
Domain
Percentage of Items on Test

Domain 1. Agile Principles & Mindset

Key Topics

- Agile Frameworks & Terminology
- Agile Manifesto
- Agile Methods & Approaches
- Agile Process Overview
- Kanban
- Leadership Practices & Principles
- Lean
- Scrum
- XP

Why Use Agile?

Different types of projects require different methods. Some projects, especially knowledge work projects in a fast-moving, complex environment, call for an agile approach.

Knowledge Work Projects are Different

The Agricultural Revolution

The Agricultural Revolution was a period of technological improvement and increased crop productivity that occurred during the 18th and early 19th centuries in Europe.

The Industrial Revolution

The Industrial Revolution, now also known as the First Industrial Revolution, was the transition to new manufacturing processes in Europe and the United States, in the period from about 1760 to sometime between 1820 and 1840.

The Information Revolution

Focused on information and collaboration, rather than manufacturing. It places value on the ownership of knowledge and the ability to use that knowledge to create or improve goods and services.

Knowledge Workers

The Information Revolution relies on knowledge workers. These are people with subject matter expertise who communicate their knowledge and take part in analysis or development efforts.

Manufacturing vs Knowledge Projects

Characteristics of Industrial Work

- ❏ Work is visible
- ❏ Work is stable
- ❏ Emphasis is on running things
- ❏ More structure / Fewer decisions
- ❏ Focus on the right answers
- ❏ Define the task
- ❏ Command and control
- ❏ Strict standards
- ❏ Focus on quantity
- ❏ Measure performance to strict standards
- ❏ Minimize cost of workers for a task

Characteristics of Knowledge Work

- ❏ Work is invisible
- ❏ Work is changing
- ❏ Emphasis is on changing things
- ❏ Less structure / More decisions
- ❏ Focus on the right questions
- ❏ Understand the task
- ❏ Give autonomy
- ❏ Continuous innovation
- ❏ Focus on quality
- ❏ Continuously learn and teach
- ❏ Treat workers as assets, not costs

Agile Methodology

The communication and collaboration involved in knowledge work projects made the work more uncertain and less definable than industrial work.

Agile pioneers collected the most effective techniques for knowledge work and adapted them for use on projects.

Defined vs Empirical Processes

Defined Processes

- Industrial work
- In a defined process, we can define the constituent steps in advance.
- The most efficient way to proceed for a well-understood project in an unchanging environment.
- Most industrial projects can be planned and managed by using a defined approach.

Empirical Processes

- Knowledge work
- When faced with a new or uncertain process, such as building an underwater home for the first time or using carbon nanotubes instead of steel, there will be many unknowns and uncertainties involved

Empirical Processes

- When faced with uncertainty, a process of trial and error and experiment is required to determine what works, surface issues, and incrementally build on small successes.
- The resulting process will be iterative and incremental, with frequent reviews and adaptation.
- The empirical approach is required for projects where the execution stage is characterized by uncertainty and risks - in other words, projects that would benefit from the agile approach.

The Agile Mindset

Declaration of Independence

1

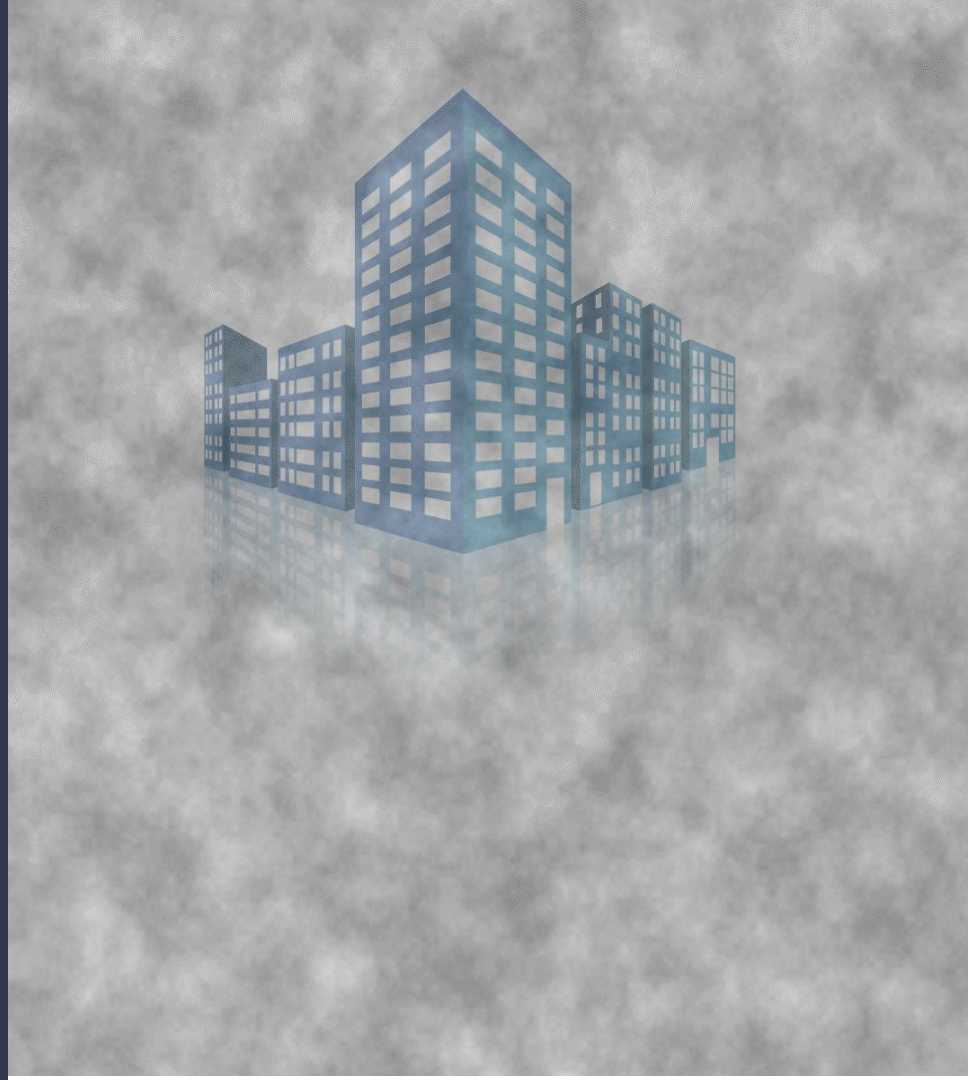
Increase return on investment by making continuous flow of value our focus.

2

Deliver valuable results by engaging customers in frequent interactions and shared ownership.

3

Expect uncertainty and manage for it through iterations, anticipation, and adaption.

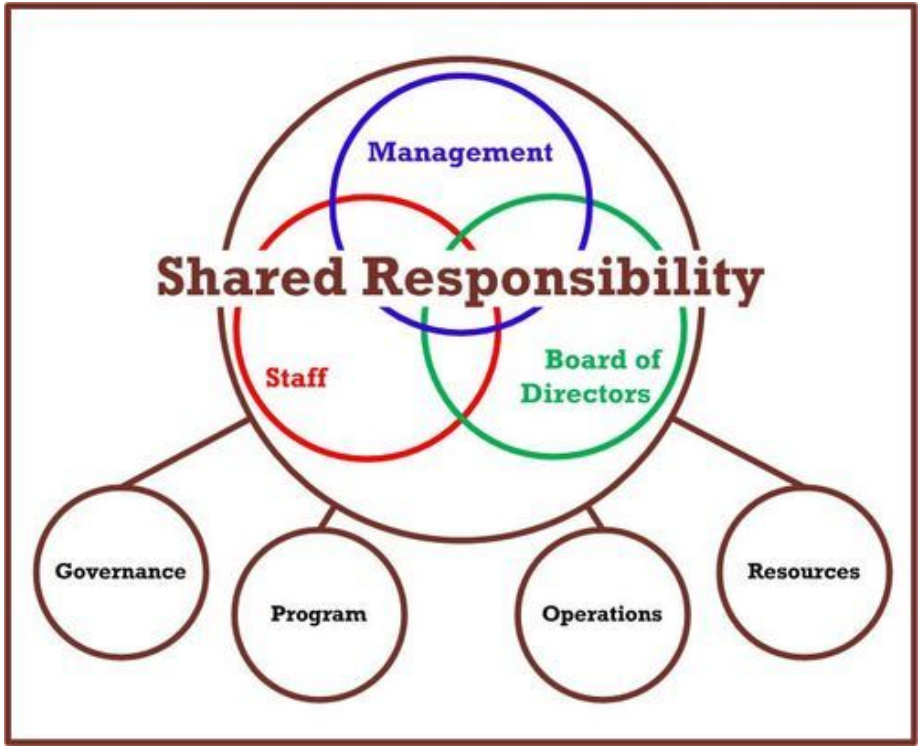


4

Unleash creativity and innovation by recognizing that individuals are the ultimate source of value, and creating an environment where they can make a difference.

5

Boost performance through group accountability for results and shared responsibility for team effectiveness.



6

Improve effectiveness and reliability through situationally specific strategies, processes, and practices

“Left-to-right adoption”

A shorthand way of saying “teach agile values first”

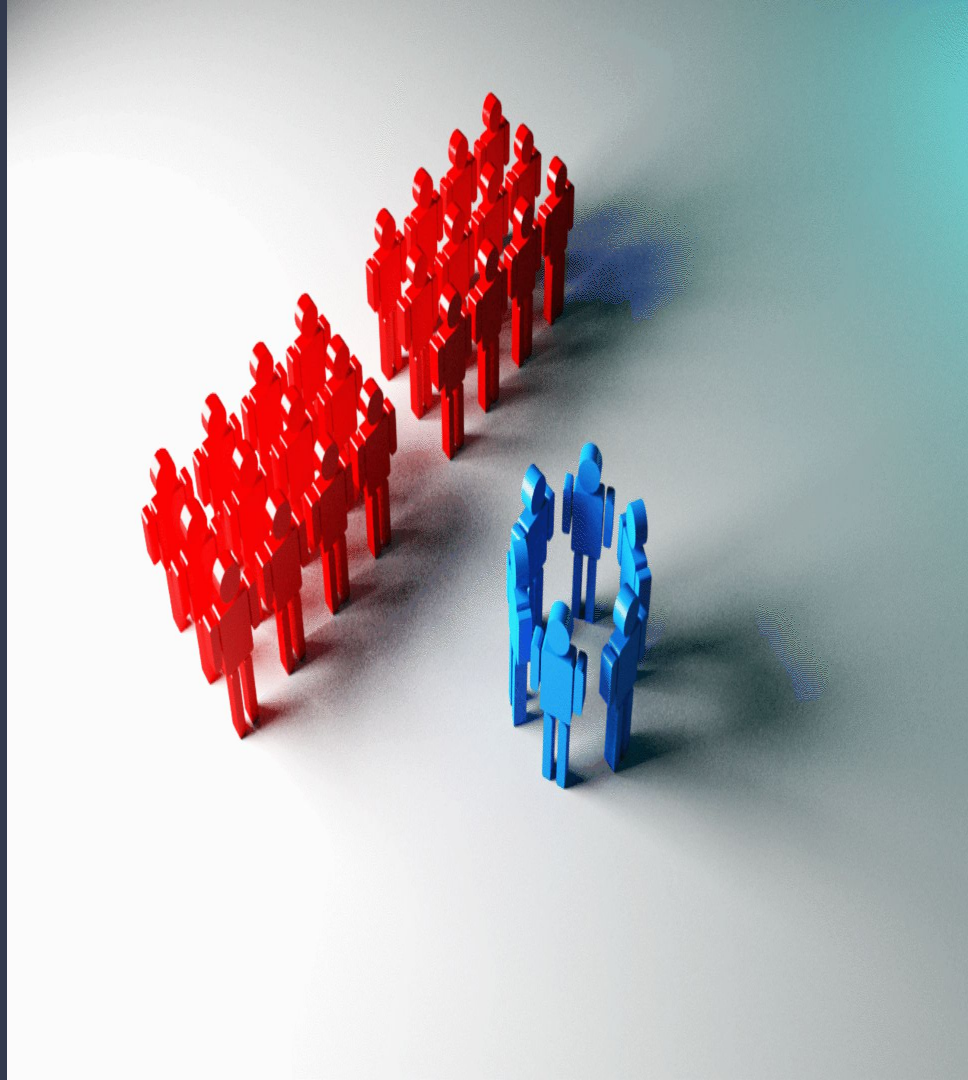
Personal, Team, and Organizational Agility

If just one member of Organization adopts the agile mindset

It can help that person become more effective. However, they will feel continually frustrated that others in the organization don’t seem to realize what is important or are focused on the wrong goals and metrics.

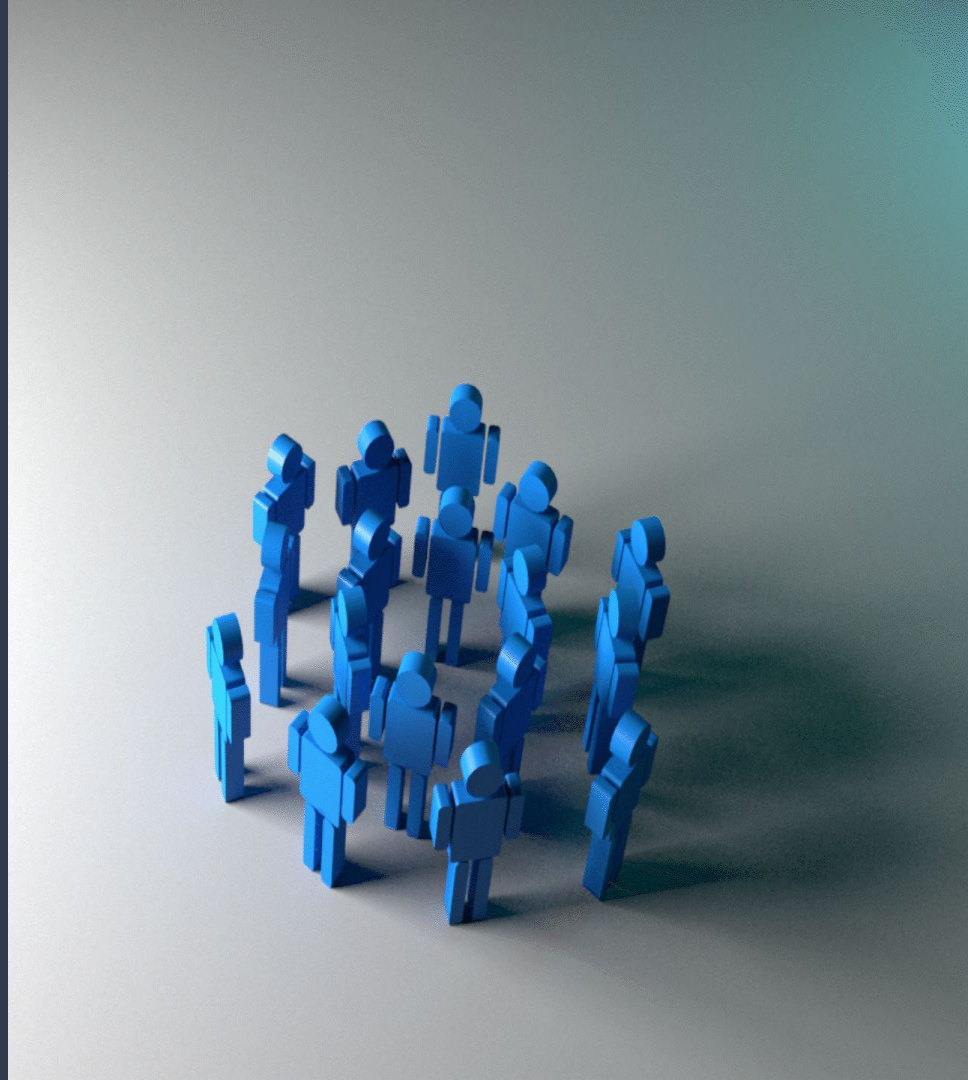
If one team in the organization adopts agile principles and practices

It can help the team members become more effective at delivering their project work. However, they will feel inhibited or misunderstood by other groups or systems in the organization, such as the project management office or functional silos.



If the entire organization adopts the agile way of thinking

Then everyone will be working together to improve agility and the delivery of value. By adopting common goals and values, such as continuous improvement and welcoming change, everyone’s effectiveness will be enhanced.



Creating Organizational Change

1

Individually learning and internalizing agile principles.

2

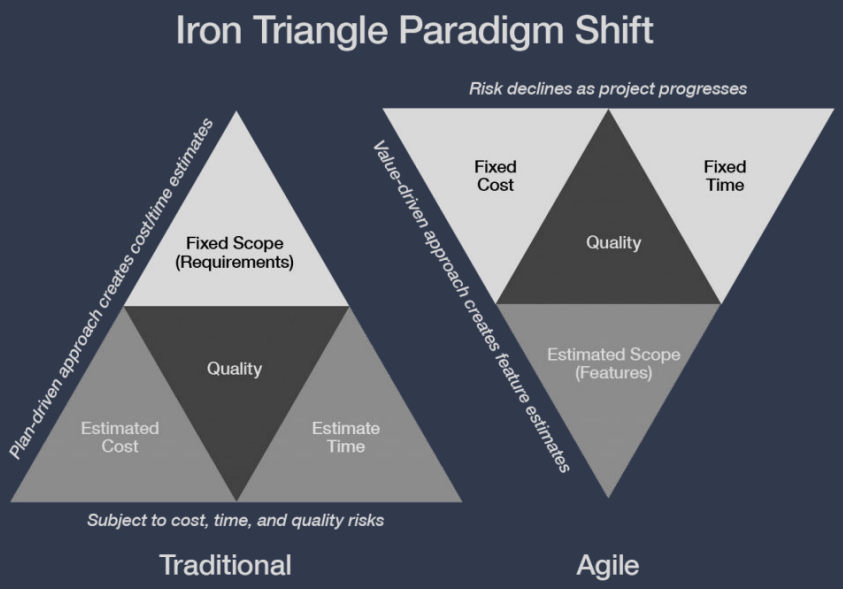
Doing is the practice of agile. For example, this might involve visualizing work items, using short iterations, or building in feedback and improvement steps.

3

Persuading others to adopt the agile mindset and practices will magnify agile learning and effectiveness across the entire organization. Also, the more people in your circle of work you can successfully educate about the merits of agile, the more allies you will have in advocating the cause. The end result of this process can be a complete transformation of the organization based on agile principles.

The Agile Triangle

Inverted Triangle Model

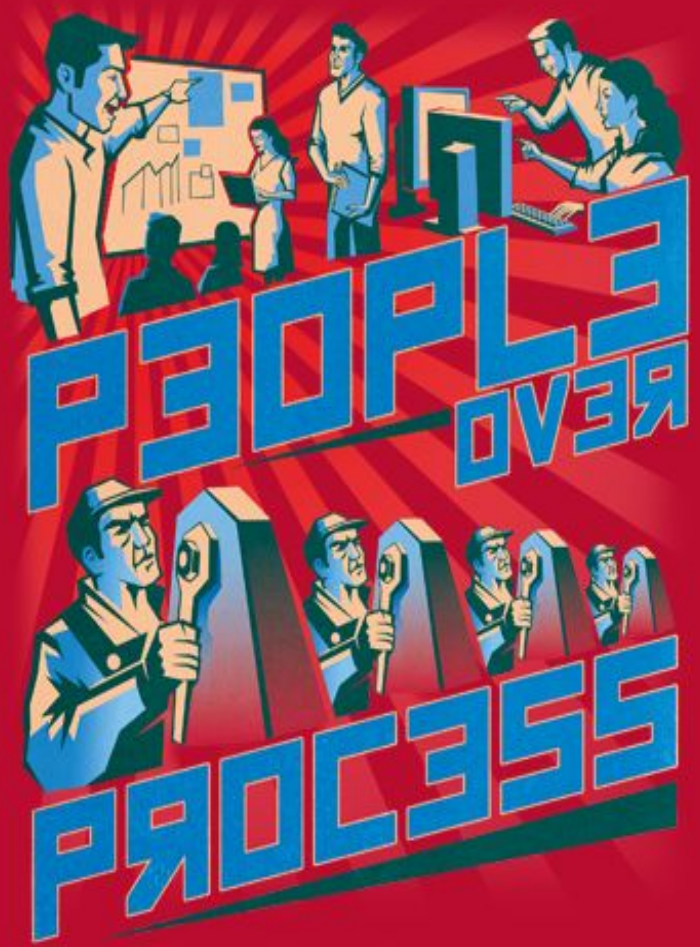


Agile teams allow scope to vary within the fixed parameters of cost and time.

We aim to deliver the most value we can by X date within Y budget.

We'll begin with a high-level vision of the end product, however, we can't define up front how much we'll be able to get done. This will emerge as we get closer to the target date.

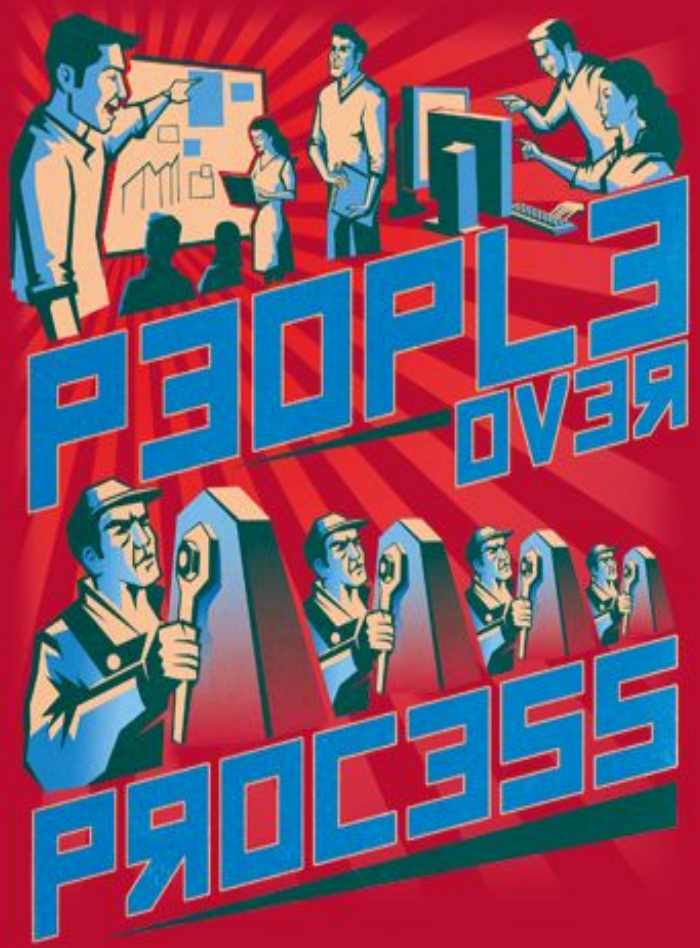
The Agile Manifesto



1

Individuals and interactions over processes and tools.

