Development Approach

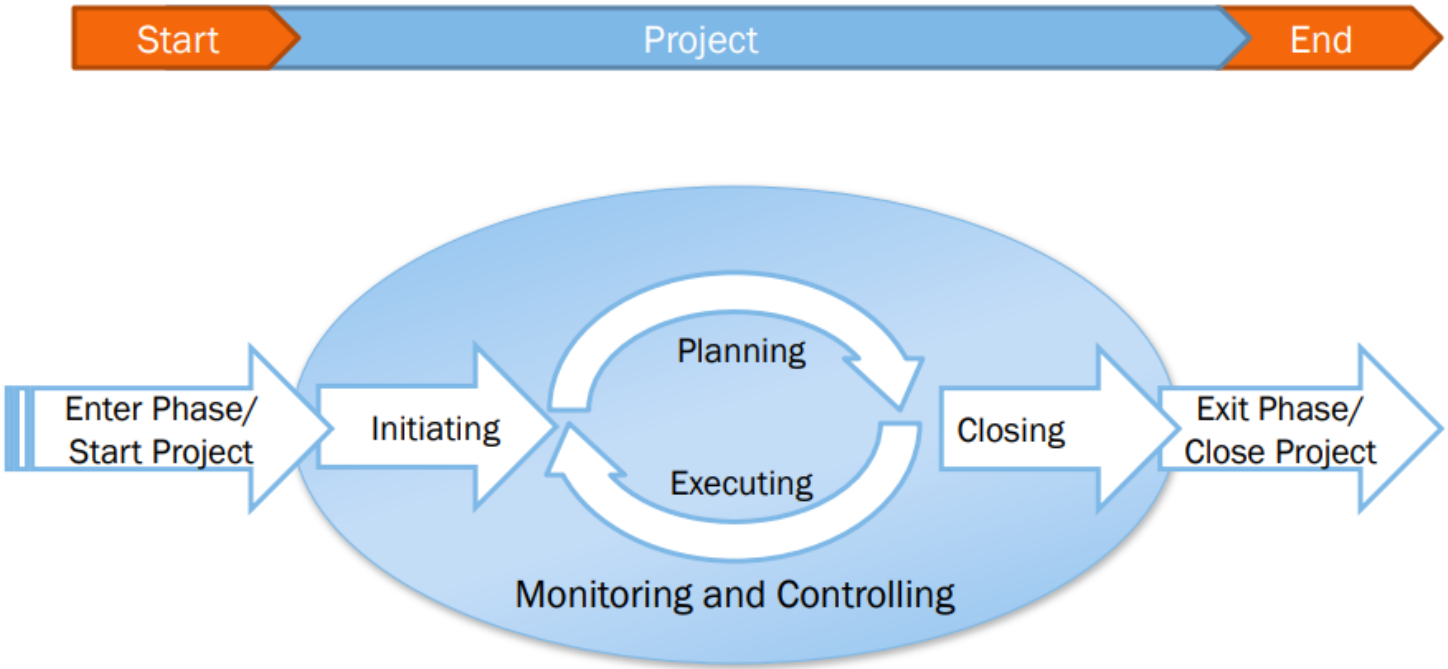
SE423: Software Project Management

Outline

- Methodology Concepts
- Software Development Process
- Agile Project Management
- DevOps

Methodology Concepts

Project Life Cycle



Methodology Concepts

- Process
  - A collection of work activities, actions and tasks that are performed when some work product is to be created
- Process Model or Life Cycle
  - An abstract description of a software process that presents one view of that process
  - Waterfall, Iterative, Spiral, Evolutionary, etc.
- Process Methodology or Methodology
  - Instantiations of process models – tend to be prescriptive.
  - The conventions that a group agrees to – “How we work around here”

What a Methodology Addresses?

- Introducing new people to the process
- Substituting people
- Delineating responsibilities
- Demonstrating visible progress
  - McConnell's definition for visibility: "The ease & accuracy with which it is possible to assess the status of a project's cost, schedule, functionality, or other characteristic."

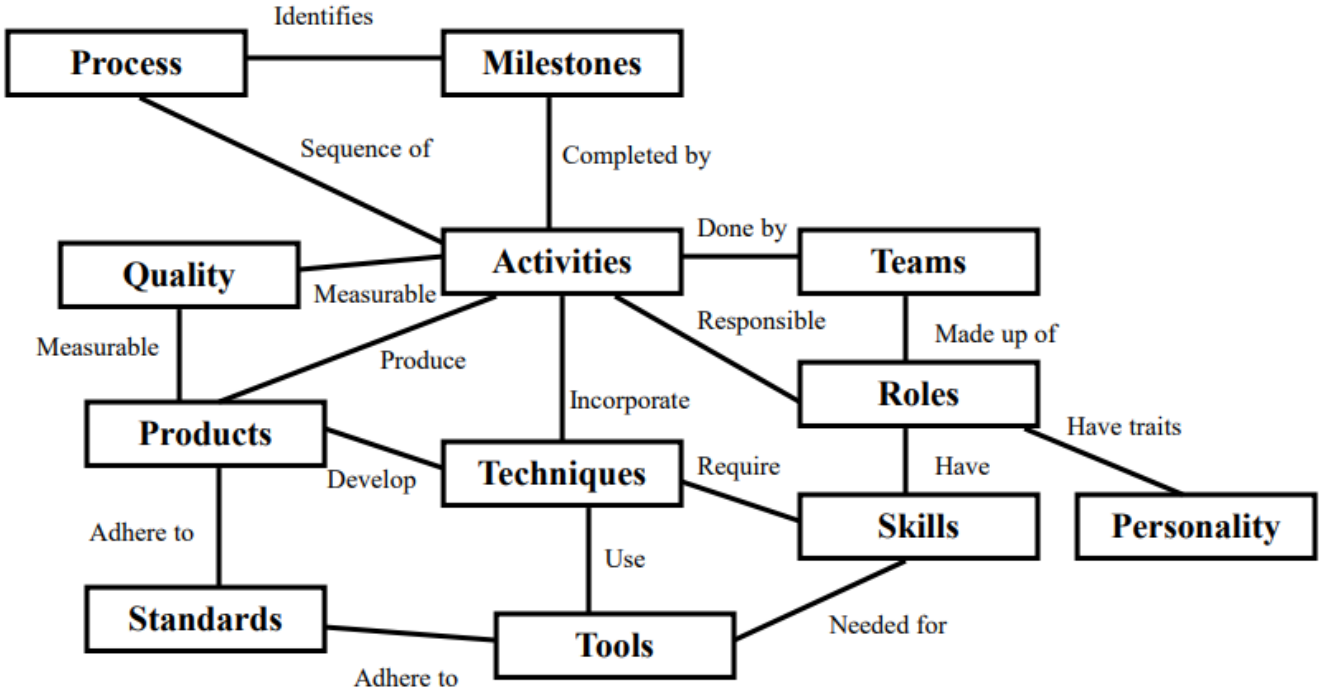
Evaluating a Methodology

- How rapidly you can substitute or train people.
- How great an effect it has on the sales process.
- How much freedom (or how constraining) it is to people on the project.
- How fast it allows people to respond to changing situations.
- How well it “protects” the organization – legally or from other damages.

Elements of a Methodology

- Teams
- Roles
- Skills
- Techniques
- Activities
- Process
- Work Products
- Milestones
- Standards
- Quality

Cockburn Methodology Structure



Elements of a Methodology

- Activities
  - the meetings, reviews, and other general activities the person must attend, generate or do.
- Process
  - the sequencing of activities over time with pre and post conditions for the activities.
- Milestones
  - events marking progress or completion. Milestones mark an instant in time and are either fully met or not met.

Elements of a Methodology

- Teams
  - how you group the people and how you assign people to roles.
- Roles
  - the job descriptions of team members: project manager, requirements gatherer, tester, program designer, etc. Roles may need to consider the personalities of the people being assigned to those roles.
- Skills
  - the skills team members should have in order to assume responsibility for their role on the project.

Elements of a Methodology

- Products
  - what each person or team hands to another person or team: use cases, class designs, test specifications, framework documentation, interface definitions, etc.
- Techniques
  - the techniques the person uses in their work: requirement session facilitation, Java programming, use case modeling, etc.
- Tools
  - what tools the people use in their jobs, either within a technique or to produce a deliverable according to the standard.

Elements of a Methodology

- Standards
  - what is permitted or not permitted in the work product. There are notational standards (which includes the programming language), management and decision standards, and project conventions. The methodology may leave certain standards open, to be determined on the project.
- Quality
  - what rules, issues or concerns are to be tracked for each deliverable or activity.

IT project life cycles

- IT projects have two concurrent life cycles:
  - *Project life* cycle (PLC) encompasses all activities of project, including the System/Software Development Life Cycle (SDLC)
  - PLC is directed toward achieving *project* requirements
  - SDLC is directed toward achieving *product* requirements
- Both life cycle models are needed to manage an IT project
  - PLC alone will not adequately address system development concerns
  - SDLC alone will not adequately address business and product integration concerns
  - Effective integration of the two life cycle models is essential to improving the likelihood of project success

Software Development Process

Software Development Process

- Ad hoc
  - Code and Fix
  - Rapid Prototyping
- Prescriptive
  - Linear/sequential (Classic and Waterfall)
  - Evolutionary (Iterative/incremental or spiral)
  - Unified Process
- Adaptive
  - Lean and agile methods

Plan-driven Methodology

- The “traditional” way to develop software
- Based on system engineering and quality disciplines (process improvement)
- Standards developed from DoD & industry to make process fit a systems approach
- Values well defined work products

Plan-driven Characteristics

- Focus on repeatability and predictability
- Defined, standardized, and incrementally improving processes
- Thorough documentation
- A software system architecture defined up-front
- Detailed plans, workflow, roles, responsibilities, and work product descriptions
- Process group containing resources for specialists: process monitoring, controlling, and educating
- On-going risk management
- Focus on verification and validation

