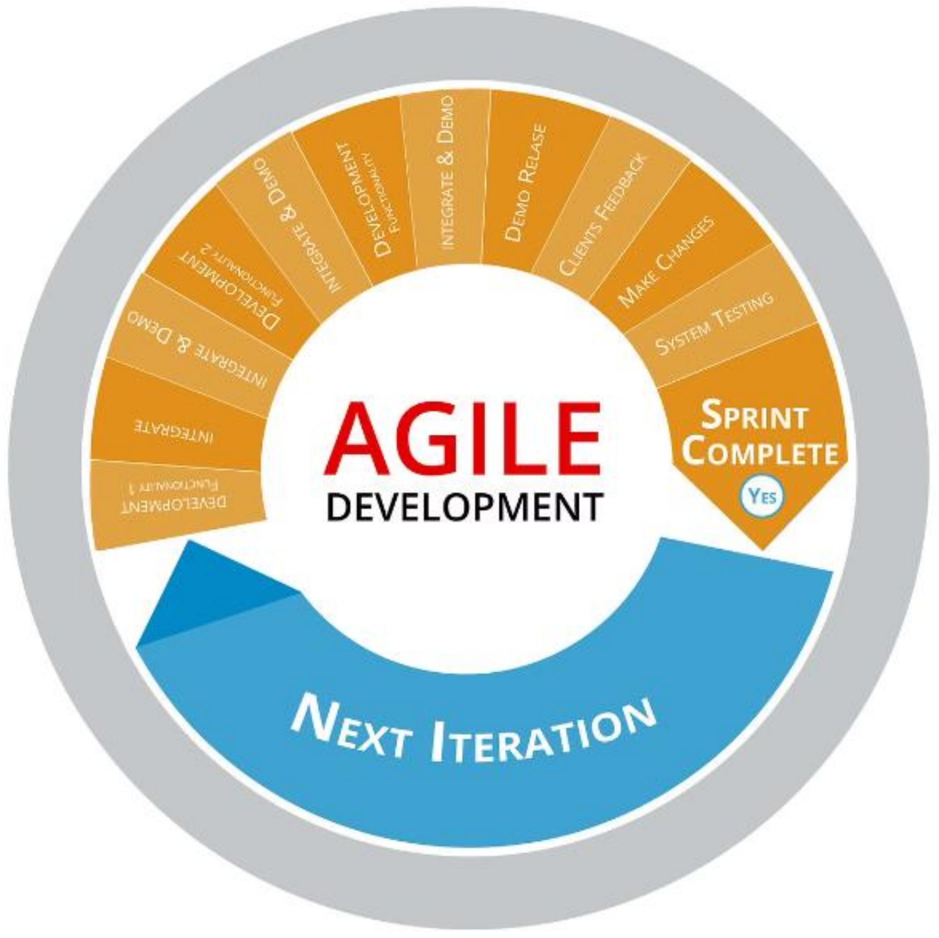
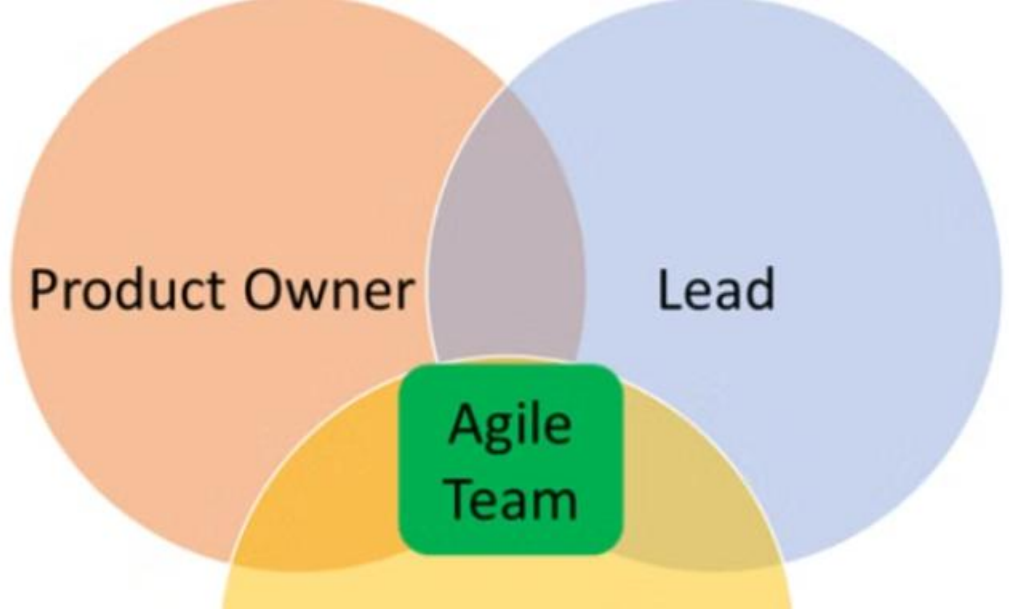
Agile Project Management



Explore the principles and practices of agile project management, a flexible and iterative approach to delivering projects effectively.

Module – 3

Software Development Methodologies



Agile Fundamentals

1 **Iterative Approach**

Agile emphasizes breaking down projects into smaller, manageable iterations for faster delivery and adaptation to changing requirements.

2 **Collaboration**

Agile encourages close collaboration between cross-functional teams to ensure alignment and continuous feedback.

3 **Flexibility**

Agile allows for flexibility to respond to evolving needs, rather than following a rigid plan.

Agile Project Lifecycle

1. **Plan**

   Define the project scope, objectives, and timeline.

2. **Design**

   Collaborate on the product design and user experience.

3. **Develop**

   Implement the project in an iterative, incremental manner.

4. **Test**

   Continuously test and validate the product throughout development.

5. **Deploy**

   Release the final product and gather feedback for future iterations.

Agile Roles and Responsibilities

Product Owner

Defines the product vision and prioritizes the backlog.

Scrum Master

Facilitates the agile process and removes obstacles.

Development Team

Collaborates to design, develop, and test the product.