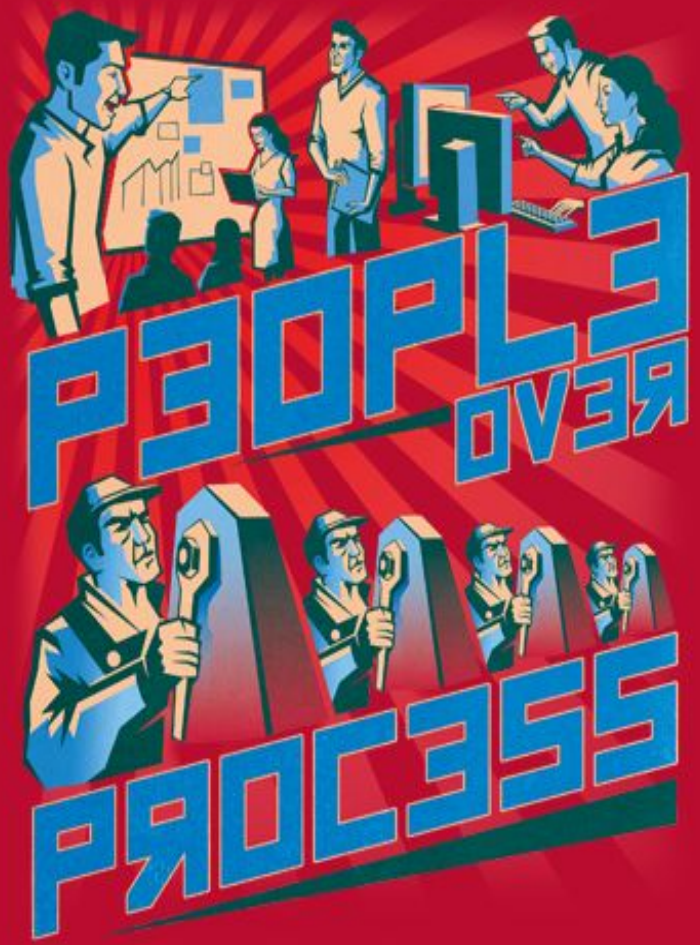
projects are undertaken by people,not tools, and problems get solved by people, not processes.



1

Individuals and interactions over processes and tools.

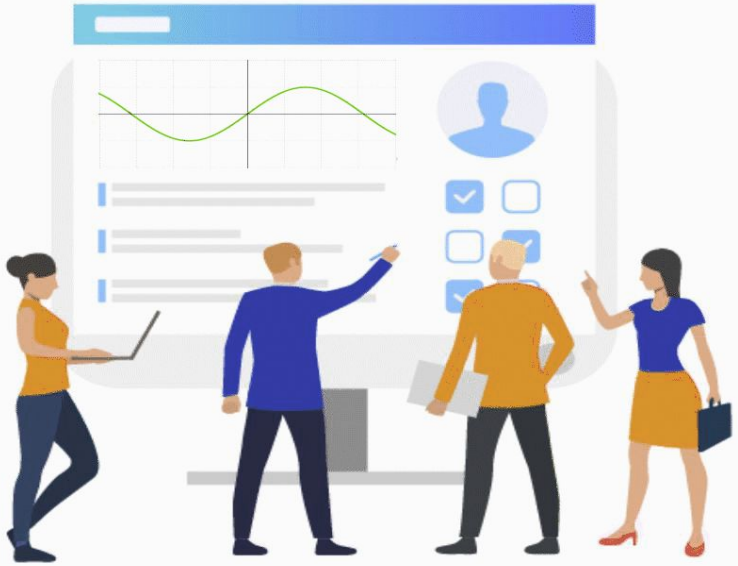
Projects are accepted by people, scope is debated by people, and the definition of a successfully “done” project is negotiated by people.



1

Individuals and interactions over processes and tools.

Focusing early on developing the individuals involved in the project and emphasizing productive and effective interactions help set up a project for success.



Working software over comprehensive documentation

2

This value reminds us to focus on the purpose or business value we’re trying to deliver, rather than paperwork.

2. Working Software over Comprehensive Documentation

The agile approach to documentation is *"just enough, just in time —and sometimes, just because*."

This phrase is shorthand to remind us of three important concepts:

1. Agile documentation should be "barely sufficient"—just enough to cover our needs. This keeps most of our efforts focused on the emerging system.
2. Agile documentation is done "just in time"—also known as the "last responsible moment" — so we don't have to spend extra time to keep it updated as our requirements and designs change.
3. Finally, "just because" reminds us that sometimes when documentation is required or requested, it's easier and preferable to just produce it than to face the consequences of not doing so.

Customer collaboration over contract negotiation

3

Similar to the difference between “being right” and “doing the right thing.”

Customer collaboration over contract negotiation

3

It’s possible to build a product as originally specified, but if the customer’s preferences or priorities change, it is better to be flexible and work toward the new goal.

4

Responding to change over following a plan

In knowledge work projects, we know that our initial plans are inadequate, since they are based on insufficient information about what it will take to complete the project.

Responding to change over following a plan

4

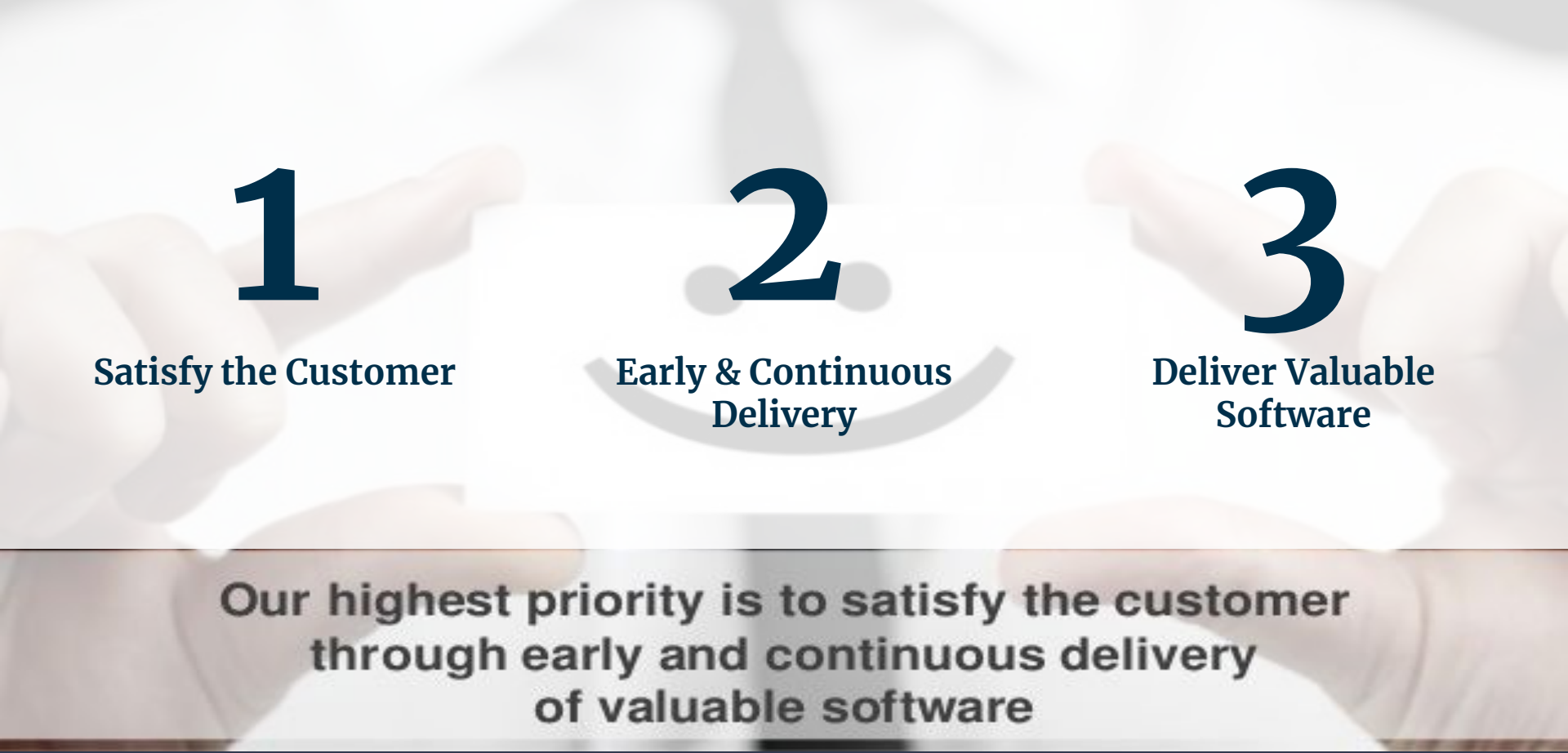
We want to spend more of our effort and energy responding to the changes that will inevitably arise.

Responding to change over following a plan

4

Agile projects have highly visible queues of work and plans in the form of backlogs and taskboards. The intent of this value is to broaden the number of people who can be readily engaged in the planning process by adjusting the plans and discussing the impact of changes.

The Twelve Principles



Principle 1 - Satisfy Customer with Great Systems

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

Instead of creating a high overhead mechanism for suppressing or processing changes, agile methods use a lightweight, high-visibility approach. For example, **continuously updating and prioritizing changes into the backlog of work to be done**. Agile methods for handling changes keep the project adaptive and flexible as long as possible. By welcoming changes that will inevitably happen and setting up an efficient way to deal with them, we can spend **more time developing the end product**.

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

This principle emphasizes the importance of releasing work to a test environment and getting feedback. Frequent testing and feedback are so important that software developers use continuous integration tools to provide feedback about any code they have written that breaks the build. Agile teams need feedback on what they have created thus far to see if they can proceed, or if a change of course is needed.

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

Delivering within a short timeframe also has the benefit of keeping the product owner engaged and keeping dialogue about the project going. With frequent deliveries, we will regularly have results to show the customer and opportunities to get feedback. Often at these demos, we learn of new requirements or changes in business priorities that are valuable planning inputs.

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

By working with business representatives daily, we can learn about the business in a way that is far beyond what a collection of requirements-gathering meetings can ever achieve. As a result, we are better able to suggest solutions and alternatives to business requests. The business representatives also learn what types of solutions are expensive or slow to develop, and what features are cheap. They can then begin to fine-tune their requests in response.

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

When it isn’t possible to have daily interactions between the business representatives and the development team, Agile methods try to get the two groups working together regularly in some way, perhaps every two days or whatever type of frequent involvement will work. (Some teams use a“proxy customer,”in which an experienced business analyst who is familiar with the business interests serves as a substitute, but this isn’t an ideal option.)

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

Since the development team is such an important factor, agile methods promote empowered teams. People work better when they are given the autonomy to organize and plan their own work. Agile methods advocate freeing the team from the micromanagement of completing tasks on a Gantt chart. Instead, the emphasis is on craftsmanship, peer collaboration, and teamwork, which result in higher rates of productivity.

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

Written documents are great for creating a lasting record of events and decisions, but they are slow and costly to produce. In contrast, **face-to-face** communication allows us to quickly transfer a lot of information in a richer way that includes emotions and body language. In a face-to-face conversation, **questions can be immediately answered**, instead of "parked" with the hope that there will be a follow-up explanation, or the answer will become clear later

Working software is the primary measure of progress

By adopting “working software” as our primary measure of progress, we shift our focus to working results rather than documentation and design.

In agile, we **assess progress based on the emerging product or service** we are creating. Questions like “How much of the solution is done and accepted?” are preferred over “Is the design complete?” since we want to focus on usability and utility rather than conceptual progress.

If a feature can’t be measured or tested, it isn’t considered complete.

Working software is the primary measure of progress

This emphasis on a working product helps ensure that we get features Accepted by the customer, rather than Marking items as “completed development” when they haven’t yet been accepted as “done.”

This definition of progress as “working systems” creates a results-oriented view of the project. Interim deliverables and partially completed work will get no external recognition. So we want to focus instead on the primary goal of the **project—a product that delivers value to the business**.

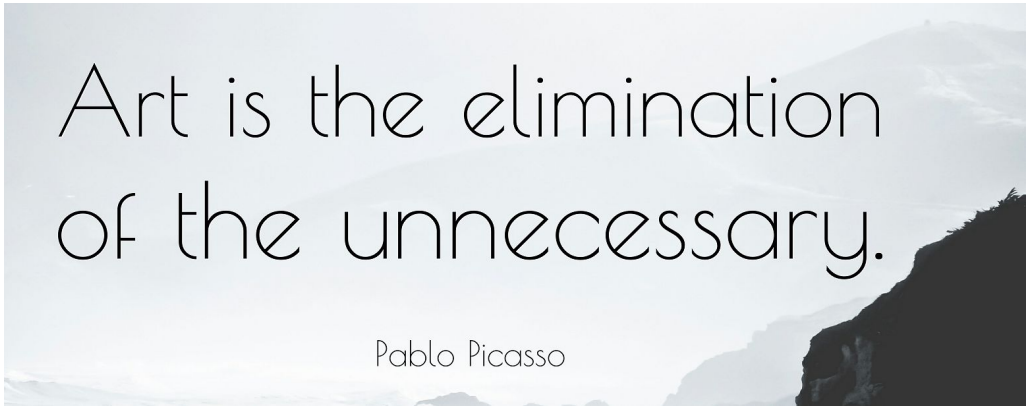
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely

Instead of long, intense development Periods, agile methods recognize the value Of a **sustainable pace that allows team members to maintain a work-life balance**. A sustainable pace is not only better for the team; it benefits the organization as well. Long workdays lead to resignations, which means the organization loses talent and domain knowledge. **Hiring and integrating new members into a team is a slow and expensive process**.

Continuous attention to technical excellence and good design enhances agility

An agile team needs to balance its efforts to deliver high-value features with **continuous attention to the design of the solutions**. This allows the product to deliver long-term value without becoming difficult to maintain, change, or extend. This helps the project run more smoothly and speeds up the team's progress. **Once the code base becomes tangled, the organization loses its ability to respond to changing needs**. In other words, it loses its agility. So we need to give the development team enough time to undertake refactoring. **Refactoring is the housekeeping, cleanup, and simplifications that need to be made to code to ensure it is stable and can be maintained over the long term**.

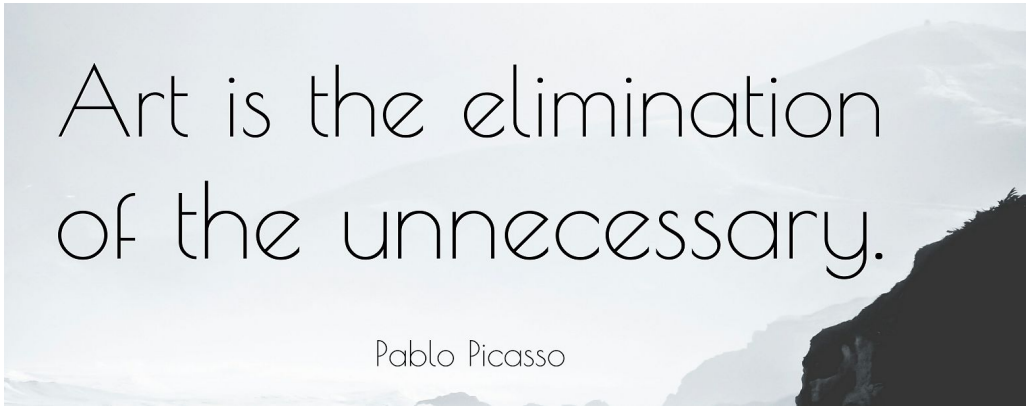
Simplicity—the art of maximizing the amount of work not done—is essential



The most reliable features are those we don't build—since there is nothing that could go wrong with them. Because so many features that are built are never actually used, and because complex systems have an increased potential to be unreliable, agile methods focus on simplicity. This means boiling down our requirements to their essential elements only.

Complex projects take longer to complete,are exposed to a longer horizon of risk, and have more potential failure points and opportunities for cost overruns.

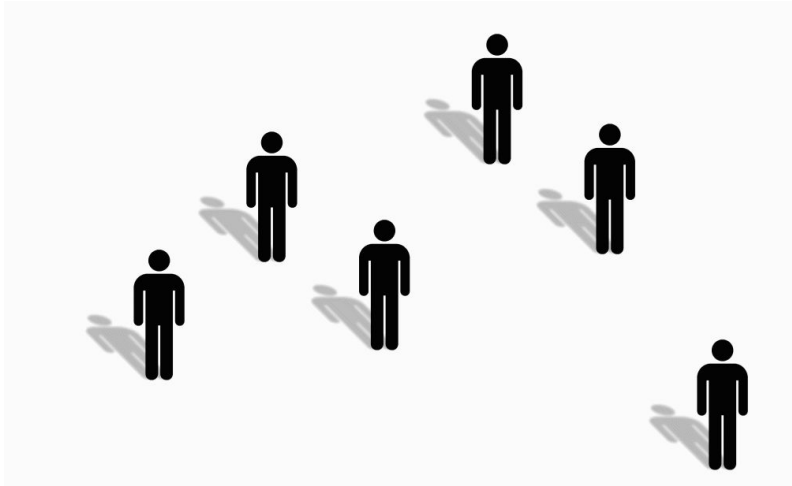
Simplicity—the art of maximizing the amount of work not done—is essential



Agile methods seek the "**simplest thing that could possibly work**" and recommend that this solution be built first.

This approach is not intended to preclude further extension and elaboration of the product—instead, it simply says, "Let's get the plain- vanilla version built first." This approach not only mitigates risk but also helps boost sponsor confidence.

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



To get the best out of people, we have to let them self-organize. People like self-organizing because it allows them to find an approach that works best for their methods, their relationships, and their environment.

Self-organizing teams that have the **autonomy to make local decisions** have a **higher level of ownership and pride in the architectures, requirements, and designs they create** than in those that are forced on them or "suggested" by external sources. Ideas created by the team have already gone through the team vetting process for alignment and approval—so **they don't need to be "sold" to the team**. In contrast, ideas that come from outside sources need to be sold to the team for the implementation to be successful, and this is sometimes a challenging task.

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

Gathering lessons learned at the end of a project is, frankly, too little, too late. Instead, we need to **gather our lessons learned while they are still applicable and actionable**. This means we need to gather them during the project and—most importantly—**make sure we do something about what we've learned** to adjust how we complete the remainder of the project.

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

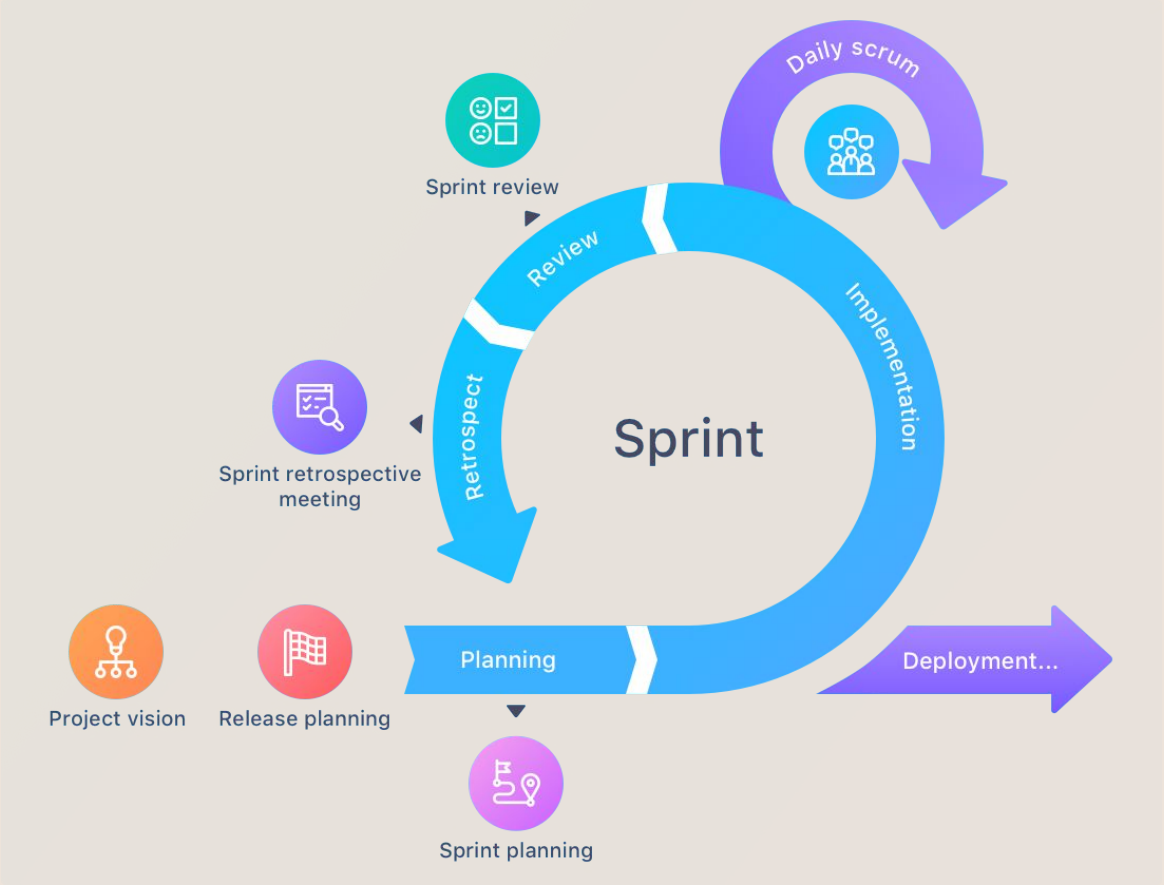
Agile methods employ frequent lookbacks, called“**retrospectives**,”to **reflect on how things are working on the project and identify opportunities for improvements**. These retrospectives are typically done at the end of each iteration, ensuring that the team has regular opportunities to review their process.

One advantage of doing retrospectives so frequently is that we **don’t forget about problems and issues**. Compare this to conducting a single lessons learned review at the end of a project, in which team members are asked to think back over a year or more to recall what went well and where they ran into issues.

Agile Methodologies

There are over a dozen Agile methodologies

Scrum


Daily scrum
Sprint review
Review
Implementation
Retrospect
Sprint
Sprint retrospective meeting
Project vision
Release planning
Planning
Deployment...
Sprint planning

Scrum is a popular agile model that is **lightweight and easy to understand**—but like all agile methods, it is difficult to truly master. The methodology documented in the "Scrum framework" is **a set of practices, roles, events, artifacts, and rules that are designed to guide the team in executing the project.**

Scrum is **guided by 3 Pillars**:

1. **Transparency** - giving visibility to those responsible for the outcome
2. **Inspection** - doing timely checks of how well the project is progressing toward its goals, looking for problematic deviations or differences from the goals
3. **Adaptation** - adjusting the team's process to minimize further issues if an inspection shows aproblemor undesirable trend

Scrum is a popular agile model that is **lightweight and easy to understand**—but like all agile methods, it is difficult to truly master. The methodology documented in the "Scrum framework" is **a set of practices, roles, events, artifacts, and rules that are designed to guide the team in executing the project.**

Scrum recognizes **five fundamental values**:

1. **Focus**
2. **Courage**
3. **Openness**
4. **Commitment**
5. **Respect**

These values reflect the concepts laid out in the Agile Manifesto, and they are also similar to the core values of XP (as we'll see when we get to that methodology).

S68.gif

Sprint – a time boxed iteration of one month or less in which the team builds a potentially releasable product.

- Most Scrum sprints are **one or two weeks long**.
- Each sprint is like a **mini-project**.
- During a sprint, **no changes are made that would affect the sprint goal**.
- The scope might be clarified or renegotiated as new information becomes available, but if, for example, a change has to be made to the members of the development team, that wouldn't be done during a sprint.

The sequence of activities within each sprint consists of a **sprint planning meeting**, a **development period** that includes **daily scrums**, a **sprint review meeting**, and a **sprint retrospective meeting**.

- A sprint can be cancelled before the timebox is over if the sprint goal becomes obsolete because of a change in business direction or technology conditions.
- **Only the product owner can cancel the sprint**.
- When that happens, any incomplete product backlog items are re-estimated and returned to the product backlog.