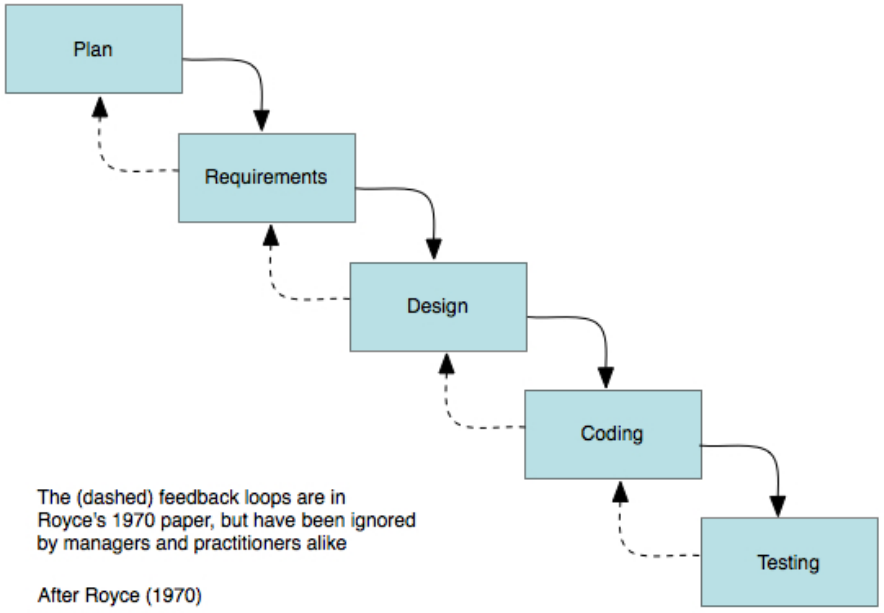
Sequential ('waterfall') methodology

- The term *waterfall* was coined by Winston Royce in a 1970 paper titled *Managing the Development of Large Software Systems*, in the Proceedings of IEEE WESCON
- The paper used the sequential waterfall approach as an example of an ***ill-conceived, risk-prone practice*** for developing large systems
- Royce advocated a series of iterative feedback loops among the development stages, incrementally gaining learning value from working software
- Instead of adopting the approach Royce advocated, managers and practitioners adopted its *anti-form, without* feedback loops

Waterfall SDLC

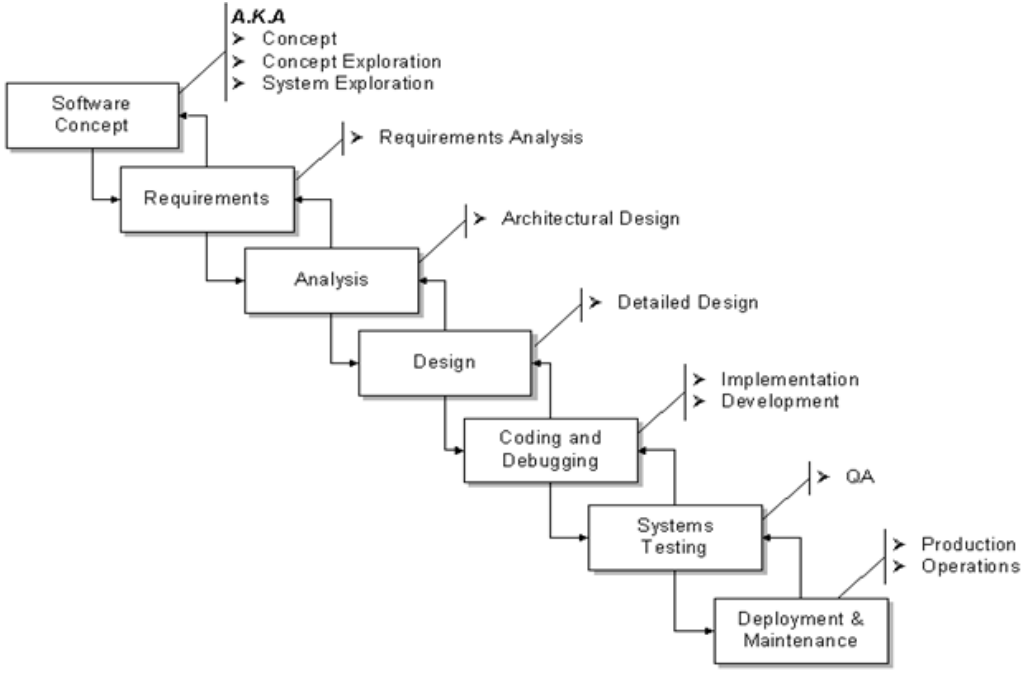
- Each phase is marked by completion of Deliverables
- The primary software project phases:
  - Requirements
  - Analysis
  - Design
  - Construction
  - Quality Assurance (aka Testing)
  - Deployment

Waterfall SDLC



The (dashed) feedback loops are in Royce's 1970 paper, but have been ignored by managers and practitioners alike

After Royce (1970)



Waterfall system development model

- Highly-sequential process
- Failure symptoms:
  - Protracted integration and late design breakage
  - Late risk resolution
  - Requirements-driven functional decomposition
  - Adversarial stakeholder relationships
  - Focus on documents and review meetings
- Still followed (in name or practice) by many organizations, usually a modified version

Waterfall system development model

Sequential: suitable projects and management approaches

- A sequential SDLC is suitable for projects with:
  - Clear, unambiguous, and stable user requirements
  - Familiar, proven technology
  - Low complexity
  - Adequate time
  - Stable schedule
- A project meeting most of these criteria can use conventional project management practices, such a big, up-front planning and conventional risk assessment

Evolutionary methodologies

- An *evolutionary methodology* follows an iterative and incremental approach that allows the start of development with incomplete, imperfect knowledge
- An iterative and incremental process is like solving a jigsaw puzzle: neither top-down nor bottom-up but accretionary and convergent
- An iterative and incremental process offers these advantages:
  - Logical progress toward evolving a robust architecture
  - Effective management of changing requirements
  - Effective means to address changes in planning
  - Ability to perform continuous integration
  - Early understanding of the system (the ‘Hello world!’ effect)
  - Ongoing risk assessment
- Evolutionary methodologies are incremental at both the macro (project- scale) and micro (working team) process levels

Iterative system development model

- Non-linear approach to system development
- Incorporates top five principles of modern development processes:
  - *Architecture first*. Provides the central design element
  - *Iterative life-cycle process*. Provides the essential risk management element
  - *Component-based development*. Provides the technology element
  - *Change management environment.* Provides the control element
  - *Round-trip engineering*. Provides the automation element

5,000 foot view of Iterative SDLC

- Iterative SD model defines four life-cycle phases:
  - Inception
  - Elaboration
  - Construction
  - Transition
- We iterate through each phase, and repeat as needed.
- Now, for a quick survey of the phases…

Inception phase

- Essential activities
  - *Formulate product scope*. Capture requirements and operational concept
  - *Perform feasibility analysis*. Determine whether the organization has the resources and technical capabilities to meet customer's needs
  - *Synthesize the system architecture*. Evaluate essential system design constraints and trade-offs, as well as available solutions
  - *Plan and prepare business case*. Address risk management, staffing, iteration plans, cost, and infrastructure

Elaboration phase

- Most critical phase of the four
- Essential activities
  - *Elaborate the vision*. Detail elements of the vision that drive architectural or planning decisions
  - *Elaborate the process and infrastructure*. The construction process and environment are established here
  - *Elaborate the architecture and select reusable (internal or COTS) components*. Baseline the architecture as quickly as possible and demonstrate that the architecture will support the vision at reasonable cost in reasonable time

Construction phase

- Essential activities
  - *Achieve useful versions* (intermediate, alpha, beta, and other test releases)
  - *Perform resource management, control, and process optimization*
  - *Complete component development and test*
  - *Assess product releases against acceptance criteria*

Transition phase

- Essential activities
  - *Perform deployment-specific engineering tasks*. Commercial packaging and production, sales kit development, field personnel training
  - *Assess deployment baselines against complete vision and acceptance criteria.* Examine and compare what is being delivered to what was envisioned and delineated by acceptance criteria
  - Plan for next iteration

Comparative expenditure profiles

Waterfall		Iterative	
Activity	**Cost**	**Cost**	**Activity**
Management	5%	10%	Management
Requirements	5%	10%	Requirements
Design	10%	15%	Design
Code & Unit Testing	30%	25%	Implementation
Integration & Test	40%	25%	Assessment
Deployment	5%	5%	Deployment
Environment	5%	10%	Environment
Total	100%	100%	Total

Suitable Projects And Management Approaches

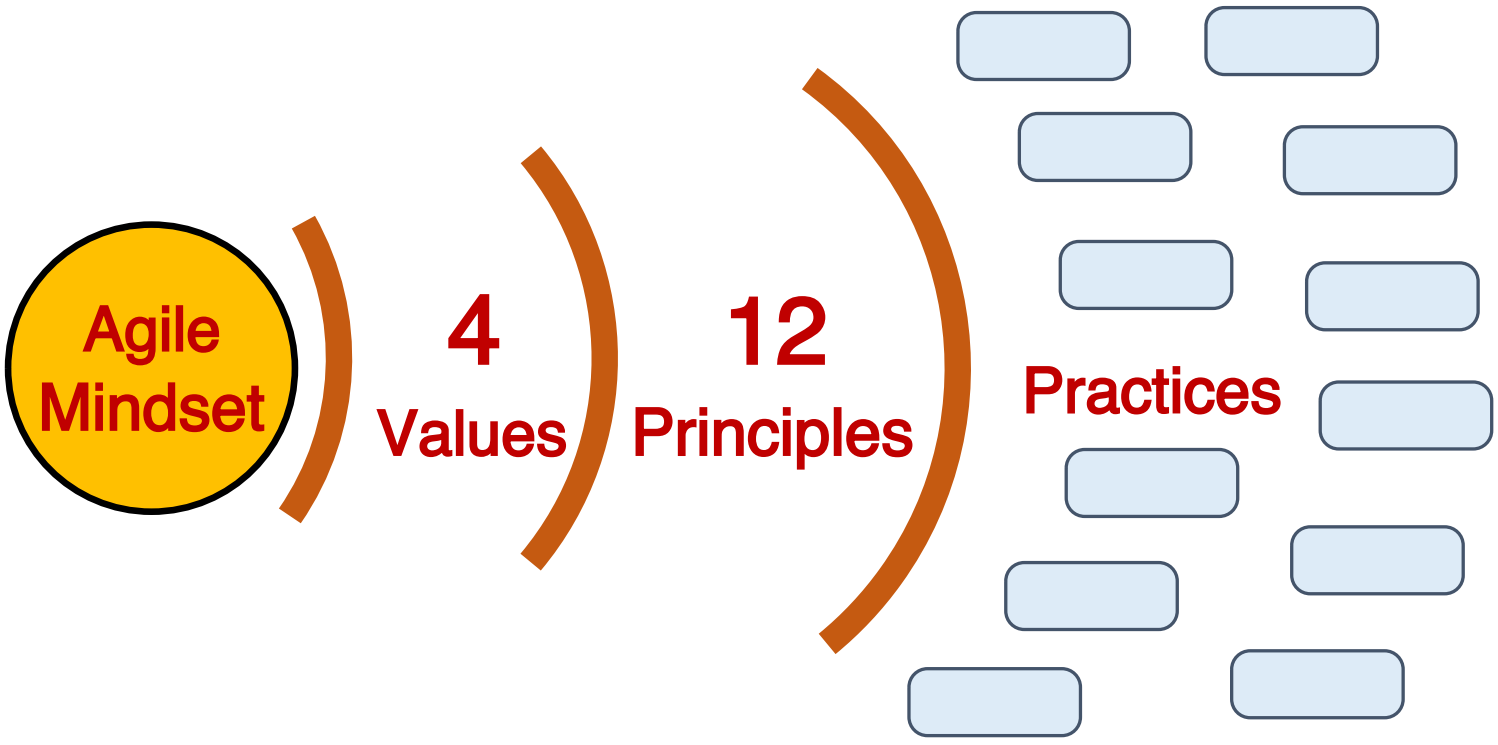
- An evolutionary SDLC is suitable for projects with:
  - Reasonably–but not perfectly–clear user requirements
  - Unfamiliar or unproven technology
  - High complexity
  - Short time schedule
  - Schedule variability
- Such a project would use rolling wave planning rather than big, up-front planning and use a continuous, adaptive approach to risk assessment and management

