Best of Study

Agile Technologies (21CS641)

Module – 1

Why Agile? : Understanding Success, Beyond Deadlines, The Importance of Organizational Success, Enter Agility, How to Be Agile?: Agile Methods, Don't Make Your Own Method, The Road to Mastery, Find a Mentor. The Genesis of Agile, Introduction and background, Agile Manifesto, and Principles, Simple Design, User Stories, Agile Testing, Agile Tools.

Why Agile?

Agile development is increasingly popular, with many leading companies such as Google, Yahoo, Symantec, and Microsoft adopting its methodologies. This trend highlights the widespread recognition of agile practices in the tech industry.

However, it's important to note that adopting agile development solely to boost productivity may not be the most effective approach. The true benefits of agile development, including the ability to release software more frequently, stem from fundamentally changing the way work is approached rather than simply increasing the speed of work.

Understanding Success

Traditionally, success in project management is defined by the ability to deliver projects on time, within budget, and according to specified requirements. These classic definitions have long been the standard metrics by which the success of a project is measured.

Here are some classic definitions of success:

1. On Time: Completing the project by the deadline.

2. On Budget: Finishing the project without exceeding the budget.

3. According to Specification: Meeting all the requirements and standards set for the project.
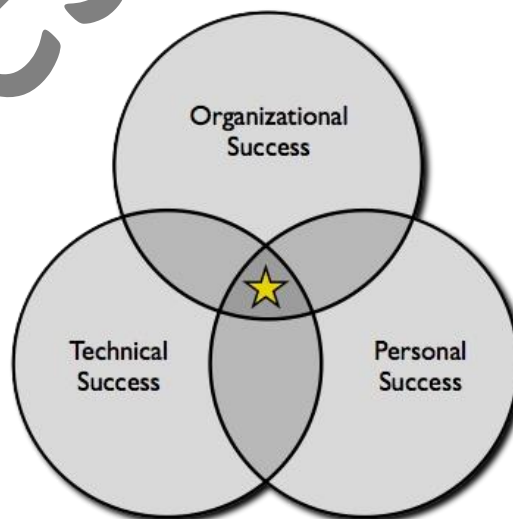
Best of Study

Beyond Deadlines

Success is more than just meeting deadlines. But what else matters?

When we were kids, we loved playing around and enjoyed the challenge of programming. Getting a program to work felt like a big victory. Even if a program didn't work, it was still a success because we had fun writing it. Back then, success was all about personal satisfaction.

As I gained experience, my software became more complex, and I often lost track of how it worked. Sometimes, I had to abandon programs before they were finished. I began to believe that maintainability was the key to success. This belief was confirmed when I started working with teams of other programmers. I took pride in writing elegant, maintainable code. For me, success meant achieving technical excellence.

However, even with good code, some projects still failed. I realized that my project teams were part of a larger system involving many people. My projects needed to satisfy those people, especially the ones paying for our work. The software had to provide more value than it cost. Success, therefore, meant delivering value to the organization.

These definitions of success are not incompatible. All three types of success are important. Without personal success, it's hard to stay motivated. Without technical success, the code will eventually fail. Without organizational success, your team may not be needed in the company anymore.

Best of Study

The Importance of Organizational Success

Software teams often focus more on technical and personal successes, neglecting organizational success. However, even if you aren't prioritizing organizational success, the broader organization is evaluating your team at this level. Senior management and executives are more concerned with results than with how elegant, maintainable, or popular your software is. They want a return on their investment in your project. If you don't deliver this kind of success, they will take action to ensure you do.

Unfortunately, senior managers often don't have the time or perspective to find precise solutions for each project. They use broad measures rather than detailed, nuanced approaches. They expect project teams to handle the fine details.

When managers are dissatisfied with your team's results, they resort to drastic measures. The most obvious targets are costs. There are two straightforward ways to cut costs: set aggressive deadlines to reduce development time, or outsource the work to a country with lower labour costs, or both.

These methods are blunt instruments. Aggressive deadlines often end up increasing timelines instead of reducing them, and offshoring has hidden costs.

Therefore, it's time for your team to take responsibility for its success—not just personal or technical success, but organizational success as well.

What Do Organizations Value?

While some projects directly generate revenue, organizational value extends beyond just sales. Projects can provide value in various ways that aren't always measured in dollars and cents.

Beyond revenue and cost savings, sources of value include:

Competitive Differentiation: Standing out from competitors.

Brand Projection: Strengthening the brand's image.

Enhanced Customer Loyalty: Increasing customer retention and loyalty.