Scrum Team Roles

Scrum teams are made up of the development team, the product owner, and the Scrum Master.

Development Team

- The development team is the **group of professionals who build the product** increments in each sprint.
- The members of the development team are **self-organizing**—they are empowered to manage their own work.
- Scrum teams are also **cross-functional**, each team member can fulfill more than one of the roles needed.

Scrum Team Roles

Scrum teams are made up of the development team, the product owner, and the Scrum Master.

Product Owner

- Maximizes the value of the product by **managing the product backlog**, or list of work to be done.
- Makes sure that the business and the team have a **shared understanding** of the project vision, the project goals, and the details of the work to be done

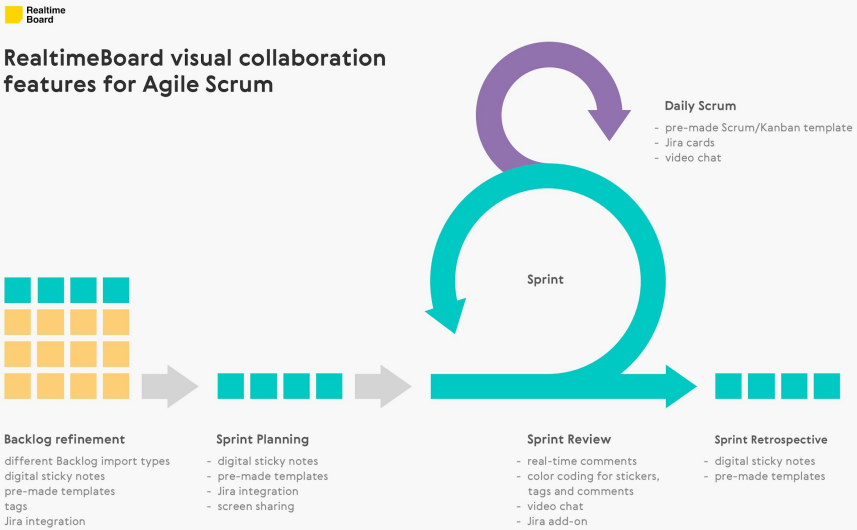
Scrum Team Roles

Scrum teams are made up of the development team, the product owner, and the Scrum Master.

Scrum Master

- Ensures that the **Scrum methodology is understood and used effectively**.
- A **servant leader to the development team**, removing any impediments to their progress, facilitating their events (meetings), and coaching the team members.
- **Assists the product owner with managing the backlog and communicating the project vision**, project goals, and the details of the backlog items to the development team.
- Finally, the ScrumMaster serves the organization by facilitating its adoption of Scrum, not just on one project, but on a **wider scale throughout the organization**.

Scrum Activities (Events & Ceremonies)



The Scrum methodology defines five “activities,” which are actually meetings that are focused on a specific purpose. These include:

1. **Product backlog refinement**
2. **Sprint planning meetings**
3. **Daily scrums**
4. **Sprint reviews**
5. **Sprint retrospectives**

The last four of these meetings are the team’s planned opportunities for inspection and adaptation (two of the three pillars) in each sprint.

RealtimeBoard visual collaboration features for Agile Scrum

Backlog refinement

- different Backlog import types
- digital sticky notes
- pre-made templates
- tags
- Jira integration

Sprint Planning

- digital sticky notes
- pre-made templates
- Jira integration
- screen sharing

Sprint Review

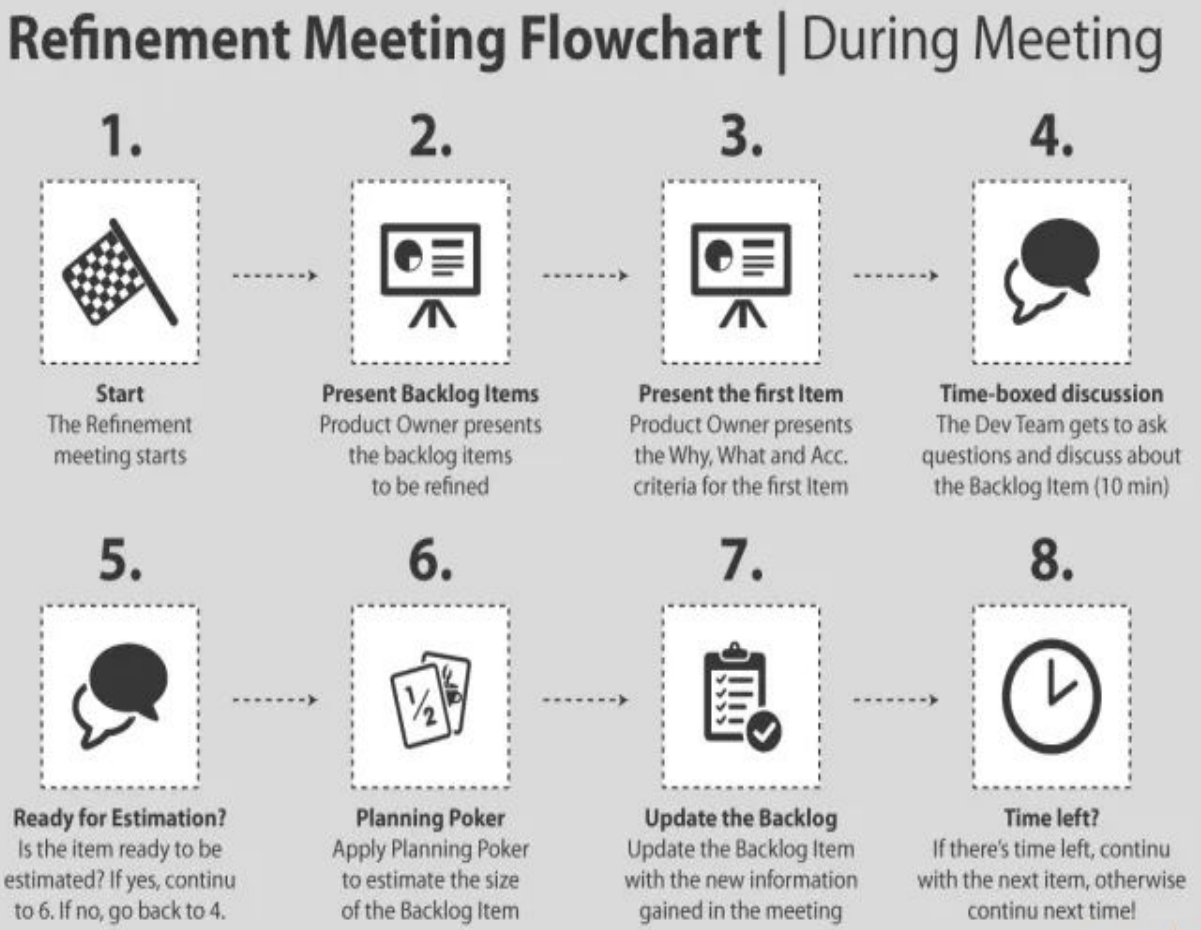
- real-time comments
- color coding for stickers, tags and comments
- video chat
- Jira add-on

Sprint Retrospective

- digital sticky notes
- pre-made templates

Backlog Refinement

The backlog refinement meetings are where "**grooming the backlog**" is done. This basically means that everyone involved in the project gathers to discuss and update the items in the backlog.



Sprint Planning Meeting

- Everyone gathers to determine **what will be delivered** in the upcoming sprint and how that work will be achieved.
- The **product owner presents the updated backlog items**, and the group discusses them to ensure they have a shared understanding.
- The development team forecasts **what can be delivered** in the sprint, based on their estimates, projected capacity, and past performance. With this in mind, they define the **sprint goal**.
- The development team then **determines how** that functionality will be built and how they will **organize themselves** to deliver the sprint goal.

Daily Scrum Meeting

Time box

Same place

Same time

Facilitated by Scrum Master

Full team presence

Focus on 3 questions

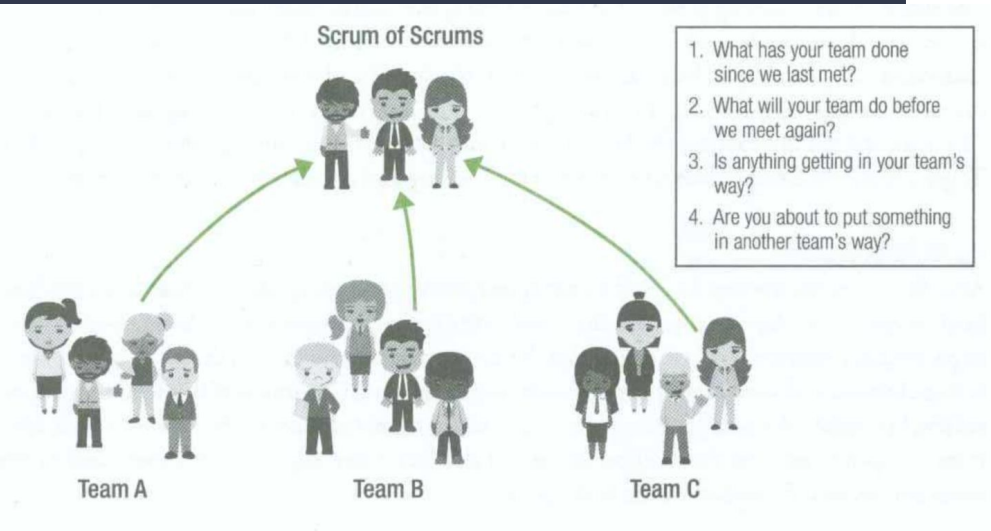
3 Main Questions:

1. What did I do yesterday?
2. What will I do today?
3. What's in my way?

- The daily scrum is a **15-minute timeboxed meeting** that is held at the **same time and place everyday** while the team is working toward the sprint goal.
- The **Scrum Master makes sure the meeting happens everyday**, and follows up on any identified obstacles.
- The daily scrum is primarily for the members of the development team, who use it to synchronize their work and report any issues they are facing.
- The scope of this meeting is strictly limited—each member of the team briefly answers three questions.

Scrum of Scrums

As Scrum projects get larger, multiple Scrum teams might need to coordinate their work. To do this, they commonly use an approach called “scrum of scrums.”

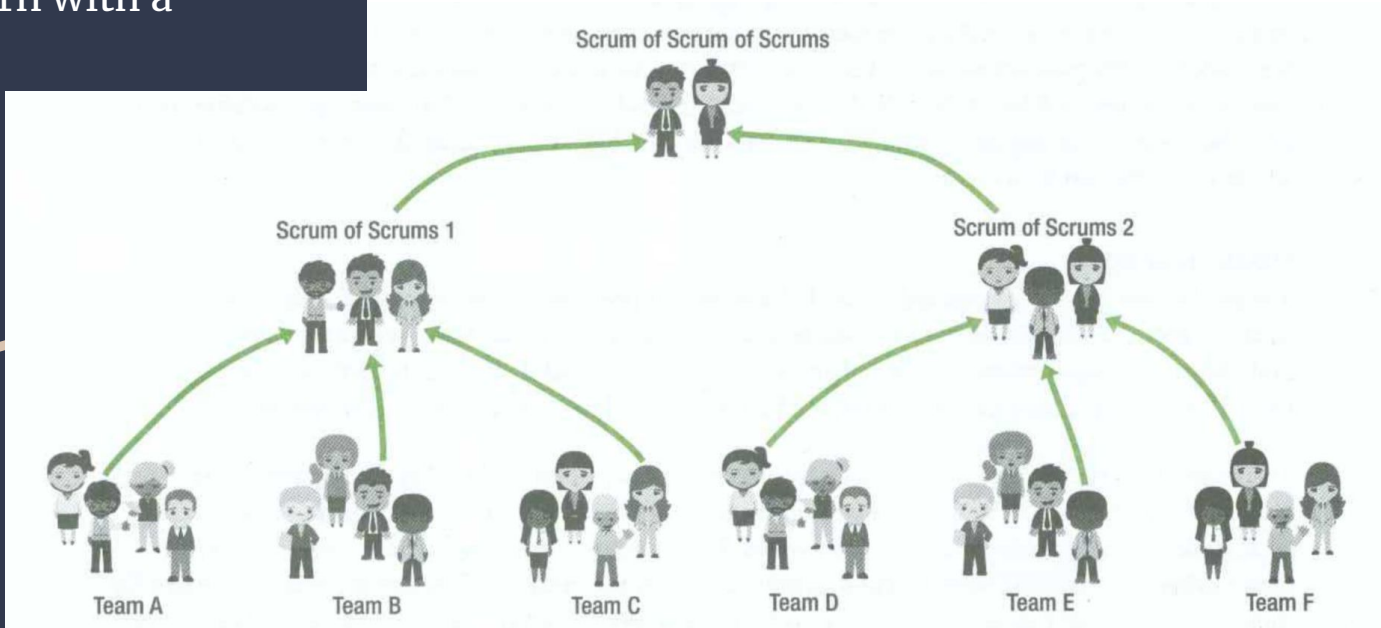


- As the name suggests, this is a meeting in which a **representative from each team reports** their progress to the representatives of the other teams.
- Just like a normal daily scrum, the participants answer three questions, and they might also address a fourth question to help surface potential conflicts between teams: “**Are you about to put something in another team’s way?**”

Scrum of Scrum of Scrums

Larger endeavors may even warrant a “scrum of scrum of scrums” where the teams repeat this pattern with a third-level meeting.

- Here, a representative from each scrum of scrums will attend a “scrum of scrum of scrums” to coordinate the work across a larger set of teams.



Sprint Review

The sprint review meeting is held at the end of the sprint and includes the development team, the product owner, and the Scrum Master (and potentially other stakeholders).

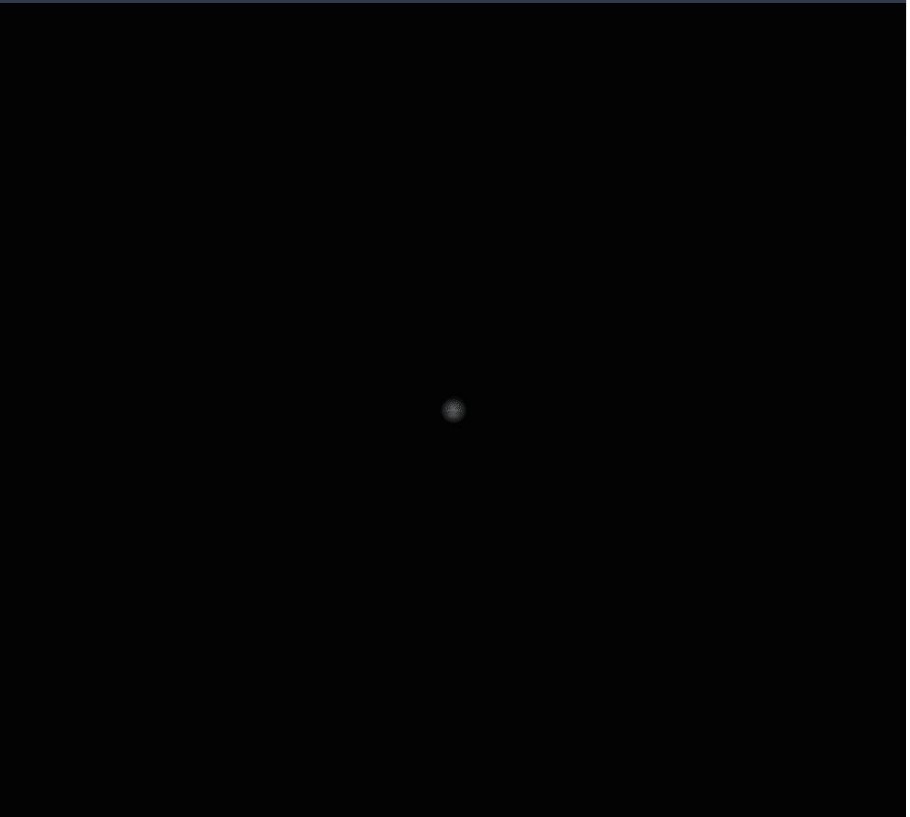
- The team **demos the increment**, or evolving product, that they built in the sprint to the product owner.
- The **product owner inspects the work** to see whether it is acceptable, deciding if it is “done" or explaining what is missing.
- The team and the product owner **discuss the increment and the remaining items in the product backlog**.
- Together, they make any changes needed to the backlog and **decide what to work on next**.

Sprint Retrospective

After the sprint review, but before the next sprint planning meeting, the development team holds their final “inspect and adapt” activity for the sprint.

- This is the **opportunity to gather lessons learned** and look for opportunities for improvement.
- The timing of this meeting allows to consider the product owner’s feedback from the sprint review in deliberations and also **factor any improvements identified during the meeting into the plan for the next sprint**.
- The reflection that takes place during the meeting might cover anything that occurred during the sprint, including the areas of people, process, and product.
- Explore what went well, look for areas for improvement, and **decide what changes to implement in the next sprint**.

Scrum Artifacts



These are three tangible items that are produced or used by the team during a sprint

1. **The product increment**
2. **The product backlog**
3. **The sprint backlog**

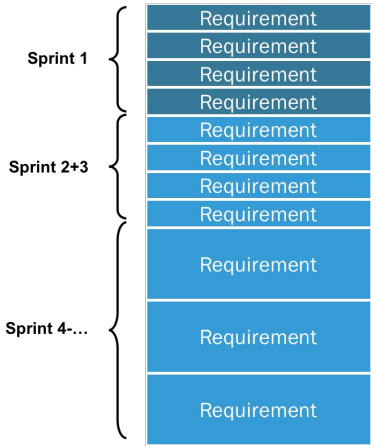
Product Increment

During each sprint, the members of the development team build an increment of the solution (the end product of the project).

- As we've seen, in the sprint review, they demo their latest increment to get feedback from the product owner and find out if that item is "done."
- To maximize the chances that the product increment will be acceptable, the team and the product owner need to **agree upon a definition of done** before the team begins working on it so that everyone has a **shared understanding of what "done" will look like for that increment**.

Product Backlog

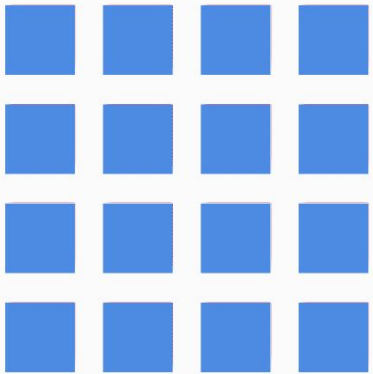
A prioritized list of all the work that needs to be done to build the product—it serves as the single source for all the product requirements.



- The backlog is **constantly being refined**.
- This activity is done by the development team and the product owner working together. **The team estimates the work items, and product owner prioritizes them**.
- For example, each time the team refines their estimates with a higher level of detail, the product owner might need to adjust the priority of those items in the backlog.

Sprint Backlog

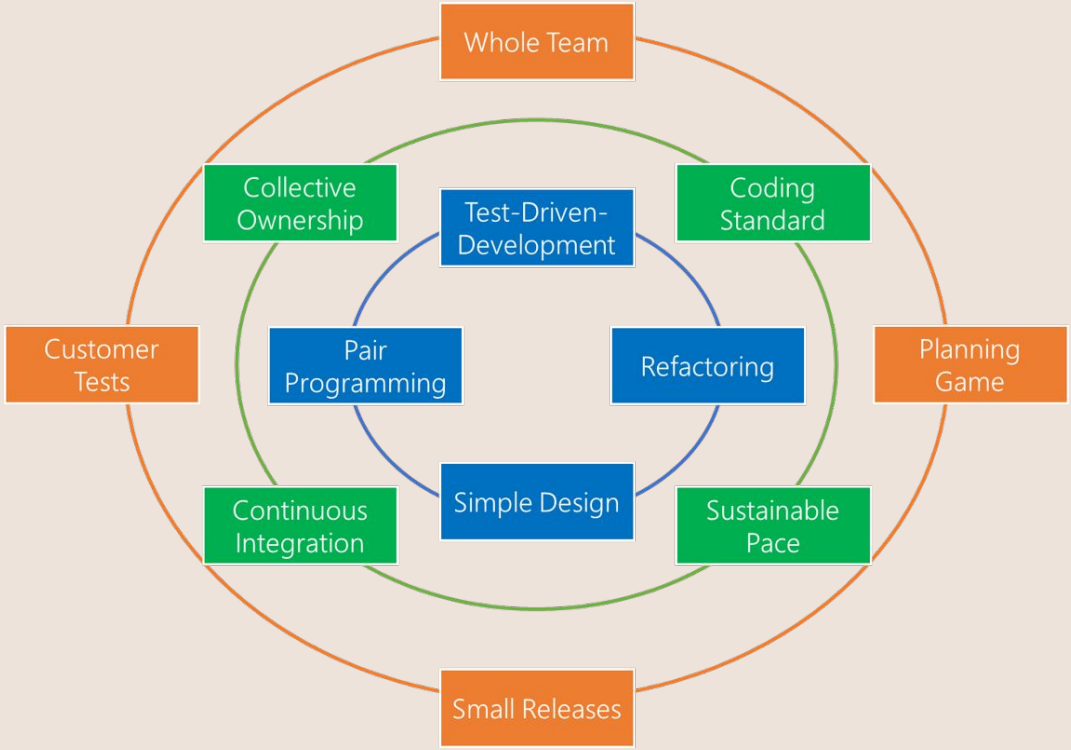
The **subset of items in the product backlog** that have been selected as the goal of a specific sprint.



- Along with the sprint backlog, the team develops a **plan for how they will achieve the sprint goal**—this is their commitment for the functionality that they will deliver in the sprint.
- The sprint backlog serves as a **highly visible view of the work being undertaken** and may only be updated by the development team.

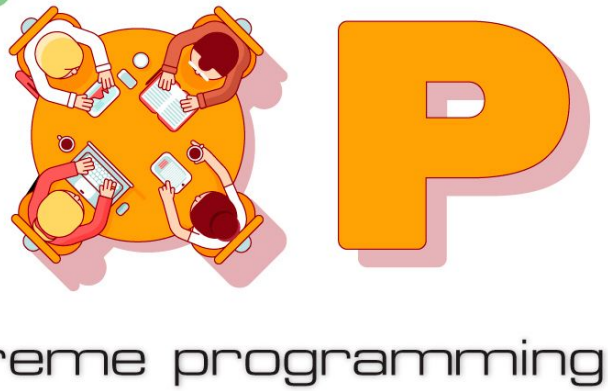
ITEM	DEVELOPMENT TEAM	PRODUCT OWNER	SCRUM MASTER
Estimates	X		
Backlog Priorities		X	
Agile Coaching			X
Coordination of Work	X		
The Definition of “Done”	X	X	X
Process Adherence			X
Technical Decisions	X		
Sprint Planning	X	X	X

Extreme Programming (XP)



Extreme Programming (XP)

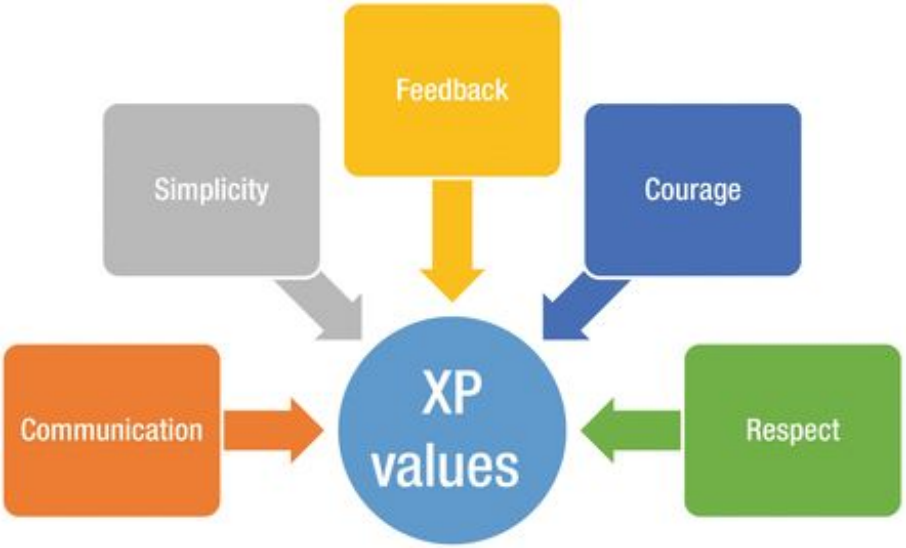
MANAGEMENT



An agile method that is **focused on software development**.

