Agile Technologies

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

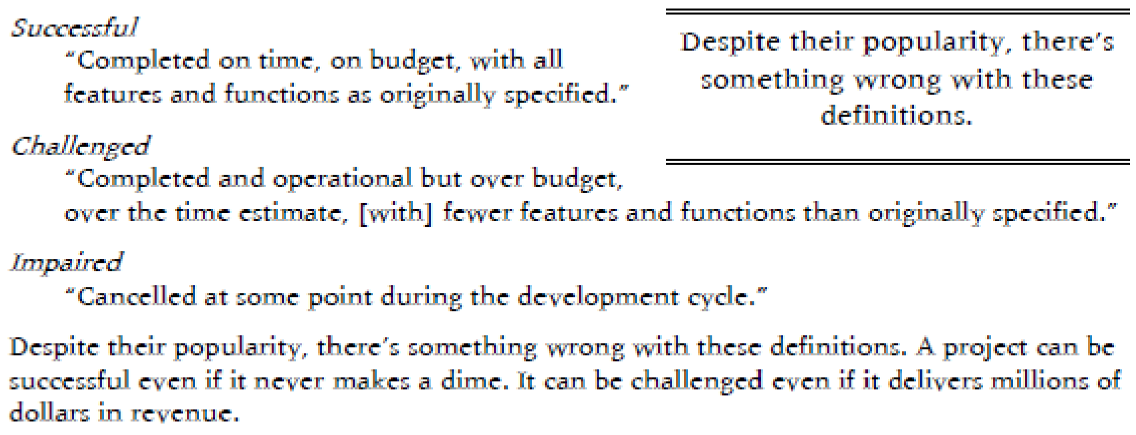
Why Agile?

Agile development is popular. All the cool kids are doing it: Google, Yahoo, Symantec, Microsoft and the list goes on.

In fact, I don't recommend adopting agile development solely to increase productivity. Its benefits—even the ability to release software more frequently—come from working differently, not from working faster.

Understanding Success

The traditional idea of success is delivery on time, on budget, and according to specification.
Here we provide some classic definitions:



Successful
"Completed on time, on budget, with all features and functions as originally specified."

Challenged
"Completed and operational but over budget, over the time estimate, [with] fewer features and functions than originally specified."

Impaired
"Cancelled at some point during the development cycle."

Despite their popularity, there's something wrong with these definitions.

Despite their popularity, there's something wrong with these definitions. A project can be successful even if it never makes a dime. It can be challenged even if it delivers millions of dollars in revenue.

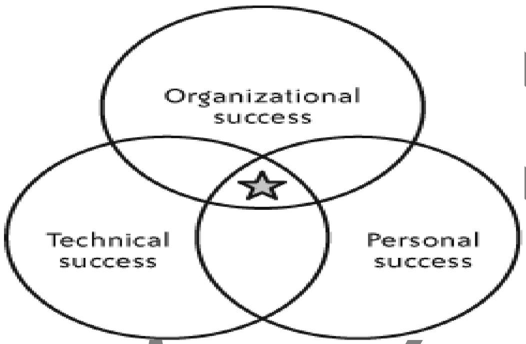
Beyond Deadlines

There has to be more to success than meeting deadlines... but what?
When we were a kids, we was happy just to play around. We loved the challenge of programming. When we got a program to work, it was a major victory. Back then, even a program that didn't work was a success of some sort, as long as we had fun writing it. **My definition of success centered on personal rewards**.

As I gained experience, my software became more complicated and I often lost track of how it worked. I had to abandon some programs before they were finished. I began to believe that **maintainability** was the key to success—an idea that was confirmed as I entered the workforce and began working with teams of other programmers. I prided myself on producing **elegant, maintainable code**. **Success meant technical excellence.**

Despite good code, some projects flopped.. I came to realize that my project teams were part of a larger ecosystem involving dozens, hundreds, or even thousands of people. My projects needed to satisfy those people ... particularly the ones signing my paycheck. In fact, for the people funding the work, the value of the software had to exceed its cost. **Success meant delivering value to the organization.**

These definitions aren't incompatible. All three types of success are important (see Figure below). Without personal success, you'll have trouble motivating yourself and employees. Without technical success, your source code will eventually collapse under its own weight. Without organizational success, your team may find that they're no longer wanted in the company.



