needs to be analyzed from various standpoint.



Concurrency in Agile Requirements Generation:

The major activities required in a software development life cycle (SDLC) were identified in the waterfall model. The agile concurrent software process model proposes that the activities identified in the waterfall model are not done sequentially; these activities progress concurrently with varying intensities during the entire software life cycle. Also, a focus on a working software subset as opposed to documents, giving high importance to customer satisfaction and sensitivity to human factors go a long way in contributing towards a software project's overall success.

WATERFALL MODEL

It was understood that software had a life-cycle starting from the decision to make that software to the time the software became operational. The term software development life-cycle was based on a basic software process model, indicating the stages software would go through in its life-cycle. The first response to the software crisis was the discovery of the waterfall software process model. It was thought that the root cause of the problem was that the requirements were not understood well enough before the software design and proper design was not done prior to coding.

Applying the engineering paradigm of freezing requirements before design and freezing design before manufacture, the waterfall model stipulated that software should be made in a sequential series of stages, like requirements' analysis, design, detailed design, coding, integration, installation, operations and maintenance. Slowly, it became apparent that it was not possible to have water-tight compartments and it was desirable to have feedback between successive stages. Thus a design error found during coding would lead to a correction of design and then the coding would continue. This was in line with the engineering practice for a design error found during manufacture is first corrected in design and then manufacturing is done as per the modified design. Also, taking the clue from engineering again, it was felt that making a prototype of the system in early stages would help in understanding the requirements better and would eventually help in making a better system.

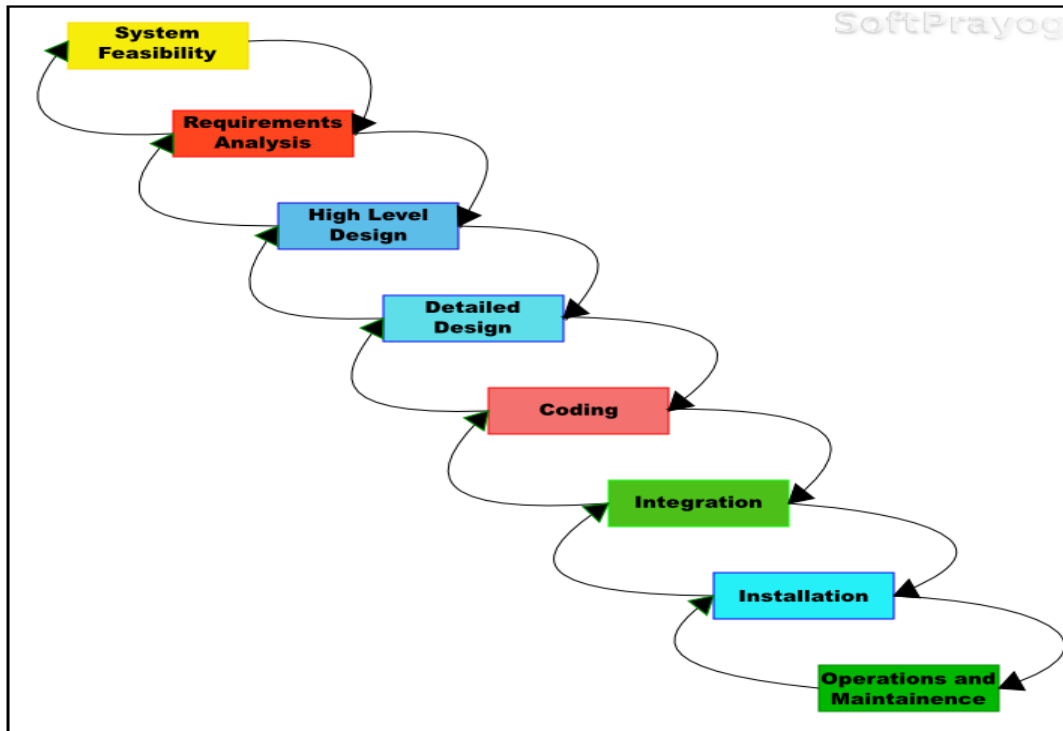


Figure 1: The Waterfall Software Process model

UNIT-V

Syllabus

UNIT- V AGILITY AND QUALITY ASSURANCE

Lecture 9 Hrs

Agile Product Development – Agile Metrics – Feature Driven Development (FDD) – Financial and Production Metrics in FDD – Agile Approach to Quality Assurance - Test Driven Development – Agile Approach in Global Software Development.