While Scrum focuses at the project management level on prioritizing work and getting feedback, **XP focuses on software development best practices**.

Core Values of Extreme Programming



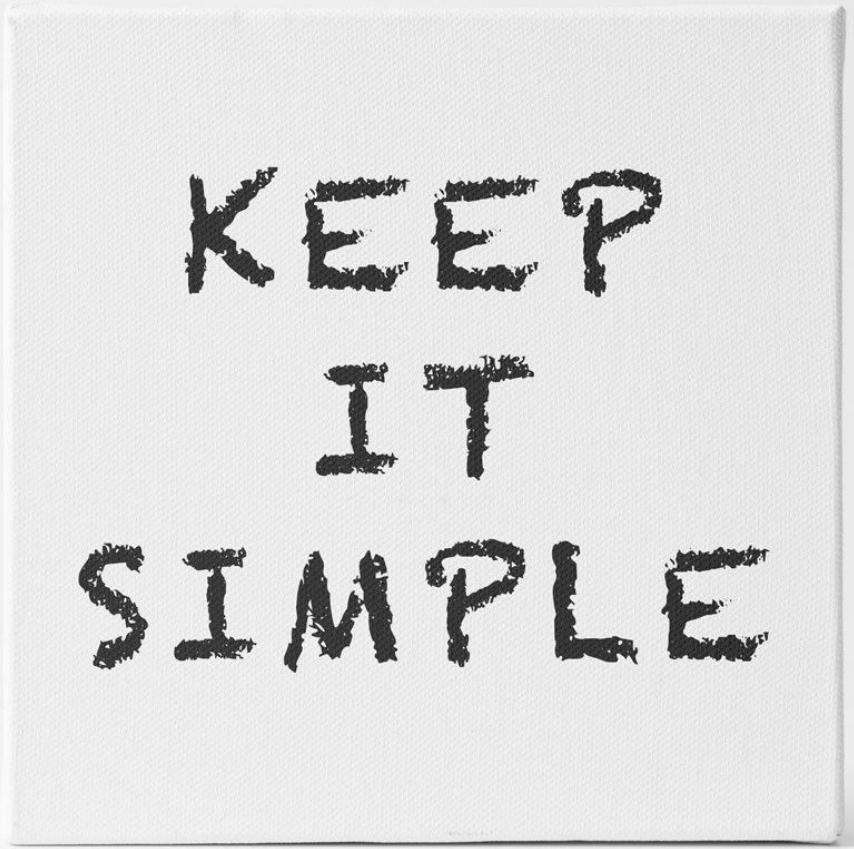
The core values of XP are:

1. Simplicity
2. Communication
3. Feedback
4. Courage
5. Respect

These values manifest themselves in the practices undertaken throughout the XP life cycle.

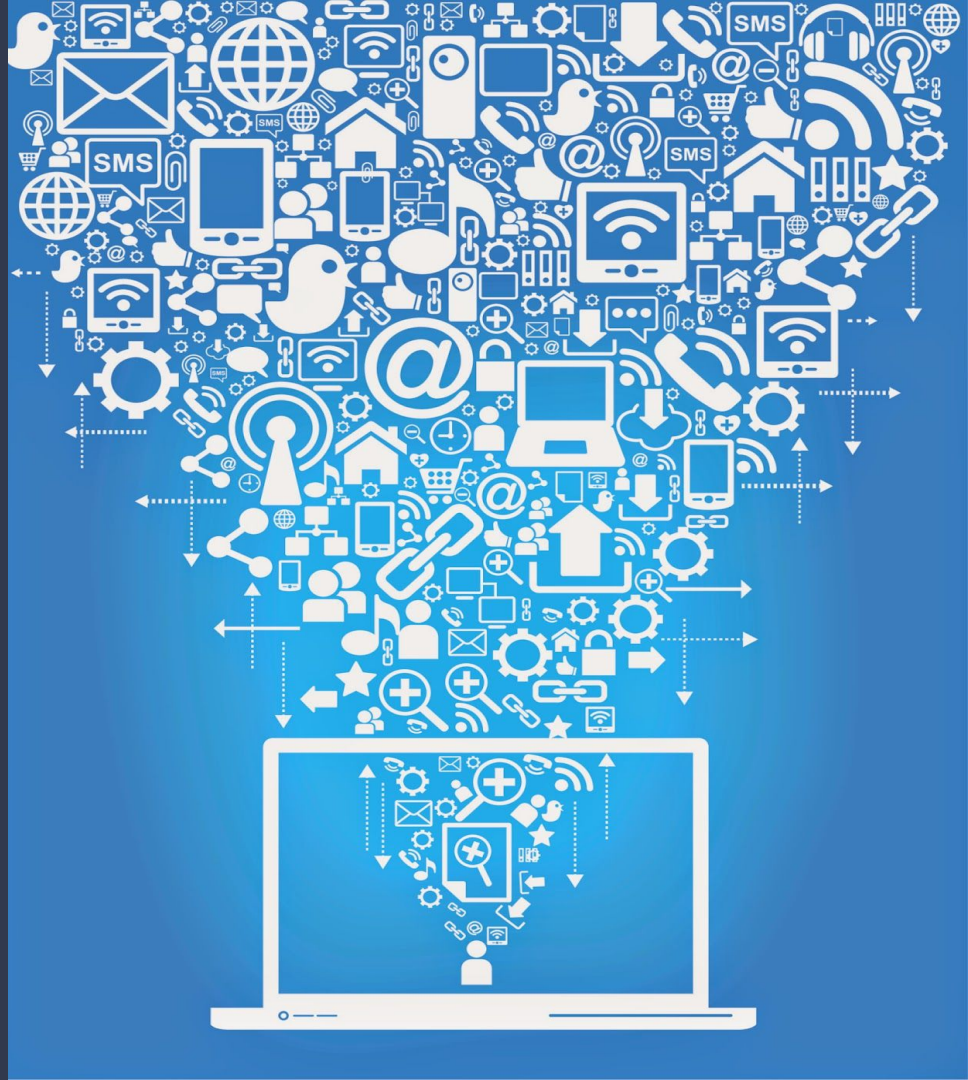
1

SIMPLICITY - Reducing complexity, extra features, and waste. XP teams keep the phrase **“Find the simplest thing that could possibly work”** in mind, and build that solution first.



2

Communication - making sure all the team members **know what is expected of them** and what other people are working on.



3

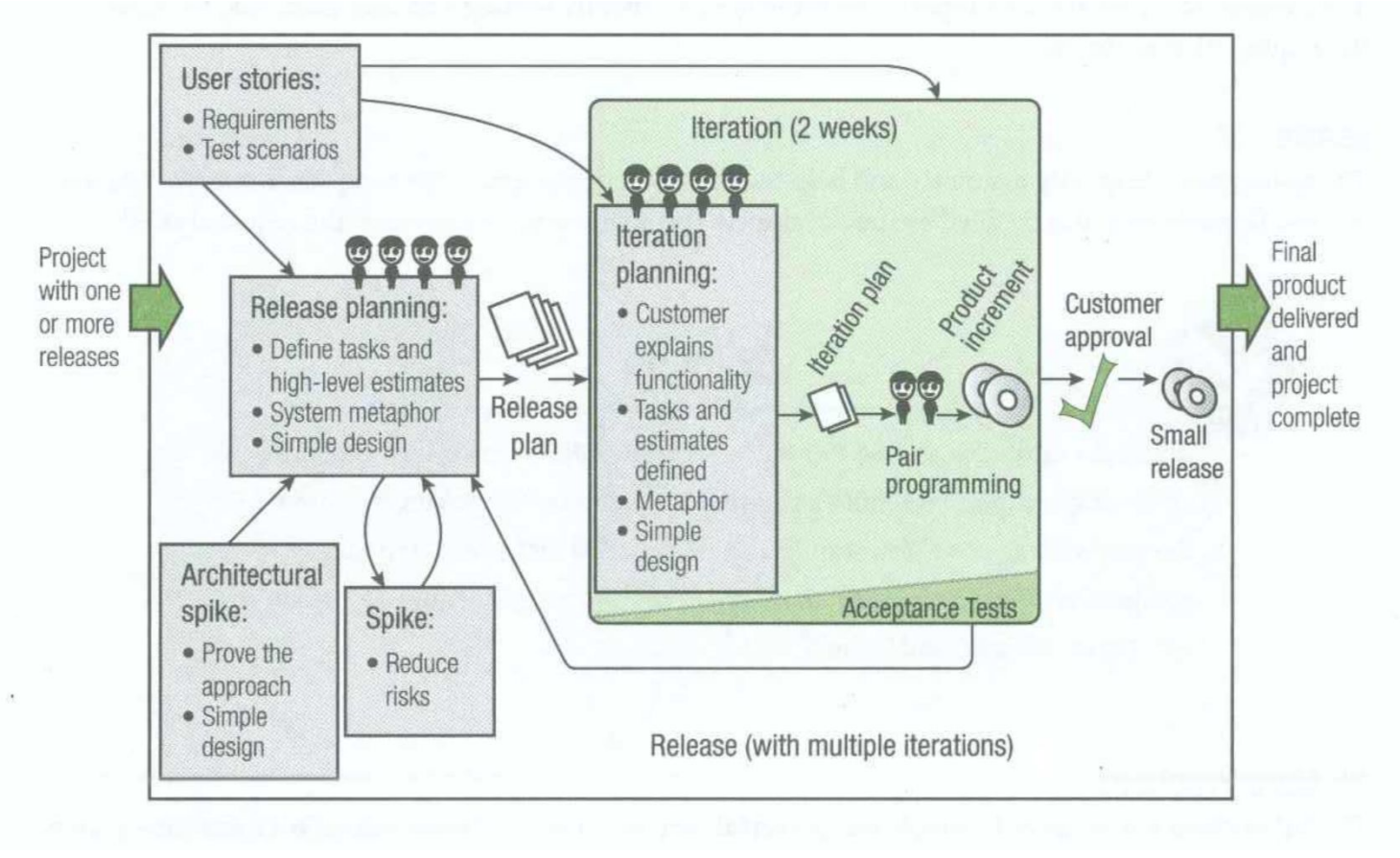
Feedback - The team should get **impressions of suitability early**. Failing fast can be useful, especially if in doing so we get new information while we still have time to improve the product.

4

Courage - In pair programming, team members **share code and often need to make bold simplifications and changes to that code**. Backed up by automated builds and unit tests, developers need to have the confidence to make important changes.

5

Respect - people **work together as a team** and **everyone is accountable** for the success or failure of the project. Team members need to **recognize that people work differently**, and **respect those differences**.



Extreme Programming Lifecycle

The Extreme Programming Lifecycle

- XP teams use lightweight requirements called "**user stories**" to plan their releases and iterations.
- Iterations are typically two weeks long, and **developers work in pairs** to write code during these iterations.
- All software developed is subjected to **rigorous and frequent testing**.
- Upon approval by the on-site customer, the software is delivered as **small releases**.
- "**Spikes**" are periods of work undertaken to reduce threats and issues,
- "**Architectural spikes**" are iterations used to prove a technological approach. The spikes are blended into the release planning processes.

XP Team Roles

COACH

Acts as a mentor to the team, guiding the process and helping the team members stay on track. Also helps the team become more effective and reinforces communication both within the team and across teams. This role shares many responsibilities with a ScrumMaster.

PROGRAMMER

The developers who build the product by writing and implementing the code for the requested user stories.

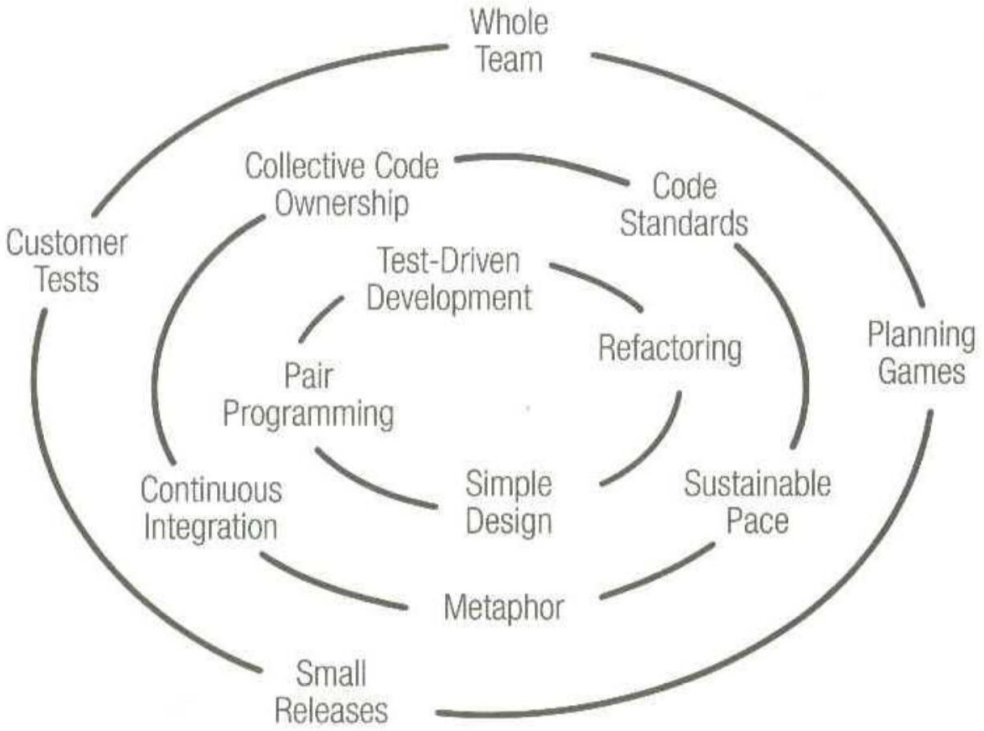
CUSTOMER

The business representative who provides the requirements, priorities, and business direction for the project. This person defines the product that will be built, determines the priority of its features, and confirms that the product actually works as intended. This role is similar to the product owner in Scrum.

TESTER

The testers provide quality assurance and help the customer define and write acceptance tests for the user stories. This role may also be filled by the developers (programmers), if they have the required skills.

XP 13 Core Practices



Whole Team

- The whole team practice is the idea that all the contributors to an XP project **sit together in the same location**, as members of a single team.
- XP emphasizes the notion of generalizing specialists, as opposed to role specialists. In other words, **anyone who is qualified to perform a role can undertake it**—the roles are not reserved for people who specialize in one particular area.
- This practice helps optimize the use of resources, since people who can perform multiple jobs are able to switch from one role to another as the demand arises.
- The practice also allows for **more efficient sharing of information** and helps eliminate the possibility that people in certain roles will be idle or over stretched at certain points in the project.

Planning Games

- XP has two primary planning activities
- **Releases** are a push of new functionality all the way to the production user.
  - During release planning, the **customer outlines the functionality required**, and the **developers estimate how difficult the functionality will be to build**.
  - Armed with these estimates and priorities, the customer lays out the plan for the project delivery. Since the initial attempts at estimating will likely be imprecise, this process is **revisited frequently and improved** as the priorities and estimates evolve.
- **Iterations** are the short development cycles within a release that Scrum calls “sprints.”
  - Iteration planning is done at the start of every iteration.
  - The **customer explains what functionality they would like to see in the next iteration**, and then the **developers break this functionality into tasks and estimate the work**.
  - Based on these estimates (which are more refined than the release planning estimates) and the amount of work accomplished in the previous iteration, the team commits to the work items they think they can complete in the two-week period.

Small Releases

- Frequent, small releases to a test environment are encouraged in XP, both at the iteration level, to demonstrate progress and increase visibility to the customer, and at the release level, to rapidly deploy working software to the end users.
- Quality is maintained in these short delivery timeframes by rigorous testing and through practices like continuous integration, in which suites of tests are run as frequently as possible.

Customer Tests

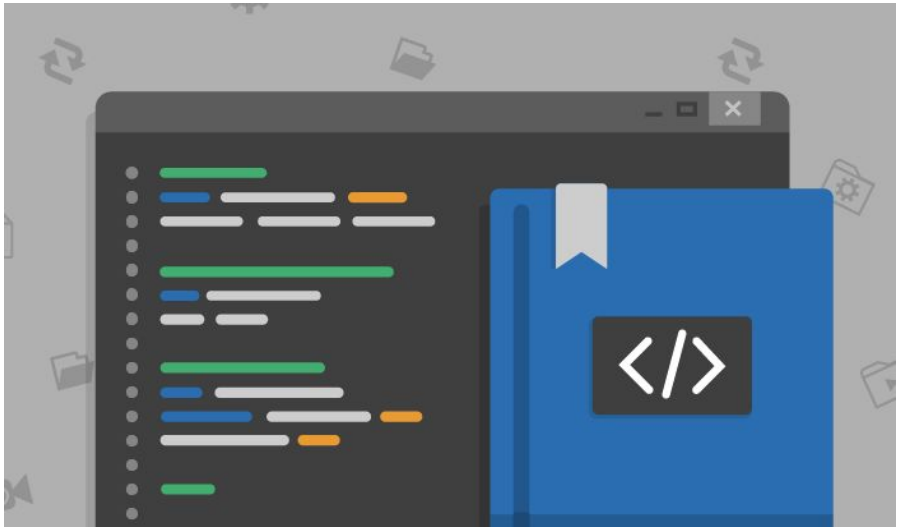
- As part of defining the required functionality, the **customer describes one or more test criteria** that will indicate that the software is working as intended.
- The team then builds **automated tests** to prove to themselves and the customer that the software has met those criteria.

Collective Code Ownership

- Any **pair of developers can improve or amend any code**. This means multiple people will work on all the code, which results in increased **visibility and broader knowledge** of the code base.
- This practice leads to a **higher level of quality** - with more people looking at the code, there is a **greater chance defects will be discovered**. There is also less of an impact to the project if one of the programmers leaves, since the knowledge is shared.

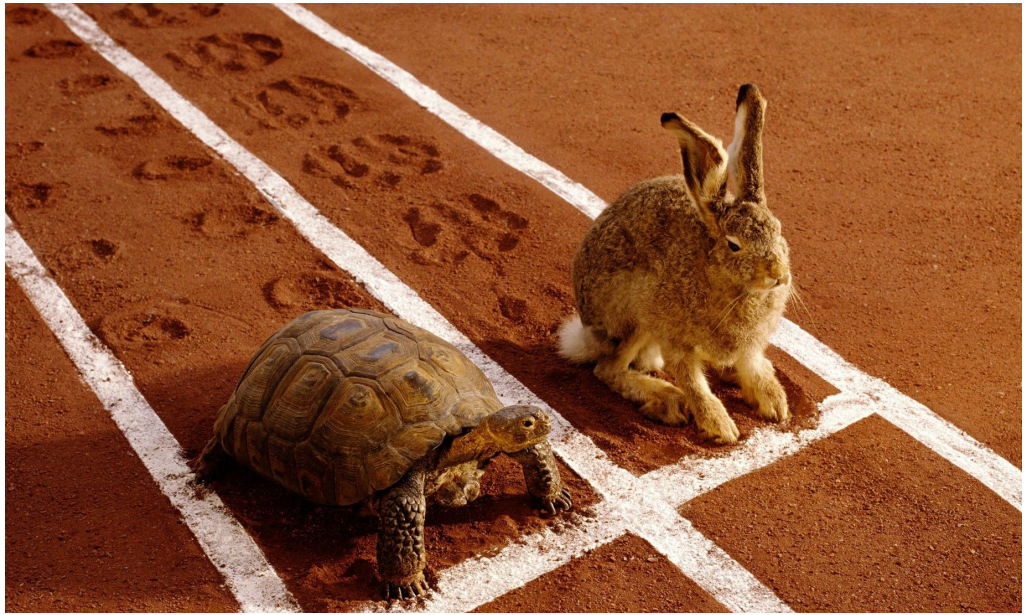
Code Standards

- Although collective code ownership has its advantages, allowing anyone to amend any code can result in issues if the team members take different approaches to coding.
- To address this risk, XP teams follow a **consistent coding standard** so that all the code looks as if it has been written by a single, knowledgeable programmer.
- The specifics of the standard each team uses are not important; what matters is that the team takes a **consistent approach to writing the code.**



Sustainable Pace

- XP recognizes that the highest level of productivity is achieved by a team operating at a sustainable pace.
- While periods of overtime might be necessary, repeated long hours of work are unsustainable and counterproductive.
- The practice of maintaining a sustainable pace of development optimizes the delivery of long-term value.



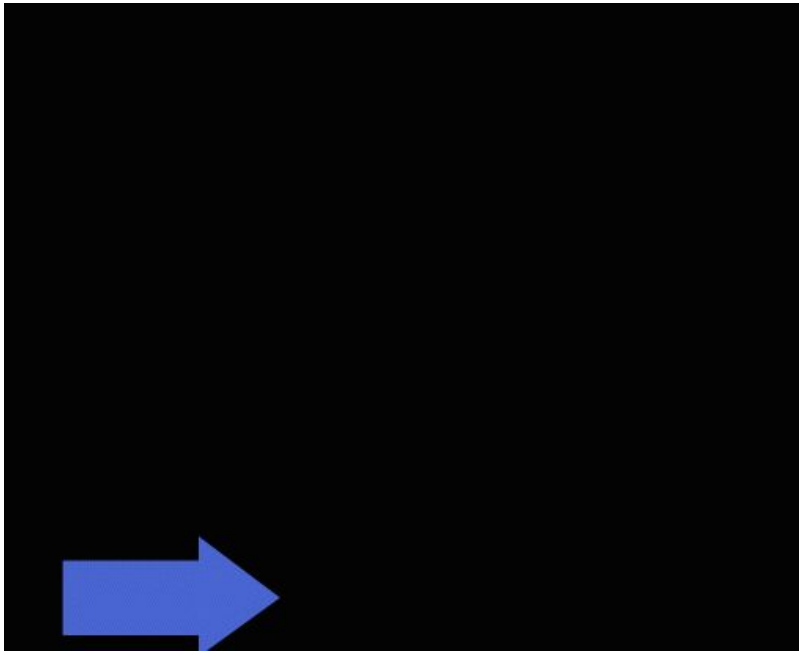
Metaphor

- XP uses metaphors and similes to explain designs and create a shared technical vision. These descriptions establish comparisons that all the **stakeholders can understand** to help explain how the system should work.
- For example, “The billing module is like an accountant who makes sure transactions are entered into the appropriate accounts and balances are created.”

