In 2001, a group of 17 software engineers met to discuss how to bring software to market faster. They wanted to develop and launch quickly, then use built-in customer feedback loops to make quick improvements. The goal was to recognize value early and often and they develop something called the agile manifesto. This manifesto has four key values. First is it places value on individuals and interactions over processes and tools, very different than our predictive models. Second, it places value on working software over comprehensive documentation. You remember in predictive models, we always started with a plan and followed our documentation. Third, it places value on customer collaboration over contract negotiation, and finally number 4, it places value on responding to change over following a predictive plan.

When we talk about the vision, we can also think of this vision as the scope, but we don't call it a scope. What we call it in agile is a product backlog. A product backlog is a prioritized list of work or list of functions for the team to work on. I like to think of it as a to-do list.

The product backlog could contain things like new features or fixes from other sprint releases. Each function or small portion of the product backlog is broken down into something called a user story. A user story is a small unit of work in the agile framework. It's usually an informal general explanation of some feature written from the perspective of the end user or written from the perspective of the customer. Its purpose is to articulate how this piece of work will deliver value to the customer.

The project team takes the user stories, takes them into the sprint. The sprint sometimes is called the scrum. It is a process that consists of planning, it consists of the daily scrum, the sprint review, and the sprint retrospective.

The current content focuses on the 12 principles of Agile as outlined in the Agile Manifesto. Here’s a brief summary of each principle:

1. Customer Satisfaction: Prioritize satisfying the customer through early and continuous delivery of valuable software.
2. Welcoming Change: Embrace changing requirements, even late in development, to provide a competitive advantage.
3. Frequent Delivery: Deliver working software frequently, with a preference for shorter timescales.
4. Daily Collaboration: Encourage daily collaboration between business people and developers.
5. Motivated Individuals: Build projects around motivated individuals, providing them with the environment and support they need.
6. Face-to-Face Communication: Promote face-to-face conversation as the most effective method of communication.
7. Working Software as Progress: Measure progress primarily through working software.
8. Sustainable Development: Maintain a constant pace of development that is sustainable for all team members.
9. Technical Excellence: Focus on continuous attention to technical excellence and good design.
10. Simplicity: Maximize the amount of work not done by focusing on simplicity.
11. Self-Organizing Teams: Allow self-organizing teams to determine the best way to accomplish their work.
12. Continuous Improvement: Regularly reflect on team performance and adjust behaviors for improvement.

These principles emphasize the importance of collaboration, flexibility, and continuous improvement in Agile project management. Would you like to explore any specific principle in more detail?

In the current course content on Agile Project Management, the focus is on how Agile methodologies are applied in practice. Key points include:

* Agile Frameworks: Different frameworks exist, with Scrum being one of the most popular. Scrum involves:
  + Sprints: Short, frequent development cycles.
  + Roles:
    - Developers: Team members who create the project.
    - Product Owner: Prioritizes the work.
    - Scrum Master: Provides support and leadership.
* Kanban: Utilizes a visual board to track workflow, with categories like backlog, work in progress, and done.
* Lean: Focuses on eliminating waste in materials and time.
* Extreme Programming (XP): Emphasizes simplicity and collaboration, often using techniques like pair programming to enhance code quality.

The module will explore these frameworks in detail, highlighting that teams can combine elements from different methodologies to suit their projects.

Lean-Agile is a hybrid between Lean principles and Agile practices.

What are these Lean principles?

Mary and Tom Poppendieck best describe the principles of Lean in their book *Lean Software Development: An Agile Toolkit*. Here’s a summary:

1. Eliminate waste. What’s waste? It’s anything that doesn’t “add value to a product, value as perceived by the customer.” Waste is anything produced or sitting around that doesn’t get used. Parts on a shelf that are just gathering dust. It can be “[when] developers code more features than are immediately needed.” The time lost shifting development from one group to another? That’s waste too, say Mary and Tom Poppendieck.
2. Amplify learning. The Poppendiecks see tackling a software solution as a lot like trying to put together a new recipe for a top restaurant dish. A chef will iterate and learn from the variations produced. Software development is more complex, and teams can be big, but when you amplify learning, it makes this discovery process possible.
3. Decide as late as possible. In something like embedded software development, there can be many unknowns. Which processor is going to fit the product and users’ demands best? Should we develop in x, y or z language? Is this screen better than that screen? Functionality likely to change due to user or market feedback? There’s no value in speculating. Instead, delay decisions until you have the facts.
4. Deliver as fast as possible. “Design, implement, feedback, improve,” and repeat. “Rapid development” is entirely plausible; its speed enables you to make informed decisions with real feedback. Keeping this cycle short, amplifies learning as well as making sure “customers get what they need now, not what they needed yesterday.”
5. Empower the team. The people doing the work are the ones who understand the details the best. Want “excellence”? Make sure your technical team are involved in “the details of technical decisions”. Enable these teams to use “pull techniques” and “local signalling mechanisms” (like Kanban) to schedule and complete work whilst letting everyone know what’s coming up. So long as a leader guides the team, “they will make better technical decisions and better process decisions than anyone can make for them.”
6. Build integrity in. Here the focus is on ensuring quality. This is more than just customers being happy with the final product, its “perceived integrity”. And it’s going further than “conceptual integrity” where the whole works seamlessly together. For software, building integrity in means it can evolve smoothly over time, because it has “coherent architecture, scores high on usability and fitness for purpose, and is maintainable, adaptable, and extensible.” For the team, integrity comes “from wise leadership, relevant expertise, effective communication, and healthy discipline”.
7. See the whole. Don’t just look at one area of a project. Step back and look at all areas involved. “Quite often, the common good suffers if people attend first to their own [specialised] interests.” Instead of measuring based on people’s “[specialised] contribution”, look instead to the overall performance of the project.