Agile Ceremonies

Sprint Planning

Defines the goals and tasks for the upcoming sprint.

Daily Standup

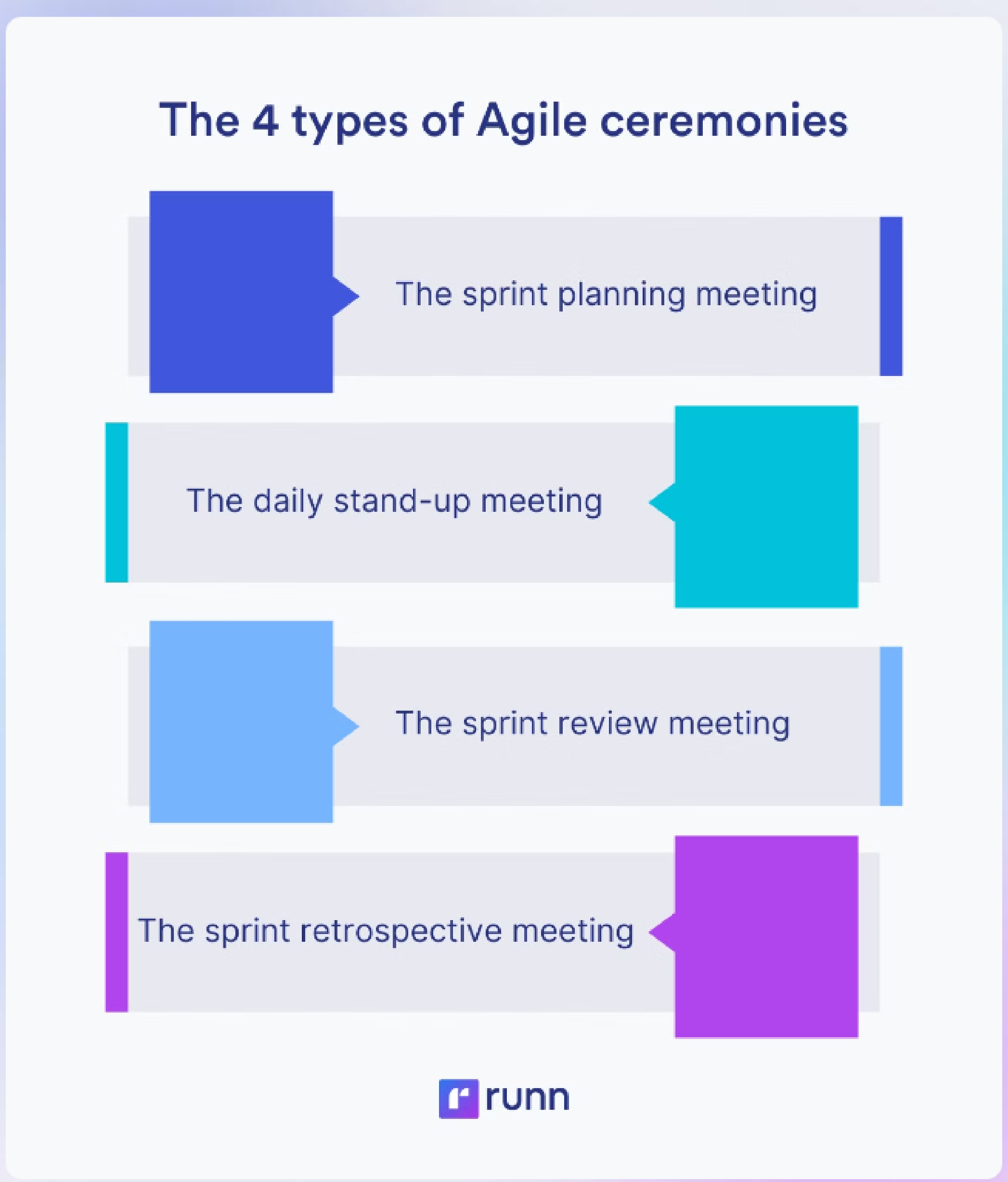
Brief daily check-in to discuss progress and blockers.