- Even if the team cannot come up with an effective metaphor to describe something, they can use a common set of names for different elements to ensure everyone understands where and why changes should be applied.

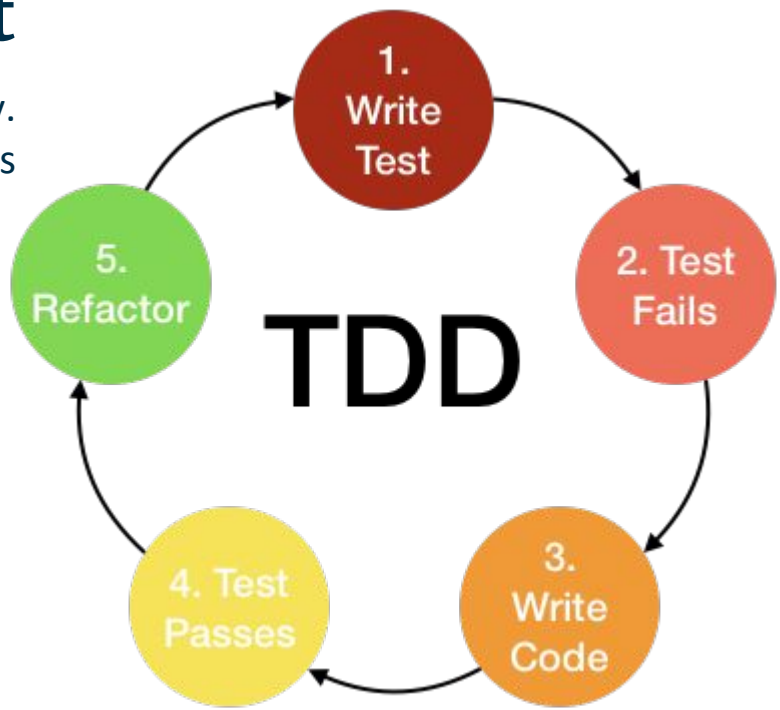
Continuous Integration

- Integration involves **bringing the code together** and making sure it **all compiles and works together**. This practice is critical, because it brings problems to the surface before more code is built on top of faulty or incompatible designs.
- XP employs continuous integration, which means **every time a programmer checks in code to the code repository** (typically several times a day), **integration tests are run automatically**.
- Such tests highlight broken builds or problems with integration, so that the **problems can be addressed immediately**.



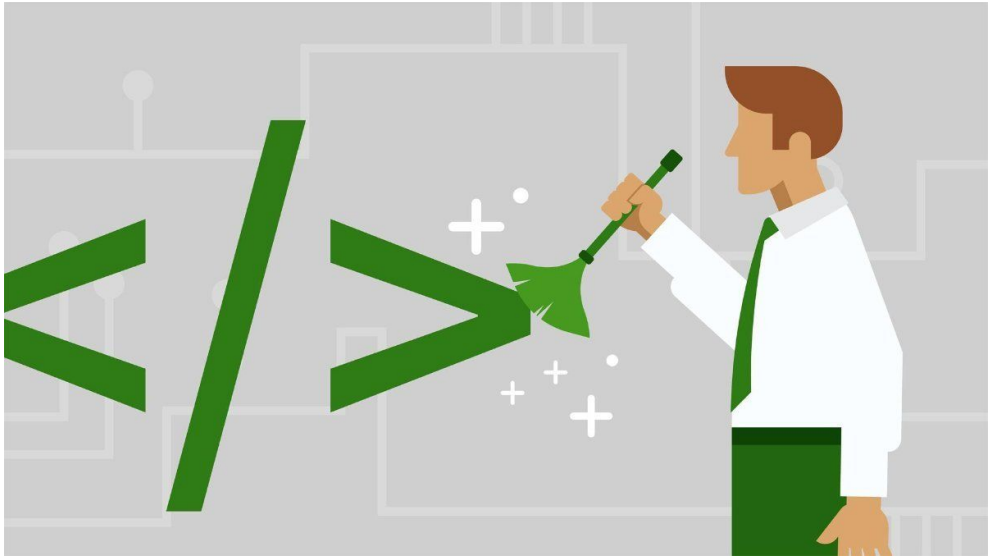
Test-Driven Development

- Testing is a critical part of the XP methodology. To ensure good test coverage so that problems can be highlighted early in development, XP teams often use the practice of test-driven development.
- The team **writes the acceptance tests prior to developing the new code**.
- If the tests are working correctly, the initial code that is entered will fail the tests, since the required functionality has not yet been developed. **The code will pass the test once it is written correctly.**
- The test-driven development process **strives to shorten the test-feedback cycle as much as possible** to get the benefits of early feedback.



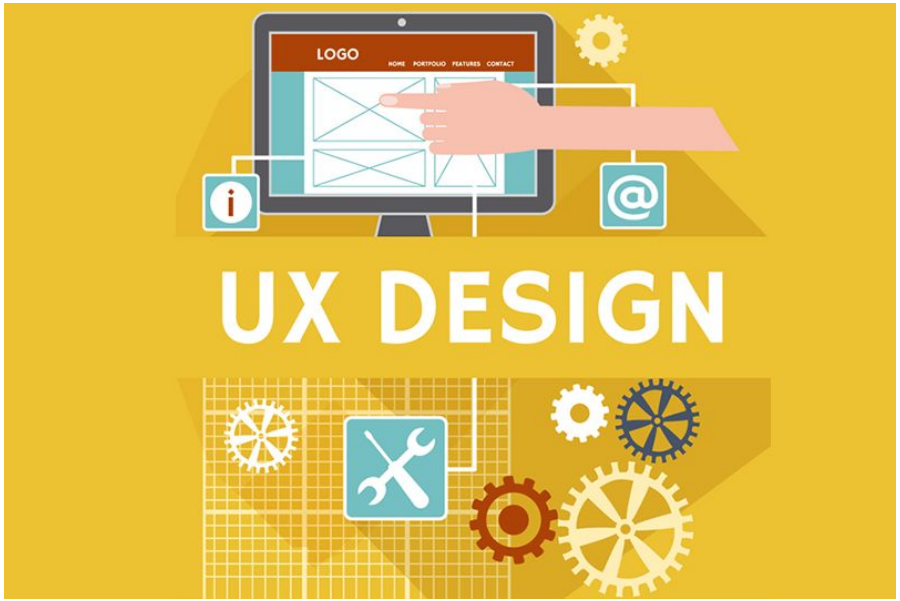
Refactoring

- Refactoring is the process of **improving the design of existing code without altering its external behavior or adding new functionality**.
- By keeping the design efficient, changes and new functionality can easily be applied to the code.
- Refactoring focuses on **removing duplicated code, lowering coupling** (dependent connections between code modules), **and increasing cohesion**.



Simple Design

- By focusing on keeping the design simple but adequate, XP teams can **develop code quickly and adapt it as necessary**. The design is kept appropriate for what the project currently requires. It is then revisited iteratively and incrementally to ensure it remains appropriate.
- XP follows a deliberate design philosophy that asks, "**What is the simplest thing that could work?**" as opposed to complex structures that attempt to accommodate possible future flexibility.

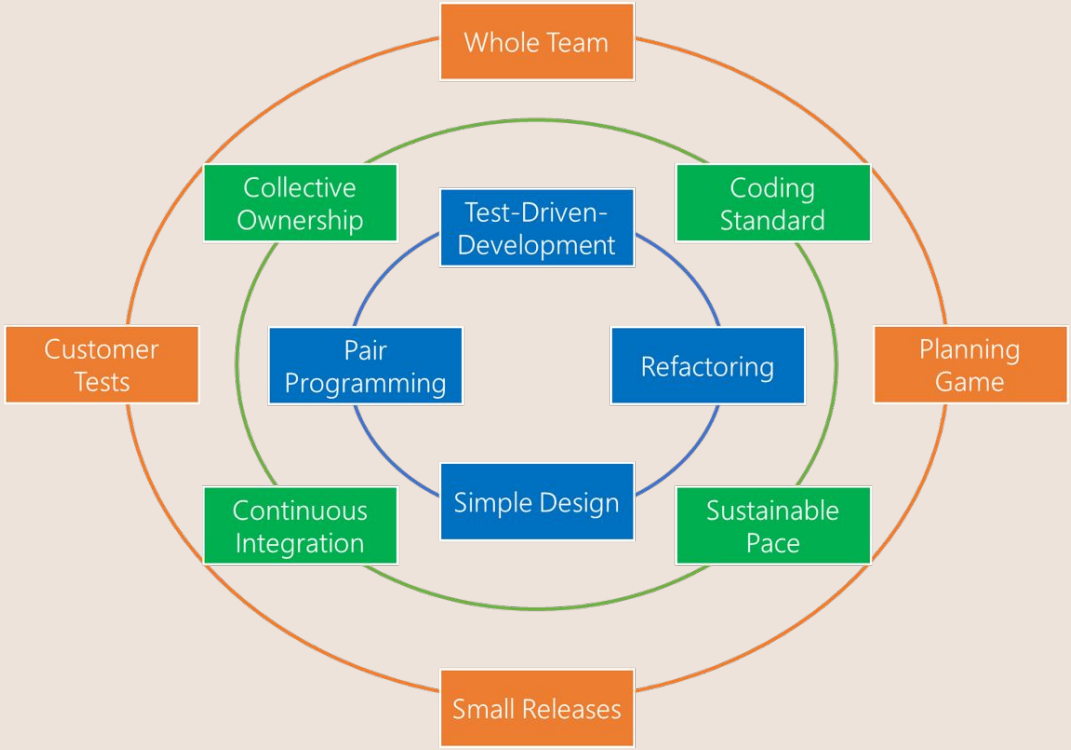


- Since code bloat and complexity are linked to many failed projects, **simple design is also a risk mitigation strategy**.

Pair Programming

- InXP, production co**de is written by two developers working as a pair**. While one person writes the code, the other developer reviews the code as it is being written—and the two change roles frequently.
- This practice may seem inefficient, but XP advocates assert that it saves time because the **pairs catch issues early** and there is a benefit in that the two people will have a **larger knowledge base**.
- Working in pairs also helps **spread knowledge about the system through the team**.

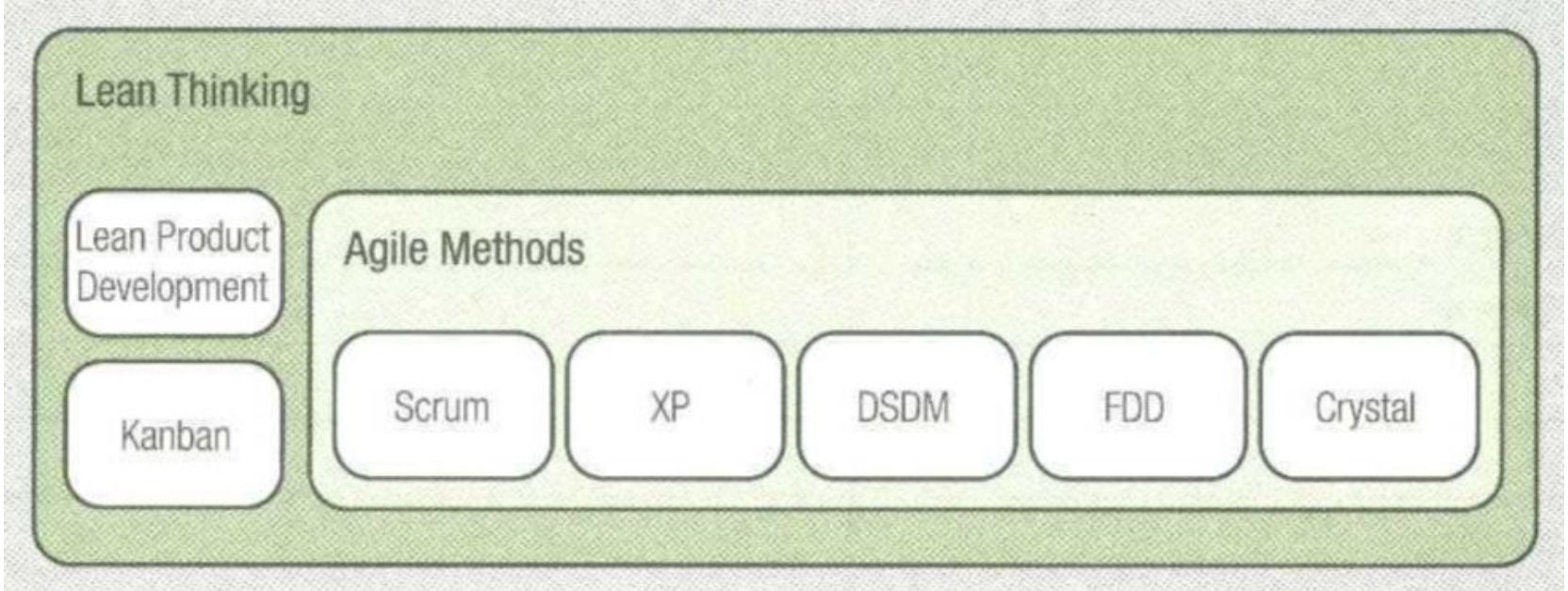
Lean Product Development



Lean Product Development

While the original lean production systems dealt with manufacturing products, **lean product development deals with developing new and better products**. The high-level principles of lean product development include:

1. **Using visual management tools**
2. **Identifying customer-defined value**
3. **Building in learning and continuous improvement**



Lean is a Superset of Agile

7 Core Concepts of Lean Software Development

1. Eliminate Waste
2. Empower the Team
3. Delivery Fast
4. Optimize the Whole
5. Build Quality In
6. Defer Decisions
7. Amplify Learning

1. Eliminate Waste

To maximize value, we must **minimize waste**. In knowledge work, waste can take the form of partially done work, delays, handoffs, unnecessary features, etc.

Therefore, to increase the value we are getting from projects, we must **develop ways to identify, and then remove, waste**.

2. Empower the Team

Rather than taking a micromanagement approach, we should respect the team members' superior knowledge of the technical steps required on the project and **let them make local decisions** to be productive and successful.

3. Deliver Fast

We can maximize the project's return on investment (ROI) by **quickly producing valuable deliverables and iterating through designs**.

We will find the best solution through the **rapid evolution of options**.

4. Optimize the Whole

We aim to see the system as **more than the sum of its parts**. We go beyond the pieces of the project and look for how it aligns with the organization.

We also focus on **forming better intergroup relations**.

5. Build Quality In

Lean development doesn’t try to “test in” quality at the end; instead, we **build quality into the product and continually assure quality** throughout the development process, using techniques like refactoring, continuous integration, and unit testing.

6. Defer Decisions

We balance **early planning** with **making decisions and commitments as late as possible**.

For example, this may mean re-prioritizing the backlog right up until it is time to do the work, or avoiding being tied to a nearly technology-bounded solution.

7. Amplify Learning

This concept involves **facilitating communication early and often**, getting **feedback as soon as possible**, and **building on what we learn**. Since knowledge work projects are business and technology learning experiences,**we should start early and keep learning**.

7 Wastes of Lean Software Development

1. **Partially Done Work**
2. **Extra Processes**
3. **Extra Features**
4. **Task Switching**
5. **Waiting**
6. **Motion**
7. **Defects**

1. Partially Done Work

Work started, but not complete; partially done work can entropy

For example:

- Code waiting for testing
- Specs waiting for development

2. Extra Processes

Extra work that does not add value

For example:

- Unused documentation
- Unnecessary approvals

3. Extra Features

Features that are not required, or thought of as “nice-to-haves”

For example:

- Gold-plating
- Technology features

4. Task Switching

Multi-tasking between several different projects when there are context-switching penalties

For example:

- People assigned to multiple projects

5. Waiting

Delays waiting for reviews and approvals

For example:

- Waiting for prototype reviews
- Waiting for document approvals

6. Motion

The effort required to **communicate or move information or deliverables from one group to another**. if teams are not co-located, this effort may need to be greater.

For example:

- Distributed teams
- Handoffs

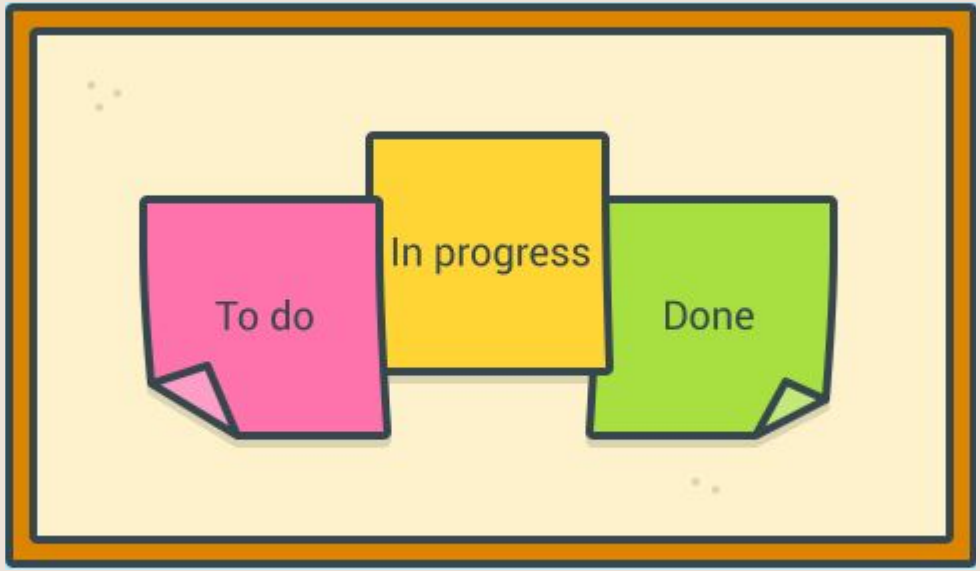
7. Defects

Defective documents or software that needs correction.

For example:

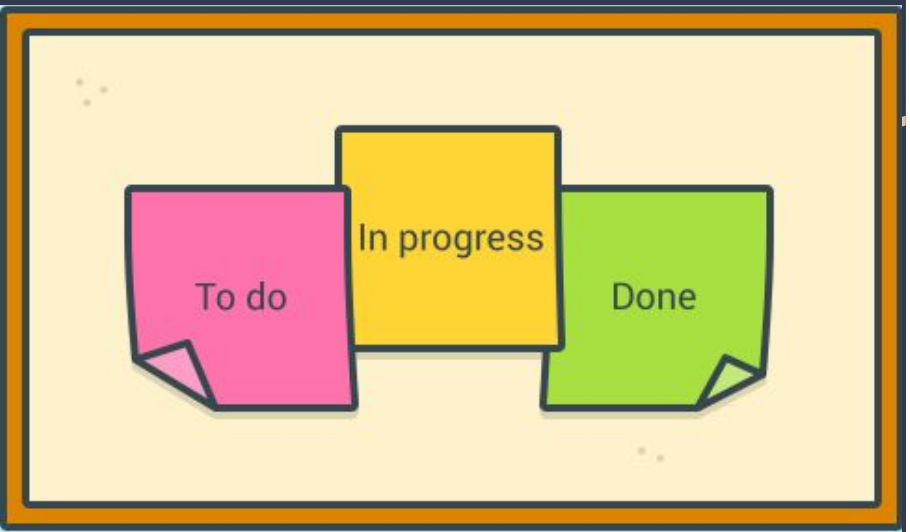
- Requirements defects
- Software bugs

Kanban



Kanban

The Kanban method is derived from the lean production system developed at Toyota.

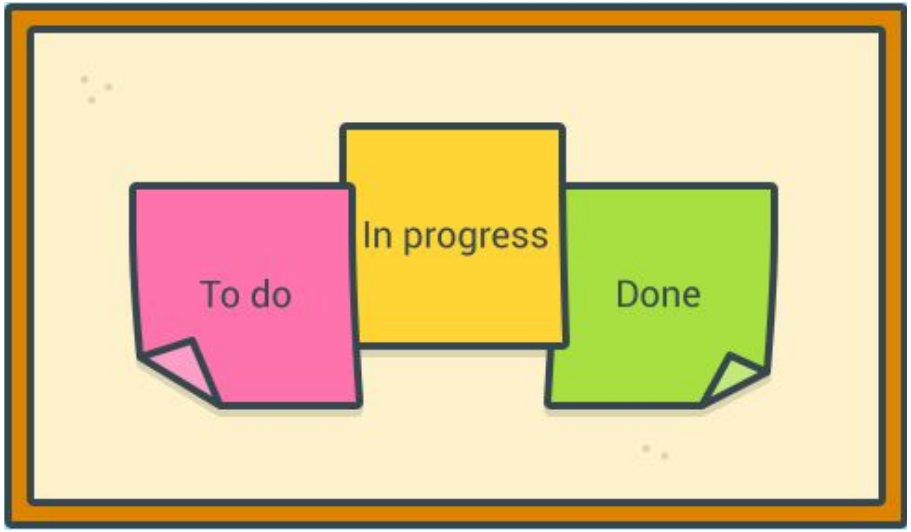


"Kanban" is a Japanese word meaning "**signboard**."

This board **shows the work items in each stage of the production process**, as defined by the team.

5 Principles of Kanban

1. Visualize the Workflow
2. Limit WIP
3. Manage Flow
4. Make Process Policies Explicit
5. Improve Collaboratively



1. Visualize Your Workflow

Knowledge work projects, by definition, **manipulate knowledge, which is intangible and invisible**.

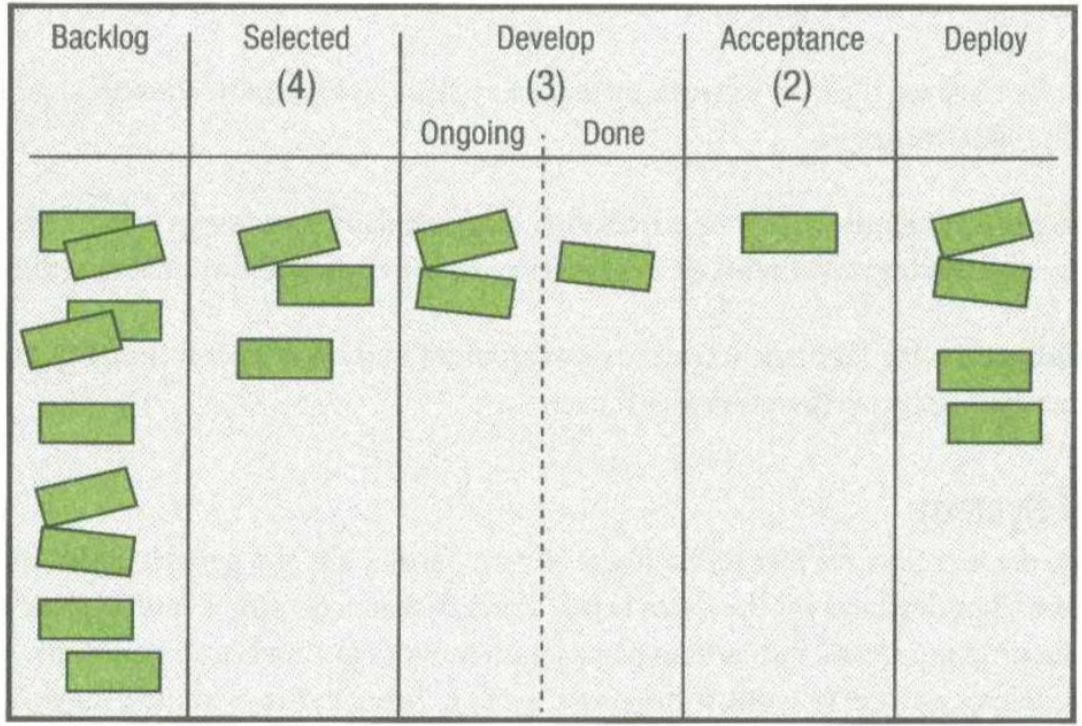
Therefore, having some way to visualize the workflow is very important for organizing, optimizing, and tracking it.