UNIT V: AGILITY AND QUALITY ASSURANCE

P.Praveen, Asst.Prof • Agile Methodologies

Syllabus Focus: Agile Product Development - Agile Metrics - Feature Driven Development (FDD) - Financial and Production Metrics in FDD - Agile Approach to Quality Assurance - Test Driven Development - Agile Approach in Global Software Development.

| Agile Product Development

WHAT IS AGILE PRODUCT DEVELOPMENT?

Agile development for new products allows companies to develop products in a manner that responds effectively to change, risk, and uncertainty. This involves self-organizing teams that create fast prototypes in collaboration with each other and with customers. These prototypes are then iterated through repeated cycles where customers interact with them, and provide feedback that is then incorporated into the product. Often, organizations start with software development, before applying Agile to other product development activities.

AGILE PRODUCT DEVELOPMENT METHODOLOGY

Agile first came to prominence through the Agile Manifesto, a document published in 2001 and written collaboratively by seventeen software development leaders. The Agile Manifesto proposed twelve Agile principles that express the spirit of Agile development through the Agile methodology.

Subsequently, Agile has evolved into a set of frameworks and practices (Scrum, sprints & sprint planning, daily stand-up meetings, burndown charts, etc.) that are frequently used especially by software development organizations using Agile. But these specific practices are not essential to Agile. Preeminently, Agile is a set of principles that favor empowered teams that are granted the freedom to innovate and create products that delight customers. It also implies a learning organization that adapts nimbly to changes in markets and technologies.

We view Agile as a continuum, not just a black and white proposition. We also apply Agile methods to our approach to the Product Development Process, regardless of the product or service you're offering.

THE 12 PRINCIPLES OF AGILE PRODUCT DEVELOPMENT & MANAGEMENT

The Agile Manifesto lays out 12 principles that describe the Agile mindset. Below we'll review each of them and explain how they can be adapted to product development.

1. **Satisfy customers through early and continuous delivery:** Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. Iteration and customer feedback are at the core of the Agile methodology. Product managers focus on delivering a working product, ideally an MVP, as soon as possible. The goal is to get feedback from real customers and use those insights to inform future releases and improve the product.

2. **Welcome changing requirements:** Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. Customer needs, the marketplace, and competition are constantly evolving. In this dynamic context, companies need to embrace change to remain competitive. While traditional approaches to product development view changing requirements as a threat, Agile product teams consider them an opportunity to increase the chance of delivering superior value to the customer.
3. **Deliver working products frequently:** Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. Agile product development focuses on shortening the gap between initial planning and delivery. Delivering working products frequently to customers allows teams to collect product feedback early, evaluate their course, and make changes if needed.
4. **Encourage regular communication:** Business people and developers must work together daily throughout the project. Agile encourages regular communication and collaboration between product teams and business stakeholders. Daily stand-ups and sprint planning, review, and retrospective meetings are just a few ways to bring everyone on the same page and align the product goals with business objectives.
5. **Build projects around motivated individuals:** Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. Another key aspect of the Agile methodology is to empower individuals and product teams through autonomy and trust. With the right environment, product management tools, and support, people will do their best work and produce quality outcomes. The most important thing is to make sure that everyone understands the product strategy and the underlying business motivation.
6. **Choose face-to-face conversation whenever possible:** The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. Due to the increasing number of distributed or remote teams, 'face-to-face conversation' can be interpreted as 'synchronous communication.' In the end, the goal behind this principle is to encourage real-time communication about the product to avoid misunderstandings and delays.
7. **Measure progress with working versions of the final product:** Working software is the primary measure of progress. Rather than releasing a final, perfect product, Agile product development involves delivering many versions of the product until arriving at a final version that meets all the requirements. The only way to measure progress is to have a working product that users can test out.
8. **Promote sustainable product development:** Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. Breaking down processes into small and achievable milestones helps product teams set realistic goals and clear expectations. The idea is to keep team members motivated, improve work-life balance, and avoid burnout and turnover.
9. **Pay attention to technical excellence and good design:** Continuous attention to technical excellence and good design enhances agility. The goal of Agile is to create not just a working product but also a high-quality solution. When teams pay attention to the technical quality and design after each iteration, they can fix issues early and avoid problems in the future.
10. **Keep it simple:** Simplicity—the art of maximizing the amount of work not done—is essential. This Agile principle invites product teams to focus their efforts on what is essential to create value. An example of