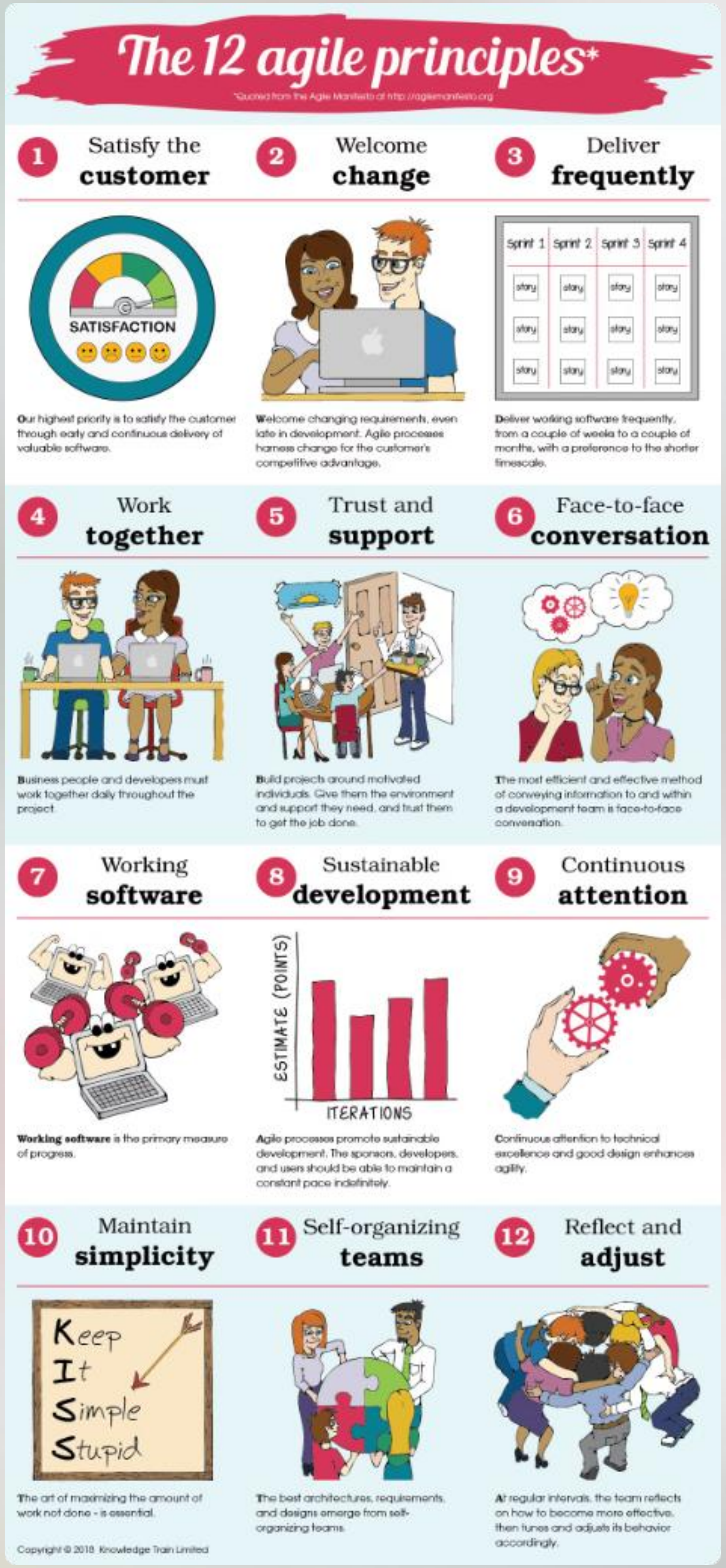
Sprint Review

Demonstrates the completed work and gathers feedback.

Retrospective

Reflects on the sprint and identifies areas for improvement.


The 4 types of Agile ceremonies
The sprint planning meeting
The daily stand-up meeting
The sprint review meeting
The sprint retrospective meeting
runn



The 12 agile principles*

*Quoted from the Agile Manifesto at http://agilemanifesto.org

1 Satisfy the **customer**

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

2 Welcome **change**

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

3 Deliver **frequently**

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

4 Work **together**

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

5 Trust and **support**

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

6 Face-to-face **conversation**

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

7 Working **software**

Working software is the primary measure of progress.

8 Sustainable **development**

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

9 Continuous **attention**

Continuous attention to technical excellence and good design enhances agility.

10 Maintain **simplicity**

The art of maximizing the amount of work not done - is essential.

11 Self-organizing **teams**

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

12 Reflect and **adjust**

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

Agile Principles

Customer Focus

Prioritize customer needs and feedback.

Embrace Change

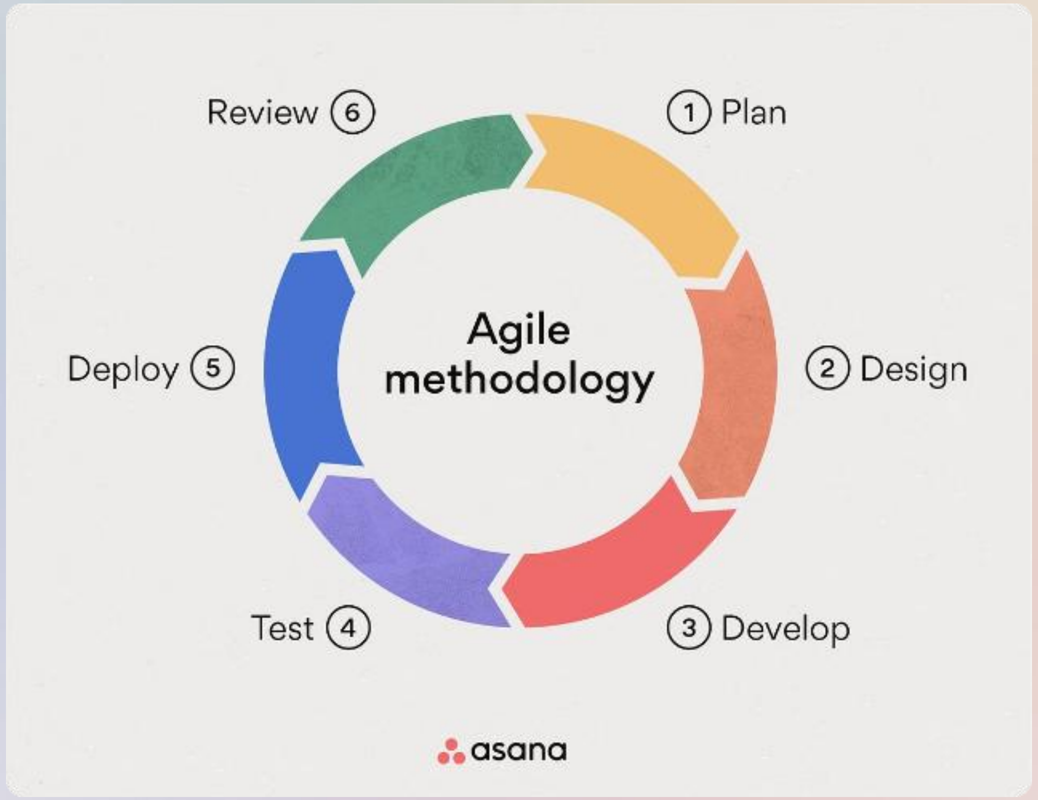
Adapt to evolving requirements and market conditions.

Collaboration

Foster teamwork and cross-functional communication.

Continuous Improvement

Regularly reflect and identify areas for enhancement.



Agile Methodologies

1

Scrum

A popular agile framework that emphasizes iterative development and regular feedback.

2

Kanban

A visual system for managing and optimizing the flow of work.

3

Lean

A methodology focused on minimizing waste and maximizing value.

Agile Benefits

Faster Delivery

Agile's iterative approach allows for quicker product releases.

Improved Quality

Continuous testing and feedback lead to higher-quality products.

Enhanced Adaptability

Agile's flexibility enables teams to respond to changing needs.