The Importance of Organizational Success

Organizational success is often neglected by software teams in favour of the more easily achieved technical and personal successes. Rest assured, however, that even if you're not taking responsibility for organizational success, the broader organization is judging your team at this level. Senior management and executives aren't likely to care if your software is elegant, maintainable, or even beloved by its users; they care about results. That's their return on investment in your project. If you don't achieve this sort of success, they'll take steps to ensure that you do.

Unfortunately, senior managers don't usually have the time or perspective to apply a clear solution to each project. They wield swords, not scalpels. They rightly expect their project teams to take care of fine details.

When managers are unhappy with your team's results, the swords come out. Costs are the most obvious target. There are two easy ways to cut them: **set aggressive**

deadlines to reduce development time, or ship the work to a country with a lower cost of labour or both.

These are clumsy techniques. Aggressive deadlines end up increasing schedules rather than reducing them and off- Shoring has hidden costs.

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

WHAT DO ORGANIZATIONS VALUE?

Although some projects' value comes directly from sales, there's more to organizational value than revenue. Projects provide value in many ways, and you can't always measure that value in dollars and cents.

Aside from revenue and cost savings, sources of value include:*

• Competitive differentiation
• Brand projection
• Enhanced customer loyalty
• Satisfying regulatory requirements
• Original research
• Strategic information

Enter Agility

Will agile development help you be more successful? It might. Agile development **focuses on achieving personal, technical, and organizational successes**. If you're having trouble with any of these areas, agile development might help.

Organizational Success

Agile methods achieve organizational successes by focusing on delivering value and decreasing costs. This directly translates to increased return on investment. Agile methods also set expectations early in the project, so if your project won't be an organizational success, you'll find out early enough to cancel it before your organization has spent much money.

Specifically, agile teams increase value by including business experts and by focusing development efforts on the core value that the project provides for the organization.

Agile projects release their most valuable features first and release new versions frequently, which dramatically increase value. When business needs change or when new information is discovered, agile teams change direction to match.

Agile teams decrease costs as well. They do this partly by technical excellence; the best agile projects generate only a few bugs per month. They also eliminate waste by cancelling bad projects early and replacing expensive development practices with simpler ones.

Agile teams communicate quickly and accurately, and they make progress even when key individuals are unavailable. They regularly review their process and continually improve their code, making the software easier to maintain and enhance over time.

Technical Success

Extreme Programming, the agile method, is particularly adept at achieving technical successes. XP programmers work together, which helps them keep track of the important details necessary for great work and ensures that at least two people review every piece of code.

Programmers continuously integrate their code, which enables the team to release the software whenever it makes business sense. The whole team focuses on finishing each feature completely before starting the next, which prevents unexpected delays before release and allows the team to change direction at will.

In addition to the structure of development, Extreme Programming includes advanced technical practices that lead to technical excellence. The most well-known practice is **test driven development**, which helps programmers write code that does exactly what they think it will. XP teams also create simple, ever-evolving designs that are easy to modify when plans change.

Personal Success

Personal success is, well, personal. Agile development may not satisfy all of your requirements for personal success. However, once you get used to it, you'll probably find a lot to like about it, no matter who you are:

Executives and senior management
They will appreciate the team's focus on providing a solid return on investment and the Software's longevity.

Users, stakeholders, domain experts, and product managers

They will appreciate their ability to influence the direction of software development, the team's focus on delivering useful and valuable software, and increased delivery frequency.

Project and product managers

They will appreciate their ability to change direction as business needs change, the team's ability to make and meet commitments, and improved stakeholder satisfaction.

