Agile delivery: A guide to finding the right model

GitLab

What's inside?

Introduction

Helping teams efficiently create customer-centric products, Agile is one of the most important and transformative methodologies introduced to the software engineering discipline in recent decades. Agile has gone through several transformations over the years — from the introduction of Waterfall in the 1970s and Scrum and XP in the 1990s to the Agile Manifesto in the early 2000s and SAFe (Scaled Agile Framework) and LeSS (Large Scale Scrum) in more recent times. With the rapid speed of today's market, software teams must keep up with customer demand and adopt a development approach that facilitates rapid delivery.

Agile delivery is an iterative approach to software delivery in which teams build software incrementally at the beginning of a project rather than ship it at once upon completion.

> **Agile development means taking iterative, incremental, and lean approaches to streamline and accelerate the delivery of projects.**

The Agile model empowers teams to rapidly respond to changes and cultivates a mindset that embraces "collaboration, a positive outlook, an openness to change, a willingness to fail (and recover fast), and transparency." In this iterative approach, teams can develop a minimum viable product and improve its features in successive iterations. Agile project planning paves the way for teams to adopt a variety of methods to increase visibility and collaboration, making it a popular software development methodology.

Development teams have created several approaches to deliver better software faster. These techniques help teams organize, plan, and deliver customer-focused software. This ebook explores the five core models of Agile delivery, helping you determine the best approach for your team.

Scrum methodology

Overview

Scrum, often synonymous with Agile, is an approach that emphasizes continuous improvement, self organization, and experience-based learning.

> **Scrum is "a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value."**
>
> *— THE SCRUM GUIDE*

By utilizing user stories, tasks, backlogs, and extensions, teams have a structured model to carry them across the software development lifecycle. Development occurs in short, fixed timeframes (sprints) to allow complex tasks to be completed incrementally, with releases at the end of each sprint. The key metric in Scrum is velocity (i.e. the number of user stories completed in a sprint), and required roles include Scrum Master and product owner. Teams that use a Scrum approach to development are likely to be committed, respectful, and focused.

Scrum works best for small teams that collaborate on complex projects and isn't well-suited for teams that experience significant changes and variety in workflow and planning.

Implementation

Implementing a Scrum methodology can be accomplished in five steps.

1. **Create the team:** The team of 5-9 members should come from a variety of functions, such as developers, designers, and testers, ensuring delivery at the end of each sprint.

2. **Determine sprint length:** Sprints typically last between 7 and 30 days. Many teams have 2-week sprints, so it's recommended that you start there and adjust your sprint length as you learn what works best for your team. Before each sprint, a planning meeting determines the work for the sprint, while the team commits to completing the tasks. Upon sprint completion, a retrospective or demo is held in which the team discusses improvements and plans the work for the next sprint.

3. **Identify Scrum Master:** The Scrum Master acts as the project manager for the team in the sense that this role ensures the team progresses on goals and resolves challenges. The Scrum Master also helps lead sprint planning.

4. **Identify product owner:** The product owner represents the end user and champions the voice of the customer by writing and prioritizing stories.

5. **Prioritize product backlog:** The backlog contains all the stories (requirements) in a project. The product owner continuously prioritizes the stories in the blacklog so that sprint planning is done according to story importance.

Kanban development

Overview

Kanban is a highly visual workflow that involves a one-piece flow from the backlog. Developers pull tasks from the backlog and work on them until completion — rather than in a fixed time-box. The main goal of Kanban teams is to reduce the cycle time of a user story, so continuous delivery is important. Tasks are organized using Kanban cards on a board, enabling end-to-end visibility throughout production. Teams use Kanban boards to continuously improve their flow of work.

Three practices guide Kanban:

- **Visualize work:** Concurrently viewing items provides contextual information.
- **Limit work in progress:** Kanban is about a steady flow of completing projects. Too much work can create bottlenecks.
- **Manage flow:** Pulling from the backlog ensures continuous delivery.

The key metrics in Kanban development are cycle time, lead time, and works in progress. Decreasing the average amount of time it takes a project to go from start to finish and limiting the number of projects in flux helps Kanban teams steadily progress. Teams that use a Kanban framework favor transparency and communication, and the culture is collaborative, transparent, balanced, and customer focused.

This development methodology works best when teams are comprised of independent developers who don't collaborate to work on projects and teams that experience frequent shifts in priorities and requirements. Kanban is not recommended for highly collaborative teams that require structure.

Implementation

Implementing a Kanban methodology can be accomplished in four steps.