Agile Certification

Google Project Management Certificate

Earn a professional certification in agile project management from Google.

Hands-on Learning

Develop practical skills through interactive lessons and real-world projects.

Career Opportunities

Prepare for in-demand agile project management roles.

Flexible Learning

Complete the program at your own pace through an online platform.

What is Iterative and Incremental Development?

Iterative and incremental development is a process that combines the iterative design method with the incremental build model. It is used by software developers to help manage projects.

Incremental: An incremental approach breaks the software development process down into small, manageable portions known as increments. Each increment builds on the previous version so that improvements are made step by step.

Iterative: An iterative model means software development activities are systematically repeated in cycles known as iterations. A new version of the software is produced after each iteration until the optimal product is achieved.

Iterative and incremental development models are complementary in nature, which is why they are often used together to boost their efficacy and achieve project deliverables.

What is Iterative and Incremental Development?

An example of iterative and incremental development in Agile could be the creation of a new **e-commerce website**.

The project would be broken down into smaller increments, such as building a wireframe, uploading products, and creating advertising copy.

As these steps are unfolding, the software development team would repeat the cycles of prototyping and testing to make improvements to the website with each iteration.

What is Iterative and Incremental Development?

- The incremental and iterative development process is integral to the field of Agile software development as it enables project managers to reap the benefits of both incremental and iterative approaches.
- Incremental development ensures that **developers can make changes early on in the process rather than waiting until the end** when the allotted time has run out and the money has been spent.
- Iterative development means **improvements are made on an ongoing basis,** so the end result is likely to be delivered on time and be of higher quality.

Extreme Programming

- Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software and higher quality of life for the development team.
- XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

Extreme Programming - Values

Communication

Software development is inherently a team sport that relies on communication to transfer knowledge from one team member to everyone else on the team. XP stresses the importance of the appropriate kind of communication – face-to-face discussion with the aid of a whiteboard or other drawing mechanism.

Simplicity

Simplicity means “what is the simplest thing that will work?” The purpose of this is to avoid waste and do only absolutely necessary things such as keep the design of the system as simple as possible so that it is easier to maintain, support, and revise. Simplicity also means addressing only the requirements that you know about; don’t try to predict the future.

Feedback

Through constant feedback about their previous efforts, teams can identify areas for improvement and revise their practices. Feedback also supports simple design. Your team builds something, gathers feedback on your design and implementation, and then adjusts your product going forward.

Extreme Programming - Values

Courage

Kent Beck defined courage as “effective action in the face of fear”. This definition shows a preference for action based on other principles so that the results aren’t harmful to the team. You need courage to raise organizational issues that reduce your team’s effectiveness. You need courage to stop doing something that doesn’t work and try something else. You need courage to accept and act on feedback, even when it’s difficult to accept.

Respect

The members of your team need to respect each other in order to communicate with each other, provide and accept feedback that honors your relationship, and work together to identify simple designs and solutions.

Extreme Programming - Practices

Sit Together
Since communication is one of the five values of XP, and most people agree that face-to-face conversation is the best form of communication, have your team sit together in the same space without barriers to communication, such as cubicle walls.
Whole Team
A cross-functional group of people with the necessary roles for a product form a single team. This means people with a need as well as all the people who play some part in satisfying that need all work together on a daily basis to accomplish a specific outcome.
Informative Workspace
Set up your team space to facilitate face-to-face communication, allow people to have some privacy when they need it, and make the work of the team transparent to each other and to interested parties outside the team. Utilize Information Radiators to actively communicate up-to-date information.
Energized Work
You are most effective at software development and all knowledge work when you are focused and free from distractions.
Energized work means taking steps to make sure you are able physically and mentally to get into a focused state. This means do not overwork yourself (or let others overwork you). It also means staying healthy, and showing respect to your teammates to keep them healthy.

Extreme Programming - Practices

Pair Programming

Pair Programming means all production software is developed by two people sitting at the same machine. The idea behind this practice is that two brains and four eyes are better than one brain and two eyes. You effectively get a continuous code review and quicker response to nagging problems that may stop one person dead in their tracks.

Teams that have used pair programming have found that it improves quality and does not actually take twice as long because they are able to work through problems quickly and they stay more focused on the task at hand, thereby creating less code to accomplish the same thing.

Stories

Describe what the product should do in terms meaningful to customers and users. These **stories** are intended to be short descriptions of things users want to be able to do with the product that can be used for planning and serve as reminders for more detailed conversations when the team gets around to realizing that particular story.

Weekly Cycle

The Weekly Cycle is synonymous with an **iteration**. In the case of XP, the team meets on the first day of the week to reflect on progress to date, the customer picks the stories they would like delivered in that week, and the team determines how they will approach those stories. The goal by the end of the week is to have running tested features that realize the selected stories. The intent behind the time-boxed delivery period is to produce something to show to the customer for feedback.

Extreme Programming - Practices

Quarterly Cycle

The Quarterly Cycle is synonymous with a release. The purpose is to keep the detailed work of each weekly cycle in the context of the overall project. The customer lays out the overall plan for the team in terms of features desired within a particular quarter, which provides the team with a view of the forest while they are in the trees, and it also helps the customer to work with other stakeholders who may need some idea of when features will be available.

Slack

The idea behind slack in XP terms is to add some low-priority tasks or stories in your weekly and quarterly cycles that can be dropped if the team gets behind on more important tasks or stories. Put another way, account for the inherent variability in estimates to make sure you leave yourself a good chance of meeting your forecasts.

Ten-Minute Build

The goal of the Ten-Minute Build is to automatically build the whole system and run all of the tests in ten minutes. The founders of XP suggested a 10-minute time frame because if a team has a build that takes longer than that, it is less likely to be run on a frequent basis, thus introducing a longer time between errors.

Extreme Programming - Practices

Continuous Integration

Continuous Integration is a practice where code changes are immediately tested when they are added to a larger code base. The benefit of this practice is you can catch and fix integration issues sooner.