how product teams apply this principle to product development is the creation of a Minimum Viable Product—an early version of a product with just enough features to attract early customers and validate the product idea.

11. **Self-organizing teams generate the most value:** The best architectures, requirements, and designs emerge from self-organizing teams. Self-organizing teams decide as a group how they will accomplish their work, rather than waiting for a manager to tell them what to do. Giving teams a high-level direction and letting them organize and manage themselves boosts motivation and engagement and speeds up decision-making.
 12. **Regularly reflect on how to be more effective:** At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. Agile is about constant improvement, and this philosophy also applies to processes and teams. Most Agile frameworks, such as Scrum or Kanban, include a retrospective at the end of each sprint. This is a great opportunity to evaluate what went well and what didn't and understand how to improve the process in the future.
-

| Agile Metrics

Agile metrics help agile development teams and their management measure the development process, gauging productivity, work quality, predictability, and health of the team and products being developed. A key focus of agile metrics is on value delivered to customers—instead of measuring "what" or "how much" we are doing, we measure how it impacted a customer.

TYPES OF AGILE METRICS

There are three important families of agile metrics:

- **Lean metrics:** Focus on ensuring a flow of value from the organization to its customers and eliminating wasteful activities. Common metrics include lead time and cycle time.
- **Kanban metrics:** Focus on workflow, organizing and prioritizing work and getting it done. A common metric is a cumulative flow.
- **Scrum metrics:** Focus on the predictable delivery of working software to customers. Common metrics include the burndown chart and team velocity.

THE IMPORTANCE OF AGILE TESTING METRICS

Agile methodologies place a special emphasis on quality because the end goal is delivering working software to users—buggy or unusable software is not working software. Quality is also manifested in internal aspects that are not directly visible to customers, such as code quality, maintainability, and technical debt.

Agile testing metrics can help teams measure and visualize the effort spent in software quality, and to a certain extent, the results of this effort. For example, the escaped defects metric measures, across versions, sprints, or product lines, how many bugs were discovered in production whereas ideally bugs should be discovered and fixed during the development stage.

WHAT MAKES FOR A POWERFUL METRIC IN AN AGILE ENVIRONMENT?

Agile environments require metrics that are well understood by teams and can help learn and improve processes. Here are a few qualities that make a metric powerful, in the sense that it can help drive positive improvement in an agile team:

- **The metric is used by the team:** Agile metrics should not be imposed or measured by management, they should be used voluntarily by agile teams to learn and improve.
- **The metric is surrounded by conversation:** Metrics should not just be numbers, they should be the starting point of a conversation about process and roadblocks affecting the team.
- **The metric is part of a specific experiment:** Metrics should be used to answer a specific question about agile processes, not just measured for the sake of measurement.
- **The metric is used in tandem with other metrics:** Even a great metric, if used alone, might lead to tunnel vision, and incentivize teams to maximize that metric at the expense of all else. Using several metrics together provides a balanced picture of agile activity.
- **The metric is easy to calculate and understand:** Metrics that are overly complex or not fully understood, even if they provide good insights about a team's work, are not useful in guiding day-to-day activities.