Agile Project Management

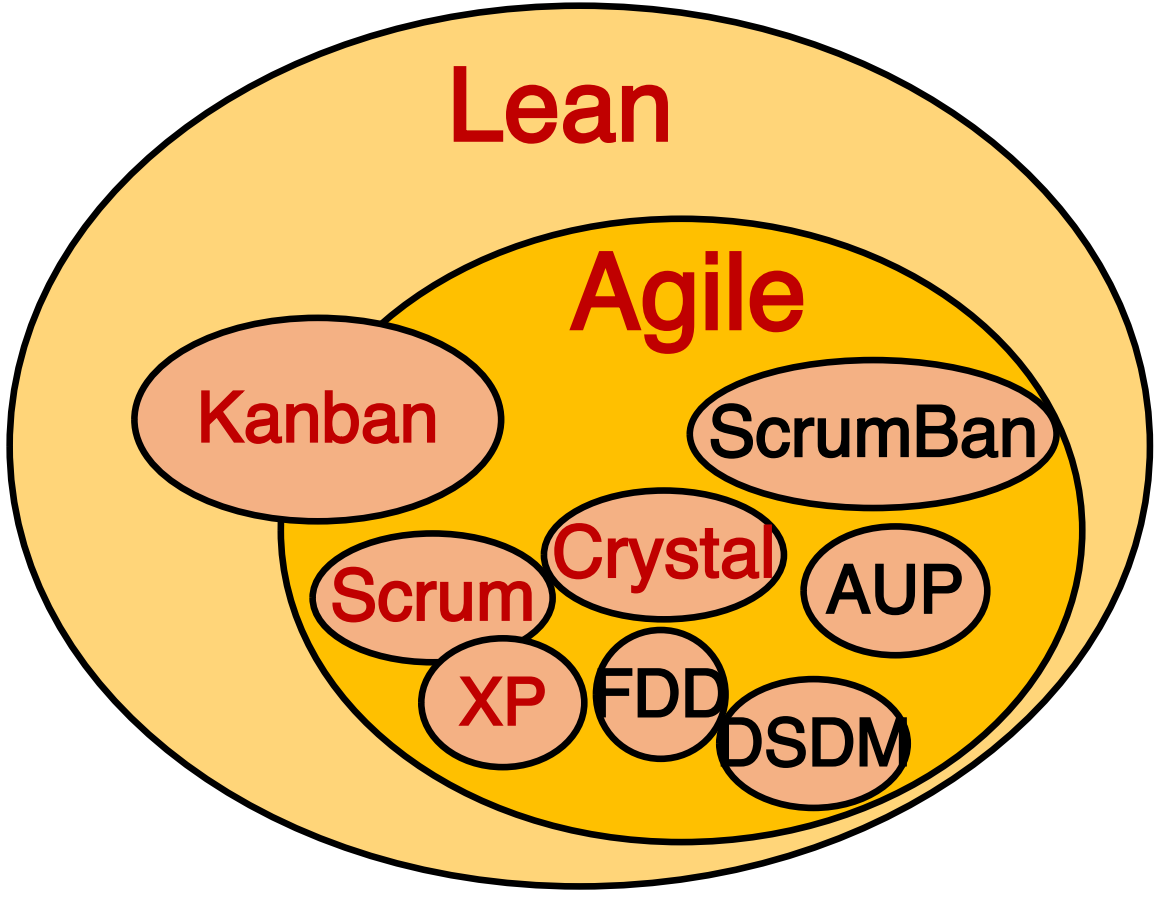
Agile software engineering

- Software products must be brought to market quickly so rapid software **development and delivery** is essential.
- Virtually all software products are now developed using an **agile approach**.
- **Agile** software engineering focuses on **delivering** functionality **quickly**, **responding** to **changing** product specifications and **minimizing** development **overheads**.

Agile Manifesto Values, Principles, and Common Practices



Agile is a Blanket Term for Many Approaches



Agile Manifesto and Mindset

Agile is a mindset defined by 4 values, guided by 12 principles, and manifested through many different practices.

Agile practitioners select practices based on their needs.

The Four Values of the Agile Manifesto

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

1. individuals and interactions over processes and tools
2. working software over comprehensive documentation
3. customer collaboration over contract negotiation
4. 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.

The Twelve Principles Behind the Agile Manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

The Twelve Principles Behind the Agile Manifesto

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Development Principles

- **Involve the customer**

  Involve customers closely with the software development team. Their role is to provide and prioritize new system requirements and to evaluate each increment of the system.

- **Embrace change**

  Expect the features of the product and the details of these features to change as the development team and the product manager learn more about it. Adapt the software to cope with changes as they are made.

- **Develop and deliver incrementally**

  Always develop software products in increments. Test and evaluate each increment as it is developed and feed back required changes to the development team.

Agile Development Principles

- **Maintain simplicity**
  Focus on simplicity in both the software being developed and in the development process. Wherever possible, do what you can to eliminate complexity from the system.
- **Focus on people, not things**
  Trust the development team and do not expect everyone to always do the development process in the same way. Team members should be left to develop their own ways of working without being limited by prescriptive software processes.

Agile software engineering

- A large number of ‘agile methods’ have been developed.
  - There is no ‘best’ agile method or technique.
  - It depends on who is using the technique, the development team and the type of product being developed

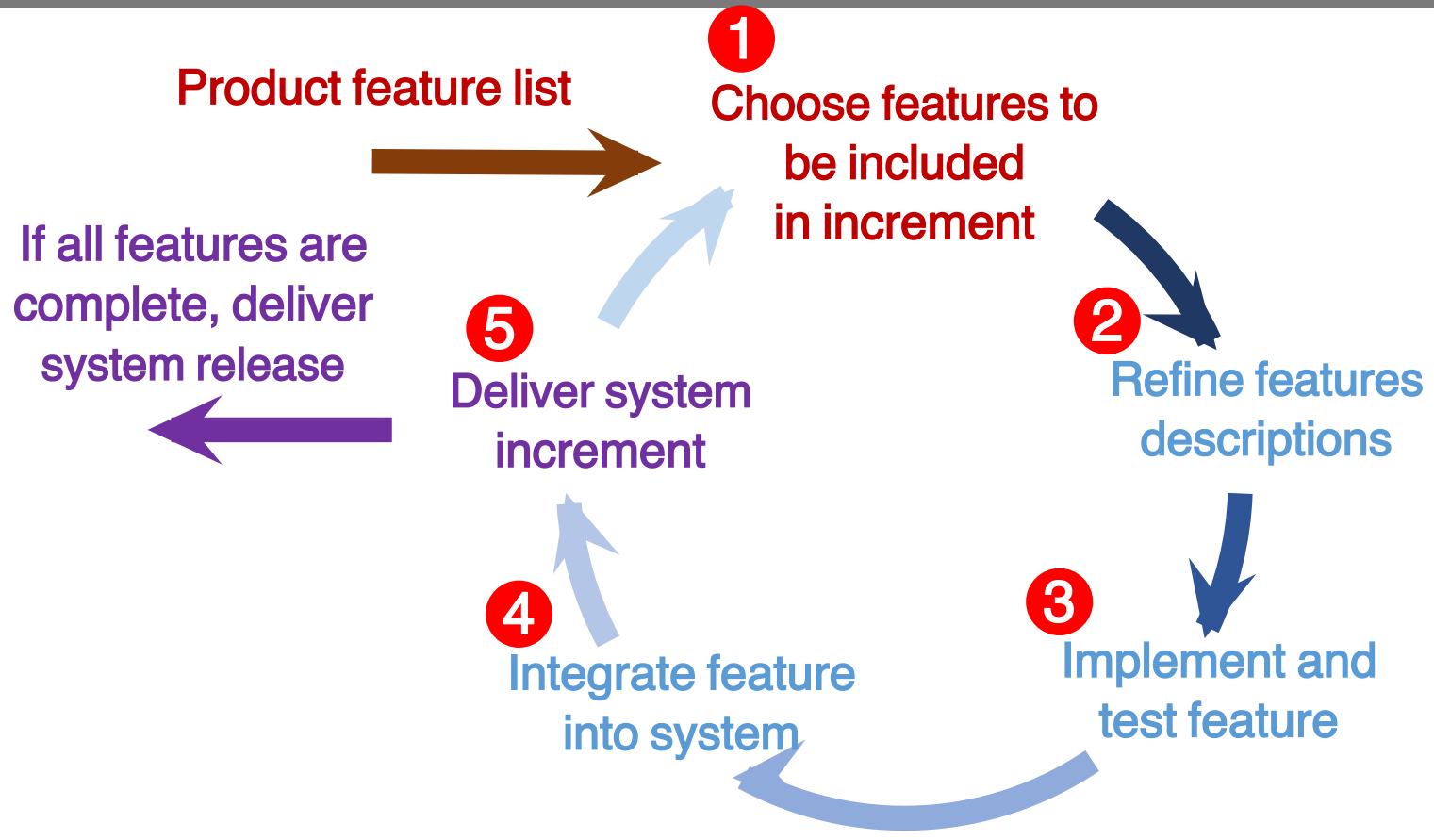
Agile methods

- Plan-driven development evolved to support the engineering of large, long-lifetime systems
  - This approach is based on controlled and rigorous software development processes that include detailed project planning, requirements specification and analysis and system modelling.
  - However, plan-driven development involves significant overheads and documentation and it does not support the rapid development and delivery of software.
- Agile methods were developed in the 1990s to address this problem.
  - These methods focus on the software rather than its documentation, develop software in a series of increments and aim to reduce process bureaucracy as much as possible.