Developers

They will appreciate their improved quality of life resulting from increased technical quality, greater influence over estimates and schedules, and team autonomy.

Testers

They will appreciate their integration as first-class members of the team, their ability to influence quality at all stages of the project, and more challenging, less repetitious work.

How to Be Agile

What does it mean to "be agile"?

The answer is more complicated than you might think. Agile development isn't a specific process you can follow.

Agile development is a philosophy. It's a way of thinking about software development.

The systematic description of this way of thinking is the Agile Manifesto, a collection of 4 values and 12 principles
To "be agile," you need to put the agile values and principles into practice.

Agile Methods

A method, or process, is a way of working. Whenever you do something, you're following a process. Some processes are written, as when assembling a piece of furniture; others are ad hoc and informal, as when I clean my house.

Agile methods are processes that support the agile philosophy. Examples include Extreme Programming and Scrum.

Agile methods consist of individual elements called **practices**. Practices include using version control, setting coding standards, and giving weekly demos to your stakeholders.

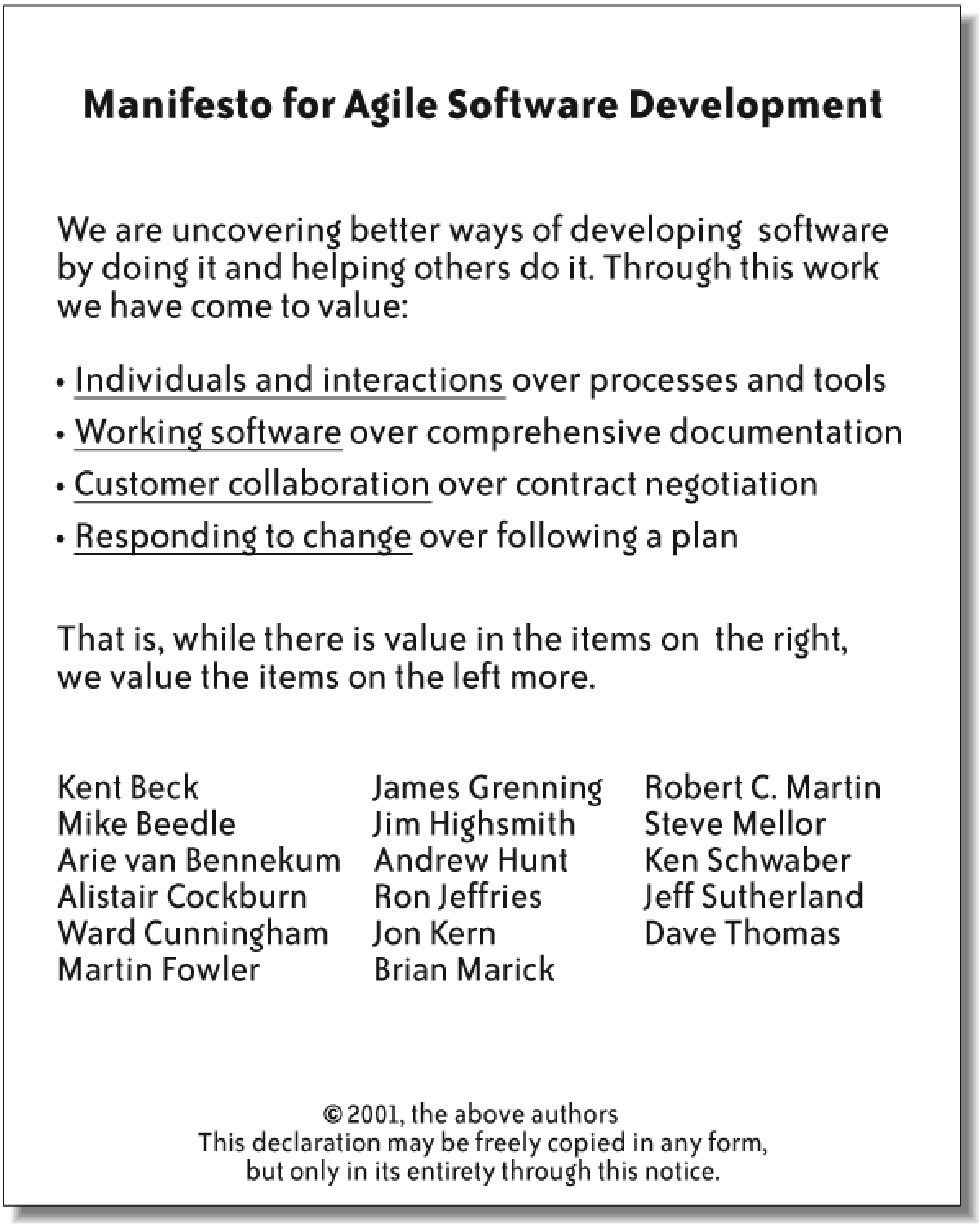
Don't Make Your Own Method

You might want to create your own agile method by mixing together practices from various agile methods. At first glance, this doesn't seem too hard. There are some of good agile practices to choose from.

However, creating a brand-new agile method is a bad idea if you've never used agile development before. Just as there's more to programming than writing code, there's more to agile development than the practices.

The practices are dependent on agile principles. Unless you understand those principles unless you've already mastered the art of agile development—you're probably not going to choose the right practices.

Every project and situation is unique, of course, so it's a good idea to have an agile method that's customized to your situation. Rather than making an agile method from scratch, start with an existing, proven method and iteratively refine it. Apply it to your situation, note where it works and doesn't, make an educated guess about how to improve, and repeat.



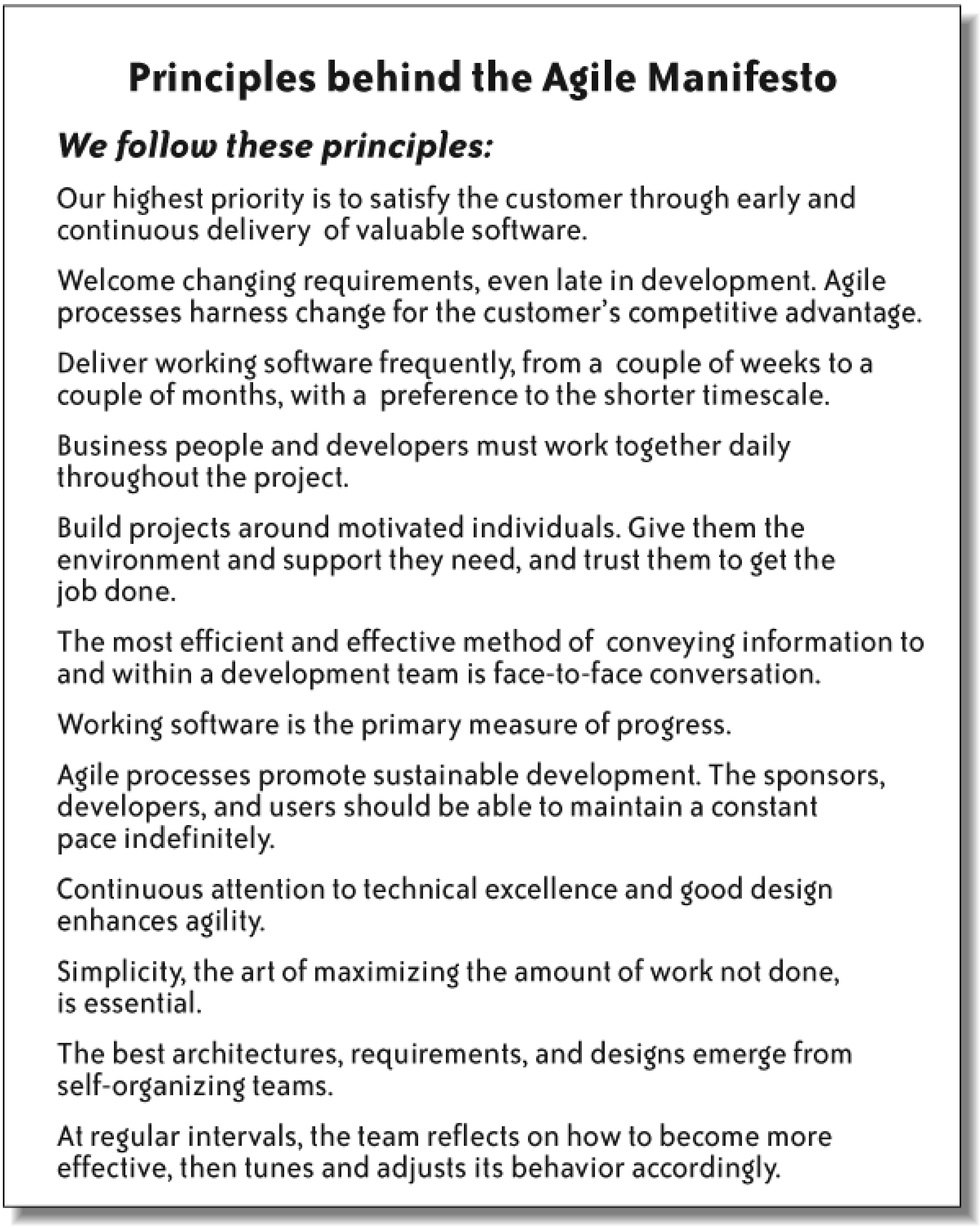
Manifesto for Agile Software Development

