Alberto Andrés Valdés González.
Degree: Mathematical Engineer.
Work position: Data Scientist.
Mail: anvaldes@uc.cl/alberto.valdes.gonzalez.96@gmail.com
Location: Santiago, Chile.

Metholodogies

Agile Metholodiges

The Agile methodology is a project management approach that involves breaking the project into phases and emphasizes continuous collaboration and improvement. Teams follow a cycle of planning, executing, and evaluating.

Agile Manifesto:

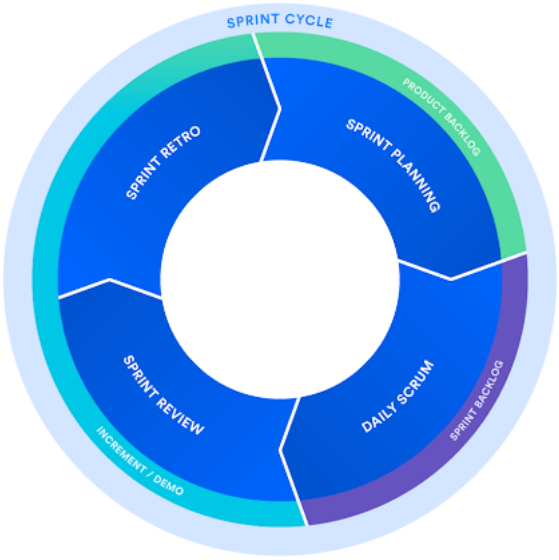
We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.
- That is, while there is value in the items on the right, we value the items on the left more.

SCRUM:

Scrum is an agile project management framework that helps teams structure and manage their work through a set of values, principles, and practices. Much like a rugby team (where it gets its name) training for the big game, scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.



Members of the team:

The scrum product owner:

Product owners are the champions for their product. They are focused on understanding business, customer, and market requirements, then prioritizing the work to be done by the engineering team accordingly. Effective product owners:

- Build and manage the product backlog.
- Closely partner with the business and the team to ensure everyone understands the work items in the product backlog.
- Give the team clear guidance on which features to deliver next.
- Decide when to ship the product with a predisposition towards more frequent delivery.

The scrum master:

Scrum masters are the champions of scrum within their teams. They coach teams, product owners, and the business on the scrum process, and look for ways to fine-tune their practice of it.

The scrum development team:

They are the champions for sustainable development practices. The most effective scrum teams are tight-knit, co-located, and usually five to seven members. One way to work out the team size is to use the famous 'two pizza rule' coined by Jeff Bezos, the CEO of Amazon (the team should be small enough to share two pizzas).

Team members have differing skill sets, and cross-train each other so no one person becomes a bottleneck in the delivery of work.

Scrum artifacts:

Product Backlog: Is the primary list of work that needs to get done and maintained by the product owner or product manager. This is a dynamic list of features, requirements, enhancements, and fixes that acts as the input for the sprint backlog.

Sprint Bakclog: Is the list of items, user stories, or bug fixes, selected by the development team for implementation in the current sprint cycle. Before each sprint, in the sprint planning meeting (which we'll discuss later in the article) the team chooses which items it will work on for the sprint from the product backlog.

Increment (or Sprint Goal): Is the usable end-product from a sprint. You may not hear the word "increment" out in the world, as it's often referred to as the team's definition of "Done", a milestone, the sprint goal, or even a full version or a shipped epic. It just depends on how your teams defines "Done" and how you define your sprint goals.

KANBAN:

Kanban is a popular framework used to implement agile and DevOps software development. It requires real-time communication of capacity and full transparency of work. Work items are represented visually on a kanban board, allowing team members to see the state of every piece of work at any time.

Kanban boards:

The work of all kanban teams revolves around a kanban board, a tool used to visualize work and optimize the flow of the work among the team. While physical boards are popular among some teams, virtual boards are a crucial feature in any agile software development tool for their traceability, easier collaboration, and accessibility from multiple locations.