Satisfying Regulatory Requirements: Meeting legal and regulatory standards.

Best of Study

Original Research: Conducting innovative research and development.

Strategic Information: Gaining valuable insights and data for future decisions.

Enter Agility

Can agile development make you more successful? It might. Agile development focuses on achieving personal, technical, and organizational successes. If you're struggling in any of these areas, agile development might help.

Organizational Success

Agile methods promote organizational success by delivering value and reducing costs, which increases return on investment. Agile also sets expectations early in the project, allowing you to cancel projects that won't succeed before too much money is spent.

Agile teams increase value by involving business experts and focusing on the core value the project provides to the organization. They release the most valuable features first and frequently update the software, increasing its value. When business needs change or new information emerges, agile teams adapt quickly.

Agile teams also reduce costs by maintaining technical excellence, producing only a few bugs per month. They cut waste by cancelling unpromising projects early and replacing expensive practices with simpler ones.

Agile teams communicate effectively and continue to make progress even when key people are unavailable. They regularly review their processes and continually improve their code, making the software easier to maintain and enhance over time.

Technical Success

Extreme Programming (XP), an agile method, excels in achieving technical successes. XP programmers collaborate closely, ensuring important details are managed effectively and every piece of code is reviewed by at least two people.

Best of Study

Continuous integration of code allows the team to release software whenever it aligns with business needs. The team prioritizes completing each feature entirely before moving on to the next, which prevents unexpected delays and allows for flexible changes in direction.

Extreme Programming also incorporates advanced technical practices that promote excellence. Test-driven development ensures code performs as intended from the outset. XP teams also create simple, adaptable designs that can easily accommodate changes in plans.

Personal Success

Personal success varies for each individual. While agile development may not fulfil every personal goal, it typically offers numerous benefits that appeal to various stakeholders:

Executives and Senior Management: Appreciate the team's focus on delivering a strong return on investment and ensuring the longevity of the software.

Users, stakeholders, domain experts, and product managers will value:

Influence on Development Direction: Their ability to shape the course of software development.

Focus on Delivering Value: The team's dedication to delivering software that is useful and valuable.

Increased Delivery Frequency: More frequent releases of software.

Project and product managers will appreciate:

Adaptability: The team's capability to pivot in response to changing business needs.

Commitment and Delivery: Meeting commitments reliably.

Stakeholder Satisfaction: Improved satisfaction among stakeholders.

Developers will find value in:

Improved Technical Quality: Enhancements in the technical quality of the software.

Best of Study

Estimation and Scheduling Influence: Having more influence over project estimates and schedules.

Autonomy: Greater autonomy within the team structure.

Testers will benefit from:

Integration and Involvement: Being integral members of the team.

Quality Influence: Their ability to impact quality throughout all project stages.

Challenging Work: Engaging in less repetitive and more stimulating tasks.

How to Be Agile?

What does it mean to "be agile"? It's not as straightforward as following a specific process. Agile development is a philosophy—a mindset about how software development should be approached.

The foundation of this mindset is the Agile Manifesto, which outlines 4 core values and 12 principles. These guide the agile approach and emphasize flexibility, collaboration, and responsiveness to change.

To truly "be agile," you must embody these values and principles in your software development practices. This involves:

 Embracing the Agile Values: Prioritizing individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan.

  Applying Agile Principles: Such as satisfying customers through early and continuous delivery of valuable software, welcoming changing requirements even late in development, delivering working software frequently (weeks rather than months), and promoting sustainable development that keeps a constant pace indefinitely.

By adhering to these values and principles, teams can adopt an agile mindset that fosters innovation, adaptability, and continuous improvement in their development processes.

Best of Study

Agile Methods

A method or process is simply a way of working—every action follows some sort of process. Some processes are explicit and documented, like assembling furniture, while others are informal and improvised, like cleaning a house.

Agile methods are specific processes designed to align with the agile philosophy. Examples of agile methods include Extreme Programming (XP) and Scrum.

These agile methods are composed of individual components known as practices. Practices encompass activities such as using version control systems, establishing coding standards, and conducting weekly demonstrations for stakeholders. Each practice contributes to the overall agility of the development process by promoting collaboration, adaptability, and continuous improvement.

Don't Make Your Own Method

Creating your own agile method by combining practices from different agile methods might seem appealing initially. After all, there are many good agile practices to choose from.

However, if you're new to agile development, designing a completely new agile method is not advisable. Just as programming involves more than writing code, agile development goes beyond implementing practices.

These practices are rooted in agile principles. Without a solid understanding of these principles or without mastering agile development, you might not select the most appropriate practices.

