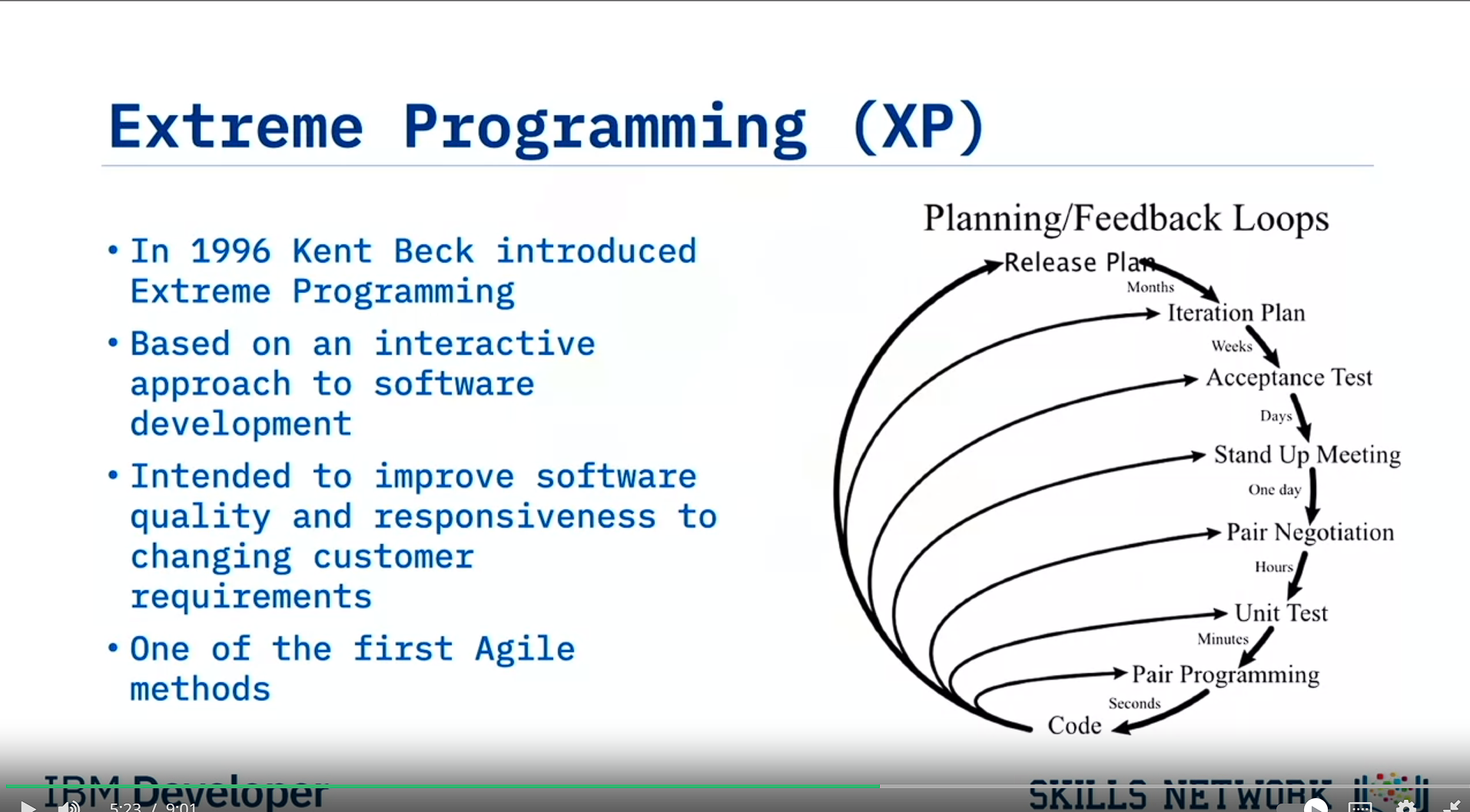
1. **Agile Definition**:
   * Iterative approach to project management.
   * Focus on responsiveness and quick delivery of value to customers.
   * Planning in small increments with customer feedback to adjust accordingly.
2. **Characteristics of Agile**:
   * Adaptive planning: Plan in small iterations instead of long-term planning.
   * Evolutionary development: Building in small increments and evolving.
   * Early delivery: Delivering products early to get customer feedback.
   * Continuous improvement: Constantly improving the product and team based on feedback.
   * Responsiveness to change: Quickly re-planning based on changing customer requirements.
3. **Agile Manifesto Values**:
   * Individuals and interactions over processes and tools.
   * Working software over comprehensive documentation.
   * Customer collaboration over contract negotiation.
   * Responding to change over following a plan.
4. **Key Points on Agile Manifesto**:
   * Value on the left side items more, but right side items still have value.
   * Importance of working software, interactions, collaboration, and responsiveness.
5. **Agile Software Development**:
   * Iterative approach conforming to the Agile Manifesto.
   * Emphasizes flexibility, peer and customer interaction, and transparency.
   * Uses small, co-located, cross-functional, self-organizing, and self-managing teams.
6. **Key Takeaway**:
   * Build what is needed, not strictly what was planned.
   * Importance of being responsive to changing needs and feedback.



**Waterfall Approach**:

1. **Description**:
   * Structured, step-by-step process.
   * Sequential phases: requirements, design, coding, integration, testing, deployment.
   * Each phase has entrance and exit criteria.
2. **Problems with Waterfall**:
   * No provision for change: difficult to go back and modify previous phases.
   * No intermediate delivery: software is delivered only at the end.
   * Information loss: risk of losing information between phases.
   * High cost of late mistakes: expensive to fix issues found in later phases.
   * Long lead times: extended duration from start to finish.
   * Siloed teams: lack of awareness of the impact on other phases.

**Extreme Programming (XP)**:

1. **Description**:
   * Iterative approach with fast feedback loops.
   * Major release planning, iteration planning, daily stand-ups, pair programming, and unit testing.
   * Focus on improving software quality and responsiveness to change.
2. **Values of XP**:
   * **Simplicity**: Do only what is needed, avoid over-engineering.
   * **Communication**: Foster constant interaction and information sharing.
   * **Feedback**: Frequent feedback loops to gauge progress and make adjustments.
   * **Respect**: Everyone's input is valued equally, fostering a collaborative environment.
   * **Courage**: Honest estimation and commitment to realistic goals.

**Kanban**:

1. **Definition**:
   * Originates from Japanese manufacturing.
   * Focus on continuous flow and visual management of tasks.
2. **Core Principles**:
   * **Visualize the workflow**: Use Kanban boards to see and manage tasks.
   * **Limit work in progress**: Avoid overloading team members with too many tasks.
   * **Manage and enhance the flow**: Continuously seek improvements in the process.
   * **Make policies explicit**: Clearly define and communicate processes and definitions of "done".
   * **Continuously improve**: Regular feedback and adjustments to enhance workflow efficiency.

**Summary**:

* **Waterfall**: A traditional, phase-based approach with significant limitations in flexibility and responsiveness.
* **Extreme Programming (XP)**: An iterative methodology emphasizing simplicity, communication, feedback, respect, and courage.
* **Kanban**: A visual and flow-focused approach prioritizing continuous improvement and efficient task management.