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

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

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

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity, the art of maximizing the amount of work not done, is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The Road to Mastery

Mastering the art of agile development requires real-world experience using a specific, **well-defined agile method**.

Extreme Programming for this purpose. It has several advantages:

• Of all the agile methods, XP is the most complete. It places a strong emphasis on technical practices in addition to the more common teamwork and structural practices.

• XP has undergone intense scrutiny. There are thousands of pages of explanations, experience reports, and critiques out there. Its capabilities and limitations are very well understood.

To master the art of agile development—or simply to use XP to be more successful—follow these steps:

1. Decide why you want to use agile development. Will it make your team and organization more successful? How?

2. Determine whether this approach will work for your team.

3. Adopt as many of XP's practices as you can. XP's practices are self-reinforcing, so it works best when you use all of them together.

4. Follow the XP practices rigorously and consistently.

5. As you become confident that you are practicing XP correctly—again, give it several months—start experimenting with changes ,Each time you make a change, observe what happens and make further improvements.

Find a Mentor

As you adapt XP to your situation, you're likely to run into problems and challenges. For these situations, you need a mentor: an outside expert who has mastered the art of agile development.

NOTE
If you can get an expert to coach your team directly, that's even better.
However, even master coaches benefit from an outside perspective when they

encounter problems.

The Genesis of Agile, Introduction and background

In the early 1990s, PC computing began to rise in organizations, but software development faced a hurdle. At that time, people used to call this crisis the "the application development crisis." At that time, organizations used to estimate three years between a validated business need and an actual application in production. But, business doesn't work like that. Even those days, businesses moved faster than three years' time span.

If you had to wait for three years to solve the problems your business faces, your business requirements, systems, and even the entire business can change in three years. Because of this time crisis, businesses used to cancel many projects halfway. And many projects failed to match the requirements and needs.

Before agile came, several industries like software, aerospace, manufacturing used to follow the waterfall approach. They would identify problems and work to create a plan that solves the problem. For example, the development team used to- set requirements and work scope for a project

- Design the product based on predefined requirements
- Build the product
- Test the product
- Identify problems during the testing
- Fix the problem
- Launch the finished product

This waterfall approach needs you to stick to the plan set at the very beginning of your project. That means you can't make any necessary changes along the way, even if it's needed. Now, this created a lot of havoc since a fixed plan could be inconvenient. Moreover, the waterfall approach was all about bringing a finished product to the market, even if it takes years to complete.