Test-First Programming

Instead of following the normal path of:
develop code -> write tests -> run tests
The practice of Test-First Programming follows the path of:
Write failing automated test -> Run failing test -> develop code to make test pass -> run test -> repeat
As with Continuous Integration, Test-First Programming reduces the feedback cycle for developers to identify and resolve issues, thereby decreasing the number of bugs that get introduced into production.

Incremental Design

The practice of Incremental Design suggests that you do a little bit of work upfront to understand the proper breadth-wise perspective of the system design, and then dive into the details of a particular aspect of that design when you deliver specific features. This approach reduces the cost of changes and allows you to make design decisions when necessary based on the most current information available.
The practice of Refactoring was originally listed among the 12 core but was incorporated into the practice of Incremental Design. Refactoring is an excellent practice to use to keep the design simple, and one of the most recommended uses of refactoring is to remove duplication of processes.

Lean Software Development (LSD)

Lean Software Development (LSD) is an agile framework used to streamline and optimize the software development process. It may also be referred to as the Minimum Viable Product (MVP) strategy as these ways of thinking are very similar since both intend to speed up development by focusing on new deliverables.

Lean Software Development (LSD)

Eliminate Waste

Lean development aims to eliminate anything that does not add value to the customer.

According to the Toyota School of lean manufacturing, there are seven wastes including:

Overproduction – manufacturing product before it is required

Unnecessary transportation – Moving inventory from one place to another, wherein there's a risk of damage without any value addition

Inventory – holding inventory increases cost without adding any value to the consumer; excess inventory occupies valuable space, increases lead times and leads to delayed innovation

Motion – denotes unnecessary motion of workers

Defects – quality issues cause having to rework or scrap and can lead to excessive additional costs to companies that do not usually find ways to eliminate sources of defects.

Over-processing – refers to the usage of advanced, expensive tools instead of simple ones.

Waiting – when inventory waits while value-adding steps are being performed.

Lean Software Development (LSD)

What is LSD?

Lean Software Development (LSD) is an approach derived from **lean manufacturing principles** aimed at optimizing efficiency and minimizing waste in the software development process.

Prevent Defects: It integrates quality assurance throughout the development process to prevent defects.

Eliminate Waste: It focuses on activities that add value to the customer and eliminates those activities that do not add value.

Fast Delivery: Reduces cycle time to deliver software quickly and respond to feedback and changing requirements rapidly.

Delay Decisions: Delay decisions until they can be made based on facts.

Lean Software Development (LSD)

What is LSD?

Tom and Mary Poppendieck translated these wastes to how to eliminate them in software development so as to boost customer satisfaction:

Redundant Code or functionality – does not provide additional value to the user; requires time for discussion, development, testing, and documentation; delays feedback loops.

Delay in the software development process - slows downtime to the customer, delays feedback loops.

Unclear or fluctuating requirements – ends in having to rework, frustration, quality issues, and loss of focus.

Slow or ineffective communication – results in delays, poor communication that can affect the reputation of the IT team.

Partially done work – does not provide value to the customer or enable the team to gain knowledge from work.

Defects and quality issues – abandoned work, lead to rework, and poor customer satisfaction.

Switching Tasks – ends in delays, poor work quality, communication issues.

Lean Software Development (LSD)

Seven Principles of LSD

1. Eliminating the Waste

To identify and eliminate wastes e.g. unnecessary code, delay in processes, inefficient communication, issues with quality, data duplication, more tasks in the log than completed, etc. regular meetings are held by Project Managers. This allows team members to point out faults and suggest changes in the next turn.

2. Fast Delivery

Previously long-time planning used to be the key to success in business, but with time, it has been found that engineers spend too much time on building complex systems with unwanted features. So they came up with an MVP strategy which resulted in building products quickly that included a little functionality and launching the product to market and seeing the reaction. Such an approach allows them to enhance the product based on customer feedback.

Lean Software Development (LSD)

Seven Principles of LSD

3. Amplify Learning

Learning is improved through ample code reviewing and meetings that are cross-team applicable. It is also ensured that particular knowledge isn’t accumulated by one engineer who’s writing a particular piece of code so paired programming is used.

4. Builds Quality

LSD is all about preventing waste and keeping an eye on not sacrificing quality. Developers often apply test-driven programming to examine the code before it is written. Quality can also be gained by getting constant feedback from team members and project managers.

5. Respect Teamwork

LSD focuses on empowering team members, rather than controlling them. Setting up a collaborative atmosphere, keeping perfect balance when there are short deadlines and immense workload. This method becomes very important when new members join a well-established team.

Lean Software Development (LSD)

Seven Principles of LSD

6. Delay the Commitment

In traditional project management, it often happens when you make your application and it turns out to be completely unfit for the market. LSD method recognizes this threat and makes room for improvement by postponing irreversible decisions until all experiment is done. This methodology always constructs software as flexible, so new knowledge is available and engineers can make improvements.

7. Optimizing the Whole System

Lean’s principle allows managers to break an issue into small constituent parts to optimize the team’s workflow, create unity among members, and inspire a sense of shared responsibility which results in enhancing the team’s performance.

Lean Software Development (LSD)

Benefits of LSD

Increased Efficiency: LSD reduces delays and inefficiencies by identifying and eliminating non-value-adding activities.

Higher Quality: It integrates quality assurance throughout the development process, thus preventing defects and ensuring quality products.

Faster Delivery: Shorter development cycles allow for quicker release of features and updates, thus meeting customer demands more rapidly.

Adaptability: Delaying decisions until they are necessary and are based on facts, allowing teams to adapt to changes and new information.

Enhanced Collaboration: Engages customers throughout the development process, ensuring that their needs and feedback are continuously addressed.

Lean Software Development (LSD)

Challenges of Lean development

Team training requirements. If a team is unable to work together and communicate, development efforts suffer. Teams and team members must be trained so they can take on each other's work and adapt to changing initiatives. They must also be able to organize work and trade duties as priorities change.

Missing metrics. A common pitfall of Lean is measuring the wrong software metrics or not measuring at all. Since one of the main goals of Lean software engineering is to eliminate waste, teams must be able to identify waste and measure it.