10 POWERFUL AGILE METRICS

1. **Sprint Burndown:** The sprint burndown chart visualizes how many story points have been completed during the sprint and how many remain, and helps forecast if the sprint scope will be completed on time.
Why it is powerful: Makes it instantly clear how much value a sprint has already delivered and how close we are to completing our commitment to customers.
2. **Agile Velocity:** Velocity measures how many story points were completed by a team, on average, over the past few sprints. It can be used to predict the team's output in the upcoming sprints.
Why it is powerful: Velocity is powerful because it's a result metric—how much value was actually delivered to customers in a series of sprints. Be careful not to compare velocity across teams because story points and definition of done can vary between teams.
3. **Lead Time:** Lead time measures the total time from the moment a story enters the system (in the backlog), until it is completed as part of a sprint, or released to customers. It measures the total time for a requirement to be realized and start earning value—the speed of your value chain.
Why it is powerful: In a sense, lead time is more important than velocity because it measures the entire agile system from end to end. Reducing lead time means the entire development pipeline is becoming more efficient.
4. **Cycle Time:** As illustrated above, the cycle time is a subset of lead time - it measures the time for a task to go from "started" or "in progress" to "done". Normally, cycle times should be around half the sprint length. If cycle times are longer than a sprint, teams are not completing work they committed to.
Why it is powerful: A very simple metric that can raise a red flag when items within sprints across your entire system are not moving forward.
5. **Code Coverage:** Code coverage measures the percentage of your code which is covered by unit tests. It can be measured by the number of methods, statements, branches or conditions which are executed as

part of a unit test suite.

Why it is powerful: Code coverage can be run automatically as part of every build and gives a crude picture showing how much of the codebase has been tested. A low code coverage almost always indicates low code quality. However, a high coverage may not equal high quality, because there are other types of tests - such as UI or integration tests - which are not counted.

6. **Static Code Analysis:** While not exactly a metric, this is an automated process that can provide insights into code quality and clean code from simple errors redundancies. Code quality, while difficult to define and measure, is known to be a key contributor to software quality in general, and in particular, software maintainability.

Why it is powerful: Static code analysis provides a safe baseline for code quality. However, it is no substitute for human input into code quality, via manual code reviews, pair programming or other methods.

7. **Release Net Promoter Score:** Net Promoter Score (NPS), calculated for a software release, measures whether users would recommend the software to others, do nothing, or recommend against using it. It is an important gauge of customer satisfaction.

Why it is powerful: The ultimate test of agile development is providing value to a customer. If customers are recommending this new release to others, that is a clear indication of success. If not, you can use this as a warning metric and use other data to understand what's wrong.

8. **Cumulative Flow:** This is a kanban metric which shows the status of tasks - in a sprint, a release or across software teams. It can visualize bottlenecks in the process - a disproportionately large number of tasks in any of the workflow stages indicates a problem. For example, a big "bubble" in the chart in a verification or testing stage indicates this stage has insufficient resources.

Why it is powerful: As with the burndown chart, the power of this metric is in its visual simplicity you can grasp a process in one glance and immediately identify issues. Cumulative flow lets you catch problems in mid-process before they result in delayed delivery.

9. **Failed Deployments:** Measures the number of deployments (either to test, production environments, or both). Can help understand how solid environments are and whether teams are really building potentially shippable software.

Why it is powerful: Especially when applied to production environments, this metric can provide a clear indication that sprints or releases are production ready, or not.

10. **Escaped Defects:** The number of bugs discovered only after a build or release enters production. Escaped defects should ideally be zero. Measuring them across releases or teams provides a crude, but still highly relevant, a measure of deployed software quality.

Why it is powerful: Production bugs, especially if frequent, are a problem in the agile process. Just like in lean manufacturing, we should "stop the production line" and discover what's wrong.

THE MISSING METRIC: QUALITY INTELLIGENCE

We presented several powerful metrics that provide important insights into the agile process. However, there is no single metric as clear or powerful as the burndown chart or cycle time, which can tell us the most important thing: "how good" is the software being built by our developers.