Kanban cards:

In Japanese, kanban literally translates to "visual signal."For kanban teams, every work item is represented as a separate card on the board.

Benefits of kanban

Planning flexibility: A kanban team is only focused on the work that's actively in progress. Once the team completes a work item, they pluck the next work item off the top of the backlog.

Shortened time cycles: Cycle time is a key metric for kanban teams. Cycle time is the amount of time it takes for a unit of work to travel through the team's workflow – from the moment work starts to the moment it ships. By optimizing cycle time, the team can confidently forecast the delivery of future work.

Fewer bottlenecks: Multitasking kills efficiency. The more work items in flight at any given time, the more context switching, which hinders their path to completion. That's why a key tenet of kanban is to limit the amount of work in progress (WIP). Work-in-progress limits highlight bottlenecks and backups in the team's process due to lack of focus, people, or skill sets.

Visual metrics: One of the core values is a strong focus on continually improving team efficiency and effectiveness with every iteration of work. Charts provide a visual mechanism for teams to ensure they're continuing to improve. When the team can see data, it's easier to spot bottlenecks in the process (and remove them). Two common reports kanban teams use are control charts and cumulative flow diagrams.

Continuous delivery: Continuous delivery (CD) is the practice of releasing work to customers frequently. Continuous integration (CI) is the practice of automatically building and testing code incrementally throughout the day. Together they form a CI/CD pipeline that is essential for development teams (especially for DevOps teams) to ship software faster while ensuring high quality.

SOLID principles

SOLID is an acronym that stands for five key design principles:

1. Single Responsibility Principle (SRP): The Single Responsibility Principle states, „Each software module or class should have only one reason to change“. In other words, we can say that each module or class should have only one responsibility to do.

2. Open-Closed Principle (OCP): The Open/Closed Principle is one of the SOLID principles of software design, which suggests that software entities such as classes, modules, and functions should be open for extension but closed for modification. This principle states that you should be able to add new functionality to a system without having to modify existing code.

3. Liskov Substitution Principle (LSP): The Liskov Substitution Principle (LSP) is a principle in object-oriented programming that states that objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.

4. Interface Segregation Principle (ISP): It states that a client should not be forced to depend on interfaces that it does not use. In other words, it is better to have multiple smaller interfaces that are tailored to specific needs rather than a single large interface that tries to cover everything.

5. Dependency Inversion Principle (DIP): It states that high-level modules should not depend on low-level modules. Instead, both should depend on abstractions. This promotes loose coupling and flexibility, making it easier to make changes without affecting other parts of the system.

Code Review

Code reviews are methodical assessments of code designed to identify bugs, increase code quality, and help developers learn the source code.

4 approachs to code review:

1. Pair programming: Pair programming involves two developers collaborating in real time — one writing code (the driver) and one reviewing code (the navigator). Pairing sessions are popular with development teams because teammates collaborate to identify the most effective solution to a challenge. Team members share knowledge and can quickly overcome difficulties by working through ideas together and drawing on their expertise.

2. Over-the-shoulder reviews: In an over-the-shoulder-review, two developers — the author and reviewer — team up in person or remotely through a shared screen and the author explains the completed change proposal and offers reasoning for the chosen solutions. The reviewer asks questions and makes suggestions, similar to how team members collaborate during pairing sessions. The author can make small changes during the review and note larger fixes for a later time.

3. Tool-assisted reviews: Teams may decide to use tools to save time and ensure the highest quality code is shipped. Tool-assisted reviews can automatically gather changed files and display the differences, or make it easier to provide feedback and have conversations via comments, and incorporate things like static application security testing (SAST) to help identify and remediate vulnerabilities.

4. Email pass-around: Email pass-arounds are often used for minor difficulties and small pieces of code. They can be conducted via email or source code management systems. During an email pass-around, an author sends an email containing code changes to reviewers. Email pass-around is similar to over-the-shoulder reviews in that they can be easily implemented and don't require a strong learning curve or a mentoring stage to teach the author how to make a change.