Incremental development

- All **agile methods** are based around **incremental development and delivery**.
- Product development focuses on the software features, where a feature does something for the software user.
- With incremental development, you start by prioritizing the features so that the most important features are implemented first.
  - You only define the details of the feature being implemented in an increment.
  - That feature is then implemented and delivered.
- Users or surrogate users can try it out and provide feedback to the development team. You then go on to define and implement the next feature of the system.

Incremental development



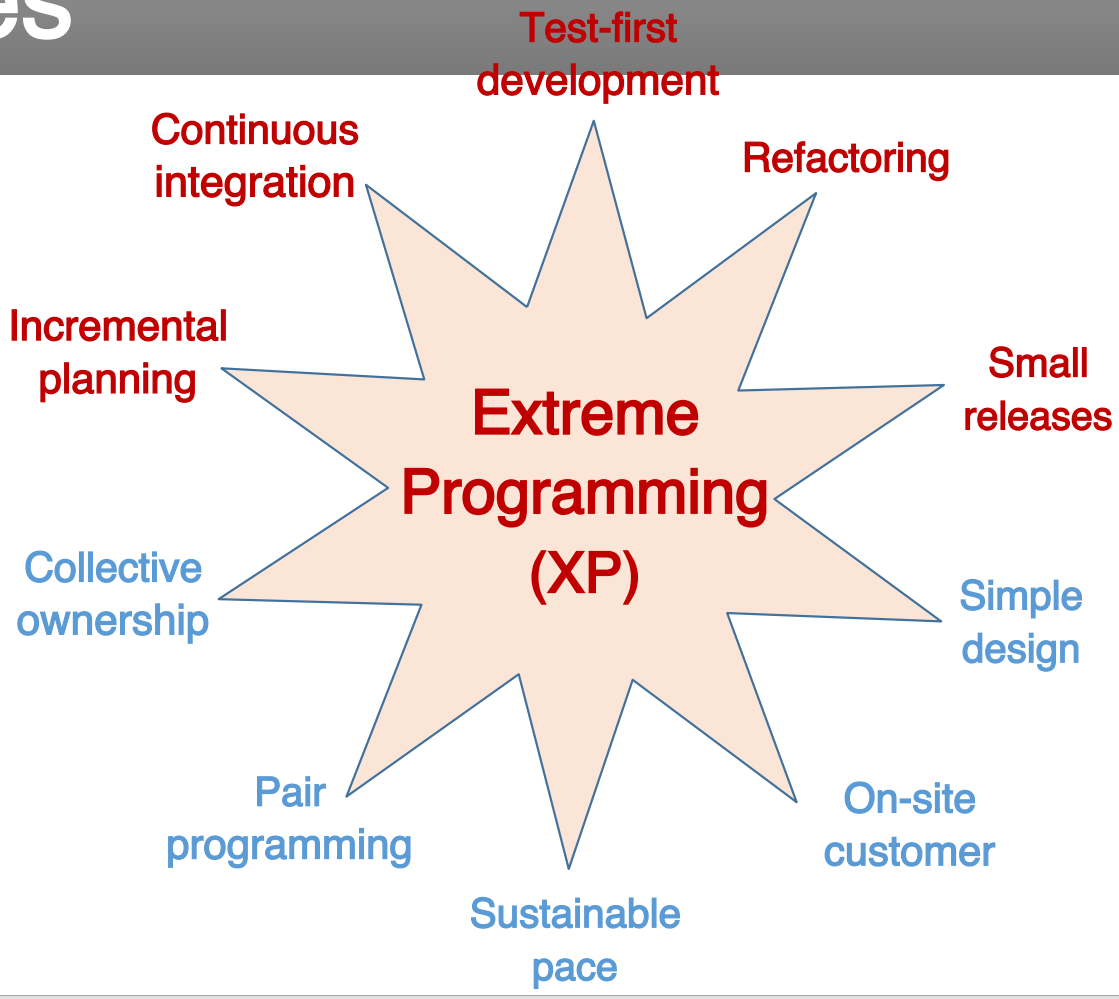
Incremental development activities

1. Choose features to be included in an increment
   Using the list of features in the planned product, select those features that can be implemented in the next product increment.
2. Refine feature descriptions
   Add detail to the feature descriptions so that the team have a common understanding of each feature and there is sufficient detail to begin implementation.
3. Implement and test
   Implement the feature and develop automated tests for that feature that show that its behaviour is consistent with its description.
4. Integrate feature and test
   Integrate the developed feature with the existing system and test it to check that it works in conjunction with other features.
5. Deliver system increment
   Deliver the system increment to the customer or product manager for checking and comments. If enough features have been implemented, release a version of the system for customer use.

Extreme programming

- The most influential work that has changed software development culture was the development of Extreme Programming (XP).
- The name was coined by Kent Beck in 1998 because the approach was developed by pushing recognized good practice, such as iterative development, to ‘extreme’ levels.
- Extreme programming focused on 12 new development techniques that were geared to rapid, incremental software development, change and delivery.
- Some of these techniques are now widely used; others have been less popular.

Extreme Programming Practices



Widely adopted XP practices

- Incremental planning/user stories
  - There is no ‘grand plan’ for the system. Instead, what needs to be implemented (the requirements) in each increment are established in discussions with a customer representative.
  - The requirements are written as user stories.
  - The stories to be included in a release are determined by the time available and their relative priority.

Widely adopted XP practices

- Small releases
  - The minimal useful set of functionality that provides business value is developed first.
  - Releases of the system are frequent and incrementally add functionality to the previous release.

Widely adopted XP practices

- Test-driven development
  - Instead of writing code then tests for that code, developers write the tests first.
  - This helps clarify what the code should actually do and that there is always a ‘tested’ version of the code available.
  - An automated unit test framework is used to run the tests after every change.
  - New code should not ‘break’ code that has already been implemented.

Widely adopted XP practices

- Continuous integration
  - As soon as the work on a task is complete, it is integrated into the whole system and a new version of the system is created.
  - All unit tests from all developers are run automatically and must be successful before the new version of the system is accepted.

Widely adopted XP practices

- Refactoring
  - Refactoring means improving the structure, readability, efficiency and security of a program.
  - All developers are expected to refactor the code as soon as potential code improvements are found.
  - This keeps the code simple and maintainable.

Widely adopted XP practices

- Incremental planning/user stories
  - There is no ‘grand plan’ for the system. Instead, what needs to be implemented (the requirements) in each increment are established in discussions with a customer representative.
  - The requirements are written as user stories.
  - The stories to be included in a release are determined by the time available and their relative priority.

Scrum

- Software company managers need information that will help them understand how much it costs to develop a software product, how long it will take and when the product can be brought to market.
- Plan-driven development provides this information through long-term development plans that identify deliverables - items the team will deliver and when these will be delivered.
- Plans always change so anything apart from short-term plans are unreliable.
- Scrum is an agile method that provides a framework for agile project organization and planning. It does not mandate any specific technical practices.

Scrum Terminology

- Scrum
  A daily team meeting where progress is reviewed and work to be done that day as discussed and agreed.
- Sprint
  A short period, typically two to four weeks, when a product increment is developed.
- ScrumMaster
  A team coach who guides the team in the effective use of Scrum.

Scrum Terminology

- Product
  The software product that is being developed by the Scrum team.
- Product owner
  A team member who is responsible for identifying product features and attributes. They review work done and help to test the product.
- Product backlog
  A to-do list of items such as bugs, features and product improvements that the Scrum team have not yet completed.

Scrum Terminology

- Development team
  A small self-organising team of five to eight people who are responsible for developing the product.
- Potentially shippable product increment
  The output of a sprint which should be of high enough quality to be deployed for customer use.
- Velocity
  An estimate of how much work a team can do in a single sprint.