**Detailed Explanation of Agile Practices**

1. **Working in Small Batches**
   * **Concept**: Small batch processing, derived from lean manufacturing, involves completing work in smaller segments rather than large chunks. This approach minimizes waste and allows for quicker detection and correction of errors.
   * **Example**: Mailing brochures in small batches rather than all at once allows for early detection of issues. If you fold, insert, seal, and stamp each brochure individually, you can immediately see if there is a problem, such as missing glue on envelopes or typos, and correct it before proceeding further. This prevents large-scale waste and ensures higher quality output.
2. **Minimum Viable Product (MVP)**
   * **Concept**: An MVP is the simplest version of a product that can be released to test a hypothesis and learn from user feedback. It’s not just an early version of the final product but a strategic tool to validate assumptions and guide further development.
   * **Example**: Instead of building a complete car from scratch, an MVP approach might start with a skateboard. The initial product (skateboard) helps test the color preference. Subsequent versions (scooter, bike, etc.) gradually add features and gather feedback, ultimately leading to a final product (car) that better matches customer needs and preferences, thus ensuring continuous learning and adaptation.
3. **Behavior Driven Development (BDD)**
   * **Concept**: BDD involves writing tests based on the expected behavior of the system from the user’s perspective. It uses a common language (Gherkin syntax) that is understandable by both technical and non-technical stakeholders, ensuring that everyone has a shared understanding of the requirements.
   * **Example**: For an e-commerce system, BDD might include scenarios like: "Given I have items in my cart, when I proceed to checkout, then the system should allow me to place an order." This approach ensures that the system meets user expectations and behaves correctly under different conditions.
4. **Test Driven Development (TDD)**
   * **Concept**: TDD focuses on writing tests before the actual code. This practice ensures that the code meets the predefined requirements and allows for immediate validation of functionality.
   * **Process**:
     + Write a test case for a specific function.
     + Write just enough code to pass the test.
     + Refactor the code for optimization while ensuring it still passes the test.
   * **Example**: Suppose you need to write a function to calculate the sum of two numbers. You first write a test case that specifies the expected output for given inputs. Then, you implement the function to pass this test and refactor it for better performance or readability.
5. **Pair Programming**
   * **Concept**: Pair programming involves two developers working together at one workstation. One writes the code (the driver), while the other reviews each line of code as it is written (the navigator). They switch roles frequently.
   * **Benefits**: This practice improves code quality by having two sets of eyes on the code, facilitates knowledge sharing and mentorship, and can significantly reduce the number of bugs.
   * **Example**: In a development team, a senior developer can pair with a junior developer. The senior developer guides the junior through complex problem-solving and coding practices, while the junior brings a fresh perspective and can ask questions that may reveal overlooked assumptions or errors.

By incorporating these practices, Agile teams can ensure rapid delivery, continuous improvement, and high-quality software that better meets user needs.

**Summary and Highlights**

Congratulations! You have completed this lesson. At this point in the course, you know:

* Agile is an iterative approach to software development that emphasizes flexibility, interactivity, and transparency using small, cross-functional teams
* The Agile Manifesto describes the core values of Agile:

- 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 waterfall approach is a structured, step-by-step process that can lead to problems not surfaced until later in development
* Extreme Programming (XP) advocated an iterative approach that valued simplicity, communication, feedback, respect, and courage
* A Kanban system is characterized by visualizing workflow, limiting work in progress, managing and enhancing flow, making process policies explicit, and continuously improving a process
* Working in small batches means delivering something useful quickly
* An MVP is the cheapest/easiest thing you can do to test a hypothesis and learn
* Behavior Driven Development makes sure you are building the right thing
* Test Driven Development makes sure you are building the thing right
* Pair programming enables you to discover defects earlier and increase your code quality
* The Gherkin syntax, which comes from a company called Cucumber, is a single syntax that developers and stakeholders can understand: given some set of preconditions, when an event occurs, then some outcome is observed.

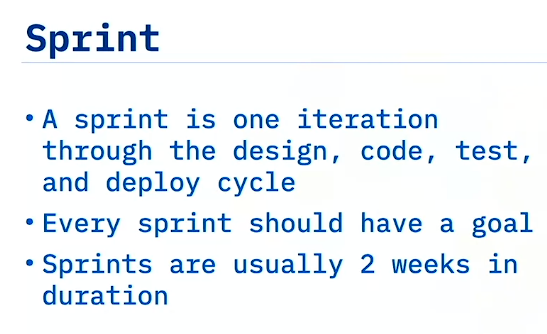
Go to next item

**Co**

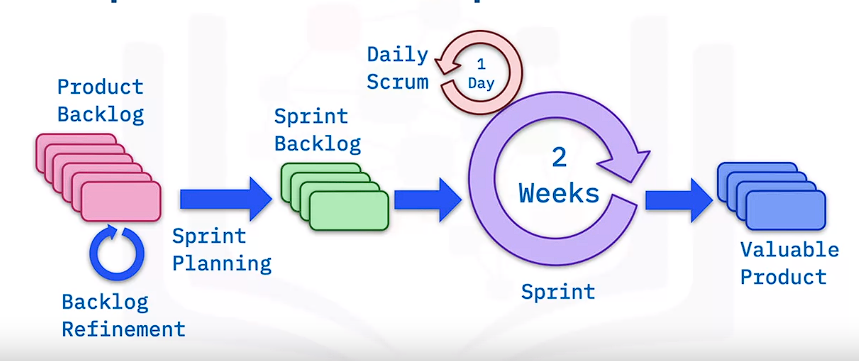
**Agile vs. Scrum**

* **Agile**:
  + **Philosophy**: Agile is a flexible approach to software development that prioritizes adaptability, collaboration, and customer feedback. It values 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.
  + **Perspective**: Agile is not prescriptive; it provides guiding principles for teams to follow but does not define specific methodologies or practices.
* **Scrum**:
  + **Methodology**: Scrum, on the other hand, is a specific framework within the Agile philosophy. It provides a structured approach to incremental product development, emphasizing small, cross-functional, self-managing teams.
  + **Prescription**: Unlike Agile, Scrum is prescriptive, offering clear roles, rules, meetings, and artifacts that teams must follow to implement the methodology effectively.

**Key Characteristics of Scrum**

1. **Incremental Product Development**: Scrum focuses on delivering product increments in small, manageable iterations called sprints.
2. **Cross-functional Teams**: Teams in Scrum are composed of individuals with diverse skill sets who collectively possess all the skills necessary to deliver a shippable product increment.
3. **Self-Management**: Scrum teams are empowered to make decisions and manage their own work processes, fostering autonomy and accountability.
4. **Roles, Rules, and Artifacts**: Scrum defines specific roles (e.g., Product Owner, Scrum Master, Development Team), meetings (e.g., Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective), rules (e.g., timeboxing), and artifacts (e.g., Product Backlog, Sprint Backlog, Increment) to guide the development process.
5. 