No boundaries. Software requirements specifications evolve in Lean as customers provide feedback. Too many requirements and too much change create complexity and make it difficult to release a cohesive piece of software.

Suboptimization. Suboptimization is when only part of the production system's value stream is optimized. This happens when a problem is broken into parts that are solved in isolation without considering the effect one part will have on other parts or the entire system.

Lean Software Development (LSD)

Challenges of Lean development

Team training requirements. If a team is unable to work together and communicate, development efforts suffer. Teams and team members must be trained so they can take on each other's work and adapt to changing initiatives. They must also be able to organize work and trade duties as priorities change.

Missing metrics. A common pitfall of Lean is measuring the wrong software metrics or not measuring at all. Since one of the main goals of Lean software engineering is to eliminate waste, teams must be able to identify waste and measure it.

No boundaries. Software requirements specifications evolve in Lean as customers provide feedback. Too many requirements and too much change create complexity and make it difficult to release a cohesive piece of software.

Suboptimization. Suboptimization is when only part of the production system's value stream is optimized. This happens when a problem is broken into parts that are solved in isolation without considering the effect one part will have on other parts or the entire system.

Lean Software Development (LSD)

Challenges of Lean development

Team training requirements. If a team is unable to work together and communicate, development efforts suffer. Teams and team members must be trained so they can take on each other's work and adapt to changing initiatives. They must also be able to organize work and trade duties as priorities change.

Missing metrics. A common pitfall of Lean is measuring the wrong software metrics or not measuring at all. Since one of the main goals of Lean software engineering is to eliminate waste, teams must be able to identify waste and measure it.

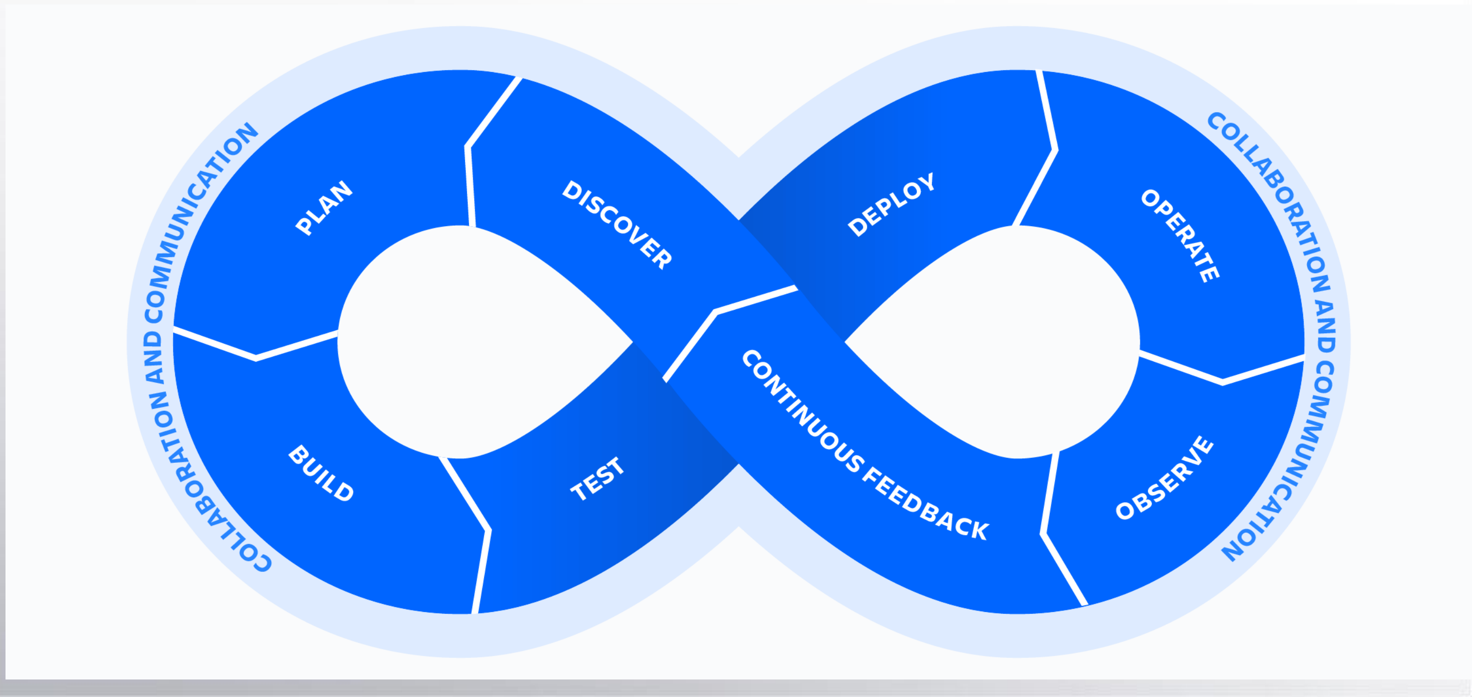
No boundaries. Software requirements specifications evolve in Lean as customers provide feedback. Too many requirements and too much change create complexity and make it difficult to release a cohesive piece of software.

Suboptimization. Suboptimization is when only part of the production system's value stream is optimized. This happens when a problem is broken into parts that are solved in isolation without considering the effect one part will have on other parts or the entire system.

DevOps

It’s best to understand DevOps as a business drive to improve communication and collaboration among development and operations teams, in order to increase the speed and quality of software deployment. It’s a new way of working that has profound implications for teams and the organizations they work for.

DevOps is more than just development and operations teams working together. It’s more than tools and practices. DevOps is a mindset, a cultural shift, where teams adopt new ways of working.


COLLABORATION AND COMMUNICATION
PLAN
DISCOVER
BUILD
TEST
DEPLOY
OPERATE
OBSERVE
CONTINUOUS FEEDBACK
COLLABORATION AND COMMUNICATION

The importance of DevOps

A DevOps culture means developers get closer to the user by gaining a better understanding of user requirements and needs.