Each project and context is unique, so it's beneficial to tailor your agile approach to fit your specific needs. Instead of starting from scratch, begin with a well-established agile method and refine it iteratively. Implement it in your context, observe what works well and what doesn't, make informed adjustments, and repeat the process.

This iterative refinement approach allows you to gradually adapt the agile method to better suit your team and project dynamics, ensuring you derive maximum benefit from agile practices while minimizing risks associated with inexperienced customization.

Best of Study

The Road to Mastery

Mastering agile development requires real-world experience with a specific, well-defined agile method. Extreme Programming (XP) is recommended for this purpose because:

Comprehensive Approach: XP is the most complete agile method, emphasizing both technical practices and teamwork.

Well-Documented: XP has extensive documentation, including explanations, experience reports, and critiques, making its capabilities and limitations well understood.

To master agile development or use XP to enhance success, follow these steps:

1**. Identify Your Goals: Understand** why you want to use agile development. Will it make your team and organization more successful? How?

2. **Evaluate Suitability:** Determine if this approach will work for your team.

3. **Adopt XP Practices:** Implement as many of XP's practices as possible. These practices are self-reinforcing and work best when used together.

4. **Follow Practices Consistently:** Adhere to XP practices rigorously and consistently.

5. **Experiment and Improve:** After several months of practicing XP correctly, start experimenting with changes. Observe the outcomes and make further improvements.

By following these steps, you can achieve mastery in agile development and use XP to drive success in your projects and organization.

Find a Mentor

Adapting XP to your situation will bring challenges and problems. To navigate these effectively, you need a mentor—an outside expert who has mastered agile development.

Note:

➔ If you can have an expert coach your team directly, that's ideal.
➔ Even master coaches benefit from an outside perspective when they face problems.

Best of Study

The Genesis of Agile: Introduction and Background

In the early 1990s, as PC computing began to rise in organizations, software development faced a significant hurdle known as the "application development crisis." Organizations estimated it would take three years from identifying a business need to having an application in production. However, business moves faster than that, even back then.

Waiting three years to solve business problems meant that requirements, systems, and even the business itself could change drastically during that time. Consequently, many projects were cancelled halfway through, and those that were completed often failed to meet the evolving needs of the business.

Before agile methodologies, industries like software, aerospace, and manufacturing followed the waterfall approach. This method involved:

1. Setting requirements and the project scope.

2. Designing the product based on predefined requirements.

3. Building the product.

4. Testing the product.

5. Identifying and fixing problems during testing.

6. Launching the finished product.

The waterfall approach required sticking to a fixed plan from the beginning, making it difficult to adapt to necessary changes along the way. This rigidity created many issues since the plan often became inconvenient as new information emerged. Moreover, the waterfall method focused on delivering a finished product to the market, even if it took years to complete.

This approach caused significant problems for both developers and customers. The lengthy development time meant that the nature of the original problem often changed, and by the time the product was launched, it was often outdated. These delays led to the delivery of unfinished products that no longer fit the market needs, highlighting the need for a more flexible and responsive development method.

Best of Study

Industries Were Frustrated with the Waterfall Approach

Many industries began to express their frustration with the waterfall approach. During the 1990s, a large number of software development teams started looking for a new method. Jon Kern was one of these frustrated thought leaders who became increasingly active in seeking something more "timely and responsive."

Agile Manifesto and Principles

In 2001, a group of software experts, dissatisfied with traditional methods, convened and crafted a manifesto that would redefine software development practices.

Agile Manifesto Overview:

- Created by 17 developers in Snowbird, Utah, the Agile Manifesto outlines 4 values and 12 principles that revolutionized software development.

- It serves as a transformative guide for software developers, emphasizing agility and responsiveness.

4 Values of the Agile Manifesto:

1. Individuals and Interactions over Processes and Tools: Emphasizes effective communication and collaboration within teams.

2. Working Software over Comprehensive Documentation: Prioritizes delivering functional software as the key indicator of progress.

3. Customer Collaboration over Contract Negotiation: Advocates for active customer and stakeholder involvement throughout development.

4. Responding to Change over Following a Plan: Embraces flexibility and the ability to adapt to changing requirements, even late in the development process.

These values guide agile development practices, promoting adaptability, collaboration, and customer satisfaction as core principles.

Best of Study

Principles of the Agile Manifesto for Agile Software Development

The Agile Manifesto outlines 12 principles that guide agile software development practices:

1. Customer Satisfaction through Early and Continuous Delivery: Focuses on providing early and consistent delivery of valuable software to ensure customer satisfaction throughout the development process.