1. **Visualize work:** Create a Kanban board (digital or physical) with columns that represent every stage across planning and completion. Shift tasks in the columns according to current stage.

2. **Limit work in progress:** Limiting the number of items allows team members to spend their time more efficiently. Creating WIP limits enforces a steady flow of work and eliminates waste.

3. **Enhance flow:** When tasks are finished and teams are ready, the next highest priority item is pulled from the backlog.

4. **Make improvements:** Continuous improvement is important, so measure and analyze performance using lead time, cycle time, and WIP to generate actionable insights into how your team can improve.

Extreme programming (XP)

Overview

Extreme Programming (XP) focuses on continuous delivery and speed. With XP, the customer directs development by prioritizing user stories, while developers iterate through rapid feedback loops, customer collaboration, and continuous planning and testing. Teams deliver software between 1 to 3 weeks, adjusting software according to customer input.

There are five values of XP:

- **Communication:** XP favors in-person discussion and communication on everything from planning to code.
- **Simplicity:** Find the simplest solution that will work in order to eliminate waste, and only work on the requirements that are known.
- **Feedback:** Carefully listen to feedback and identify areas for improvement. Adapt processes to feedback and commit to thoughtful iterations.
- **Courage:** Be honest about estimates and progress and courageously stop processes that aren't working.

» **Respect:** Team members must respectfully communicate and provide feedback. Everyone on the team is a valuable contributor, regardless of role.

The key metric is customer satisfaction. Due to the high level of collaboration between development teams and customers, the final software product precisely meets customer expectations.

Extreme programming works best for small teams, with a flat management structure, that experience frequent changes in user requirements. These teams should consist of senior developers who are skilled in working with non-technical individuals.

Implementation

Implementing an extreme programming approach can be accomplished in three steps.

1. **Organize user stories:** Customers identify a set of stories, helping the team both understand the desired results and estimate the size of each story. Using the size estimate and customer feedback, the team can prioritize stories. Invite relevant stakeholders, including customers, developers, and managers, to create a release schedule.

2. **Release cycles:** At the beginning of the cycle (typically 1-3 weeks), the team and customer determine which stories will be delivered in that time-box. Stories are disseminated into tasks that are completed during that cycle.

3. **Feedback session:** At the end of the cycle, the team and customer assess progress and review the work. The customer provides feedback on product value, and the team makes adjustments based on customer input.

Feature-driven development (FDD)

Overview

Feature-driven development (FDD) is a developer-centric process that combines software engineering best practices, such as code ownership and domain object modeling. FDD focuses on creating value for the customer and delivers working software in two-week cycles. In feature-driven development, the scope of the project is defined, but the details are not determined. Continuous delivery and

continuous improvement make this model a good choice for scalability.

Dedicated roles in FDD include chief architect, chief programmer, and class owner (i.e. developer who owns a piece of code). FDD can be used with teams comprised of developers with varied experience levels. One of the advantages of feature-driven development is that all aspects of a project are tracked by a feature, so there are well-defined tracking and reporting capabilities. The key metrics include the number and type of bugs detected during testing.

Feature-driven development is best for development teams in regulation-driven industries, such as finance and government, that have mature development processes and value quality.

Implementation

Implementing feature-driven development can be done in five activities:

1. **Develop overall model:** This is a high-level overview of the scope of the system and its context. Chief programmers and the chief architect conduct a domain walkthrough. They'll create model diagrams and determine what classes are in the domain. Creating an overall model shape generates a feature list.

2. **Build feature list:** The chief programmers filter the domain into subject areas (or sets). The features in this list are small and should be built in two weeks. If a feature takes longer than two weeks to build, it should be broken into smaller pieces.

3. **Plan by feature:** Based on dependencies, workload, and complexity, the team creates a development plan. The project manager, chief programmer, and development manager assign features (as classes) to developers.

4. **Design by feature:** The lead developer and class owners make sequence diagrams and adjust the overall model. After class and method prologues are written, the team holds a design inspection.

5. **Build by feature:** After successful unit testing and code inspection, the completed feature is promoted to the main build.

Dynamic Systems Development Method (DSDM)

Overview

Dynamic systems development method (DSDM) embraces customer involvement and focuses on fitness for business purpose, leaving some technical issues for a later iteration. The idea is that developers prioritize the requirements that help the business right now rather than develop for future impact. All development work must be reversible and completed in short, fixed iterations. Like XP, dynamic systems development method involves significant customer feedback and incremental development.