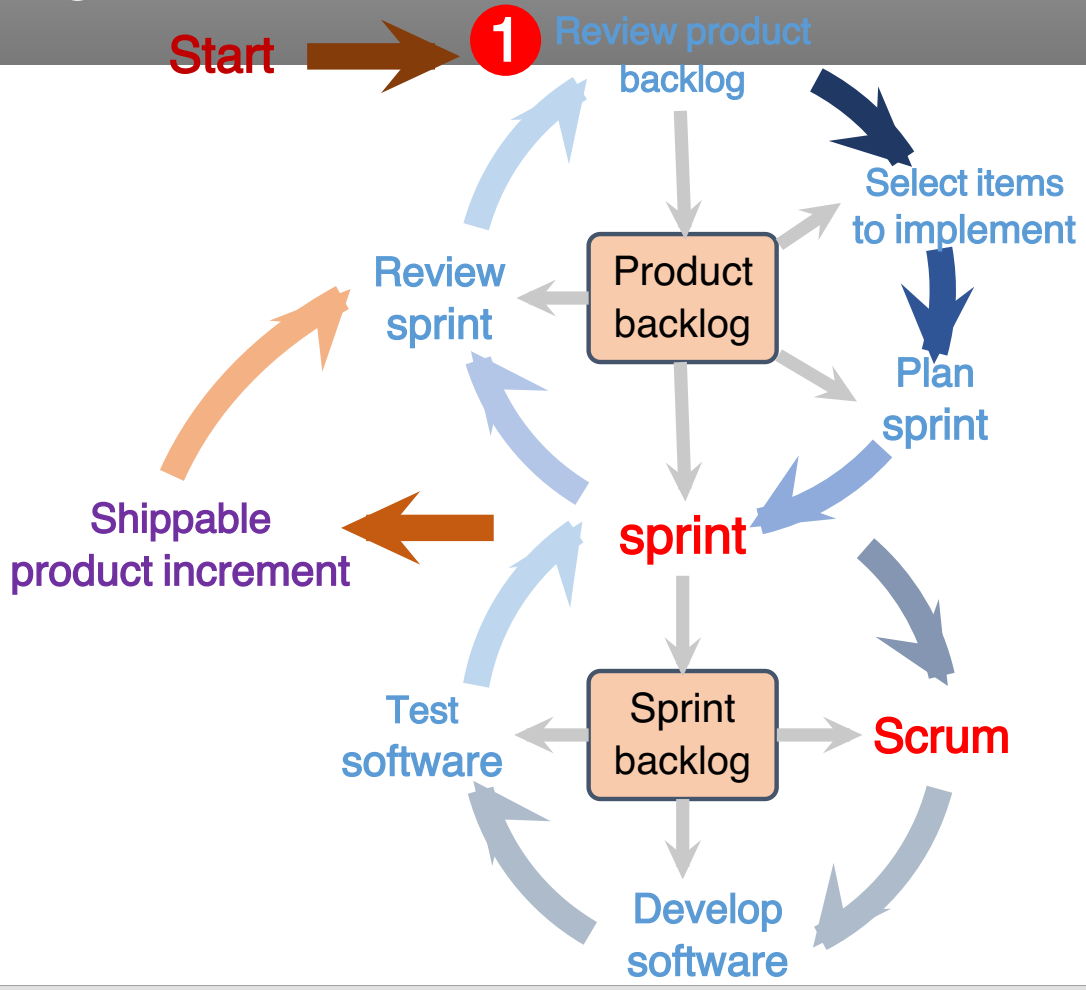
Key roles in Scrum

- The Product Owner is responsible for ensuring that the development team are always focused on the product they are building rather than diverted into technically interesting but less relevant work.
  - In product development, the product manager should normally take on the Product Owner role.
- The ScrumMaster is a Scrum expert whose job is to guide the team in the effective use of the Scrum method. The developers of Scrum emphasize that the ScrumMaster is not a conventional project manager but is a coach for the team. They have authority within the team on how Scrum is used.
  - In many companies that use Scrum, the ScrumMaster also has some project management responsibilities.

Scrum and sprints

- In Scrum, software is developed in sprints, which are fixed-length periods (2 - 4 weeks) in which software features are developed and delivered.
- During a sprint, the team has daily meetings (Scrums) to review progress and to update the list of work items that are incomplete.
- Sprints should produce a ‘shippable product increment’. This means that the developed software should be complete and ready to deploy.

Scrum cycles



Key Scrum practices

- Product backlog
  This is a to-do list of items to be implemented that is reviewed and updated before each sprint.
- Timeboxed sprints
  Fixed-time (2-4 week) periods in which items from the product backlog are implemented,
- Self-organizing teams
  Self-organizing teams make their own decisions and work by discussing issues and making decisions by consensus.

Product backlogs

- The product backlog is a list of what needs to be done to complete the development of the product.
- The items on this list are called product backlog items (PBIs).
- The product backlog may include a variety of different items such as product features to be implemented, user requests, essential development activities and desirable engineering improvements.
- The product backlog should always be prioritized so that the items that be implemented first are at the top of the list.

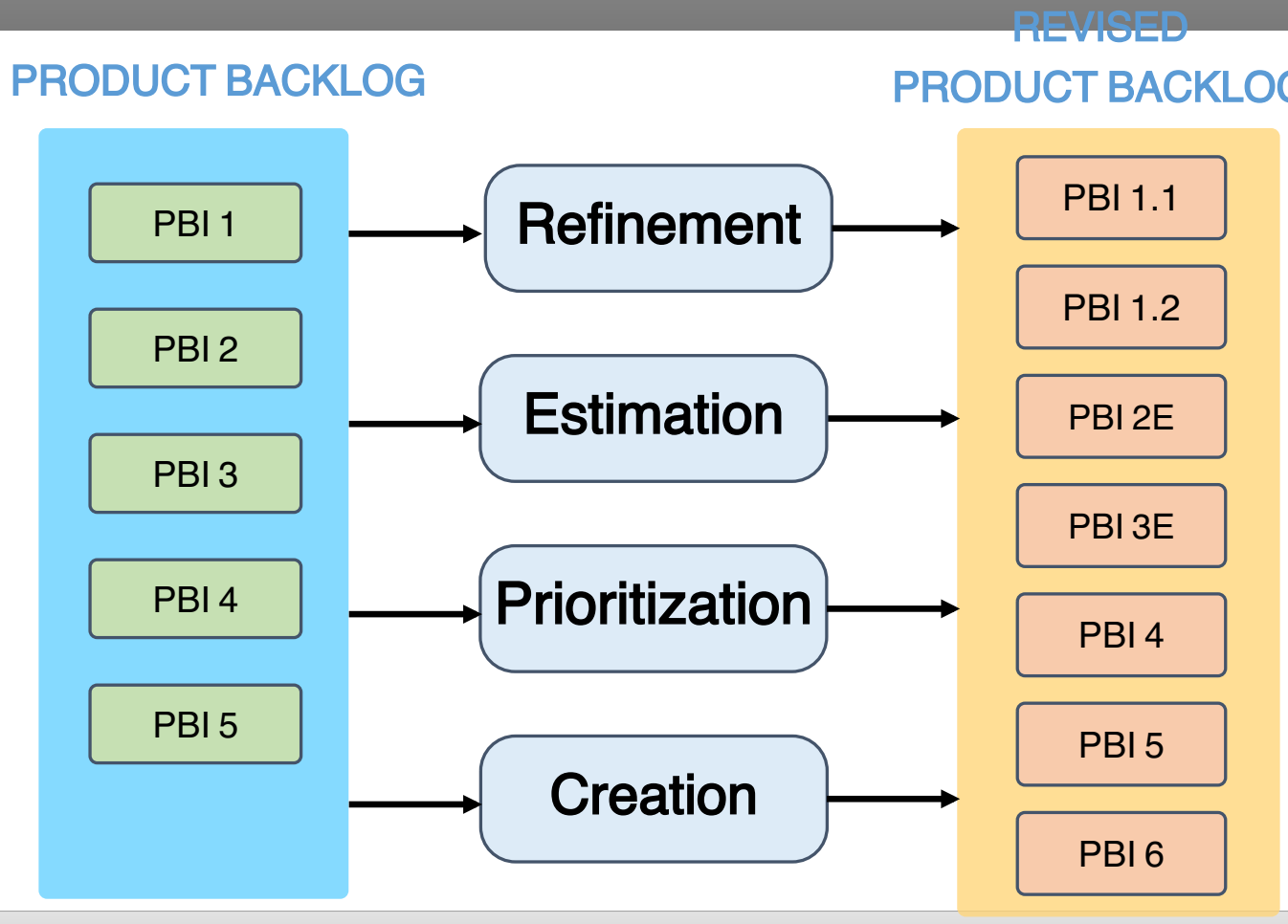
Examples of Product Backlog Items (PBIs)

1. As a teacher, I want to be able to configure the group of tools that are available to individual classes. (feature)
2. As a parent, I want to be able to view my children's work and the assessments made by their teachers. (feature)
3. As a teacher of young children, I want a pictorial interface for children with limited reading ability. (user request)
4. Establish criteria for the assessment of open source software that might be used as a basis for parts of this system. (development activity)
5. Refactor user interface code to improve understandability and performance. (engineering improvement)
6. Implement encryption for all personal user data. (engineering improvement)

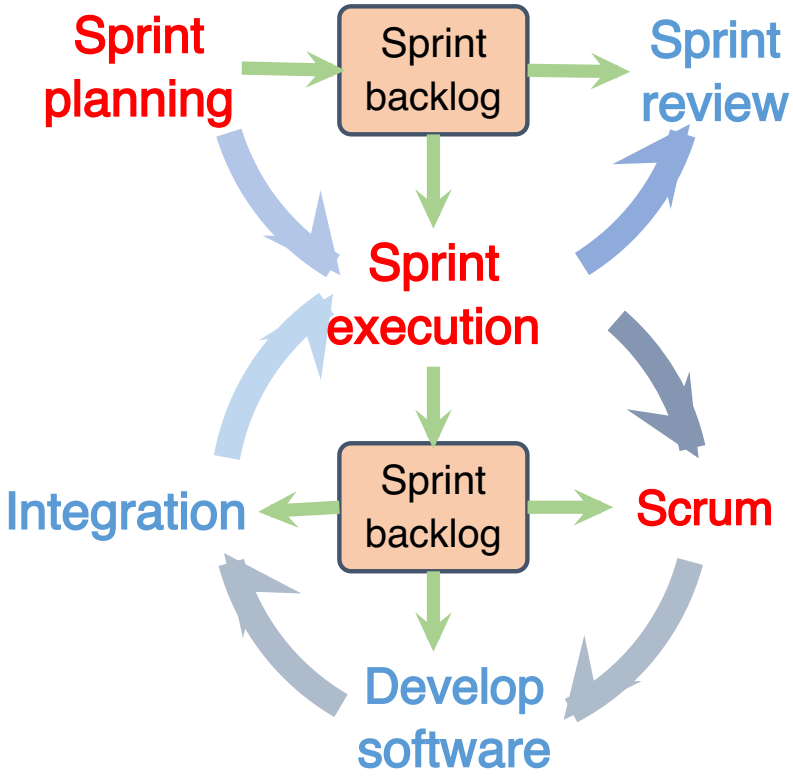
Product backlog item states

- Ready for consideration
  These are high-level ideas and feature descriptions that will be considered for inclusion in the product. They are tentative so may radically change or may not be included in the final product.
- Ready for refinement
  The team has agreed that this is an important item that should be implemented as part of the current development. There is a reasonably clear definition of what is required. However, work is needed to understand and refine the item.
- Ready for implementation
  The PBI has enough detail for the team to estimate the effort involved and to implement the item. Dependencies on other items have been identified.