**Steps in the Scrum Process**

****

1. **Product Backlog**: A prioritized list of all features, enhancements, and fixes that constitute the product's roadmap. It serves as **the single source of requirements for any changes to be made to the product.**
2. **Backlog Refinement**: Also known as backlog grooming, this process involves reviewing and refining items in the product backlog to ensure they are clear, actionable, and ready for implementation in upcoming sprints.
3. **Sprint Planning**: A meeting held at the beginning of each sprint where the Scrum Team plans the work to be done during the sprint. This involves selecting items from the product backlog and defining the sprint goal and scope.
4. **Sprint Execution**: The development work takes place during the sprint, following the plan established during sprint planning. The team collaborates daily to build and test the increment.
5. **Daily Scrum**: A brief daily meeting where team members synchronize their work and plan their activities for the next 24 hours. Each team member answers three questions**: What did you do yesterday? What will you do today? Are there any blockers?**
6. **Sprint Review**: At the end of each sprint, the Scrum Team presents the completed work to stakeholders and gathers feedback. This allows for inspection and adaptation of the product and process.
7. **Sprint Retrospective**: A meeting held after the sprint review where the team reflects on the sprint process, identifies what went well and what could be improved, and makes adjustments for future sprints.

**Conclusion**

Scrum provides a structured framework within the Agile philosophy, offering clear guidelines for incremental product development. By following the key characteristics and steps of Scrum, teams can effectively deliver high-quality software while remaining responsive to customer needs and adaptable to change.

**Roles in Scrum**

**A product manager is a job title, a product owner is a Scrum role. Your product manager might be your product owner, might not be a product owner. Don't get them confused. This is a role in Scrum.**

1. **Product Owner**:
   * **Responsibilities**:
     + Represents stakeholders' interests and serves as their liaison with the Scrum Team.
     + Articulates the product vision, ensuring alignment with stakeholders' needs and goals.
     + Decides priorities, requirements, and the readiness of product increments for release.
     + Constantly reprioritizes and refines the product backlog, adjusting expectations and ensuring sprint readiness.
     + Accepts or rejects each product increment based on stakeholder feedback and alignment with the vision.
     + Makes decisions regarding product direction, including whether to ship or pivot based on market feedback.
2. **Scrum Master**: (smartest person in the room about scrum
   * **Responsibilities**:
     + Facilitates the Scrum process, ensuring adherence to Scrum principles, practices, and ceremonies.
     + Acts as an Agile coach and mentor, guiding the team in implementing Scrum effectively.
     + Creates an environment that fosters self-organization within the Scrum Team, allowing them to make decisions and commit to sprint goals.
     + *Shields the team* from external interferences, ensuring they remain focused and productive.
     + Identifies and resolves **impediments(something that prevents or interferes with a process, power, or right.) that hinder the team's progress, prioritizing their removal to maintain productivity.**
     + Enforces sprint time boxes and other Scrum practices, ensuring adherence to agreed-upon processes.
     + Captures empirical data and adjusts forecasts to support decision-making and continuous improvement.
     + Has no management authority over the team, serving as a facilitator rather than a manager.
3. **Scrum Team**:
   * **Characteristics**:
     + Cross-functional team comprising developers, testers, business analysts, domain experts, and operations personnel.
     + Self-organizing and self-managing, with no externally assigned roles or hierarchy within the team.
     + Small in size (ideally 5-9 members), allowing for effective collaboration and communication.
     + Co-located whenever possible to maximize effectiveness, but if dispersed, efforts are made to ensure effective collaboration.(at least 2 people in same time-zine)
     + Dedicated to the project, with long-term, full-time members who are not involved in multiple projects simultaneously.
   * **Responsibilities**:
     + Negotiates commitments with the Product Owner for each sprint, focusing on short-term goals and outcomes.
     + Determines how to reach commitments autonomously, with freedom to decide on implementation details and approaches.
     + Works collaboratively to deliver product increments iteratively, embracing Agile principles and practices.
     + Maintains a cooperative and supportive environment within the team, fostering creativity, innovation, and continuous improvement.

**Conclusion**

In this video, you've learned about the three essential roles in Scrum: the Product Owner, the Scrum Master, and the Scrum Team. Each role plays a crucial part in the Scrum framework, with distinct responsibilities aimed at driving collaboration, delivering value, and promoting continuous improvement. Understanding these roles and their functions is essential for implementing Scrum effectively and maximizing the benefits of Agile product development.

**Scrum *Artifacts(a usually simple object (such as a tool or ornament) showing human workmanship or modification as distinguished from a natural object.)***

1. **Product Backlog**:
   * A comprehensive list of all requirements or user stories for the product, representing future work.
   * Contains items not scheduled for the current sprint, serving as a repository for upcoming tasks.
   * May include subcategories like an "icebox" or a "release backlog" in some teams.
2. **Sprint Backlog**:
   * Contains the subset of user stories selected for implementation during the current sprint.
   * Represents the work planned and committed to by the Scrum Team for the upcoming sprint.
   * Provides clarity on the tasks to be executed within the sprint timeframe.
3. **Done Increment**:
   * Represents a product increment that is completed by the end of a sprint.
   * Achieved through the implementation of user stories and tasks during the sprint.
   * The goal is to produce a potentially shippable product increment with each sprint.

**Scrum Events**

1. **Sprint Planning**:
   * Occurs at the beginning of each sprint.
   * Involves the entire Scrum Team, including the Product Owner and the Scrum Master.
   * Focuses on selecting user stories from the product backlog and defining the sprint goal and scope.
2. **Daily Scrum (Daily Stand-up)**:
   * A brief daily meeting held by the Scrum Team.
   * Participants include developers, testers, and other relevant team members.
   * Intended to synchronize efforts, discuss progress, and identify any impediments.
3. **Sprint**:
   * The fixed time period (usually two weeks) during which the Scrum Team works to complete the sprint backlog.
   * Emphasizes collaboration, communication, and iterative development.
4. **Sprint Review**:
   * Conducted at the end of each sprint.
   * Involves showcasing the completed increment to stakeholders and obtaining feedback.
   * Facilitates transparency and alignment with stakeholder expectations.
5. **Sprint Retrospective**:
   * Held after the sprint review and before the next sprint planning.
   * Provides an opportunity for the Scrum Team to reflect on the sprint process and identify areas for improvement.
   * Focuses on enhancing team performance and refining the Scrum process.

**Benefits of Scrum**

1. **Higher Productivity**:
   * Daily meetings, clear goals, and collaborative work foster a productive environment.
2. **Better Quality**:
   * Emphasis on testing, iterative development, and continuous improvement leads to improved software quality.
3. **Reduced Time to Market**:
   * Incremental delivery allows for quicker release of product increments, enabling faster feedback and validation.
4. **Increased Stakeholder Satisfaction**:
   * Regular demonstrations and stakeholder involvement ensure alignment with expectations, enhancing satisfaction.
5. **Improved Team Dynamics**:
   * Transparency, collaboration, and autonomy contribute to stronger team cohesion and morale.
6. **Happier Employees**:
   * Empowerment, autonomy, and a focus on continuous improvement contribute to greater employee satisfaction and engagement.

**Scrum vs. Kanban**

* **Cadence**: Scrum follows fixed-length sprints, while Kanban operates on continuous flow.
* **Release Methodologies**: Scrum releases at the end of sprints, while Kanban practices continuous delivery.
* **Roles**: Scrum defines specific roles (Product Owner, Scrum Master, and Scrum Team), while Kanban has no predefined roles.
* **Key Metrics**: Scrum measures velocity, while Kanban focuses on cycle time.
* **Change Philosophy**: Scrum discourages changes during sprints, whereas Kanban embraces continuous improvement and flexibility.

In summary, Scrum provides a structured framework for Agile product development, offering clear artifacts, defined events, and numerous benefits for teams and stakeholders alike.