There are nine guiding principles of dynamic systems development method:

1. Active customer involvement is important
2. Teams must be empowered to make critical decisions
3. Continuous delivery is necessary
4. Fitness for business purpose is a necessary component of successful delivery
5. Incremental and iterative development is essential
6. All changes must be reversible
7. Requirements are baselined at a high level
8. Integrated testing occurs throughout the development lifecycle
9. Stakeholders must be collaborative and cooperative

The end result of DSDM is a system that meets customers' business requirements, and projects are delivered on time and within budget. Teams that use DSDM value transparency and visibility and possess strong collaboration and communication skills.

Requirements are prioritized using the MoSCoW method:

- **Must have (M):** These requirements must be delivered in the fixed time-box.
- **Should have (S):** If possible, these requirements should be included, but if it's likely that they'll delay a release schedule or prevent the inclusion of higher priority requirements, they should be removed.
- **Could have (C):** These requirements are not critical but can be included if time permits.

» **Won't have (W):** Although these requirements won't be included now, they could be added later.

DSDM is best for teams that must meet their customers' business needs. This methodology requires co-located collaboration between customers and developers, and teams should be self-managed and have the authority to make time-sensitive decisions. DSDM will not work well for teams that require permission to make critical decisions that will impact the project.

Implementation

Implementing a dynamic systems development method can be achieved in five steps.

1. **Feasibility:** Determining whether the proposed system is technically possible is the goal of this stage. Moreover, teams must determine whether DSDM is the most appropriate method to build the system. This step lasts only a few weeks.

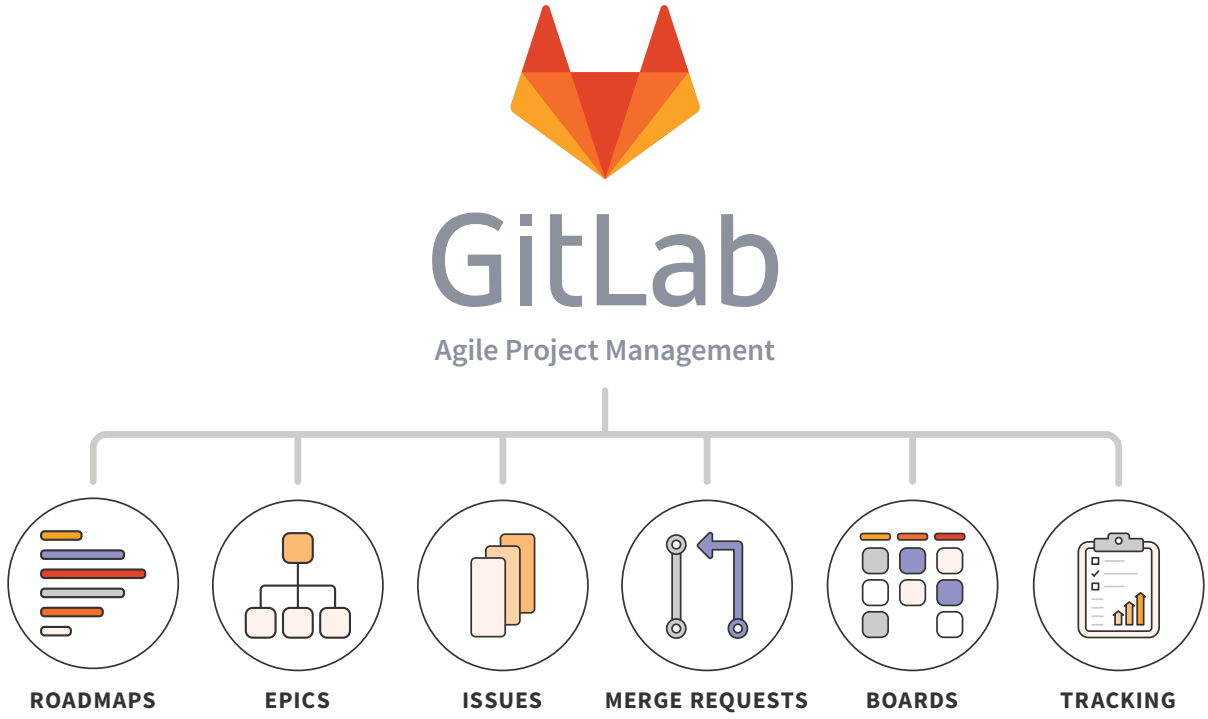
2. **Business study:** This highly collaborative step involves working with the customer to understand development priorities to identify business and system information requirements at a high level. The team also creates the system's architectural framework with major components and interfaces.

3. **Functional model iteration:** Teams iteratively build the prototype and receive customer feedback to refine requirements.