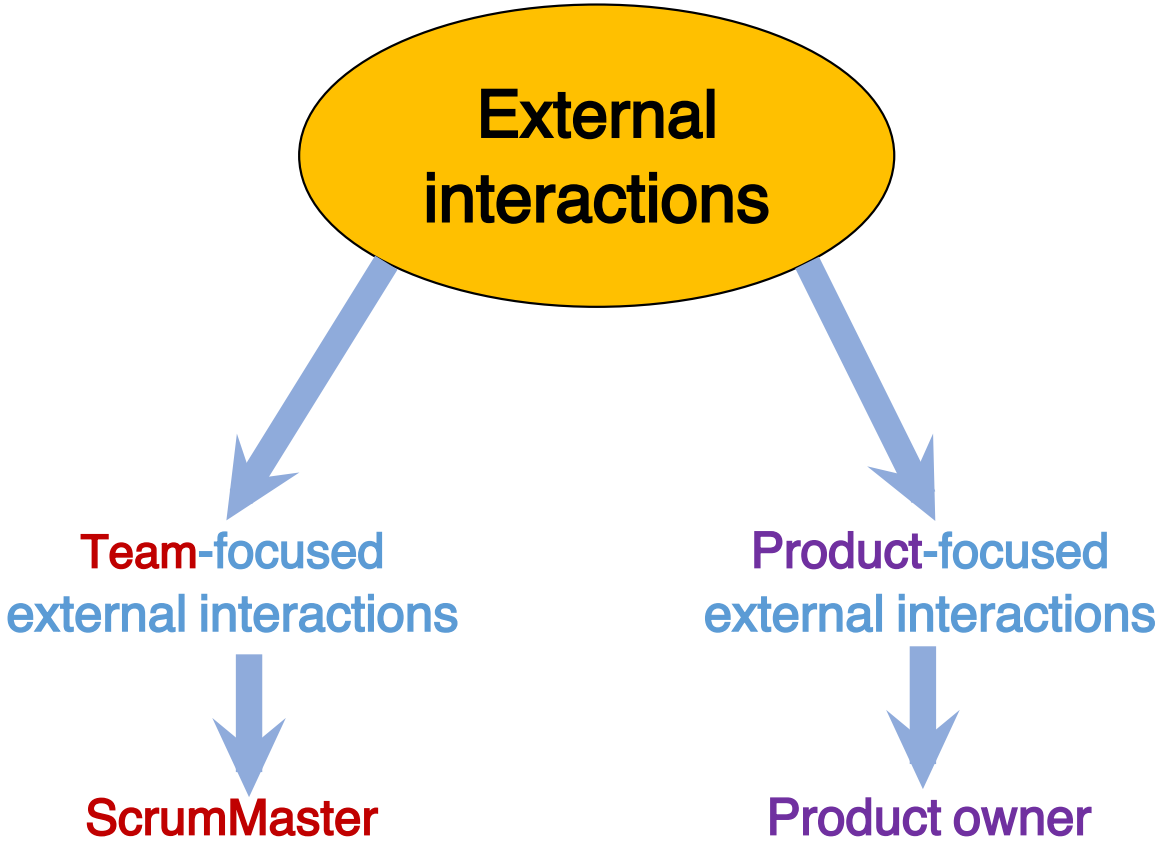
Product backlog activities



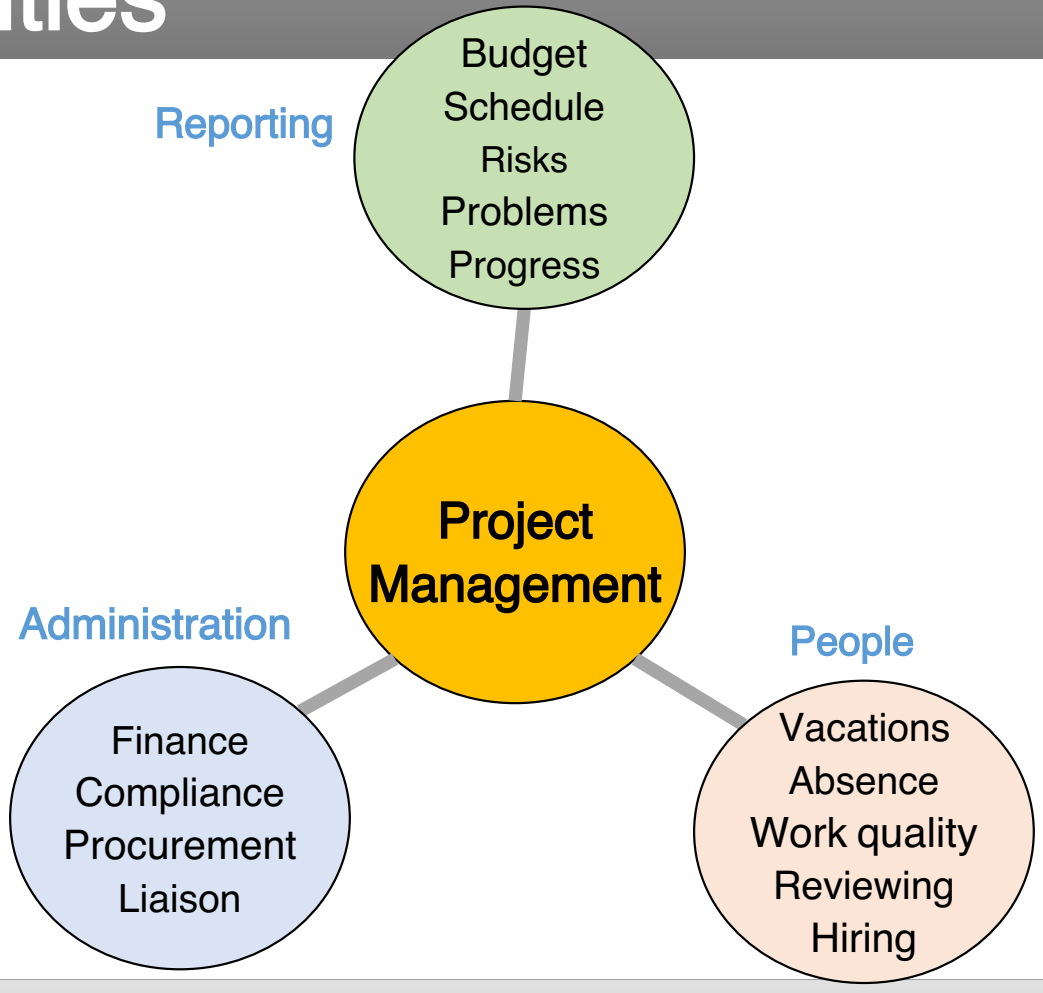
Sprint activities



Managing External Interactions



Project Management Responsibilities



Summary

- The best way to develop software products is to use agile software engineering methods that are geared to rapid product development and delivery.
- Agile methods are based around iterative development and the minimization of overheads during the development process.
- Extreme programming (XP) is an influential agile method that introduced agile development practices such as user stories, test-first development and continuous integration. These are now mainstream software development activities.

Summary

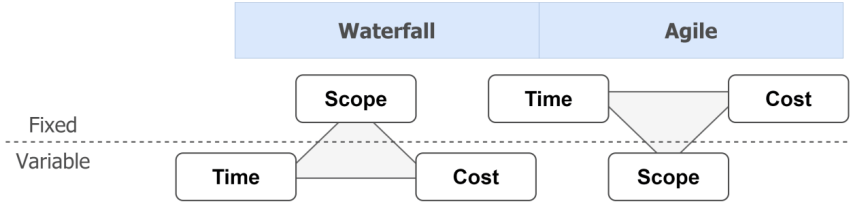
- Scrum is an agile method that focuses on agile planning and management. Unlike XP, it does not define the engineering practices to be used. The development team may use any technical practices that they believe are appropriate for the product being developed.
- In Scrum, work to be done is maintained in a product backlog – a list of work items to be completed. Each increment of the software implements some of the work items from the product backlog.

Summary

- Sprints are fixed-time activities (usually 2–4 weeks) where a product increment is developed. Increments should be 'potentially shippable' i.e. they should not need further work before they are delivered.
- A self-organizing team is a development team that organizes the work to be done by discussion and agreement amongst team members.
- Scrum practices such as the product backlog, sprints and self-organizing teams can be used in any agile development process, even if other aspects of Scrum are not used.

Agile vs. Waterfall

The main difference between Waterfall and Agile can be illustrated with the following triple constraints which can provide a rough overview. The rest of the differences that can provide a better overview can be found in the following table.

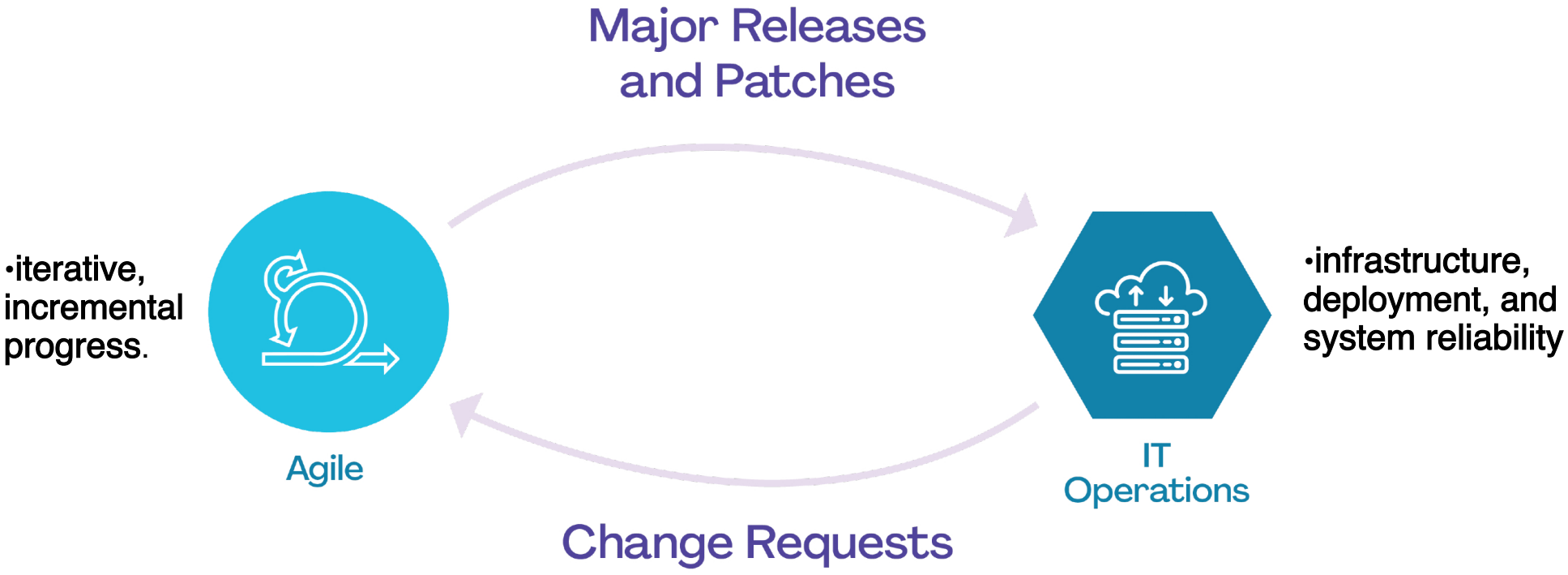


	Waterfall	Agile
Focus	On processes	On people
Management	By plans	By changes
Development style	Waterfall / Long iterations	Incremental, iterative, short iterations
Requirements	Planned at the beginning	Updated before every iteration
Customer involved	At the beginning and the end	All the time
Feedback	Minimal	Frequent
Social aspects	Plan and control	Trust, responsibility, motivation
Team organization	Hierarchical	Self-organized teams
Leadership	Directing	Coaching
Change management	Minimization of changes	Accept and adapt to changes
Documentation	Exhaustive	Just enough
Communication	Mainly written	Mainly personal
Product delivery	As a whole at the end	Frequent, but partial

development and operations.