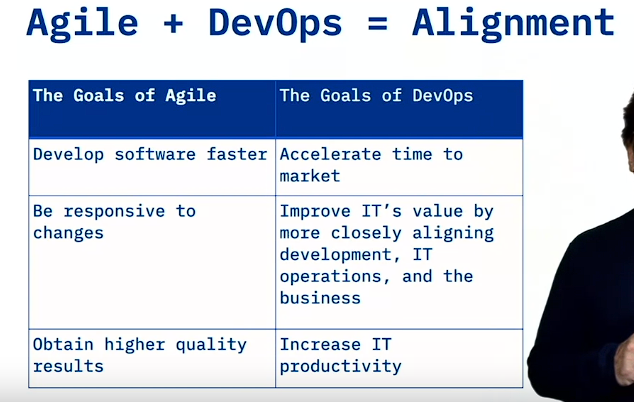
**Proper Organization and Success**

* **Conway's Law**: Organizations designing systems will produce designs reflecting their communication structures.
  + Reorganization may be necessary for agile adoption.
  + Teams must be aligned and loosely coupled yet tightly aligned.
  + Teams should have a clear mission aligned with business goals.
  + End-to-end responsibility and long-term missions enhance ownership.(don’t pull people on and off the project

**Autonomy**

* **Motivation and Efficiency**: Autonomy motivates teams and enables faster decision-making.
* **Local Decision-Making**: Decisions happen at the team level, reducing delays.
* **Reduced Handoffs**: Minimizing handoffs between teams speeds up development.

**Agile Adoption Across the Organization**

* **Wall of Confusion**: Illustrates conflicting metrics between development and operations.
  + Development favors change, while operations prioritize stability.
* **Need for Alignment**: Agile adoption is ineffective if operations remain traditional.
* **DevOps Alignment**: DevOps aligns with Agile goals, enhancing speed, responsiveness, and quality.
* **Agile and DevOps Synergy**: Agile aims for faster delivery and responsiveness, while DevOps focuses on aligning IT with business needs and improving productivity.
* 

In summary, proper organization, autonomy, and alignment across the organization are critical for Agile success. Adopting DevOps practices alongside Agile can further enhance alignment and efficiency, ensuring that both development and operations teams contribute effectively to achieving business goals.

**What Agile Is Not**

1. **Not Just Iterative Development**: Agile is more than iterative development; it's about responsiveness to change and delivering value frequently.
   * **Fuzzy Front End Pitfall**: Organizations may mistakenly believe they're Agile by starting with Waterfall-like planning and then iterating without feedback or deployment.
   * **Water-Scrum-Fall**: Iterative development without feedback or deployment resembles Waterfall with Agile elements in the middle, not true Agile.
2. **Not Just Developers Working in Sprints**: Agile involves cross-functional teams comprising developers, testers, business analysts, and possibly operations personnel.
   * **Cross-Functional Teams**: Agile teams are diverse, not solely comprised of software engineers.
3. **No Mention of Agile Project Managers**: The Agile Manifesto does not advocate for Agile project managers.
   * **No Command and Control**: Agile teams are self-managed, with members assigning work to themselves.

In summary, Agile is not merely iterative development or developers working in sprints. It encompasses responsiveness to change, cross-functional collaboration, and self-managed teams, with a focus on delivering value frequently and adapting to evolving requirements.

**Upfront Planning vs. Iterative Planning**

**you'll be able to articulate why upfront planning leads to missed deadlines and summarize why iterative planning leads to greater accuracy.**

**Upfront Planning:**

* **Risk of Missed Deadlines**: Deciding everything at the project's outset, when knowledge is minimal, often leads to missed deadlines.
* **Navigating the Unknown**: Like navigating through a field of moving penguins, upfront planning can become impractical as conditions change.
* **Deciding with Limited Information**: At the project's beginning, teams have limited knowledge, making comprehensive planning challenging.
* **Limited Accuracy in Estimates**: Estimating tasks far in advance carries a high degree of uncertainty and often results in inaccurate predictions.

**Iterative Planning:**

* **Adjusting Plans as You Go**: Agile advocates for iterative planning, adjusting plans as knowledge increases and circumstances change.
* **Greater Accuracy in Estimates**: By planning for what is known and adjusting plans iteratively, estimates become more accurate.
* **Focus on Short-Term Planning**: Instead of trying to predict far into the future with low certainty, Agile emphasizes planning for the short term with high certainty.
* **Flexibility and Adaptability**: Iterative planning allows teams to adapt to changing circumstances, improving their ability to meet deadlines and deliver value.

In summary, while upfront planning may seem logical, it often leads to missed deadlines and inaccurate estimates due to the inherent uncertainty at the project's outset. Iterative planning, on the other hand, offers flexibility, adaptability, and greater accuracy by adjusting plans as knowledge increases and circumstances change.

1. **Untrained Role Transition Leads to Failure**:
   * Placing existing people in new roles without training leads to organizational failure in Agile transitions.
2. **Role Differences**:
   * Product Manager vs. Product Owner:
     + Product Manager: Business role, manages budget, focused on operational aspects of the business.
     + Product Owner: Scrum role, visionary, leads team experiments for sprint goals. Passing information between the stakeholders and the team, translating between business and technical goals.
   * Project Manager vs. Scrum Master:
     + Project Manager: Task manager, keeps to a plan, documents risks.
     + Scrum Master: Coach, focuses team on sprints, eliminates impediments, buffers interruptions.
   * Development Team vs. Scrum Team:
     + Development Team: Typically just software engineers.
     + Scrum Team: Cross-functional, includes developers, testers, security, business analysts, ops.
3. **Training and Role Understanding**:
   * Training is essential as the skills and focus required for Scrum roles are different from traditional roles.
   * Product Owners need vision and goal orientation, unlike product managers who focus on operational aspects.
   * Scrum Masters need to foster self-management within teams and actively remove impediments, contrasting with project managers' risk documentation.
4. **Agile Mindset from Leadership**:
   * Management needs to shift from fixed project plans and deadlines to short-term goals and customer satisfaction.
   * Emphasis should be on what can be delivered in the next sprint rather than long-term predictions.
5. **Cross-functional Teams**:
   * Effective Scrum implementation requires reorganizing into cross-functional teams rather than maintaining traditional development teams.
6. **Quote from Bill Cantor**:
   * Business leaders must move away from managing projects with fixed functions, timeframes, and costs as in Waterfall to effectively use Agile.
7. **Conclusion**:
   * Success in Agile requires proper training for new roles and a mindset shift from both team members and management to adapt to Agile principles.