2. Limit WIP

Restricting the amount of work in progress **improves productivity, increases the visibility of issues and bottlenecks, and facilitates continuous improvement**.

Easier for the team to identify issues and minimize the waste and cost associated with changes.

It also results in a steady “pull” of work through the development effort, since new work can only be moved forward as existing work is completed.

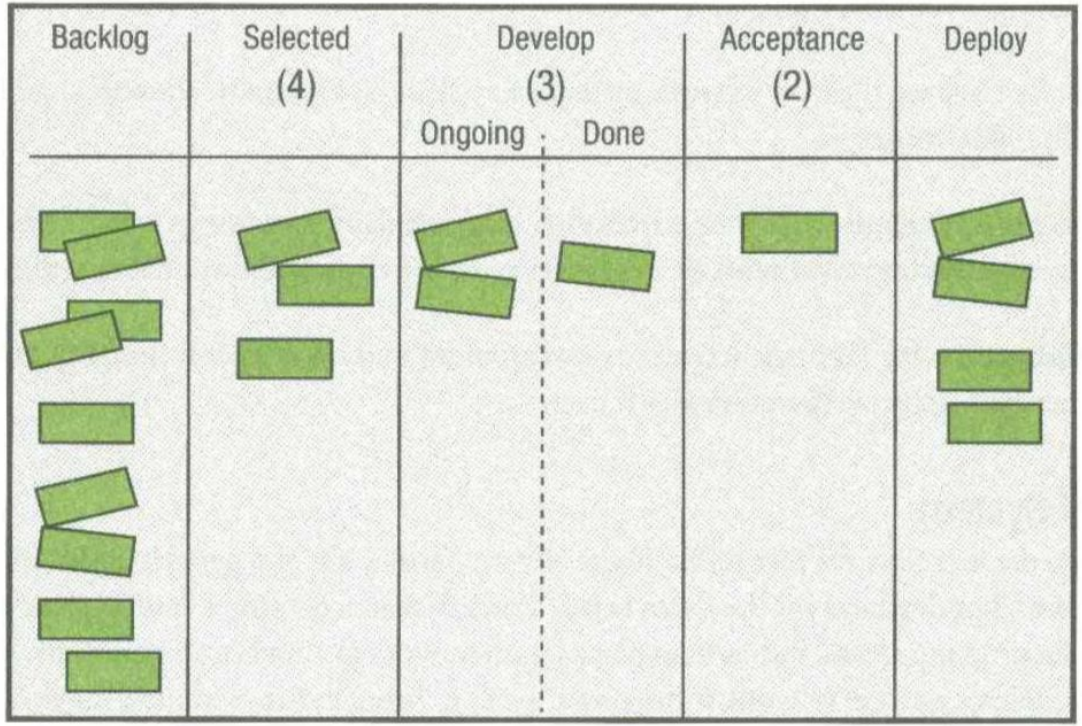


2. Limit WIP

Once the **limit at the top of a column** is reached, no new items may be moved into that column until another item is moved out.

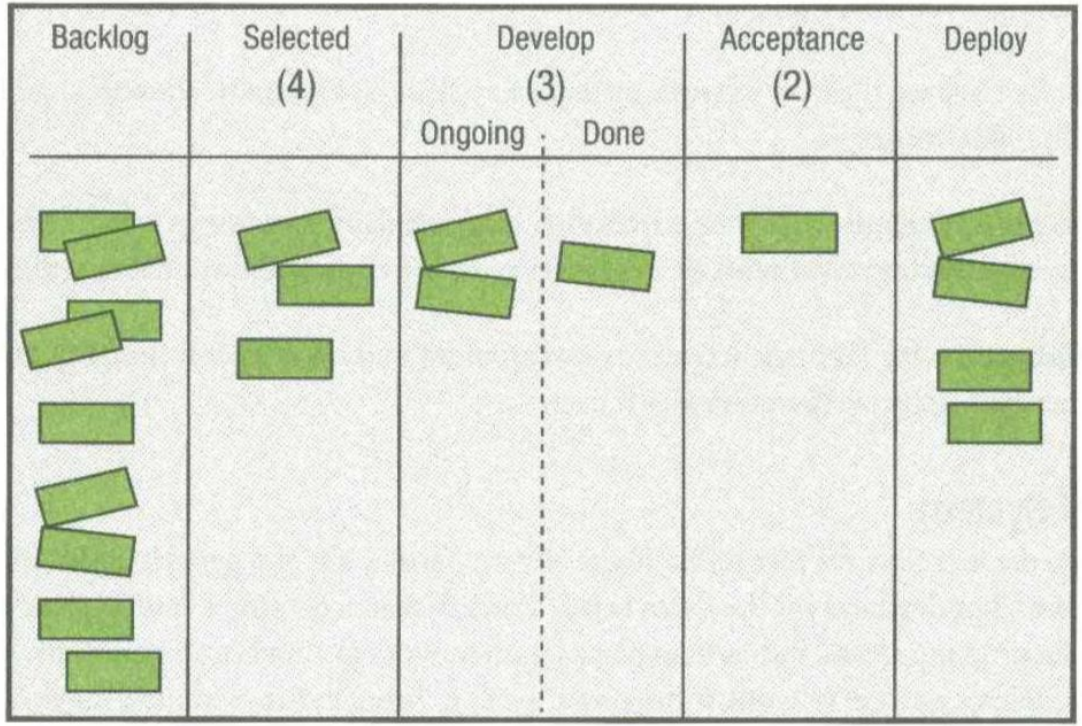
Lowering WIP actually increases a team's productivity.

Little's Law demonstrates that the duration of a queue is proportional to its size.



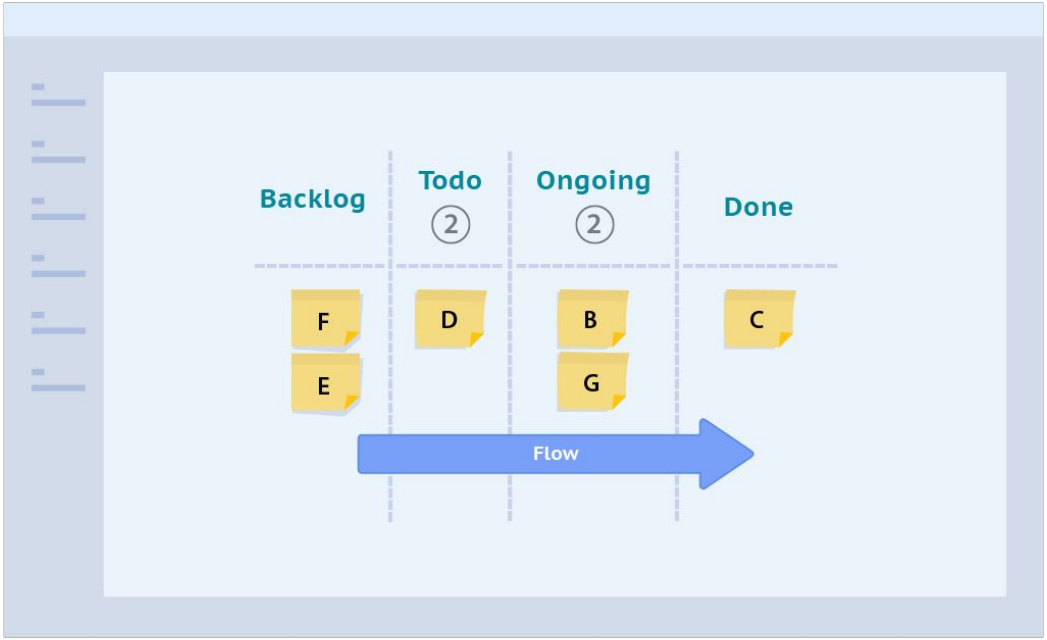
2. Limit WIP

With a continuous pull model, iterations may not be required, and therefore activities like creating estimates can be considered waste and reduced or eliminated entirely.



3. Manage Flow

By tracking the flow of work through a system, **issues can be identified and changes can be measured for effectiveness**.



4. Make Process Policies Explicit

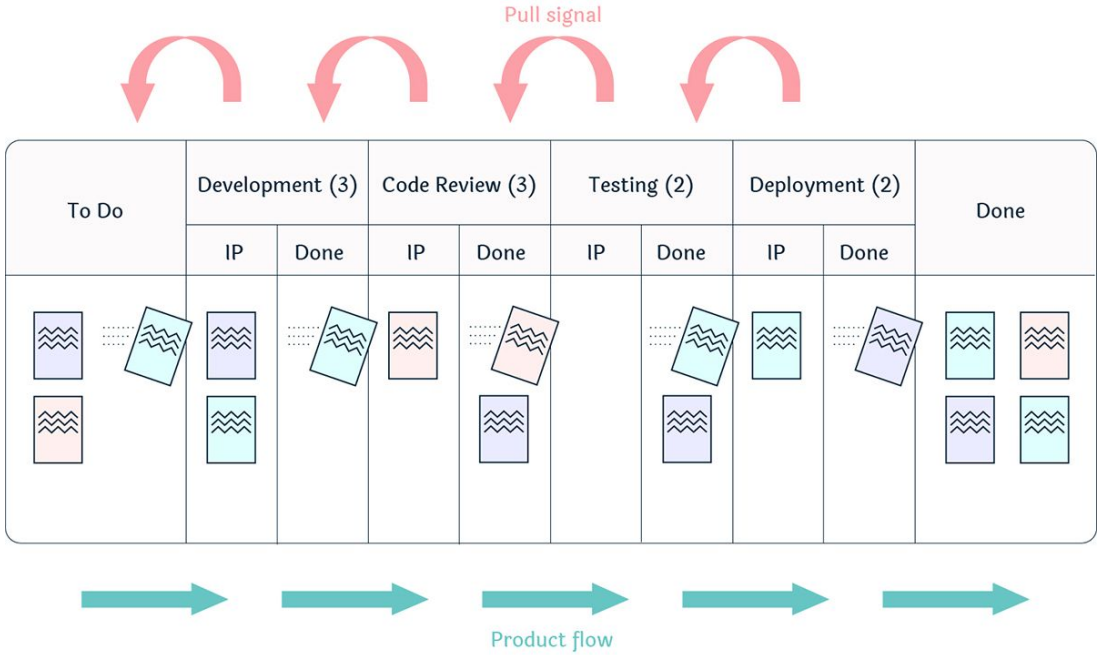
It is important to **clearly explain how things work** so the team can have open discussions about improvements in an objective, rather than an emotional or subjective, way.

5. Improve Collaboratively

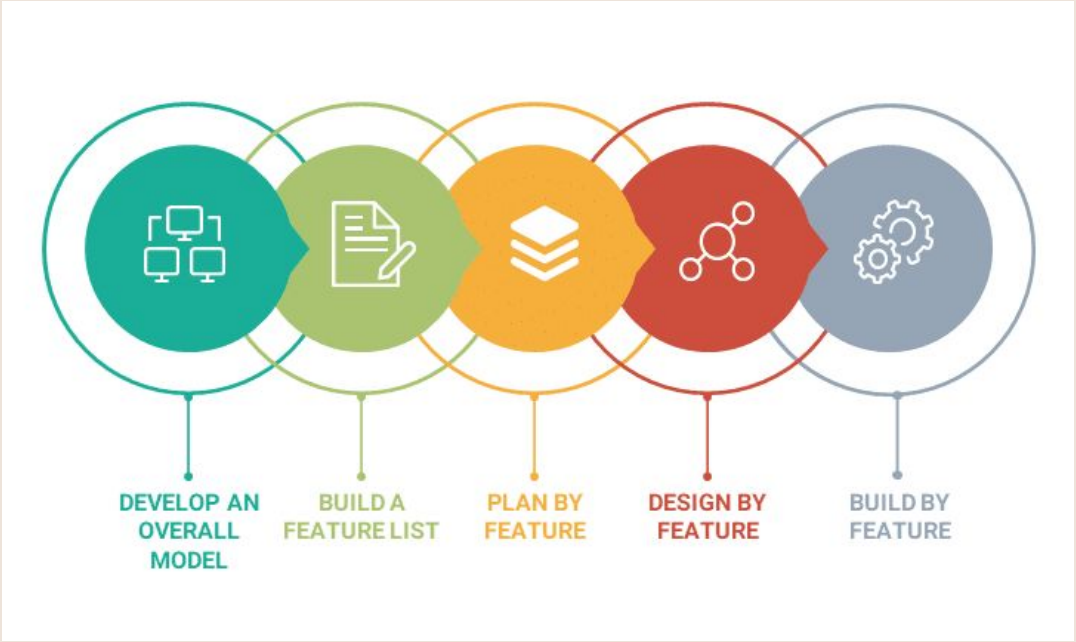
Through scientific measurement and experimentation, the team should **collectively own and improve the processes it uses**.

Kanban Pull System

Kanban teams employ a “pull system” to move work through the development process, rather than planning their work in timeboxed iterations. Each time a Kanban **team completes an item of work, it triggers a “pull” to bring in the next item** they will work on.



Feature Driven Development (FDD)



Feature Driven Development (FDD)

1. First develop an overall model for the product, build a feature list, and plan the work.

2. Then move through design and build iterations to develop the features.

8 Best Practices

1. Domain object modeling
2. Developing by Feature
3. Individual Class Ownership
4. Feature Teams
5. Inspections
6. Configuration Management
7. Regular Builds
8. Visibility of Progress & Results

1. Domain Object Modeling

Teams explore and explain the domain (or business environment) of the problem to be solved.

2. Developing by Feature

Breaking functions down into two-week or shorter chunks of work and calling them features.

3. Individual Class (Code) Ownership

Areas of code have a single owner for consistency, performance, and conceptual integrity.

4. Feature Teams

These are small, dynamically formed teams that vet designs and allow multiple design options to be evaluated before a design is chosen.

Feature teams help mitigate the risks associated with individual ownership.

5. Inspections

These are reviews that help ensure good-quality design and code.

6. Configuration Management

This involves labeling code, tracking changes, and managing the source code.

7. Regular Builds

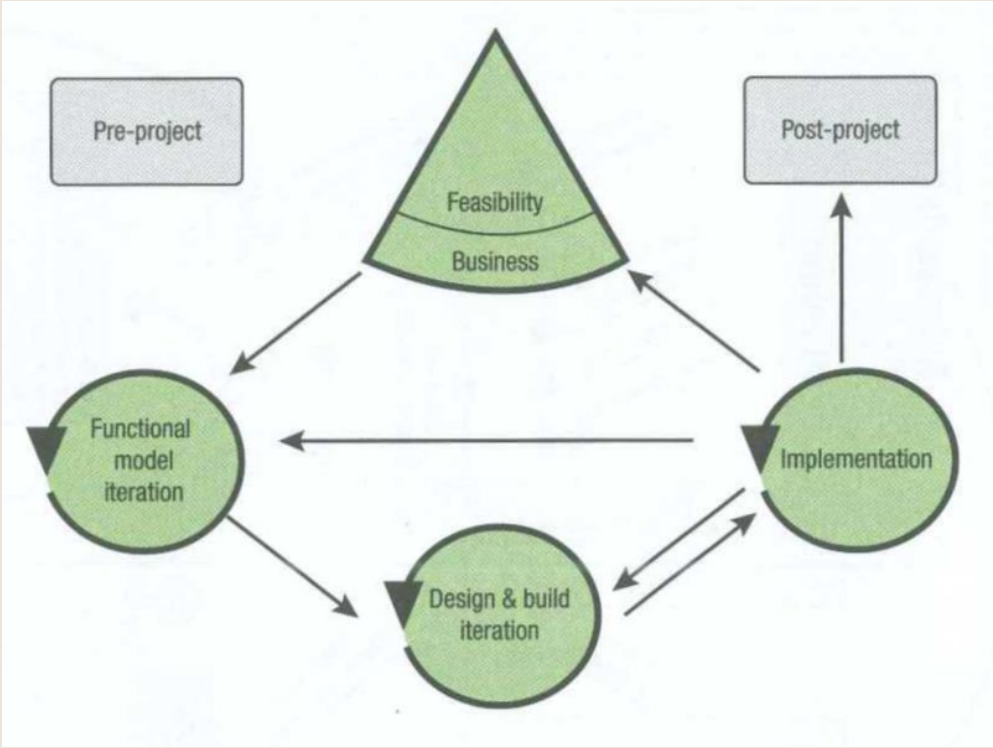
Through regular builds, the team makes sure the new code integrates with the existing code.

This practice also allows them to easily create a demo.

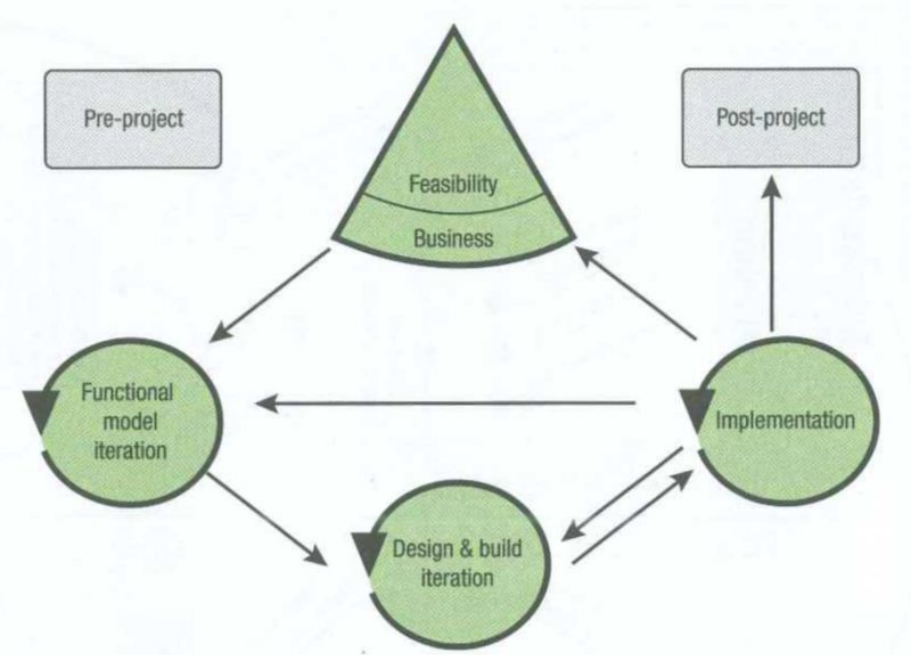
8. Visibility of Progress

This practice tracks progress based on completed work.

Dynamic Systems Development Method (DSDM)

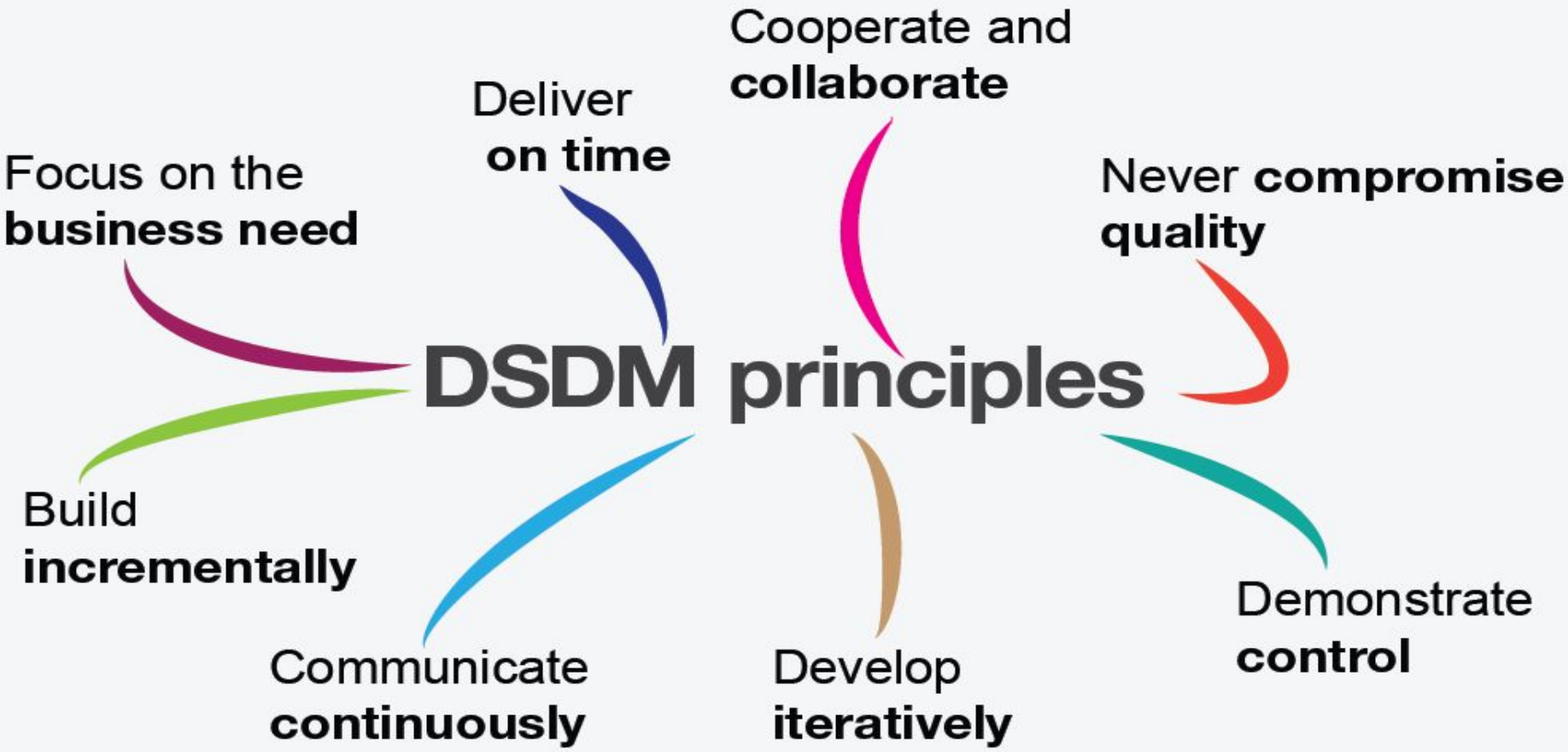


Dynamic Systems Development Method (DSDM)



DSDM was one of the earlier agile methods, and it started out quite prescriptive and detailed.

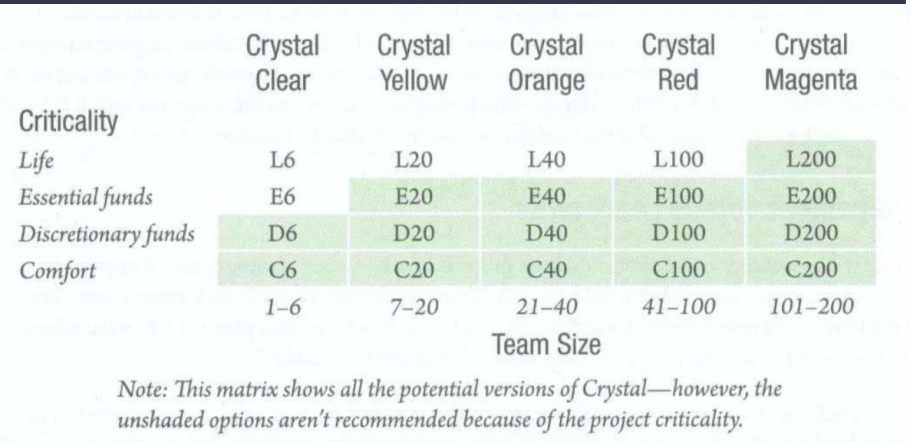
Its coverage of the project life cycle is broad, **encompassing aspects of an agile project ranging from feasibility and the business case to implementation**.