Operations teams get involved in the development process and add maintenance requirements and customer needs.

It means adhering to the following key principles that help DevOps teams deliver applications and services at a faster pace and higher quality than organizations using the traditional software development model.

DevOps is a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.

The importance of DevOps

The DevOps movement began around 2007 when the software development and IT operations communities raised concerns about the traditional software development model, where developers who wrote code worked apart from operations who deployed and supported the code. The term DevOps, a combination of the words development and operations, reflects the process of integrating these disciplines into one, continuous process.

How does DevOps work?

A DevOps team includes **developers and IT operations** working collaboratively throughout the product lifecycle, in order to increase the speed and quality of software deployment.

It’s a new way of working, a cultural shift, that has significant implications for teams and the organizations they work for.

Under a DevOps model, development and operations teams are no longer “siloed.” Sometimes, these two teams merge into a single team where the engineers work across the entire application lifecycle — from development and test to deployment and operations — and have a range of multidisciplinary skills.

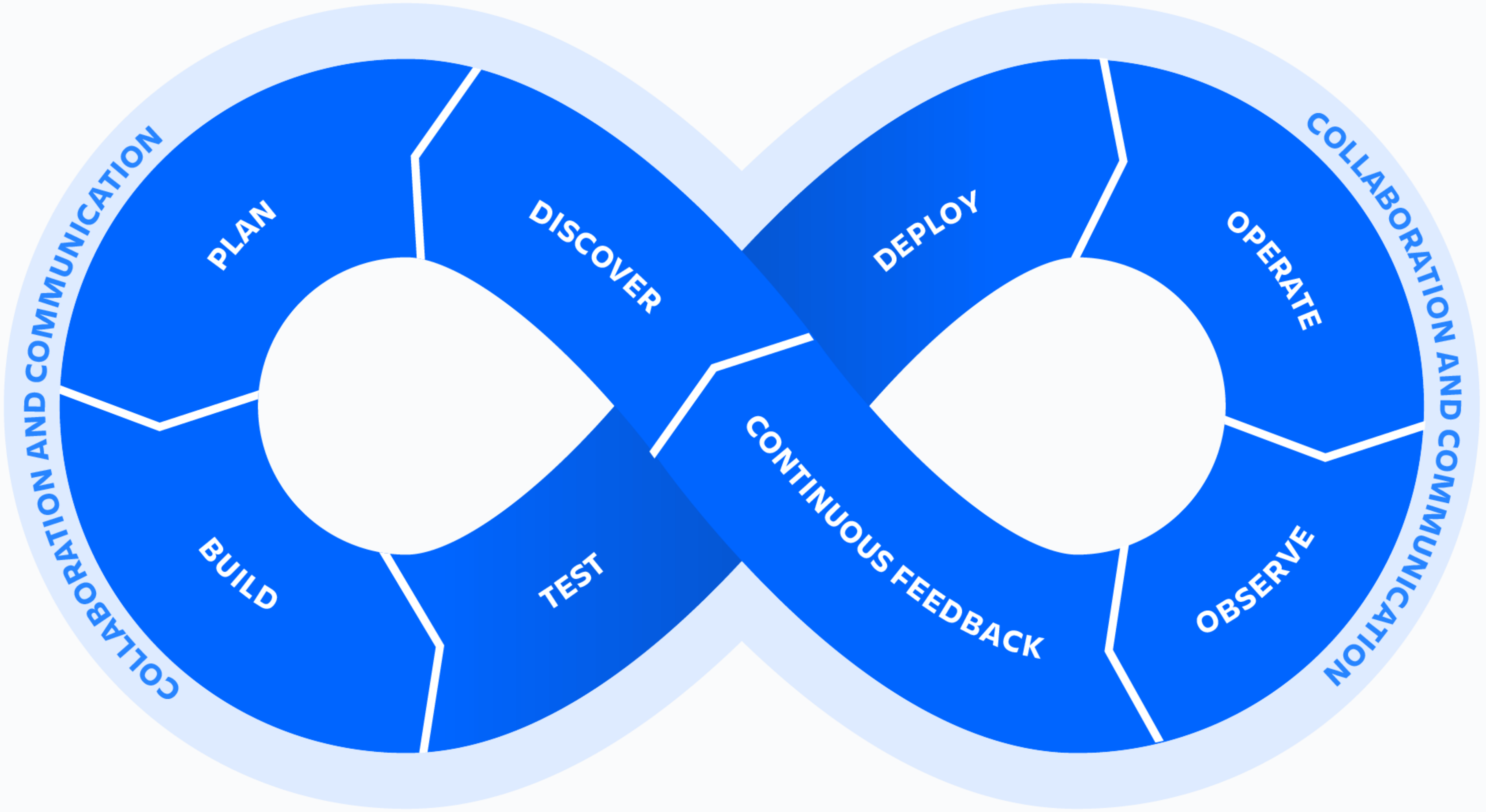
How does DevOps work?

DevOps teams use tools to automate and accelerate processes, which helps to increase reliability. A DevOps toolchain helps teams tackle important DevOps fundamentals including continuous integration, continuous delivery, automation, and collaboration.

DevOps values are sometimes applied to teams other than development. When security teams adopt a DevOps approach, security is an active and integrated part of the development process. This is called DevSecOps.

The DevOps lifecycle

Because of the continuous nature of DevOps, practitioners use the infinity loop to show how the phases of the DevOps lifecycle relate to each other. Despite appearing to flow sequentially, the loop symbolizes the need for constant collaboration and iterative improvement throughout the entire lifecycle.


COLLABORATION AND COMMUNICATION
PLAN
DISCOVER
BUILD
TEST
DEPLOY
OPERATE
CONTINUOUS FEEDBACK
OBSERVE
COLLABORATION AND COMMUNICATION

The DevOps lifecycle

Discover

Building software is a team sport. In preparation for the upcoming sprint, teams must workshop to explore, organize, and prioritize ideas. Ideas must align to strategic goals and deliver customer impact. Agile can help guide DevOps teams.