2. Welcome Changing Requirements, Even Late in Development: Embraces changes in requirements late in development to harness the customer's competitive advantage and adapt to evolving needs.

3. Deliver Working Software Frequently: Encourages delivering functional software in short, iterative cycles to obtain rapid feedback and respond effectively to changes.

4. Collaboration between Business Stakeholders and Developers: Advocates daily communication and collaboration between business stakeholders and developers to understand and prioritize requirements effectively.

5. Build Projects around Motivated Individuals: Promotes empowering and supporting motivated individuals, recognizing they are key to project success and high-quality outcomes.

6. Face-to-Face Communication is the Most Effective: Values direct and frequent face-to-face communication as the most effective way to convey information and minimize misunderstandings.

7. Working Software is the Primary Measure of Progress: Emphasizes delivering working software as the primary metric of progress, ensuring continuous delivery of valuable features and early feedback.

8. Maintain a Sustainable Pace of Work: Promotes sustainable development practices where stakeholders, developers, and users maintain a steady and productive pace indefinitely to prevent burnout.

9. Continuous Attention to Technical Excellence and Good Design: Highlights the importance of maintaining high standards of technical craftsmanship and design to ensure software maintainability and adaptability.

10. Simplicity—the Art of Maximizing the Amount of Work Not Done: Advocates focusing on the most valuable tasks and features while avoiding unnecessary complexity to maximize efficiency and effectiveness.

11. Self-Organizing Teams: Empowers teams to make decisions and organize themselves to optimize efficiency, creativity, and innovation in architecture, requirements, and design.

12. Regular Reflection on Team Effectiveness: Encourages teams to regularly reflect on their processes and interactions to enhance effectiveness and adaptability, fostering continuous improvement.

These principles provide a framework for agile teams to prioritize customer satisfaction, flexibility, collaboration, and continuous improvement in software development projects.

Simple Design

In agile software development, the concept of simple design revolves around creating systems that are adaptable to changing requirements and responsive to customer needs. The primary aim is to develop a functional system quickly and continuously refine it based on feedback from stakeholders.

Agile methodologies stress iterative and incremental development, where projects are divided into short cycles called sprints. Each sprint focuses on delivering specific features or functionality. Throughout this process, there is ongoing collaboration between customers and developers, ensuring that the system aligns closely with end-user requirements.

The agile approach also emphasizes flexibility, allowing teams to quickly adjust to evolving needs and incorporate feedback into the development process without significant delays. This iterative approach not only enhances adaptability but also minimizes risks by delivering software in manageable increments, which reduces the chances of large-scale project failures.

Overall, agile software development prioritizes simplicity in design to facilitate rapid adaptation, continuous improvement, and effective alignment with stakeholder expectations throughout the development lifecycle.

Best of Study

Phases of Agile Model in Designing System
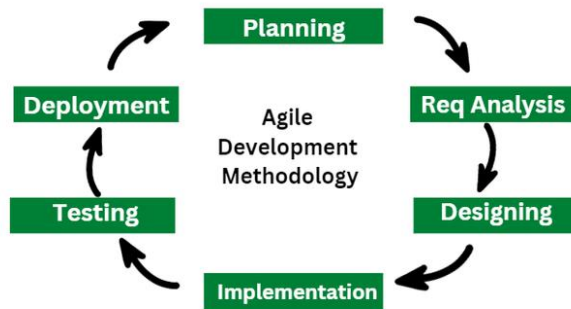
1. Planning: The team sets project goals, defines scope, identifies stakeholders, and outlines a high-level roadmap. They establish the agile methodologies and practices to be followed.

2. Requirement Analysis: Collaborating with stakeholders, the team gathers and prioritizes system requirements. They assess risks and devise strategies to mitigate them.

3. Designing: Detailed system interfaces and components are designed, including prototypes, architecture, and design patterns. The focus is on creating a scalable and cohesive system framework.

4. Implementation: The team builds system components and integrates them into the design. Each sprint concludes with deliverable software. Adjustments are made to the backlog to ensure alignment with the design.

5. Testing: Components undergo rigorous testing to verify compliance with specifications. Issues identified are addressed to ensure high-quality system readiness.

6. Deployment: The system is delivered to end users in collaboration with stakeholders. Support and training are provided as needed to ensure effective utilization.

These phases highlight the agile model's iterative and adaptive nature, enabling continuous adjustments throughout the project lifecycle. This approach ensures that the system meets user needs while maintaining project timelines and budgets.