The waterfall approach was creating a lot of problems for both the developer and the customer. As it would take years to come up with a solution, the problem's nature would change. Eventually, when they used to launch the planned solution in the market, it would become outdated. These delays in product delivery led to the delivery of an unfinished product that no longer had any market fit.

Industries were Frustrated with the Waterfall Approach

Several industries started to show their frustration with the waterfall approach. During the 1990s, a large crowd of software development teams began to plan for a new approach. Among them, **Jon Kern** was one such frustrated thought leader who became increasingly active to find something more "timely and responsive."

Agile Manifesto and Principles

In 2001, a small group of software gurus, tired of the traditional approach, got together and wrote a manifesto which became a guiding principle for Agile Software development.

The Manifesto for Agile Software Development is a document produced by 17 developers at Snowbird, Utah in 2001. This document consists of 4 Agile Values and 12 Agile Principles. These 12 principals and 4 agile values provide a guide to Software Developers. The Manifesto for Agile Software Development emerged as a transformative guide to **Software Development**.

4 Values of Manifesto for Agile Software Development

1. Individuals and Interactions over Processes and Tools: Focuses on the importance of effective communication and collaboration among team members.

2. Working Software over Comprehensive Documentation: Prioritizes the delivery of functional software as the primary measure of progress.

3. Customer Collaboration over Contract Negotiation: Encourages customers and stakeholders to have active involvement throughout the development process.

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

Principles of Manifesto for Agile Software Development:

1) Customer Satisfaction through Early and Continuous Delivery:
This principle concentrates on the importance of customer satisfaction by providing information to customers early on time and also with consistency throughout the development process.

2) Welcome Changing Requirements, Even Late in Development:
Agile processes tackles change for the customer's competitive advantage. Even late in development, changes in requirements are welcomed to ensure the delivered software meets the evolving requirements of the customer.

3) Deliver Working Software Frequently:
This principle encourages the regular release of functional software increments in short iterations. This enables faster feedback, and adaptation to changing requirements.

4) Collaboration between Business Stakeholders and Developers:
This says the businesspeople and developers must work together daily throughout the project. There should be communication and collaboration between stakeholders and the development team regularly. This is crucial for understanding and prioritizing requirements effectively.

5) Build Projects around Motivated Individuals:
This promotes in giving developers the environment and support they need and trust them to complete the job successfully. Motivated and empowered individuals are more likely to produce work with quality and make valuable contributions to the project.

6) Face-to-Face Communication is the Most Effective:
Face-to-Face communication is the most effective method of discussion and conveying information. This principle depicts the importance on direct interaction which helps in minimizing misunderstandings, and hence effective communication is achieved.

7) Working Software is the Primary Measure of Progress:
This principle emphasizes on delivering functional and working software as the primary metric for project advancement. It encourages teams to prioritize the continuous delivery of valuable features, so it ensures that good progress is consistently achieved throughout the process. The primary goal is to provide customers with incremental value and also gather feedback early in the project life cycle.

8) Maintain a Sustainable Pace of Work:
Agile promotes sustainable development. All people involved: The sponsors, developers, and users should be able to maintain a constant pace indefinitely. This principle depicts the need for a sustainable and consistent development pace. This helps in avoiding burnout and ensure long-term project success.

9) Continuous Attention to Technical Excellence and Good design:
This principle is on the importance of maintaining high standards of technical craft and design, so it ensures the long-term ability in maintenance and adaptability of the software.

10) Simplicity—the Art of Maximizing the Amount of Work Not Done:
Simplicity is essential. The objective here is to concentrate on the most valuable features and tasks and avoiding unnecessary complexity as the art of maximizing the amount of work not done is crucial.

11) Self-Organizing Teams: Self-organizing teams provides the best architectures, requirements, and designs. These help in empowering teams to make decisions and organize to optimize efficiency and creativity.