DevOps

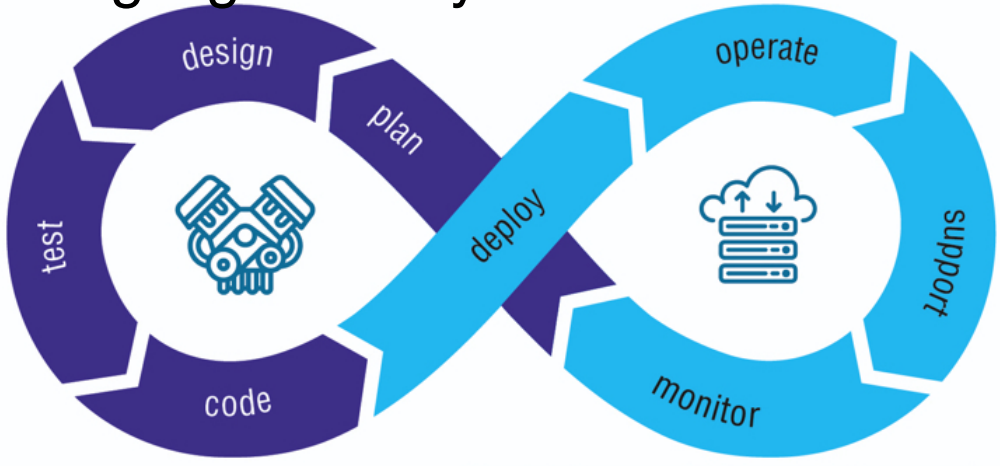
DevOps: challenges



DevOps

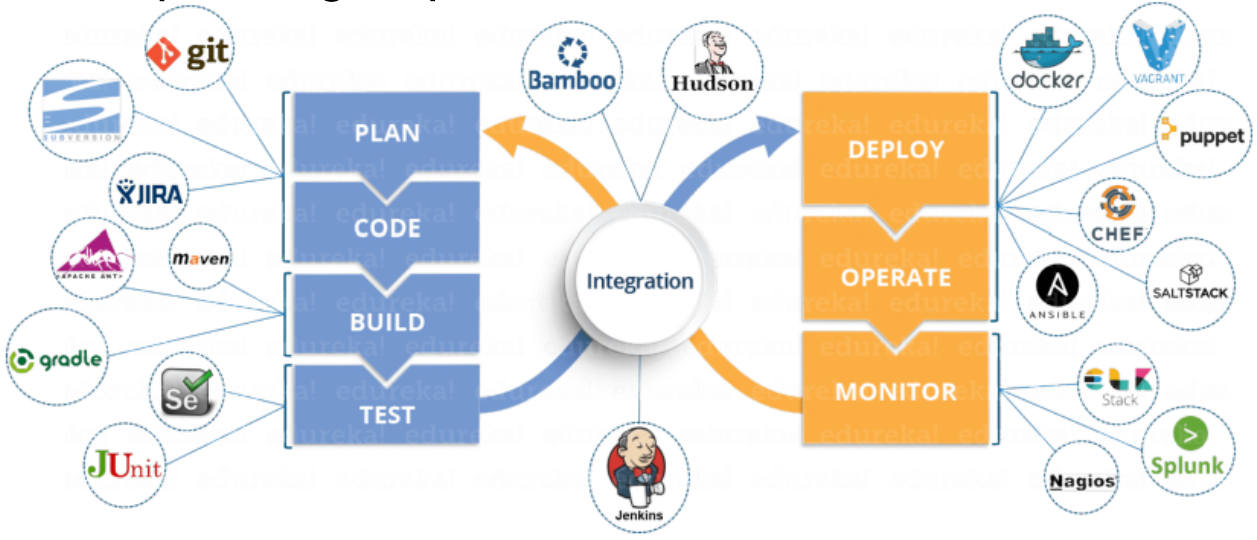
h

- When organizations address these challenges by removing the barriers that inhibit effective collaboration (operations <-> dev), they close the DevOps gap.
- Organizations do this by: adopting a mindset that promotes collaborative; learning-centric ways of working that are supported by agile practices; and very often investing significantly in automation
- On the development side they tend to adopt a continuous delivery (CD) approach and their operations efforts become streamlined and more automated.



DevOps

- DevOps is an approach to software development that extends the principles of agile development by integrating both the development and operations units of the IT team into one homogenous unit.
- This integration enables a more seamless flow between the traditionally siloed sides of the software development lifecycle and leads to fewer defects in production because operating requirements were considered from the outset of the project.



DevOps

- According to a study, the global DevOps market size was $7.398 billion in 2021, and it is anticipated to increase to $37.227 billion by 2030, with a significant CAGR (compound annual growth rate) of 20% from 2022 to 2030.
- DevOps project management is essential to any significant commercial venture to complete the project on schedule and under budget.
- It entails careful planning, resource allocation, and coordination between numerous stakeholders.

DevOps vs. Project Management

- Given the fluid nature of DevOps and its continuous integration/continuous deployment (CI/CD) process, the methodology may seem at odds with project management strategies that tend to follow timelines with a start and endpoint (waterfall style of PM).
- These Project Management approaches (timelines, Gantt charts, milestones, deadlines, etc.) seem more closely aligned with traditional software delivery methods like the waterfall approach.
- However, project management styles can adapt to fit the parameters of the DevOps pipeline (flexibility and rapid iteration): breaking work into sprints, focusing on iterative progress, and embracing change.
- The key is to shift the focus away from the one final product and instead refactor the timeline to follow the agile approach of incremental deliveries at the end of each sprint.

What is DevOps project management?

- DevOps project management is a union between the two disciplines where project management is tailored to and supports the DevOps model.
- In a DevOps project management approach, project managers serve as coordinators between multiple contributors and as trackers of timelines and dependencies (ensure team members collaborate smoothly, removing bottlenecks and resolving conflicts)
- However, project managers also need to be closely aligned with the DevOps team and bring a deep understanding of the development process and skills needed to create the end product.
- There are several best practices that project managers should follow to achieve this integration with the DevOps pipeline.

DevOps Project Management Best Practices

- Develop a collaborative culture
- Embrace the MVP (Minimum Viable Product) mindset
- Focus on both on-time and quality deliveries
- Emphasize dependencies
- Use tools to do the heavy lifting
- Track the right metrics: such as Deployment frequency, lead time for changes, change failure rate, mean time to recovery (MTTR)

DevOps Project Management Best Practices

- Develop a collaborative culture: Foster open communication, shared goals, and mutual respect. Break down silos and encourage team ownership of outcomes:
  - Get leadership on board from the beginning
  - Implement the right communication tools and technologies
  - Align the right people for the team
  - Educate and train the team
  - Empower the team with a sense of autonomy and ownership
  - Establish a pattern of continuous improvement and learning
  - Inculcate a sense of shared responsibility
  - Prioritize open and transparent communication
  - Establish an environment of trust and respect
  - Share common goals and vision

DevOps Project Management Best Practices

- Embrace the MVP mindset: Prioritize features that deliver immediate value. Avoid over-planning or chasing perfection in early stages:
  - The minimum viable product (MVP) is the simplest version of an application that will still meet the central requirements of the project.
  - DevOps embraces the MVP as a philosophy to ensure there is always a deliverable at the end of each sprint without delaying turnaround with an overly wide scope.
  - In contrast to traditional project management strategies that emphasize the final monolithic product, this is a considerable shift. DevOps is viewed less as a linear process with set start and end dates and more as a constant process of improvement.
  - The advantages of the MVP mindset are that customers receive a deliverable faster and their feedback then helps the team course correct earlier in the cycle.

DevOps Project Management Best Practices

- Focus on both on-time and quality deliveries: Set realistic timelines, monitor quality metrics (e.g., defect rates, rollback frequency), and ensure testing is embedded in the pipeline:
  - Project management places a heavy emphasis on meeting deadlines. While this is important, many of the metrics DevOps teams use to measure success focus on more qualitative factors to gauge if the customer is meeting their goals with the application.
  - This means that quick turnaround times are important. So is allocating enough time and resources to ensure the deliverable quality meets a high standard. Project managers need to weigh both these factors when deciding on the goals for a sprint and assigning tasks.
  - Part of an effective delegation of responsibilities is understanding the dependencies for each task.

DevOps Project Management Best Practices

- Emphasize dependencies: Map out dependencies early, track them continuously, and resolve blockers proactively:
  - Even in system architecture that follows a microservice approach of building small applications to handle individual tasks, there are dependencies between these applications, the systems they run on, the data they handle, and more.
  - Tracking dependencies goes beyond configuration management. It requires maintaining visibility into all the moving parts of both the IT environment and the DevOps team's tasks to properly define sprint goals.
  - Each deliverable should fully function in the existing environment and be ready to deploy at the end of each DevOps cycle. If that delivery is slowed by a backlog of tasks or a system requirement not being met, then more thorough dependency tracking is needed.

For. ex
You can't test
Befor development

By the Project manager.

DevOps Project Management Best Practices

- Use tools to do the heavy lifting: Leverage dashboards, issue trackers, CI/CD platforms, and communication tools to streamline workflows and gain visibility:
  - Automation is a major focus in DevOps to scale the impact of a team's effort, and DevOps project management is not different.
  - PM tools provide a single source of truth for how work has been assigned, how the project is proceeding, and what issues have occurred along the way. This information is valuable for the entire team to provide greater context and foster more informed decision-making.
  - These applications will overlap and integrate with the tools the DevOps team uses in their day-to-day work so that both project managers and developers are on the same page (Example: A developer updates a ticket in Jira, and the PM sees the status instantly. A failed deployment in Jenkins triggers an alert that both teams can act on.)
  - Dashboards can provide real-time metrics on how the team is performing against their goals and help PMs detect potential issues to address earlier in the cycle.