A new category of tools called Software Quality Intelligence can provide this missing metric: a clear view of software quality. SeaLights is a platform which combines data about code changes, production uses and test execution, to provide the following quality metrics:

- **Test gap analytics:** Identifying areas where the code was recently changed or executed in production but is untested. Test gaps are the best place to invest resources to improve quality.
- **Quality trend intelligence:** Showing which parts of a system are improving in quality coverage, and which are getting worse-meaning more testing time should be invested.
- **Release quality analytics:** SeaLights performs real-time analytics on hundreds of thousands of test executions, code changes, builds and production events to assess the readiness of a release.

Feature-Driven Development (FDD)

FDD stands for Feature-Driven Development. It is an agile iterative and incremental model that focuses on progressing the features of the developing software. The main motive of feature-driven development is to provide timely updated and working software to the client. In FDD, reporting and progress tracking is necessary at all levels.

HISTORY

FDD was first applied in the year 1997 on a real-world application by Jeff De Luca for large software development with specific needs of 15-month and 50 persons and published as a discussion in book *Java Modeling in Color with UML* in the year 1999.

FDD LIFECYCLE

- Build overall model
- Build feature list
- Plan by feature
- Design by feature
- Build by feature

CHARACTERISTICS OF FDD

- **Short iterative:** FDD lifecycle works in simple and short iterations to efficiently finish the work on time and gives good pace for large projects.
- **Customer focused:** This agile practice is totally based on inspection of each feature by client and then pushed to main build code.
- **Structured and feature focused:** Initial activities in lifecycle builds the domain model and features list in the beginning of timeline and more than 70% of efforts are given to last 2 activities.
- **Frequent releases:** Feature-driven development provides continuous releases of features in the software and retaining continuous success of the project.

ADVANTAGES OF FDD

- Reporting at all levels leads to easier progress tracking.
- FDD provides continuous success for larger size of teams and projects.
- Reduction in risks is observed as whole model and design is build in smaller segments.
- FDD provides greater accuracy in cost estimation of the project due to feature segmentation.

DISADVANTAGES OF FDD

- This agile practice is not good for smaller projects.
- There is high dependency on lead programmers, designers and mentors.
- There is lack of documentation which can create an issue afterwards.

Financial and Production Metrics in FDD

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. The QA organization can incorporate models learned from past projects to develop the method for coming projects.

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. The process supports organizations to

avoid and decrease chances. It's rare for an application to be 100% error-free, but a committed QA unit should strive to eliminate or restrict the most problematic errors. 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 to know the effects as early as practicable. Within each sprint, QA technicians test and retest the product with every new feature added. This enables them to confirm that the new features were performed as required and to find any problems that may have been introduced. 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. It merely has a knowledge of what it should do from the user's view. White-box testing allows the QA engineer to generate a more profound knowledge of the system's internals. Armed with this information, the QA engineer can start testing much quicker. Within an Agile QA process, the search engineers want this new level of practice understanding to verify features as soon as they are developed. White-box testing enables QA units to predict potential failure conditions and produce better test situations. Knowing how the system works to ensure they have examined all possible input situations. It can also improve to the identification of potential security difficulties. Maybe most notable: white-box testing supports close collaboration between development and QA.
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 and continue to work. At the same time, QA managers and engineers concentrate on testing recently delivered features. Being able to reproduce tests reliably will save up resources for exploratory testing. Automation will give your development team the courage to make adjustments to the system with the knowledge that any problems will be classified quickly, and can be fixed before delivery to the QA team.

All that being stated, it is essential to be careful of over-automating. Your team should prioritize test cases and later decide which of them should be automated. Conditions in which the data might vary or where a situation isn't consistently reproducible may not benefit from automation because the issues can cause false breakdowns. Implementing automation costs more upfront, but protects money in the long run by improving efficiency between development and QA teams.
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. When you are intimate with who will be practicing the real end-product, you can better prioritize the QA method to save time and money.
6. **Teamwork Makes the Dream Work:** Behind each high-quality product, there is a unit of professionals that regularly work to keep the high measure of quality upheld by the organization. Although each team running on the project must take accountability for assuring excellence, the primary responsibility for quality rests with the QA team. The roles and responsibilities of the QA team revolve around what the customer wants the system to do and can determine the client's comfort with the order. Managing the Agile QA process, engineers are the super-sleuths who root out obstacles and support the team to deliver high-quality results and secure client trust, company reliability, and reliable product performance.

| Test Driven Development (TDD)

The TDD approach derives its roots from the Agile manifesto principles and Extreme programming. As the name suggests, the test process drives software development. Moreover, it's a structuring practice that enables developers and testers to obtain optimized code that proves to be resilient in the long term.