Why is adapting better than predicting? On the face of it, adapting can look like it’s less efficient than predicting and it is. To understand Lean-Agile and why it’s so successful, you need to understand the difference between efficiency and effectiveness and how it stops waste.

Efficiency focuses on time and cost, the amount of work done, while effectiveness is about building the right thing. Adapting to new knowledge as it emerges, ending up with the right solution and avoiding the waste that comes from making the wrong thing.

If you have a focus on efficiency, you won’t be able to see the advantages of being effective. You need to take an extra step back and look at the whole picture of what the team is creating.

Test-Driven Development (TDD) is a way of developing software. Developers will write a “unit test” before they write code, checking that these tests fail and then write code to pass that test.

Research from Microsoft has found that TDD can lead to a 15-35% increase in initial development time (link opens PDF), but they also found that it increases the quality of code. Code written using TDD had fewer defects, according to Microsoft. The creation of unit tests also means that these assets exist for reuse later in a product’s lifecycle.

A 2018 study that looked at TDD and unit tests found the approach led to code with fewer defects as well. Looking at mutation scores (where you check tests through introducing defects to code), this study found TDD led to 87.5% detection of these defects. Conversely, a test last approach, testing code at the end of a project, typically favoured in Waterfall style development, found only 67% of defects.

Taking a TDD approach means that fixing these defects happens early on, before they can create waste.

The Kanban method is based on six foundational change management and service delivery.

1. Start with what you do now: Kanban is about continuous improvement, but it starts with an understanding of the current processes and workflows.
2. Agree to pursue incremental, evolutionary change: Rather than attempting a large-scale transformation all at once, Kanban advocates for small, incremental changes that build on each other over time.
3. Encourage acts of leadership at all levels: Kanban is not just for managers or team leads but for everyone involved in the work. Anyone can take leadership and suggest improvements based on their observations.
4. Focus on customer needs and expectations: Kanban promotes understanding the needs and expectations of your customers to elevate the quality of the provided services and the value it creates.
5. Manage the work, not the workers: Kanban respects the existing roles and responsibilities of team members and empowers people’s abilities to self-organize around the work.
6. Regularly review the network of services: Kanban encourages collaboration and encourages team members to share their observations, ideas, and feedback for improving the work through regular reviews of the entire network of services.

For a successful Kanban implementation, the method relies on six essential practices:

1. Visualizing the workflow: Creating a visual representation of the workflow helps to identify bottlenecks, visualize the flow of work, and make the work more transparent.
2. Limiting work in progress: Limiting the amount of work in progress helps to prevent multitasking and improve focus on completing one task at a time, thereby improving efficiency and reducing lead time.
3. Managing flow: Kanban aims to help in optimizing flow which can be achieved by monitoring flow metrics, identifying and addressing bottlenecks, and continuously improving the workflow.
4. Making process policies explicit: Defining and communicating process policies clearly helps to ensure that everyone understands how work is supposed to be done, which reduces misunderstandings and promotes consistency.
5. Implementing feedback loops: Kanban emphasizes the importance of getting feedback from customers, stakeholders, and team members as a way to identify areas for improvement.
6. Improving collaboratively: Kanban is a continuous improvement process that encourages collaboration and experimentation to identify and solve problems, improve continuously, and evolve their processes to better meet the needs of their customers.

The most important difference between Kanban and Scrum is that the former is a method, while the latter is a framework. As an Agile methodology, Kanban builds a continuous delivery model where teams release value as soon as they are ready, while Scrum organizes work in Sprints. Applying either one depends on the nature of your process, however, it can be said that Kanban offers a more tailor-made approach while Scrum relies on predetermined rules. Another key distinguishing characteristic between the two is the mindset and founding belief systems of Scrum and Kanban.

|  |  |  |
| --- | --- | --- |
|  | **Kanban** | **Scrum** |
| Nature | Kanban is an adaptive method | Scrum is a prescriptive framework |
| Principles | 1. Start with what you do now  2. Agree to pursue evolutionary change  3. Encourage acts of leadership at all levels  4. Focus on customer’s needs  5. Manage the work  6. Regularly review the network of services | 1. Empiricism  2. Transparency  3. Inspection  4. Adaptation |
| Cadences | - Team-level cadences  - Service-oriented cadences | - Sprint with a fixed length  - Sprint planning  - Daily Scrum  - Sprint Review  - Sprint Retrospective |
| Roles | - Service Delivery Manager\*  - Service Request Manager\*  (\*no pre-defined roles are required) | - Product Owner  - Scrum Master  - Development Team |
| Metrics | - Cycle Time  - Throughput  - Work In Progress | - Velocity  - Planned Capacity |