DevOps Project Management Best Practices

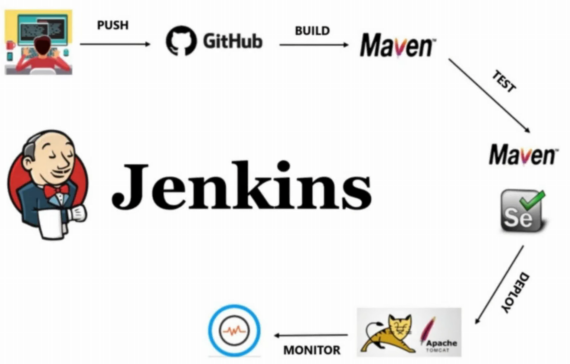
g

- Track the right metrics: such as Deployment frequency, lead time for changes, change failure rate, mean time to recovery (MTTR): Use these metrics to guide decisions, identify bottlenecks, and celebrate progress:
  - DevOps best practices recommend constant performance tracking.
  - You can use project management tools with built-in analytics reports to give you insight into data and performance tracking capabilities.
  - Tracking the right performance indicators, such as lead time, mean time to detect, and issue severity to assess the efficacy of a DevOps approach. Tracking these indicators is essential since it enables you to identify problems early and take swift corrective action.
  - The goals and standards of your enterprise will determine the DevOps metrics you need to monitor. Any engineering and development team can benefit from some indicators connected to profitability, like unit cost (the average cost to deliver a single unit of value, such as a feature, user story, bug fix, or deployment.).

DevOps project management leads to better outcomes.

- By combining the practice of DevOps (CD/CI, automation, etc.) with the principles of project management (planning, task definition, progress tracking, etc.), you achieve greater coordination between individual team members and the larger goals of the sprint.
- This leads to better outcomes by ensuring that task scopes are properly defined and dependencies are tracked so the application is ready to deploy.
- DevOps project management tools (Jira, github, gitlab, github actions, Jenkins, etc.) simplify this process and promote visibility between project managers and team members so everyone is on the same page and working toward the same goals.

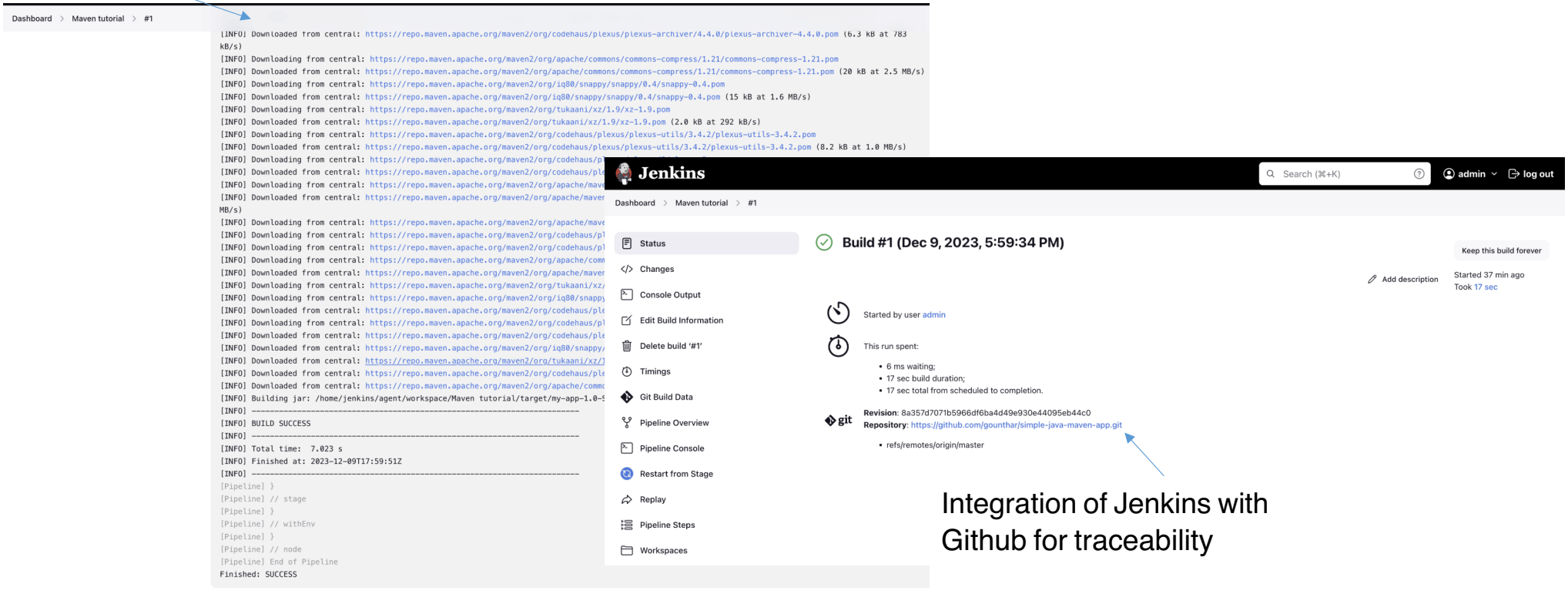
DevOps project management leads to better outcomes (Jenkins)



DevOps project management leads to better outcomes (Jenkins)

Integration of Jenkins with Maven to build the software

Fig. Jenkins dashboard



Integration of Jenkins with Github for traceability

What is the Role of a DevOps Project Manager?

- Project managers' roles have evolved due to Scrum and other agile approaches.
- Project managers in a DevOps project management methodology act as a bridge between various contributors and keep track of deadlines and commitments.
- They also need to have a strong awareness of the development process and the abilities required to produce the final product and work closely with the DevOps team.
- DevOps project managers have the responsibility to address the management and coordination of the product from development to production. They also work on technical details (they are technical orchestrators), whereas a traditional project manager is not involved in handling these tasks.
- DevOps project managers ensure they manage the integration, flow of development, testing, coordination, and deployment of the project.

Specificities of the Role of a DevOps Project Manager

- Treat the development and operations team as a single organization
- Effectively embrace change

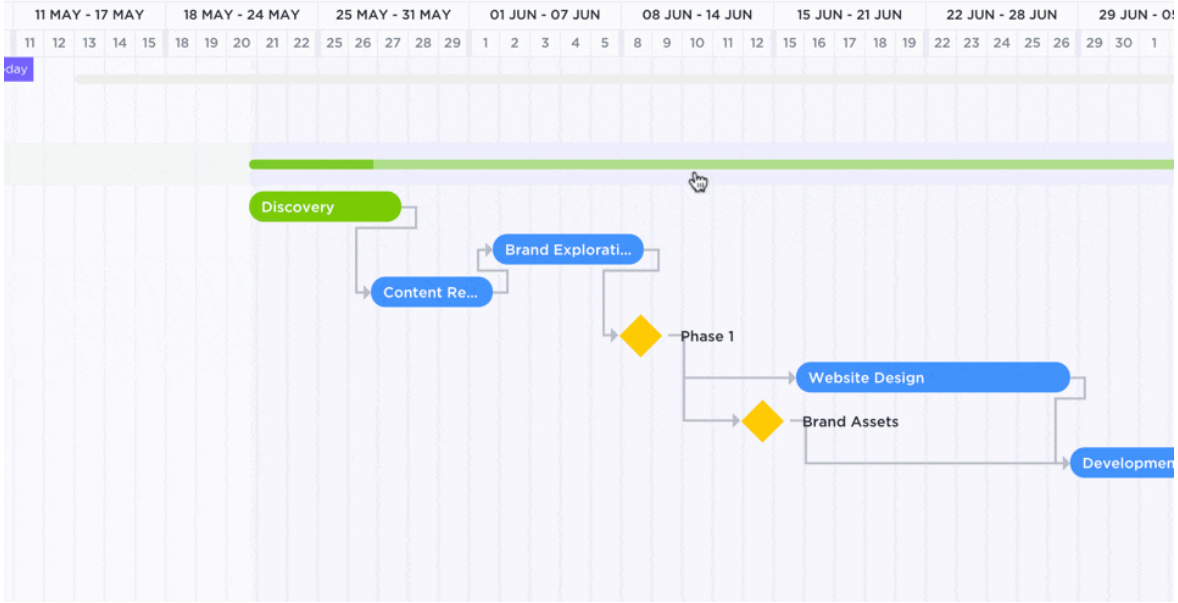
Specificities of the Role of a DevOps Project Manager

- Treat the development and operations team as a single entity.
  - Traditional setup: Developers write code, then hand it off to operations to deploy and maintain. This separation often leads to miscommunication, delays, and finger-pointing.
  - DevOps mindset: Development and operations work together from the start—sharing goals, tools, and responsibilities. They become a single entity focused on delivering reliable software quickly.
- DevOps requires a leader who can assist the team in overseeing the flow from beginning to end, much like a typical project manager's role is to plan, acquire, and execute the project.

Specificities of the Role of a DevOps Project Manager

- Treat the development and operations team as a single entity
  - Creating value stream maps should be a natural extension of the project manager's Gantt Chart planning abilities.

•Traditional project managers use these to plan tasks, timelines, and dependencies.
•Value stream maps visualize the entire flow of work—from idea to deployment—highlighting where value is added and where waste or delays occur.
•For project managers familiar with Gantt charts, creating value stream maps is a logical next step. It shifts the focus from task scheduling to process optimization.



Specificities of the Role of a DevOps Project Manager

- Treat the development and operations team as a single entity
  - Treating the Development and Operations team as a single organization is important to prevent work silos.
  - These two teams are merged into a single team to work across the entire application lifecycle and develop various skills, from development and testing to deployment to operations.
  - Attempts to use DevOps will be unsuccessful unless Dev and Ops teams collaborate effectively.
  - The development and operations teams have several impacts on task handling: educating and sharing responsibilities between Devs and Ops teams, improving productivity and collaboration, understanding deployment risks, awareness of tool quality, and managing added tasks.

Specificities of the Role of a DevOps Project Manager

- Effectively embrace change
  - The agile project management methodology clarifies that embracing change rather than sticking to the plan is crucial.
  - It takes a lot of communication and teamwork to manage change and release features incrementally. Communication is a project manager's primary skill. Enterprises ought to make use of this potential.

Specificities of the Role of a DevOps Project Manager

- Effectively embrace change
  - Change isn't just inevitable—it's a catalyst for growth. To thrive in fast-paced DevOps environments, project managers must evolve their mindset and practices:

- Begin with small projects to test ideas and reduce risk
- using the MVP approach: to deliver value early and often
- using the right agile planning tools: to stay adaptive and transparent
- eliminating silos: by fostering cross-functional collaboration
- reducing project handoffs: to streamline workflows, minimize delays
- creating real-time project visibility: with shared dashboards and live updates
- reducing project overhead: by simplifying processes and focusing on value delivery
- managing change collaboratively: Involve your team in planning and adapting to change