In TDD, developers start creating small test cases for every feature based on their initial understanding. The primary intention of this technique is to modify or write new code only if the tests fail. This prevents duplication of test scripts.

THREE PHASES OF TEST DRIVEN DEVELOPMENT

- **Create precise tests:** Developers need to create precise unit tests to verify the functionality of specific features. They must ensure that the test compiles so that it can execute. In most cases, the test is bound to fail. This is a meaningful failure as developers are creating compact tests based on their assumptions of how the feature will behave.
- **Correcting the Code:** Once a test fails, developers need to make the minimal changes required to correct the code so that it can run successfully when re-executed.
- **Refactor the Code:** Once the test runs successfully, check for redundancy or any possible code optimizations to enhance overall performance. Ensure that refactoring does not affect the external behavior of the program.

BENEFITS OF TEST DRIVEN DEVELOPMENT (TDD)

1. Fosters the creation of optimized code.
2. Helps developers better analyze and understand client requirements and request clarity when they are not adequately defined.
3. The addition and testing of new functionalities become much easier in the latter stages of development.
4. Test coverage under TDD is much higher compared to the conventional development models. This is because the TDD focuses on creating tests for each functionality right from the beginning.
5. Enhances the productivity of the developer and leads to the development of a codebase that is flexible and easy to maintain.

FRAMEWORKS FOR TEST DRIVEN DEVELOPMENT

Based on unique programming languages, there are multiple frameworks that support test driven development. Listed below are a few popular ones:

- **CsUnit and NUnit** – Both are open source unit testing frameworks for .NET projects.
- **PyUnit and DocTest:** Popular Unit testing framework for Python.
- **JUnit:** Widely used unit testing tool for Java.
- **TestNG:** Another popular Java testing framework. This framework overcomes the limitations of Junit.
- **RSpec:** A testing framework for Ruby projects.

Agile Approach in Global Software Development

Introduction of agile software development has brought considerable changes to software engineering field. Nowadays many organizations and software developers are using agile methods to deliver the efficient and high-quality product in the shortest time. The various types of agile methodologies being used by these organizations are extreme programming (XP), scrum, lean software development, feature-driven development (FDD), DSDM and crystal methodologies.

The characteristics of Agile software development processes can be summarized as iterative and incremental development, people-oriented development, customer collaboration, light and fast development cycles, adaptability throughout software development life cycle, time-bound, parsimony and fast delivery. In year 2001, several practitioners met to discuss if there was anything in common among the various agile methods and then they created agile manifesto, in which the items which were considered valuable by Agile Software Development were revealed and great attention has been provided by research community to Agile Software Development.

Agile Software Development methods focus on the four values of the Agile Manifesto:

- Individuals and interactions over Processes and tools
- Working software over Comprehensive documentation
- Customer collaboration over Contract negotiation
- Responding to change over Following a plan

The first value refers that role of humans and team interactions should be considered more valuable than defined processes and development tools. The second value implies that main objective of software team should be to deliver the working software at frequent intervals rather than large amount of documentation. One can observe from third value importance should be given to the relationship between developers and customers over strict contracts. The fourth value refers the members in the team should be prepared to make changes whenever required during the software development life cycle rather than following a plan.

Agile methods helps to improve communication and reduce coordination problems in software products. These agile methods are opposite in nature with respect to traditional approaches. When following the traditional approach software is developed in a sequential manner. Requirements are gathered from clients, there is little interaction between clients and members of the development team. For designing and coding different roles are assigned to developers and finally software is tested and integrated. In doing all these, a large amount of documentation is produced that contains information about developmental processes. Agile Software Development approach aims at overcoming these shortcomings of the traditional approaches.