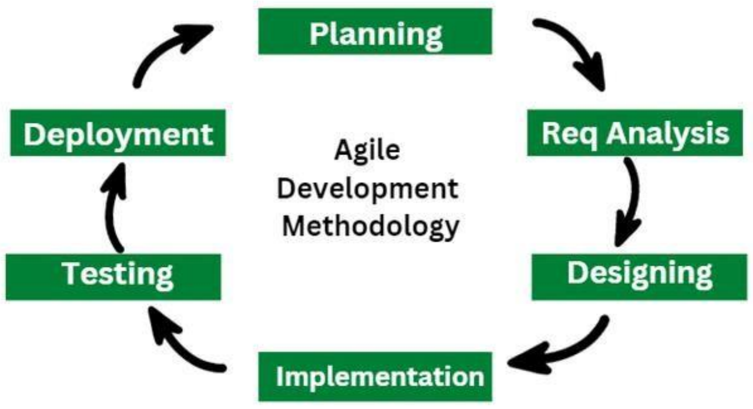
12) Regular Reflection on Team Effectiveness:
This makes the team reflects on how to become more effective in regular intervals and then adjusts accordingly. Continuous improvement is very crucial for the adapting to changing circumstances and optimizing the team performance over time.

Simple Design

Agile model In the context of system design is a flexible and adaptive approach to designing systems that can respond to changing requirements and customer needs. The key objective is to create a functioning system as soon as possible and to refine it depending on input from stakeholders. The agile model also promotes customer and developer involvement, which helps guarantee that the system being created satisfies end-user requirements.

Agile model places an emphasis on iterative and incremental development, customer and developer communication, and flexible response to evolving needs. The Agile model typically consists of several iterations or sprints, each of which is focused on delivering a specific set of features or functionality. The stakeholders evaluate the work completed at the conclusion of each sprint and offer feedback, which is then utilized to plan and prioritize the following batch of tasks. This iterative process makes it possible to quickly and without substantial delays incorporate changes in requirements or client requests into the system architecture.

Phases of Agile Model in Designing System



Planning: The team defines the project’s overall goals during the planning phase of the agile model and decides what must be accomplished throughout each sprint. This entails determining the project’s scope, identifying the main stakeholders, and developing a high-level roadmap for the system design. The development methodology, including the agile procedures and practices to be followed, is also established by the team.

Requirement Analysis: The team collaborates with the stakeholders throughout the requirement analysis phase to compile and examine the system's requirements. This entails gathering requirements and prioritizing them according to their urgency and importance. Additionally, the team identifies any potential hazards or limitations that could affect the project and creates a strategy to mitigate them.

Designing: The team develops intricate designs for the system interfaces and components at this stage. The team creates any necessary prototypes as well as the architecture and design patterns that will be employed. This phase's objectives are to lay a strong framework for the system and make that the design is scalable and consistent.

Implementation: The team constructs the system's individual parts and incorporates them into the overall design during the implementation phase. The team completes each sprint with the delivery of usable software. The team also makes adjustments to the backlog of needs and makes that the system is being developed in accordance with the design.

Testing: During the testing phase, the team validates the system by putting each component through its paces and making sure it complies with the specifications. The crew also finds and resolves any flaws or problems that come up while testing. Making ensuring the system is high-quality and prepared for deployment is the aim of this phase.

Deployment: The system is given to the end users during this last phase. The team deploys the system in collaboration with the stakeholders and offers any required support and training. This phase's objective is to guarantee that the system is correctly implemented and that end users can efficiently utilize it.

These are the six phases of the agile model in designing systems. The key characteristic of the agile model is that it is iterative and adaptive, allowing for changes and adjustments to be made throughout the project. This helps to ensure that the system being designed meets the needs of the end users and that the project is delivered on time and within budget.

User Stories in Agile

User stories are a key component of agile software development. **They are short, simple descriptions of a feature or functionality from the perspective of a user**.