**Key Points from the Passage on Kanban Boards**

1. **Kanban Board Overview**:
   * A Kanban board visually tracks the progress of tasks.
   * It shows what needs to be done, what is in progress, and what has been completed.
2. **Agile Mindset and Tools**:
   * Tools alone don’t make an organization agile; an Agile mindset is necessary.
   * Tools like ZenHub support Agile processes but require the right process and mindset first.
3. **Kanban Board Structure**:
   * **Columns/Pipelines**:
     + **New Issues**: Acts like an inbox for new tasks. New issues default here and are triaged to appropriate pipelines.
     + **Icebox**: Cold storage for long-term tasks not currently being worked on, preventing clutter in active pipelines.
     + **Product Backlog**: Contains all tasks planned for future work, not yet assigned to a sprint.
     + **Sprint Backlog**: Tasks selected for the current sprint, typically a two-week period.
     + **In-Progress**: Tasks that are actively being worked on. Assignments show who is working on each task. Once the development is complete developer typically create a pull request.
     + **Review/QA**: Tasks that are complete and awaiting review or quality assurance before final integration.
     + **Done**: Tasks that have been completed by developers, pending final acceptance by the product owner during the sprint review.
4. **Workflow on a Kanban Board**:
   * Tasks move from left to right as they progress through different stages.
   * New tasks enter from the left (New Issues) and move through stages until they reach the Done column.
   * Developers continuously pull tasks from the Sprint Backlog to In-Progress and then move them to Review/QA and finally to Done as they complete them.
5. **Benefits of ZenHub**:
   * Integrated with GitHub(plug-in to GitHub), provides a kanban board and project management reporting, reducing the need for developers to use multiple tools.
   * Keeps status updates current as it’s within the developers' primary workflow.
   * Provides a clear, real-time status of project progress for both developers and management.
6. **Kanban Board Simplicity**:
   * Kanban boards can be as simple as sticky notes on a physical whiteboard.
   * It is crucial to keep the board simple and focused on key stages to avoid unnecessary complexity.
7. **Summary**:
   * Kanban boards are essential for visualizing and managing the workflow in Agile projects.
   * The flow is from new tasks to completion, ensuring clarity and up-to-date tracking of project status.
   * Tools like ZenHub streamline this process by integrating project management within the development environment.

**ey Points from the Passage on User Stories and Epics**

1. **User Story Definition**:
   * A user story represents a piece of business value that the team can deliver within a completed increment.
   * It differs from traditional requirements by focusing on who needs the feature, what they need, and why they need it, emphasizing the business value.
2. **Components of a Good User Story**:
   * **Description**: A clear explanation of the business value, including who needs it, what they need, and why it is valuable.
   * **Assumptions**: Any assumptions or known details should be documented to guide the developer.
   * **Definition of Done (Acceptance Criteria)**: Clear criteria defining when the story is considered complete to avoid misunderstandings during reviews.
3. **User Story Description Format**:
   * **Role**: Identifies who the story is for (e.g., marketing manager, customer, sysadmin).
   * **Functionality**: Describes what is needed.
   * **Business Benefit**: Explains the value or benefit of the functionality.
4. **Acceptance Criteria and Gherkin Syntax**:
   * **Given**: Sets up the initial conditions.
   * **When**: Describes the action or event.
   * **Then**: Specifies the expected outcome or result.
   * This format ensures clarity and helps both stakeholders and developers understand the expected behavior and completion criteria.
5. **INVEST Acronym for Good User Stories**:
   * **Independent**: Stories should be self-contained and not depend on other stories.
   * **Negotiable**: Stories should be open to discussion and adjustment.
   * **Valuable**: Each story should provide clear value to the customer.
   * **Estimable**: Stories should be possible to estimate in terms of effort and time.
   * **Small**: Stories should be small enough to complete within a sprint.
   * **Testable**: There should be clear criteria to test if the story is done.
6. **Epics**:
   * **Definition**: Large user stories or ideas that are too big to be completed within a single sprint.
   * **Use**: Used for big ideas that need to be broken down into smaller, manageable user stories.
   * **Hierarchy**: Epics contain multiple smaller user stories that collectively fulfill the larger goal.
   * **Creation**: When a new issue or feature is too large, it is treated as an Epic and then broken down into smaller stories during backlog refinement.
7. **Workflow and Planning**:
   * **Sprint Planning**: Large stories (Epics) are broken down into smaller user stories to fit within the sprint cycle.
   * **Backlog Refinement**: Epics start as big ideas and are refined into smaller user stories over time to ensure they are manageable within sprints.

**Summary**

* **User stories** capture the "who," "what," and "why" of a feature, focusing on delivering business value.
* **Good user stories** should be complete, contain necessary assumptions, and have clear acceptance criteria using the Gherkin syntax.
* The **INVEST acronym** helps ensure user stories are effective and manageable.
* **Epics** are used for larger ideas that span multiple sprints and are broken down into smaller user stories for implementation.

**Key Points on Story Points and Estimating User Stories**

1. **Definition of Story Points**:
   * Story points are an abstract *metric used to estimate the difficulty of delivering and implementing a user story.*
   * They focus on relative size and complexity rather than precise time estimates.
2. **Components of Story Point Estimation**:
   * **Effort**: The amount of work required to complete the story.
   * **Complexity**: How complicated the story is to implement.
   * **Uncertainty**: The level of unknowns or new territory involved in the story.
3. **Challenges with Time Estimation**:
   * Humans are generally poor at estimating precise durations (e.g., hours or days).
   * Instead of predicting exact times, story points provide a relative measure, like t-shirt sizes (S, M, L, XL).
4. **Using Relative Sizing**:
   * Compare stories to one another to determine their relative size.
   * Use a simplified Fibonacci sequence (e.g., 3, 5, 8, 13) to represent small, medium, large, and extra-large stories.
   * Ensure the team agrees on what each size means (e.g., what a "medium" story looks like).
5. **Practical Example**:
   * Imagine comparing buildings of different heights: If one building is a "5," another slightly shorter one might be a "3," and a taller one might be an "8" or "13."
   * This analogy helps visualize how to assign story points based on relative size rather than exact measurements.
6. **Sizing Stories**:
   * Aim for stories that can be completed within a few days.
   * Large stories (e.g., those that might be rated "21" or higher) should be broken down into smaller stories to fit within a sprint.
7. **Anti-Patterns to Avoid**:
   * **Wall Clock Time**: Avoid equating story points with **exact times or dates** (e.g., "a 3 is three days").
   * Scrum masters and teams should resist translating story points into hours or days, as this contradicts the purpose of abstract estimation.
   * Focus on maintaining the relative and abstract nature of story points.
8. **Summary**:
   * Story points help estimate the difficulty and relative size of user stories, not the exact time required.
   * They should be agreed upon by the team and used to compare the size of different stories.
   * Avoid associating story points with specific durations to prevent inaccuracies and maintain the flexibility of agile planning.

**Key Points on Product Backlog and Creating Stories**

1. **Definition of Product Backlog**:
   * A product backlog is a list of all unimplemented stories.
   * It contains stories that are not in a sprint yet and are waiting to be worked on.
   * The backlog is typically ranked in order of business importance.
2. **Organization of the Backlog**:
   * The top of the backlog should be ranked more accurately than the bottom.
   * Only the next sprint or two need to be finely ranked; the rest can be less detailed.
   * Stories at the top should have more details than those further down, which can be fleshed out later.
3. **Example Project: Building a Counting Service**:
   * **Initial Requirement**: Create a service that counts things, like hits to a page.
   * **Additional Requirements**:
     + Allow multiple counters.
     + Ensure counters persist across restarts (needs a database).
     + Provide a way to reset counters.
4. **Creating Stories**:
   * Use the story template: "As [role], I need [function], so that [benefit]."
   * Example Stories:
     + "As a user, I need a service that has a counter so that I can keep track of how many times something has been done."
     + "As a user, I need to have multiple counters so that I can keep track of several counts at once."
     + "As a service provider, I need the counters to persist across restarts so that users don't lose their counts."
     + "As a system administrator, I need the ability to reset the counter so that I can start counting from zero again."
5. **Prioritizing the Backlog**:
   * New issues are created and then prioritized in the backlog.
   * Example prioritization:
     + Fundamental functionality (basic counting service) goes to the top.
     + Multiple counters can be deferred.
     + Persistence of counters is prioritized after the basic functionality.
     + Resetting counters follows the persistence feature.(Adminstrator
6. **)Ensuring Clear and Valuable Stories**:
   * Each story should clearly state who benefits from it, what is needed, and the business value.
   * Example structure: "As [role], I need [feature], so that [value]."
7. **Prioritization Process**:
   * Place fundamental stories at the top of the backlog.
   * Defer less immediate needs to later in the backlog.
   * Move high-priority items into the sprint backlog for immediate action.