Plan

DevOps teams should adopt agile practices to improve speed and quality. Agile is an iterative approach to project management and software development that helps teams break work into smaller pieces to deliver incremental value.

Build

Git is a free and open source version control system. It offers excellent support for branching, merging, and rewriting repository history, which has led to many innovative and powerful workflows and tools for the development build process.

The DevOps lifecycle

Test

Continuous integration (CI) allows multiple developers to contribute to a single shared repository. When code changes are merged, automated tests are run to ensure correctness before integration. Merging and testing code often help development teams gain reassurance in the quality and predictability of code once deployed.

Deploy

Continuous deployment (CD) allows teams to release features frequently into production in an automated fashion. Teams also have the option to deploy with feature flags, delivering new code to users steadily and methodically rather than all at once. This approach improves velocity, productivity, and sustainability of software development teams.

The DevOps lifecycle

Operate

Manage the end-to-end delivery of IT services to customers. This includes the practices involved in design, implementation, configuration, deployment, and maintenance of all IT infrastructure that supports an organization’s services.

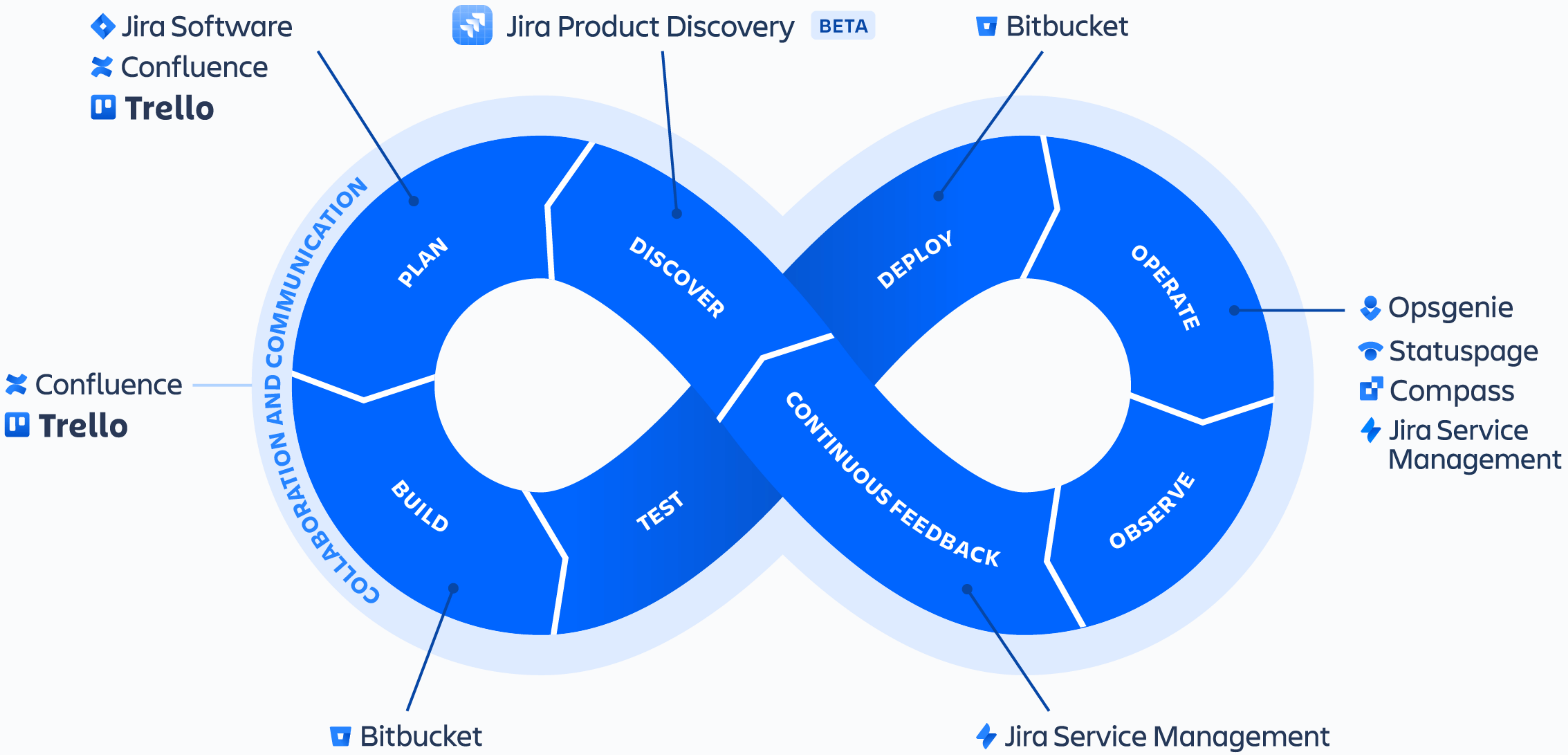
Observe

Quickly identify and resolve issues that impact product uptime, speed, and functionality. Automatically notify your team of changes, high-risk actions, or failures, so you can keep services on.

Continuous feedback

DevOps teams should evaluate each release and generate reports to improve future releases. By gathering continuous feedback, teams can improve their processes and incorporate customer feedback to improve the next release.

DevOps tools



What are the benefits of DevOps?

Speed

Teams that practice DevOps release deliverables more frequently, with higher quality and stability.

Improved collaboration

The foundation of DevOps is a culture of collaboration between developers and operations teams, who share responsibilities and combine work. This makes teams more efficient and saves time related to work handoffs and creating code that is designed for the environment where it runs.

Rapid deployment

By increasing the frequency and velocity of releases, DevOps teams improve products rapidly. A competitive advantage can be gained by quickly releasing new features and repairing bugs.

What are the benefits of DevOps?

Quality and reliability

Practices like continuous integration and continuous delivery ensure changes are functional and safe, which improves the quality of a software product. Monitoring helps teams keep informed of performance in real-time.

Security

By integrating security into a continuous integration, continuous delivery, and continuous deployment pipeline, DevSecOps is an active, integrated part of the development process. Security is built into the product by integrating active security audits and security testing into agile development and DevOps workflows.