8 Principles of DSDM

Crystal

Crystal



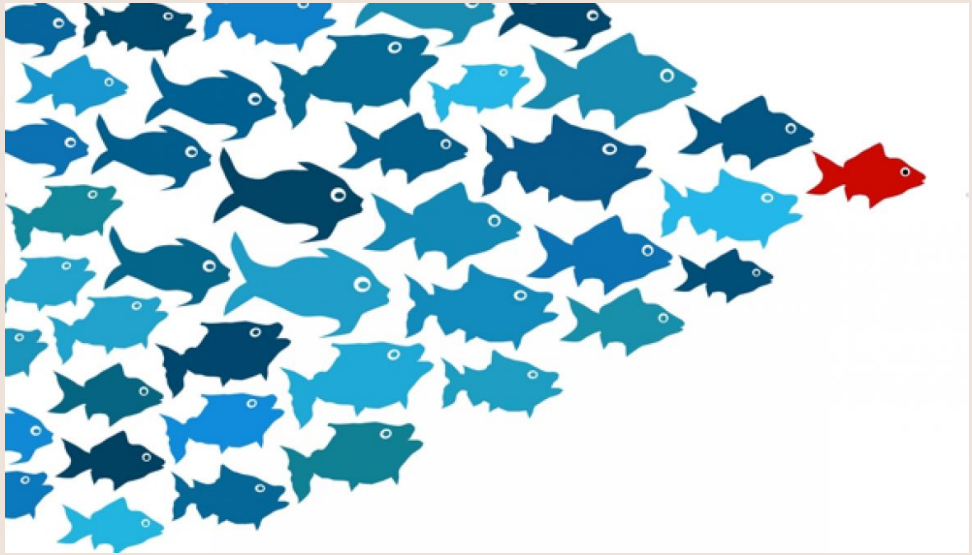
Criticality	Crystal Clear	Crystal Yellow	Crystal Orange	Crystal Red	Crystal Magenta
Life	L6	L20	L40	L100	L200
Essential funds	E6	E20	E40	E100	E200
Discretionary funds	D6	D20	D40	D100	D200
Comfort	C6	C20	C40	C100	C200
	1–6	*7–20*	*21–40*	*41–100*	*101–200*

Team Size

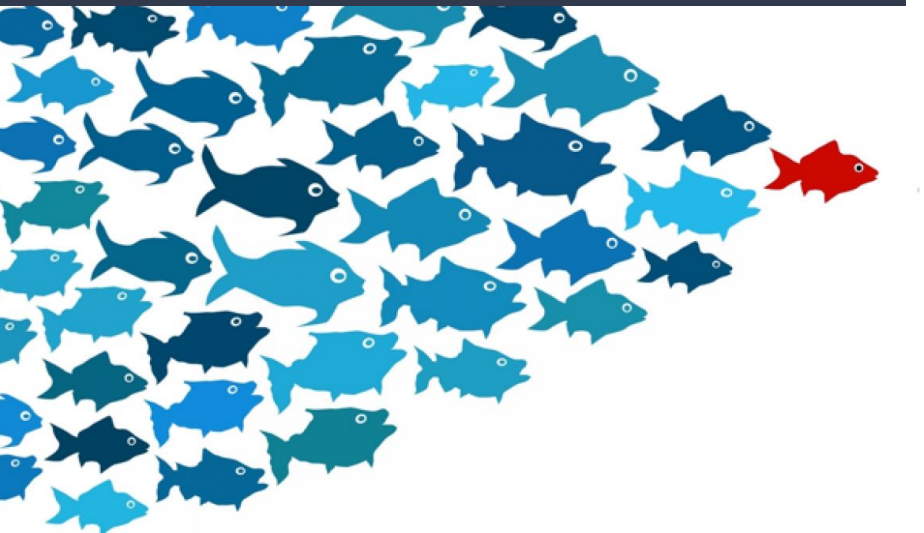
Note: This matrix shows all the potential versions of Crystal—however, the unshaded options aren't recommended because of the project criticality.

Crystal isn't just one method. It consists of a **family of situationally specific, customized methodologies that are coded by color names**. Each methodology is customized by criticality and team size, which allows Crystal to cover a wide range of projects, from a small team building a low-criticality system (Crystal Clear) to a large team building a high-criticality system (Crystal Magenta).

Agile Leadership



Agile Leadership



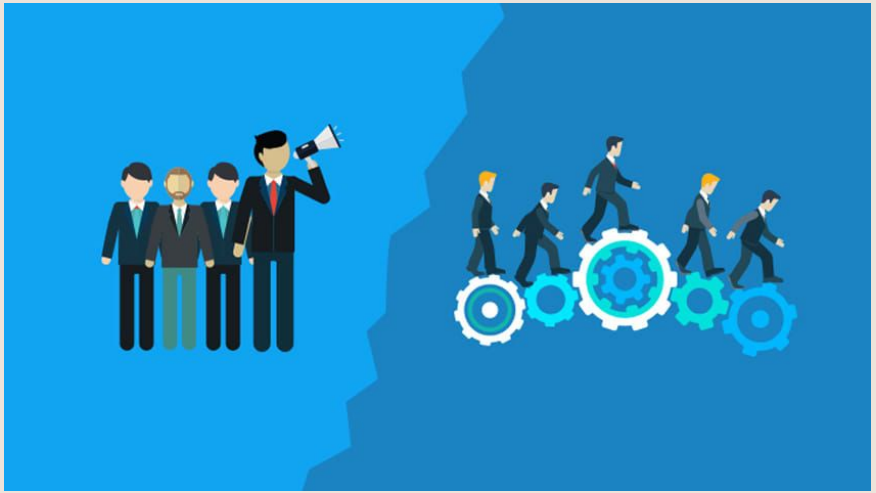
Leadership is about tapping into people's intrinsic motivations.

To be effective leaders, we need to discover why team:

1. Team members want to do things
2. Understand what motivates them
3. Align their project tasks and goals accordingly.

It is by aligning project objectives with personal objectives that we can get higher levels of productivity.

Management *vs.* Leadership



“**Management** is getting people to do what needs to be done.

Leadership is getting people to want to do what needs to be done.”

- *Warren Bennis*

Management Focus	Leadership Focus
Tasks/things	People
Control	Empowerment
Efficiency	Effectiveness
Doing things right	Doing the right things
Speed	Direction
Practices	Principles
Command	Communication

Difference of Focus Between Management and Leadership

Leadership or Management?
Human Resources Management
Career Planning
Team Time Tracking
Team Member Recognition
Task Assignment
Team Brainstorming
Planning Workshops
Creating Gantt Charts

Servant Leadership

Servant Leadership

It is the **team members**, not the leader, coach, or ScrumMaster, who **get the technical work done and achieve the business value**.

The servant leadership approach redefines the leader's role in relation to the team. It focuses the **leader** on **providing what the team members need**, **removing impediments to their progress**, and **performing supporting tasks** to maximize their productivity.

4 Primary Duties of Leaders

The primary duties of leaders are:

1. Shield the team from interruptions
2. Remove impediments to progress
3. Communicate (and re-communicate) the project vision
4. Carry food and water

1. Shield the Team from Interruptions

Servant leaders need to **isolate and protect the team members** from diversions, interruptions, and requests for work that aren't part of the project.

Protect from both internal and external matters.

2. Remove Impediments to Progress

Servant leaders need to **clear obstacles from the team's path that would cause delay or waste**.

Some agile project management tools now support **impediment backlogs**, a kind of prioritized obstacle removal list.

3. Communicate the Project Vision

Only if stakeholders have a **clear image of the goals** for the completed product and project can they align their decisions with, and work toward, the **common project objective**.

A common vision helps to keep people all pulling in the same direction.

4. Carry Food & Water

Providing the essential resources a team needs to keep them nourished and productive. This include tools, compensation, and encouragement.

Leaders also need to celebrate victories—the large ones, of course, but also the small ones—as the project progresses.

4. Carry Food & Water

Training and other professional development activities are also examples of resources the team may need to be productive.

Principles for Leading Agile Projects

1. Learn the Team Members Needs

Find out what motivates and demotivates people.

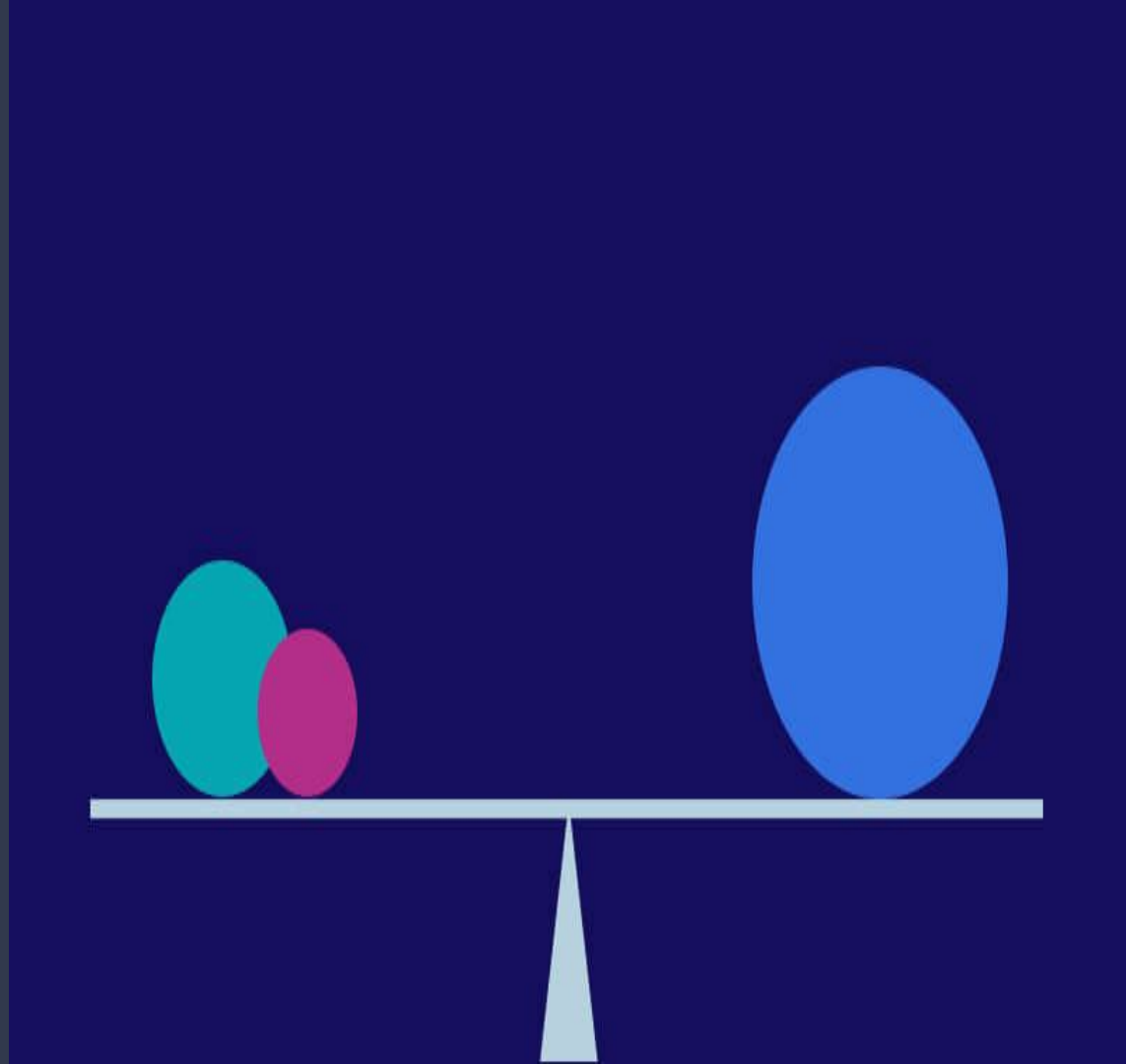
2. Learn the Project's Requirements

Talk to the customers and sponsors.

Find out their priorities.

3. Act for the Simultaneous Welfare of the Team & Project

Balance and promote the needs and desires of both the team and the other project stakeholders.



4. Create an Environment of Functional Accountability

Make sure people know what success and failure look like, and empower the team to self-organize to reach the goal.

Be proud of accomplishments, but don’t hide or shy away from failures—instead, examine them, learn from them, and adapt.

5. Have a Vision of the Completed Project

Create a “**beckoning summit**” to which others can chart their own course. When we are head-down in the weeds, it’s good to know where we are trying to get to, so we can navigate our own course.

6. Use the Project Vision to Drive Your Own Behavior

Model action toward the project goals.

7. Serve as the Central Figure in a Successful Project Team Development

Model desired behavior for the team.

8. Recognize Team Conflict as a Positive Step

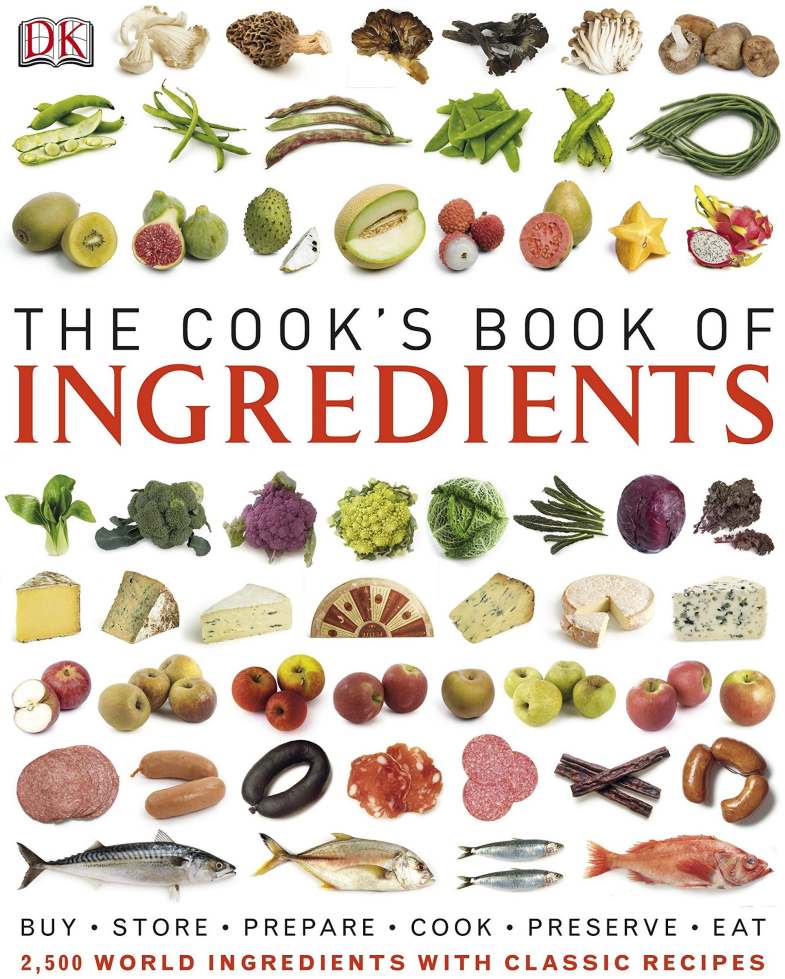
Unfiltered debate builds strong buy-in for well- discussed topics.

9. Manage with an Eye Towards Ethics

Be honest and ethical, because people don’t want to be associated with goals or missions they feel are unethical.

10. Ethics is not an Afterthought, but an Integral Part of Our Thinking

You cannot add trust in later—like quality, it has to be a core ingredient.



11. Take Time to Reflect on the Project

Review, diagnose, and adapt; improve through progressive change and learning.

12. Develop the Trick of Thinking Backwards

Imagine that we have reached the end goal and then **work backwards to determine what had to happen to get there** and what problems and risks we were able to avoid.

So first discuss and decide what “done” will look like; then chart the path to get there, and plan how you will avoid any obstacles in your way.

Agile Leadership Practices

What values, personal traits, or characteristics do you look for or admire in your leader?

1. Honesty
2. Forward-looking
3. Competent
4. Inspiring

Leadership Tasks

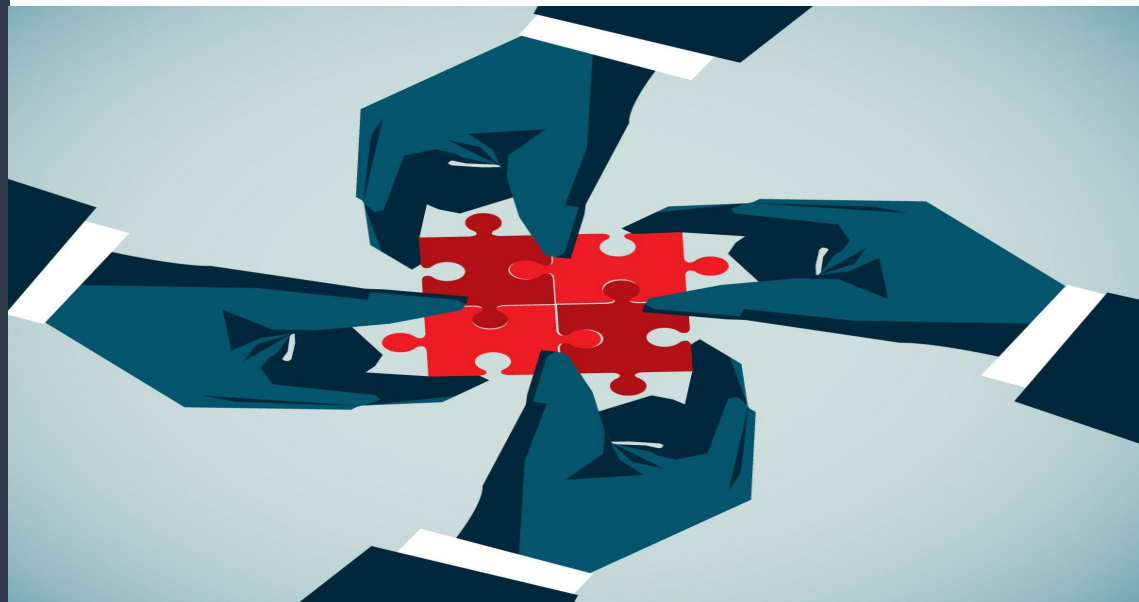
1. Experiment with New Techniques & Processes

The only real way to find out if a new process or technique is going to work for your project is to try it out.

2. Share Knowledge Through Collaboration

It’s important for teams to share knowledge about the product they are working on and how things work within their team. However, writing all this down takes a long time, and people often don’t like to read it anyway.

By working with people we learn what they do, how they solve problems, and how to help them most effectively when they are stuck. By simply changing who people work with, project knowledge can be dispersed throughout the team, reducing the impact of information loss if a team member leaves.



3. Encourage Emergent Leadership via a Safe Environment

When a team member takes the initiative and tries a new approach after gaining team approval.

If someone identifies a more efficient process, they are encouraged to try an experiment to prove if it can work.

Domain 2. Value Driven Delivery

Key Topics

- Agile contracting
- Agile project accounting principles
- Agile risk management
- Agile tooling
- Compliance/regulatory compliance
- Cumulative flow diagrams (CFDs)
- Customer-valued prioritization
- Earned value management for agile projects

Domain 2. Value Driven Delivery

Key Topics (continued)

- Frequent verification & validation
- Incremental delivery
- Managing with agile KPIs
- Minimal viable product (MVP)
  - Minimal marketable feature (MMF)
- Prioritization schemes
  - Kano analysis
  - MoSCoW
- Relative prioritization/ranking

Domain 2. Value Driven Delivery

Key Topics (continued)

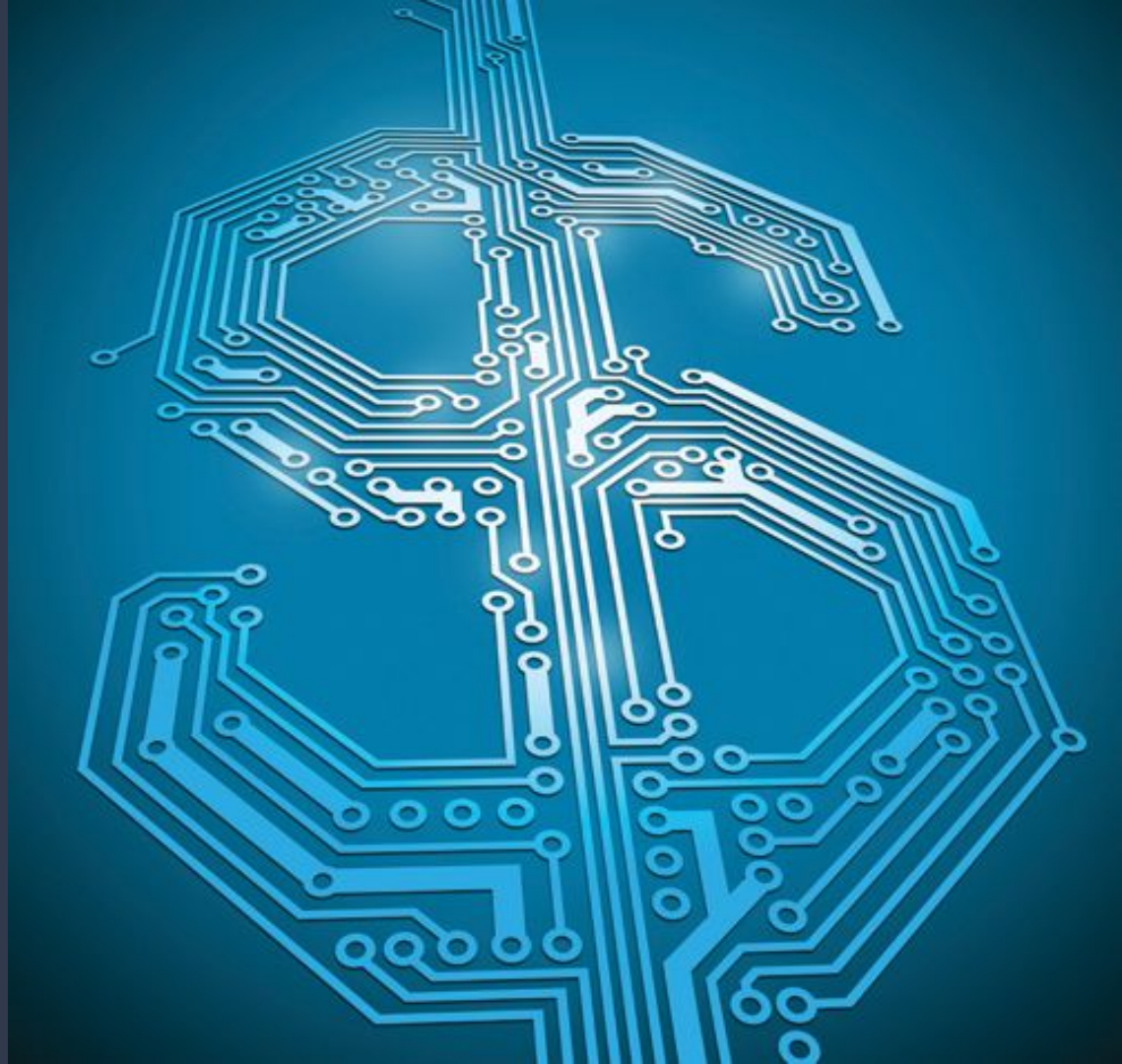
- ROI / NPV / IRR
- Software development practices
  - Continuous integration
  - Exploratory and usability testing
  - Red, green, refactor
  - TDD / TFD / ATDD
- Task/Kanban boards
- Value-driven delivery
- Work in Progress (WIP) - Limits

What is Value-Driven Delivery?