**Summary:**

In this module, you learned about the product backlog, how to build and prioritize it, and how to create user stories from requirements. The key steps include defining the product backlog, converting requirements into user stories using the "As a, I need, So that" template, and prioritizing the stories based on business importance. This ensures that the team understands who benefits from each story and the value it provides, helping to maintain a clear and organiz

Here are the key points extracted from the passage:

1. **Backlog Refinement Process**:
   * Rank the backlog in priority order.
   * Break down larger stories into smaller ones.
   * Ensure stories near the top are small enough for a sprint and detailed enough for developers to start working on them.
2. **Participants in Backlog Refinement Meeting**:
   * Product Owner: Key person with the vision and responsible for creating stories.
   * Scrum Master: Assists the Product Owner in refining the backlog.
   * Development Team: Usually optional, but having a development lead or architect can help with technical questions.
3. **Goal of the Backlog Refinement Meeting**:
   * Have a ranked product backlog in order of importance.
   * Ensure the top stories contain sufficient detail for sprint planning.
4. **Handling New Issues**:
   * Start with the new issue column to ensure it is empty by the end of the meeting.
   * Triage new issues to decide if they should be moved to the product backlog, icebox, or rejected.
5. **Story Preparation**:
   * Stories should be made sprint-ready with all necessary details and acceptance criteria before sprint planning.
   * Use the complete story template with acceptance criteria and definition of done.
6. **Estimations and Clarifications**:
   * Provide rough estimates for stories to understand the backlog size.
   * Split large, vague items into smaller, clearer ones.
7. **Using the Kanban Board**:
   * Focus on clearing new issues, the icebox, and the product backlog.
   * Continuously refine and rank items in the product backlog.
8. **Example of Story Refinement**:
   * Add necessary functions and acceptance criteria to a story about a service counter.
   * Use Gherkin syntax to define the expected behavior and outcomes.
9. **Product Owner's Responsibility**:
   * Maintain a groomed backlog.
   * Start refinement by triaging new issues.

This summary encapsulates the process and key elements of backlog refinement, the roles involved, and the objectives of the refinement meetings.

1. **Backlog Refinement Goals**:
   * Add details to make stories sprint-ready.
   * Identify stories that are technical debt.
   * Rank the product backlog in preparation for sprint planning.
2. **Using Labels for Visualization**:
   * Labels help visualize work on the Kanban board.
   * Use a few distinct colors for labels to avoid confusion.
   * Standard GitHub labels include bugs (red), enhancements (cyan), and help wanted (green).
   * Add a technical debt label (yellow) to indicate caution.
3. **Labeling Stories**:
   * Assign appropriate labels to stories based on their nature (e.g., enhancement, technical debt).
   * Use labels to quickly identify the type of work in the backlog and on the Kanban board.
4. **Story Preparation Example**:
   * Define assumptions and acceptance criteria for each story.
   * Document any known details to guide developers (e.g., using Redis for a counter).
   * Ensure stories have a clear definition of done.
5. **Technical Debt**:
   * Technical debt refers to tasks that don't directly add customer value but are necessary for maintaining the system (e.g., code refactoring, setting up environments, updating libraries).
   * Regularly address technical debt to avoid system degradation.
   * Include technical debt tasks in each sprint to manage and reduce them over time.
6. **Backlog Refinement Tips**:
   * Refine the backlog every sprint, ideally mid-sprint (e.g., Wednesday before the sprint ends).
   * Aim to have two sprints worth of refined stories in the backlog to ensure readiness.
   * The more detail added during refinement, the less time needed for sprint planning.
7. **Kanban Board Management**:
   * Use labels and story points to visualize and manage the backlog.
   * Ensure stories are detailed and labeled correctly to facilitate smooth sprint planning.
8. **Continuous Refinement**:
   * Regular backlog refinement keeps the backlog healthy and ready for sprint planning.
   * Detailed backlog refinement minimizes the time spent on sprint planning meetings.

By following these practices, you can ensure your product backlog is well-organized, prioritize tasks effectively, and maintain a balance between new enhancements and technical debt.

**Sprint Planning in Scrum**

**Sprint Planning Meeting Overview:**

* **Purpose**: To determine what stories will be included in the next sprint and how much can be accomplished within the sprint duration, typically two weeks.
* **Outcome**: A sprint backlog containing the stories the team commits to completing in the sprint.

**Attendees:**

* Product Owner
* Scrum Master
* Development Team (including software engineers, testers, operations folks, business analysts)

**Importance of a Sprint Goal:**

* Each sprint should have a clear, articulated goal provided by the product owner.
* The sprint goal guides the team, ensuring they focus on delivering the intended capability or feature.
* Helps prevent over-engineering and keeps the team aligned on priorities.

**Sprint Planning Mechanics:**

1. **Review and Select Stories**:
   * The development team reviews the top stories from the product backlog.
   * Stories are moved to the sprint backlog **based on priority and the sprint goal.**
2. **Estimate Story Points**:
   * If not already assigned, the development team estimates the size of each story using story points.
   * Techniques like planning poker can be used to reach consensus on story estimates.
   * Ensuring everyone understands and agrees on the story points is crucial.
3. **Ensure Readiness**:
   * Confirm that each story has enough detail to start work without needing further clarification.
   * Stories should include clear requirements, assumptions, and acceptance criteria.
4. **Stop at Team Velocity**:
   * The team stops adding stories once they reach their velocity, the average number of story points they can complete in a sprint.
   * Velocity helps manage expectations and ensures the team does not overcommit.

**Creating a Milestone in ZenHub:**

* **Title**: Short and descriptive, e.g., "Sprint 1: Single Counter."
* **Description**: Clear goal for the sprint, e.g., "Get a single counter working and deploy to the cloud."
* **Dates**: Define the start and end dates for the sprint, typically two weeks.

**Steps in the Interface:**

1. **Title and Goal**:
   * Example: "Sprint 1: Single Counter" with the goal of deploying a single counter to the cloud.
2. **Assign Dates**:
   * Start: Monday, 7th
   * End: Friday, 18th

**Building the Sprint Backlog:**

1. **Assign Story Points**:
   * Example: A large story might be assigned 8 points.
   * Assign it to the sprint milestone and move it to the sprint backlog.
2. **Track Progress**:
   * Use the Kanban board to visualize stories and their points.
   * Ensure the total story points do not exceed the team's velocity.

**Backlog Refinement Tips:**

* Regularly refine the backlog every sprint, preferably mid-sprint.
* Aim to have at least two sprints' worth of refined stories.
* Detailed refinement reduces time spent during sprint planning.