User stories are used to capture requirements in an agile project and help the development team understand the needs and expectations of the users.

Here are some key characteristics of user stories:

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

In agile software development, the development team uses user stories to plan and prioritize work, estimate the effort required for implementation, and track progress towards completing the user stories.

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

User stories are considered an important tool in Incremental software development. Mainly a user story defines the type of user, their need, and why they need that. So in simple, a user story is a simple description of requirements that needs to be implemented in the software system.

Pattern of User Story:

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

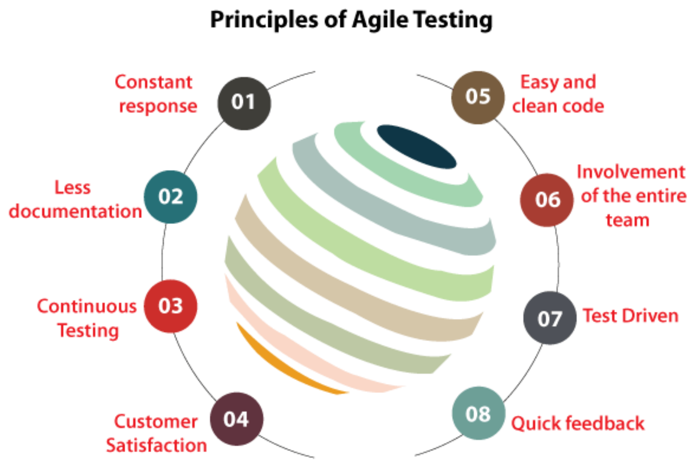
Example:
As the project manager of a construction team, I want our team-messaging app to include file sharing and information update so that my team can collaborate and communicate with each other in real-time as a result the construction project development and completion will be fast.

Agile Testing

Agile Testing is a type of software testing that follows the principles of agile software development to test the software application.

All members of the project team along with the special experts and testers are involved in agile testing. Agile testing is not a separate phase and it is carried out with all the development phases i.e. requirements, design and coding, and test case generation. Agile testing takes place simultaneously throughout the Development Life Cycle. Agile testers participate in the entire development life cycle along with development team members and the testers help in building the software according to the customer requirements and with better design and thus code becomes possible.
Agile Testing has shorter time frames called iterations or loops. This methodology is also called the delivery-driven approach because it provides a better prediction on the workable products in less duration time.

- Agile testing is an informal process that is specified as a dynamic type of testing.
- It is performed regularly throughout every iteration of the Software Development Lifecycle (SDLC).
- Customer satisfaction is the primary concern for agile test engineers at some stage in the agile testing process.



1. Constant Response

The implementation of Agile testing delivers a response or feedback on an ongoing basis. Therefore, our product can meet the business needs.

2. Less Documentation

The execution of agile testing requires less documentation as the Agile teams or all the test engineers use a reusable specification or a checklist. And the team emphases the test rather than the secondary information.

3. Continuous Testing

The agile test engineers execute the testing endlessly as this is the only technique to e constant improvement of the product.

4. Customer Satisfaction

In any project delivery, customer satisfaction is important as the customers are exposed to their product throughout the development process.
As the development phase progresses, the customer can easily modify and update requirements. And the tests can also be changed as per the updated requirements.

5. Easy and clean code

When the bugs or defects occurred by the agile team or the testing team are fixed in a similar iteration, which leads us to get the easy and clean code.

6. Involvement of the entire team

As we know that, the testing team is the only team who is responsible for a testing process in the Software Development Life Cycle. But on the other hand, in agile testing, the business analysts (BA) and the developers can also test the application or the software.

7. Test-Driven

While doing the agile testing, we need to execute the testing process during the implementation that helps us to decrease the development time. However, the testing is implemented after implementation or when the software is developed in the traditional process.

8. Quick response

In each iteration of agile testing, the business team is involved. Therefore, we can get continuous feedback that helps us to reduce the time of feedback response on development work.