Key points of the simplicity principle from the Agile Manifesto include:

* Maximizing Work Not Done: This principle emphasizes avoiding unnecessary work that does not add value.
* Lean Philosophy: Agile is rooted in Lean principles, which prioritize eliminating waste and valuing simplicity.
* Over-Engineering: Creating overly complex solutions is discouraged; simpler solutions are preferred as they are quicker to write, easier to understand, and reduce the potential for errors.
* Minimum Viable Product (MVP): The goal is to iterate quickly to deliver an MVP, allowing customers to go to market faster.

Agile team velocity is a metric used to measure how much work a team has completed over time, which helps predict project timelines. Key points include:

* Velocity Measurement: It is calculated based on the amount of work delivered in past sprints.
* User Story Estimation: Teams estimate the effort required for user stories using methods like:
  + User Story Points: Assigning numerical values to represent effort (e.g., 1 point = 5-8 hours).
  + T-shirt Sizing: Categorizing effort as small, medium, large, or extra-large.
* Calculating Velocity: By analyzing past iterations, teams can determine their average story points completed per iteration.
* Project Timeline Prediction: Using velocity, teams can estimate how many iterations and weeks are needed to complete remaining work in the backlog.

A burndown chart is a graphical representation of the work remaining for a project and the time remaining to complete it. Burndown charts are commonly used in software development, especially in teams using Agile project management.

Burndown charts can illustrate what work was completed in each iteration, how quickly it was accomplished and what work remains. A burndown chart makes it easy for stakeholders, management and sponsors to see a representation of this progress.

The biggest limitation of a burndown chart is that it’s a high-level summary. All tasks are treated equally when you summarize or count down, but it might not be clear that some are more difficult than others.

A burnup chart, on the other hand, tracks the work already completed and can help motivate the team by displaying the progress made thus far.

How does the Scrum team prioritizes user stories for sprints to maximize customer value. Key points include:

* Product Owner's Role: The product owner, as the voice of the customer, is responsible for prioritizing the product backlog based on customer value.
* Prioritization Techniques:
  + MoSCoW Method: Categorizes requirements into must-haves, should-haves, could-haves, and won't-haves.
  + Kano Analysis: Visual representation of features on a graph to assess customer satisfaction.
  + Dot Voting: Team members use sticky dots to vote on features they believe should be prioritized.
  + $100 Method: Team members allocate a hypothetical $100 to prioritize features based on perceived value.

These methods help ensure that the team focuses on delivering the most valuable features to the customer early and often.

Emphasizing that agility is not just a process but also a mindset. Key points include:

* Definition of Agile Organization: An agile organization embraces the values and principles of the Agile Manifesto, prioritizing people and results over rigid processes.
* Agility: Defined as the ability to quickly sense and adapt to changes, delivering relevant results efficiently.
* Key Areas for Agile Organizations:
  + Corporate Strategy: Must be adaptable to changing markets.
  + Organizational Structure: Should be flat and distributed to enhance communication and best practices.
  + Leadership: Emphasizes a distributed leadership model reflecting servant leadership principles.

The lecture also addresses the impact of remote work on Agile practices, noting that technology enables effective collaboration despite physical distance. Recent data shows a significant adoption of Agile methodologies across various departments beyond software development, with many organizations planning to maintain hybrid or remote work models.

A table with text and symbols

AI-generated content may be incorrect.

Strategic agility can be further broken down into six principles.

* Prioritize speed over perfection

Opportunities come and go quickly during a crisis, so organizations need to be ready and willing to act quickly, even if they sacrifice quality and predictability in the process.

* Prioritize flexibility over planning

Strategy is often taught in business schools as a cascade of choices around where to play and how to win. These choices are typically built into strategic plans that are devised and approved over a period of several months, and then executed over three or five years, before the cycle repeats. However, in a crisis, a strategic plan can easily become an anchor that locks an organization onto a path that is no longer relevant.

* Prioritize diversification and “efficient slack” over optimization

Many organizations struggled — and some failed — during the pandemic not because they weren’t nimble or innovative, but because they were felled by a single devastating blow. The root of this problem, in many cases, was either a lack of diversification or an overemphasis on efficiency and optimization.

* Prioritize empowerment over hierarchy

Systems are most vulnerable at their weakest points. A hierarchy, for example, is most vulnerable at the top.

* Prioritize learning over blaming

It has been well established that organizational cultures that reward risk taking and tolerate failure move more quickly that those that don’t. If people are criticized for failing, they are less likely to take risks; in a crisis, this can be fatal.

* Prioritize resource modularity and mobility over resource lock-in

Since it is difficult to predict how the future will unfold in a crisis, it is hard to effectively plan the allocation of resources. Thus, it important to build resources that are modular and/or mobile so they can be reconfigured or moved as needed.