Value-Driven Delivery

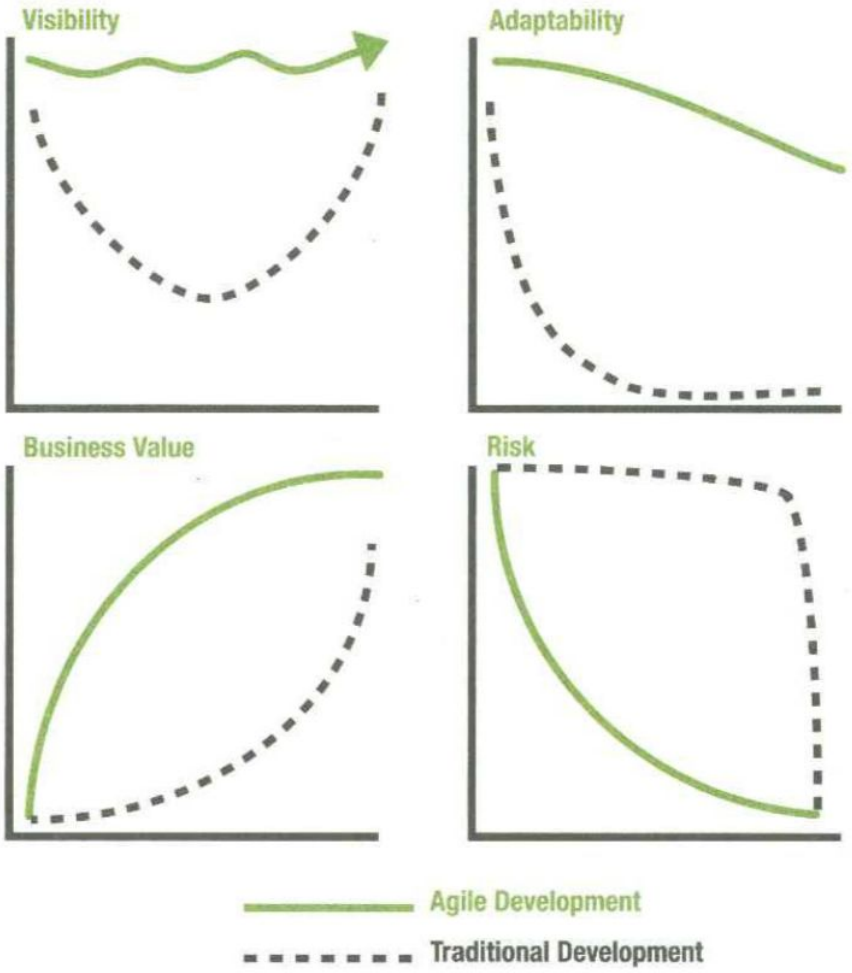
The reason projects are undertaken is to **generate business value**, be it to produce a benefit or to improve a service.

If delivering value is the reason for doing projects, then value-driven delivery must be our focus.



The Agile Value Proposition

Agile methods are able to deliver more value in comparison to non-agile methods.



Deliver Value Early

Deliver Value Early

Agile teams aim to deliver the highest-value portions of the project as soon as possible.

The longer a project runs, the longer the horizon becomes for risks that can reduce value such as failure, decreased benefits, erosion of opportunities, etc.

Deliver Value Early

Stakeholder satisfaction plays a huge role in project success.

By delivering high-value elements early, the team demonstrates an understanding of the stakeholders needs, shows that they recognize the important aspects, and proves they can deliver

Minimize Waste

Value-driven delivery means making decisions that prioritize the value-adding activities and risk-reducing efforts for the project, and then executing based on these priorities.

Minimize Waste

Agile has adopted the **lean concept of minimizing waste** and other non-value-adding activities.

List of **7 wastes**:

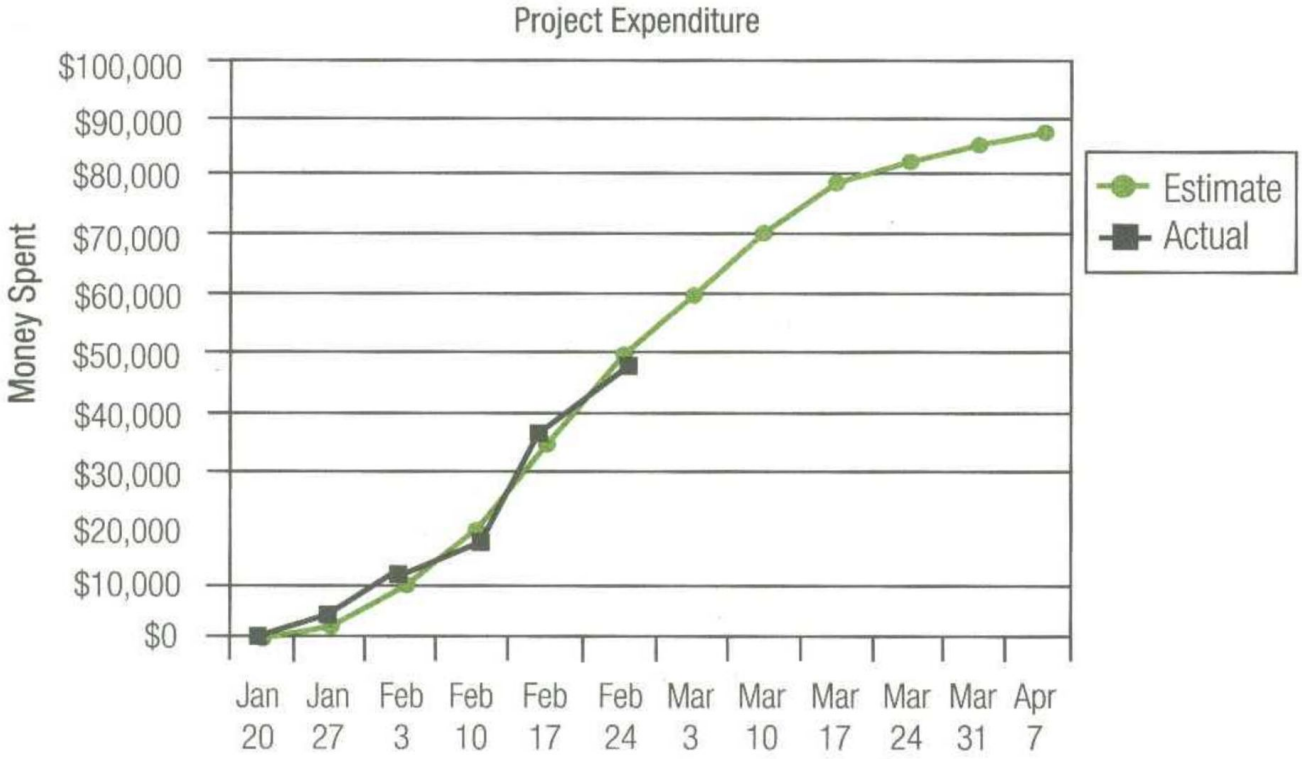
1. Partially done work
2. Extra processes
3. Extra features
4. Task switching
5. Waiting
6. Motion
7. Defects

Assessing Value

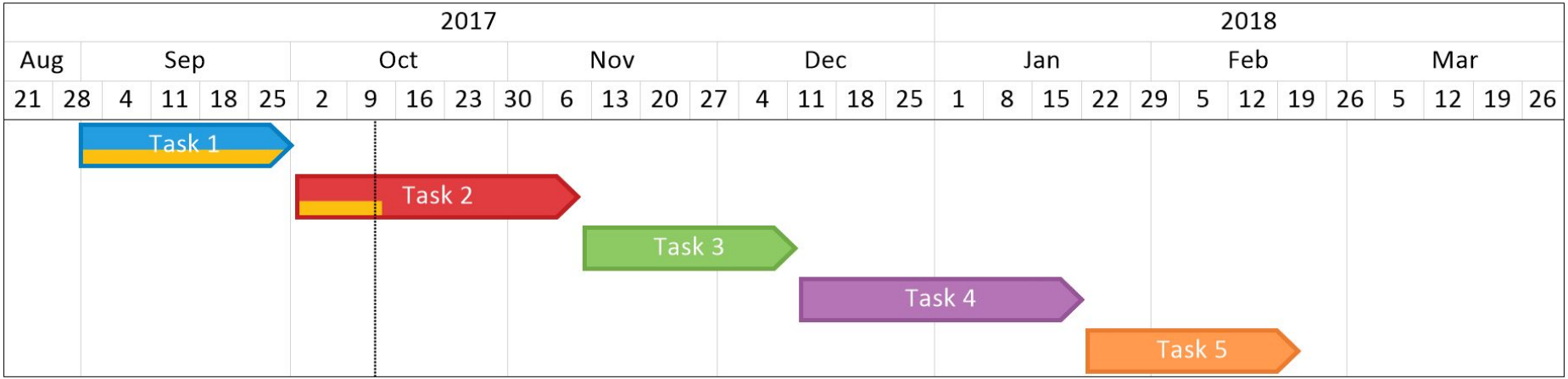
Assessing Value

Earned Value Management

Figure 2.5: S-Curve Graph

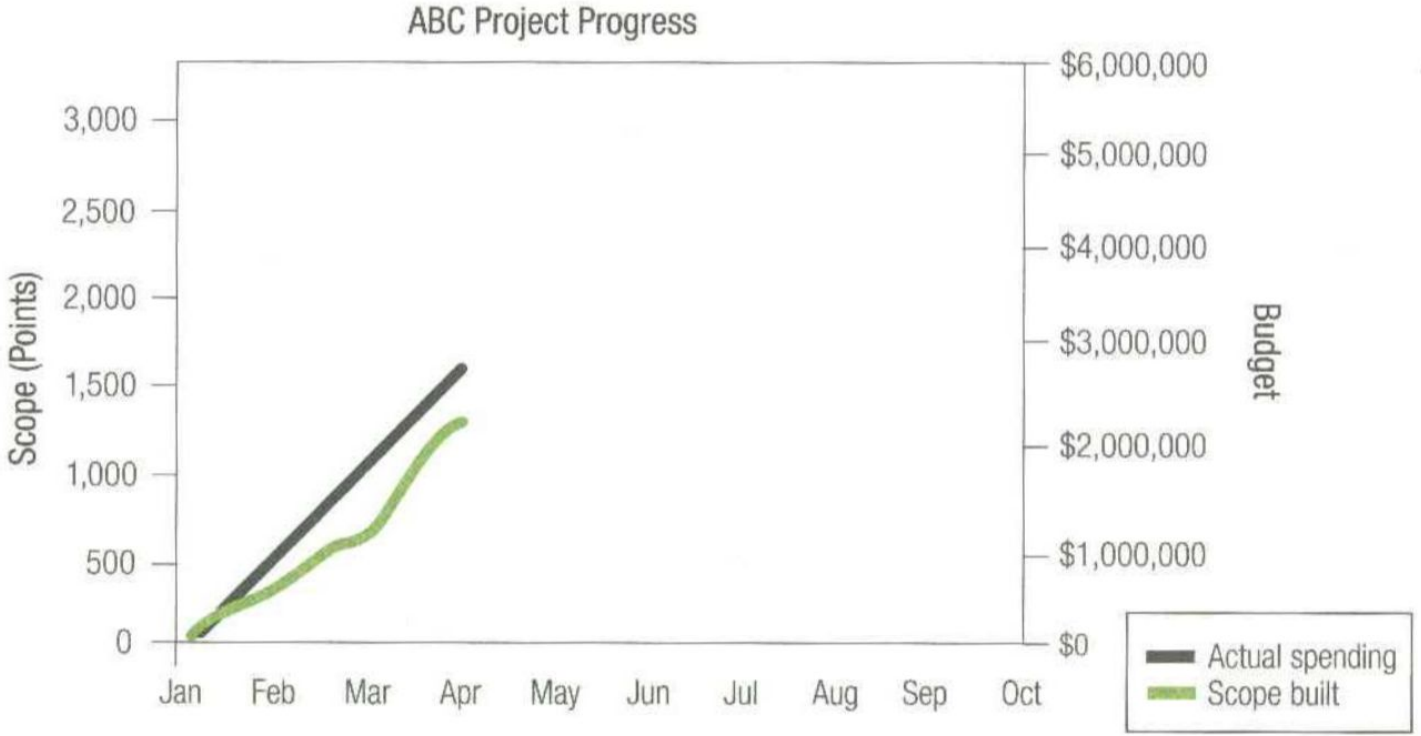


S Curve


2017
2018
Aug
Sep
Oct
Nov
Dec
Jan
Feb
Mar
21 28 4 11 18 25 2 9 16 23 30 6 13 20 27 4 11 18 25 1 8 15 22 29 5 12 19 26 5 12 19 26
Task 1
Task 2
Task 3
Task 4
Task 5

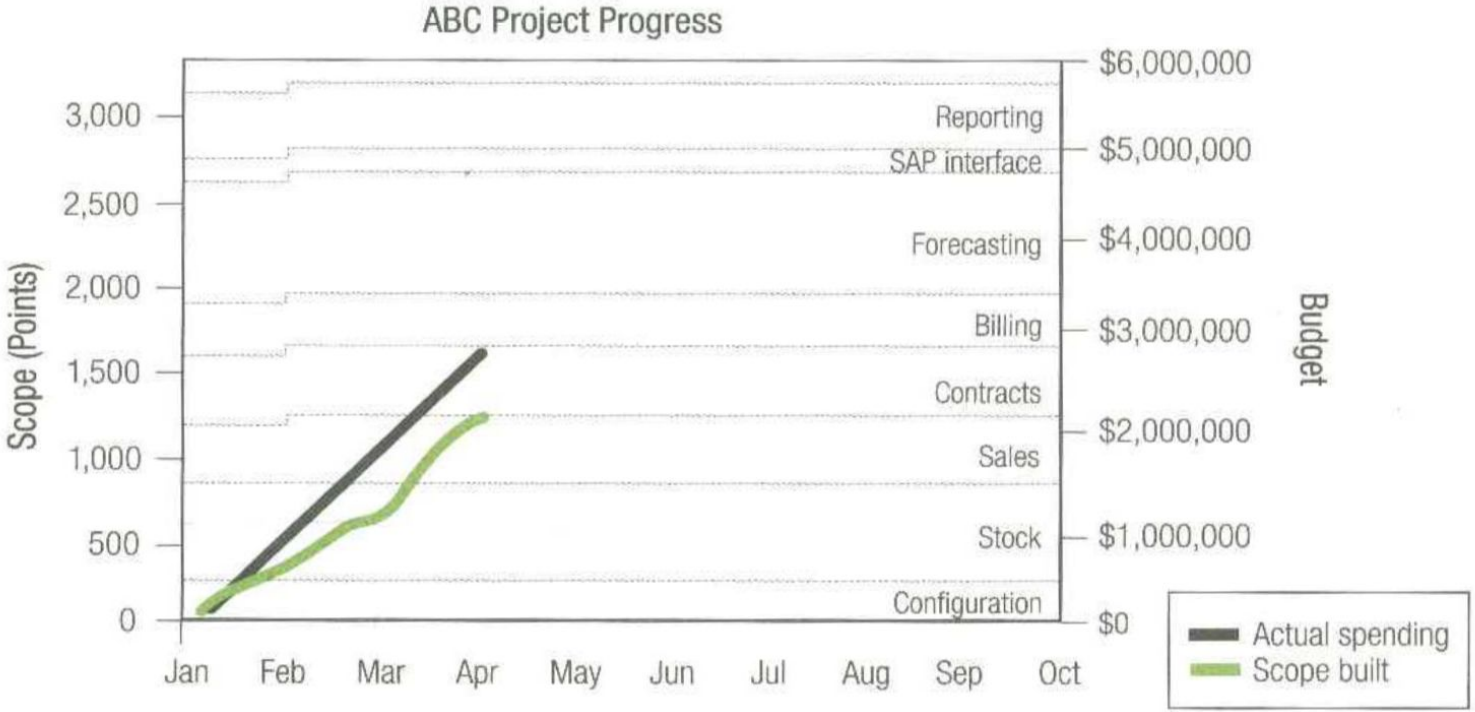
Gantt Chart

Figure 2.7: Scope, Cost, and Schedule Performance (Double S-Curve)

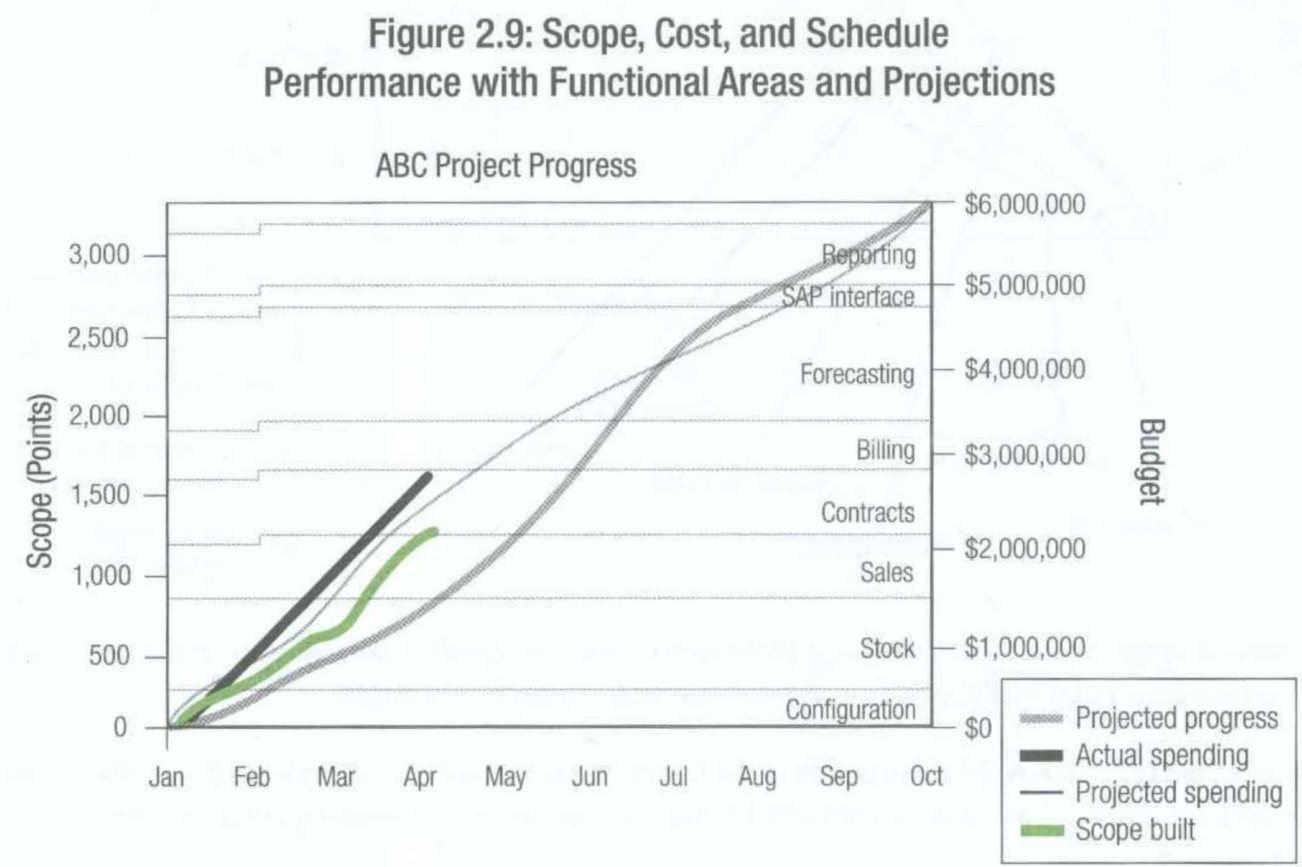


Double S-Curve

Figure 2.8: Scope, Cost, and Schedule Performance with Functional Areas

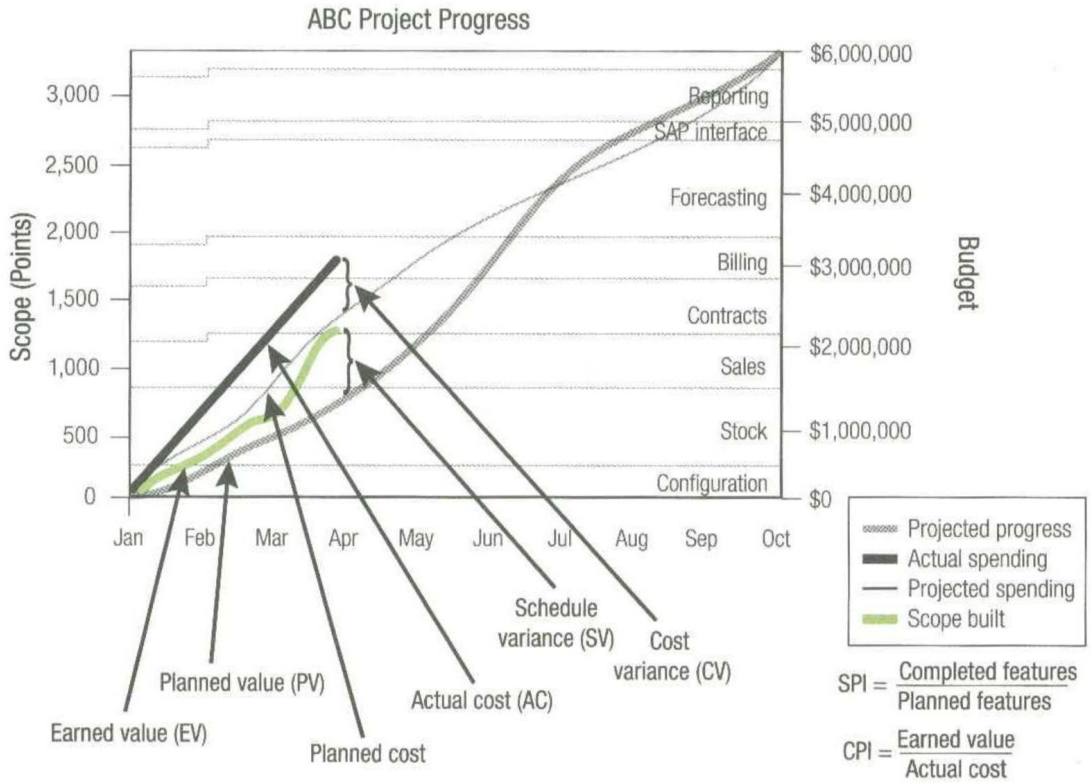


Double S-Curve with Functional Areas



Double S-Curve with Functional Areas & Projections

Figure 2.10: A Visual Tool for EVM Metrics



Visual Tool for EVM Metrics

3. Stakeholder Engagement

4. Team Performance

5. Adaptive Planning

6. Problem Detection & Resolution

7. Continuous Improvement