Best of Study

User Stories in Agile

User stories are fundamental in agile software development, serving as concise descriptions of features or functionalities from the user's perspective. Here are the key characteristics of user stories:

1. User-centric: They focus on what the user wants to achieve with the software.

2. Simple: User stories are brief and straightforward descriptions of features.

3. Independent: Each user story stands alone and does not rely on others.

4. Negotiable: They are open to discussion and refinement based on stakeholder feedback.

5. Valuable: User stories deliver value to both the user and the business.

6. Estimable: They can be estimated in terms of time and effort for implementation.

7. Testable: User stories can be tested to ensure they meet user needs.

8. Prioritized: They are ranked by importance to users and business goals.

9. Iterative: User stories are developed incrementally, allowing for feedback and changes.

10. Consistent: They follow a standard format for clarity and ease of understanding.

11. Contextual: User stories provide context to the development team about user needs and goals.

12. Acceptance criteria: They include specific criteria defining when the story is complete.

13. Role-based: Written from a specific user's perspective to ensure relevance.

14. Traceable: User stories are linked to features in the software, making it easy to trace back to user needs.

In agile development, user stories help teams plan, prioritize, estimate effort, and track progress. They ensure software meets user needs and adds value to the business, playing a crucial role in incremental development by defining requirements clearly and simply.

Best of Study

Pattern of User Story:

"As a [type of user], I want [an action], so that [some reason]."

Example:

"As a project manager of a construction team, I want our team-messaging app to include file sharing and information updates so that my team can collaborate and communicate with each other in real-time, speeding up the development and completion of the construction project."

Agile Testing

Agile Testing aligns with the principles of agile software development, integrating testing throughout the entire development lifecycle. Unlike traditional methods, agile testing involves all project team members, including testers and domain experts, right from the requirements phase through to design, coding, and test case generation. It's a collaborative approach where testers contribute to software development by ensuring customer requirements are met and enhancing design quality.
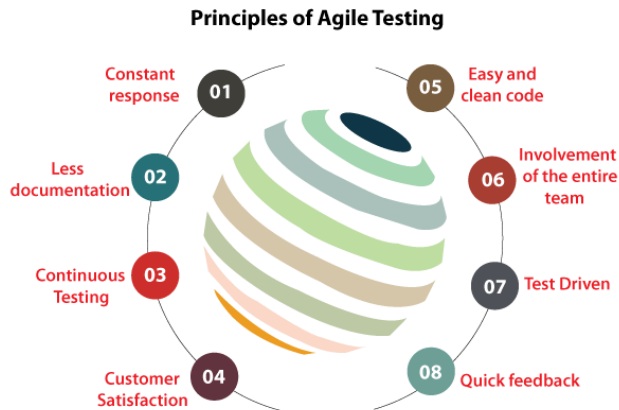
Key aspects of Agile Testing include:

- Integration throughout the Development Life Cycle: Agile testing is not a separate phase but is conducted concurrently with development activities in iterations or sprints.

- Delivery-driven approach: It focuses on delivering functional products in shorter time frames, promoting quicker feedback and adaptation to changes.

- Informal and dynamic: Agile testing is flexible and adapts to evolving requirements, ensuring customer satisfaction remains a primary goal.

In essence, agile testing supports continuous improvement and customer-centric development by embedding testing practices within agile methodologies.

Best of Study

Principles of Agile Testing

Principles of Agile Testing

01 Constant response
02 Less documentation
03 Continuous Testing
04 Customer Satisfaction
05 Easy and clean code
06 Involvement of the entire team
07 Test Driven
08 Quick feedback

Here's a refined version of the points on Agile Testing:

1. Continuous Feedback

Agile testing provides ongoing feedback, ensuring the product meets evolving business needs throughout development.

2. Minimal Documentation

Agile testing minimizes documentation by using reusable specifications and checklists, focusing more on testing rather than secondary information.

3. Iterative Testing

Agile test engineers conduct continuous testing to ensure ongoing product improvement.

4. Customer Satisfaction

Customer satisfaction is prioritized throughout the development process, with flexibility for customers to modify requirements and tests as needed.

5. Clean Code

Agile testing resolves bugs promptly within the same iteration, resulting in cleaner and more maintainable code.

Best of Study

6. Team Collaboration

In Agile testing, business analysts and developers are involved in testing applications, fostering a collaborative approach beyond the testing team.

7. Test-Driven Approach

Agile testing integrates testing early in the development cycle, reducing development time compared to traditional methods where testing follows development.

8. Rapid Feedback

Continuous involvement of the business team in Agile testing ensures quick feedback cycles, enhancing responsiveness to development work.

*****************************END*****************************