**Key Takeaways:**

* **Product Owner's Role**: Presenting the sprint goal.
* **Development Team's Role**: Creating the sprint plan by moving stories into the sprint backlog.
* **Sprint Plan**: Complete once the team's velocity is reached, balancing work and capacity.

By following these steps, you can effectively plan and execute your sprint, ensuring the team is focused, aligned, and able to deliver value incrementally.

After watching this video, you'll be able to describe the daily workflow, determine which story you should work on next, and explain why you should not work on more than one story at a time.

**Daily Workflow in Scrum**

**Execution of the Sprint:**

* **Sprint Execution**: This phase involves iterating through design, code, test, and deploy within a two-week sprint. Each sprint should have a clear goal to guide the team's efforts.

**Daily Execution Steps:**

1. **Select the Next Highest Priority Item**:
   * At the start of each day, team members should take the next highest priority item from the sprint backlog that matches their skills.
   * Avoid picking favorite items or jumping to the middle of the list; focus on business importance.
2. **Assign the Story to Yourself**:
   * Once you select a story, assign it to yourself in the tracking tool (e.g., GitHub).
   * This allows everyone to see who is working on what and ensures transparency.
   * Move the story to the "In Progress" column to indicate active work.
3. **Focus on One Story at a Time**:
   * It is crucial to work on only one story at a time to maintain focus and efficiency.
   * Multitasking can lead to incomplete tasks and reduced productivity.
   * If you get blocked on a task, it is acceptable to start another while waiting for the blocker to be resolved.
4. **Updating the Kanban Board**:
   * Regularly update the Kanban board to reflect the current status of tasks.
   * This visual representation helps team members and stakeholders understand progress and workload distribution.

**Why Work on One Story at a Time:**

* **Completion Focus**:
  + You can only deliver a feature when it is fully completed, not partially done.
  + Concentrating on one story ensures that tasks are finished and ready for deployment.
* **Avoiding Context Switching**:
  + Switching between tasks can lead to inefficiency and errors.
  + Staying focused on a single task until completion maximizes productivity and quality.

**Handling Blockers:**

* If a blocker prevents progress on a story, it's okay to start a new task while the scrum master works on resolving the issue.
* Return to the blocked story as soon as the issue is resolved to maintain workflow continuity.

**Pull Requests and Reviews:**

1. **Creating a Pull Request**:
   * Once a story is completed, create a pull request and move the story to the "Review QA" column.
   * GitHub and ZenHub can automate this process, linking the pull request to the story.
2. **Review and Merge**:
   * The story moves to the "Done" column after the pull request is reviewed and merged into the main branch.
   * This indicates that the feature is completed and integrated into the codebase.
3. **Repeat the Process**:
   * After completing a story, return to the sprint backlog and start with the next highest priority item.

**Key Takeaways:**

* Keep the Kanban board updated to reflect the current work status.
* Always work on the highest priority story that matches your skills.
* Avoid working on multiple stories simultaneously to ensure task completion and maintain productivity.

By following this structured daily workflow, you can ensure efficient sprint execution, maintain focus, and deliver high-quality features within the sprint timeframe.

Here are the main points extracted from the passage:

1. **Daily Stand-Up Meeting**:
   * Happens at the same place and time every day.
   * Also known as the daily scrum.
2. **Purpose**:
   * For team members to update each other on their work.
   * Not a project status meeting, but a meeting for the development team to understand daily tasks and help each other.
3. **Participants**:
   * Scrum master: Essential for unblocking any team member.
   * Development team: All members should attend.
   * Product owner: Optional, and should not interfere unless necessary.
4. **Format and Conduct**:
   * Stand-up format to encourage brevity.
   * Time-boxed to 15 minutes to ensure efficiency.
   * Stand in a circle and answer three questions:
     1. What did I get done yesterday?
     2. What am I going to work on today?
     3. Are there any blockers or impediments in my way?
5. **Handling Issues**:
   * Scrum master to address any blockers immediately.
   * Developers should move to the next task if blocked for a while.
6. **Tabled Topics**:
   * Any topic not related to the three questions should be tabled for discussion after the meeting.
   * Maintain a list of tabled topics to ensure they are addressed later without exceeding the 15-minute limit.
7. **Definition and Purpose**:
   * A burndown chart measures the completion of story points versus the remaining story points within a given timeframe.
   * It helps quickly and easily assess if the team is on track to meet their sprint goal.
8. **Usage and Flexibility**:
   * Originally based on milestones, but can now also be used for sprints with tools like ZenHub.
   * Can measure progress against any milestone, not just sprints (e.g., preparing for a demo or a conference).
9. **Chart Components**:
   * **Vertical Axis**: Number of story points in the sprint.
   * **Horizontal Axis**: Number of days in the sprint.
   * **Vertical Bars**: Indicate weekends (non-working days).
   * **Optimal Path Line**: Represents the ideal burndown trajectory if story points are completed evenly over time.
   * **Actual Progress Line**: Shows the actual number of story points completed over time, with each dot representing a completed story.
10. **Reading a Burndown Chart**:
    * The chart shows a downward trend as story points are completed.
    * A deviation from the optimal path can indicate potential issues in meeting the sprint goal.
    * The chart helps the team understand their progress and rally to complete remaining tasks if they fall behind.
11. **Benefits**:
    * Simple and easy to read for developers.
    * Primarily for the development team's use to ensure they meet sprint goals.
    * Provides a visual forecast of the team's ability to complete all stories by the end of the sprint.
12. **Summary**:
    * A burndown chart tracks story points completed versus remaining.
    * Useful for monitoring progress toward any milestone.
    * Helps forecast the likelihood of achieving the sprint goal.

Here are the main points extracted from the passage about Sprint review meetings:

1. **Definition and Purpose**:
   * A Sprint review is a demo session where developers showcase the features implemented during the sprint.
   * It is a time to demonstrate the valuable, completed product increment and gather feedback.
2. **Participants**:
   * Everyone involved in the project should attend: the product owner, scrum master, developers, stakeholders, and customers.
   * Anyone interested in seeing the demo is welcome to participate.
3. **Conducting the Sprint Review**:
   * Live demonstrations of new features are presented.
   * The product owner reviews the features against acceptance criteria to determine if they are acceptable.
   * Accepted stories move from the Done column to the Closed column, indicating the delivery of functionality.
4. **Feedback and Iterative Development**:
   * Feedback from stakeholders and the product owner is crucial.
   * Feedback should be converted into new backlog items, potentially leading to new stories or additional functionality.
   * Iterative development thrives by incorporating real-time feedback and adjusting the product accordingly.
5. **Handling Rejected Stories**:
   * Rejected stories should be labeled and closed to avoid impacting velocity inaccurately.
   * New stories with the correct requirements are created for the next sprint.
   * This ensures the velocity reflects the work done, even if it wasn't what the product owner wanted.
6. **Key Outcomes**:
   * A Sprint review helps shape the future of the product based on stakeholder feedback.
   * The backlog is updated with new items derived from the feedback received during the review.

In summary, a Sprint review is a crucial meeting for demonstrating completed features, gathering feedback, and planning future work. It involves all project participants and focuses on iterative development to improve the product continuously.