4. **Design and build iteration:** Teams ensure that the system works well in the specified operational environment and is built to a high enough degree that it can be used by the customer. The tested system will meet the core requirements agreed upon during the business study.

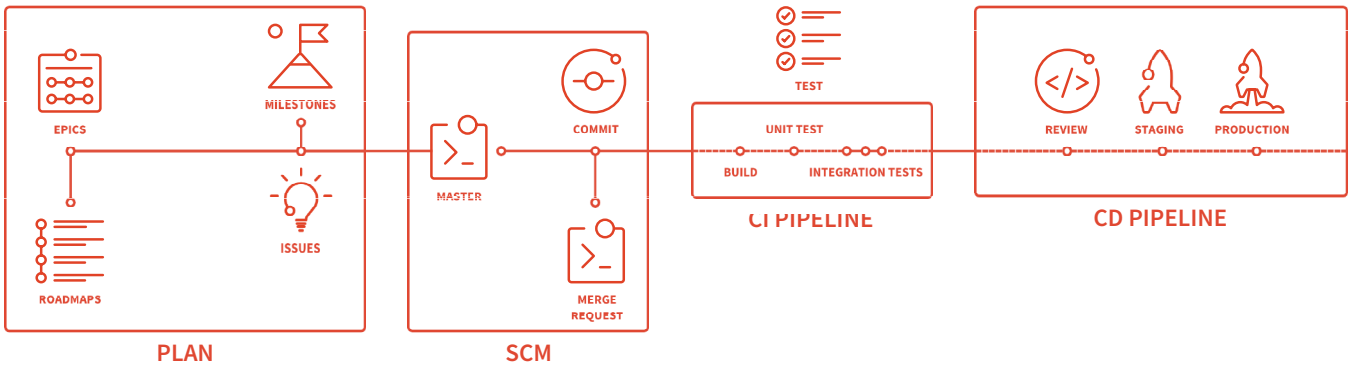
5. **Implementation:** Teams provide customers with the system and provide training.

Agile delivery with GitLab



GitLab is flexible enough to adapt to your methodology and help your team embrace Agile efforts by providing Milestones (or sprints), Issues (or user stories), weights (or points and estimation), and other common Agile artifacts.

GitLab provides end-to-end DevOps, encapsulating 10 defined stages: Manage, Plan, Create, Verify, Package, Secure, Release, Configure, Monitor, and Defend. Agile project management resides under the Manage and Plan categories and is the starting point in the DevOps lifecycle.



GitLab provides one complete application for Agile project management, version control, continuous integration/delivery/deployment, packaging, security, and monitoring, GitLab removes the complexity of DevOps toolchains.

- **Integrated:** Agile is part of GitLab, enabling collaboration and visibility from planning to deployment (and beyond).
- **Easy to learn:** See our Quick Start guide on setting up Agile teams.
- **Seamless:** Agile project management is woven throughout the single GitLab application, with intuitive user experience.
- **Scalable:** Organize multiple Agile teams to achieve enterprise agile scalability.
- **Flexible:** Flexible enough to adapt to your methodology, whether Agile or influenced by it.
- **Team Tracking:** Track work with boards, backlogs, dashboards, and traceability into your CI/CD.

For more information on GitLab's Agile project management features, please view our educational videos on how to set up your organization.

Start your free GitLab trial

About GitLab

GitLab is the first single application for the entire DevOps lifecycle. Only GitLab enables Concurrent DevOps, unlocking organizations from the constraints of today's toolchain. GitLab provides unmatched visibility, radical new levels of efficiency and comprehensive governance to significantly compress the time between planning a change and monitoring its effect. This makes the software lifecycle 200% faster, radically improving the speed of business.

GitLab and Concurrent DevOps collapses cycle times by driving higher efficiency across all stages of the software development lifecycle. For the first time, Product, Development, QA, Security, and Operations teams can work concurrently in a single application. There's no need to integrate and synchronize tools, or waste time waiting for handoffs. Everyone contributes to a single conversation, instead of managing multiple threads across disparate tools. And only GitLab gives teams complete visibility across the lifecycle with a single, trusted source of data to simplify troubleshooting and drive accountability. All activity is governed by consistent controls, making security and compliance first-class citizens instead of an afterthought.

Built on Open Source, GitLab leverages the community contributions of thousands of developers and millions of users to continuously deliver new DevOps innovations. More than 100,000 organizations, including Ticketmaster, ING, NASDAQ, Alibaba, Sony, and Intel trust GitLab to deliver great software at new speeds.


GitLab