1. **Definition and Purpose**:
   * A Sprint retrospective is a meeting to reflect on the sprint.
   * It measures the health of the process and the team, aiming for continuous improvement.
2. **Participants**:
   * The scrum master and the development team should attend.
   * The product owner is typically not invited to ensure the team can speak freely without feeling intimidated.
3. **Conducting the Sprint Retrospective**:
   * The development team must feel comfortable discussing any issues openly.
   * The retrospective involves answering three key questions:
     1. What did we do well? (What should we keep doing?)
     2. What did not go well? (What should we stop doing?)
     3. What should we change? (What do we want to do differently next time?)
4. **Actionable Outcomes**:
   * The scrum master documents the feedback and ensures that changes are made based on the team's input.
   * Not all suggested changes need to be implemented, but it's crucial that some changes are made to show that the feedback is valued and acted upon.
5. **Goal**:
   * The primary goal is improvement for the next sprint.
   * Ensuring the team feels heard and sees tangible improvements from these meetings is vital for maintaining a healthy and productive team dynamic.
6. **Summary**:
   * A Sprint retrospective is essential for reflecting on the sprint's successes and areas for improvement.
   * The development team must be comfortable speaking freely.
   * The meeting results in actionable changes to enhance the team's performance in future sprints.
7. **Importance of Metrics**:
   * Measuring performance is crucial for improvement.
   * High-performing teams set baselines, establish goals, and measure progress against those goals.
8. **Vanity Metrics vs. Actionable Metrics**:
   * **Vanity Metrics**: These are superficial and don't provide actionable insights (e.g., number of hits on a website).
   * **Actionable Metrics**: These provide meaningful data that can inform decisions and improvements (e.g., results from A/B testing).
9. **Setting Baselines and Goals**:
   * Establish a baseline to understand current performance (e.g., current time required for deployment).
   * Set goals for improvement and measure progress over time.
10. **Top Four Actionable Metrics**:
    * **Mean Lead Time**: Time from an idea to delivery to the customer.
    * **Release Frequency**: How often you can release updates or new features.
    * **Change Failure Rate**: The rate at which changes or releases fail.
    * **Mean Time to Recovery**: The time it takes to recover from a failure or downtime.
11. **Examples of Metrics for Improvement**:
    * **Reduce Time to Market**: Shorten the time required to deliver new features.
    * **Increase Product Availability**: Improve the uptime and availability of the product.
    * **Reduce Deployment Time**: Speed up the release process.
    * **Increase Detection of Defects in Testing**: Catch more bugs during testing rather than in production.
    * **Provide Timely Performance and User Feedback**: Improve the speed and quality of feedback loops from customers to the development team.
12. **Overall Goals**:
    * Ensure that the metrics used are actionable and lead to continuous improvement.
    * Taking a baseline is essential before measuring any changes.
    * Use actionable metrics to improve the team’s performance continuously.

ere's a summary of the end-of-sprint activities, handling unfinished stories, and preparing for the next sprint as described in the video:

**End of Sprint Activities**

1. **Closing Out Completed Work**:
   * Move all items in the "Done" column to the "Closed" column.
   * Close out the current sprint milestone to ensure velocity is accurately recorded.
2. **Creating a New Sprint Milestone**:
   * Either create a new milestone immediately or wait until the next sprint planning meeting.

**Handling Unfinished Stories**

1. **Untouched Stories**:
   * Stories that haven't been started should be moved to the top of the product backlog.
   * Avoid moving them directly to the next sprint to ensure they are still a priority.
   * Unassign these stories from the current sprint milestone.
2. **Partially Completed Stories**:
   * Do not move unfinished stories directly to the next sprint as it will distort velocity.
   * Adjust the story points to reflect the work done (e.g., if a story was 8 points but only half was completed, adjust it to 4 points).
   * Close the partially completed story and write a new story for the remaining work.
   * Assign the remaining story points to the new story and place it in the product backlog.

**Preparing for the Next Sprint**

1. **Closing Out the Current Sprint**:
   * Ensure all stories in the current sprint are closed.
   * Reassign unfinished stories by closing them and creating new stories if needed.
   * Move any remaining stories out of the sprint milestone and into the product backlog.
2. **Creating a New Sprint**:
   * Create a new sprint milestone either at the end of the current sprint or at the beginning of the next sprint.

**Key Points to Remember**

* Always give developers credit for the work done on unfinished stories to maintain accurate velocity.
* Split unfinished stories into new stories for the remaining work.
* Close each sprint milestone to reflect accurate velocity.
* Create a new milestone for the upcoming sprint to organize the next set of tasks.

By following these steps, you ensure a smooth transition between sprints, maintain accurate tracking of the team's progress, and set up the team for success in future sprints.

fter watching this video, you'll be able to identify anti-patterns to avoid when practicing Scrum and describe a checklist for measuring a healthy Scrum team.

**Anti-Patterns to Avoid in Scrum**

1. **No Real Product Owner**:
   * Ensure there is a single, clear product owner with a vision for the team.
   * Avoid having multiple product owners, which can lead to conflicting priorities.
2. **Teams Too Large**:
   * Keep teams small, ideally 5 to 9 members, to maintain effective communication and collaboration.
3. **Non-Dedicated Teams**:
   * Team members should be fully dedicated to the Scrum team without being pulled into other projects.
4. **Geographically Dispersed Teams**:
   * Preferably, have team members in the same time zone or at least ensure there are at least two members per geography to facilitate collaboration.
5. **Siloed Teams**:
   * Avoid functional silos; ensure teams are cross-functional and can handle all aspects of development without external dependencies.
6. **Teams Not Self-Managing**:
   * Teams should be self-managing, selecting tasks from the backlog rather than being assigned tasks by others.

**Checklist for Measuring a Healthy Scrum Team**

1. **Accountability**:
   * Every team member, including the Scrum Master and Product Owner, is accountable for their role and contributes to the team's success.
2. **Small Sprints**:
   * Work in manageable sprint lengths, ideally two weeks, to ensure frequent delivery and feedback.
3. **Ordered Product Backlog**:
   * Maintain a prioritized and well-defined product backlog to ensure clarity and focus on the most important tasks.
4. **Sprint Backlog**:
   * Have a clear sprint backlog that visualizes the remaining work and tracks progress throughout the sprint.
5. **Sprint Planning and Forecasting**:
   * Conduct proper sprint planning to establish a sprint goal and select the backlog items to be completed.
6. **Daily Scrum**:
   * Use daily scrums to plan and replan work as needed, ensuring that the team stays on track and adjusts to any changes.
7. **Delivering a Done Increment**:
   * Aim to deliver a usable increment of the product at the end of each sprint to demonstrate progress and gather feedback.
8. **Stakeholder Feedback**:
   * Engage stakeholders in sprint reviews to gather valuable feedback and make necessary adjustments.
9. **Updating the Product Backlog**:
   * Update the product backlog based on feedback from sprint reviews and new insights gained during the sprint.
10. **Alignment on Work in Progress**:
    * Ensure alignment between the product owner, development team, and Scrum Master on the current work and priorities.
11. **Sprint Retrospective**:
    * Conduct retrospectives to reflect on the sprint, identify what went well and what needs improvement, and make plans for continuous improvement.

By following these guidelines and avoiding common anti-patterns, you can maintain a healthy and effective Scrum team that continuously improves and delivers valuable products.