Software Development Life Cycle

.NET

The Software Development Life Cycle is the process of dividing software development work into distinct, repeatable phases to improve design and product management.

https://en.wikipedia.org/wiki/software_development_process

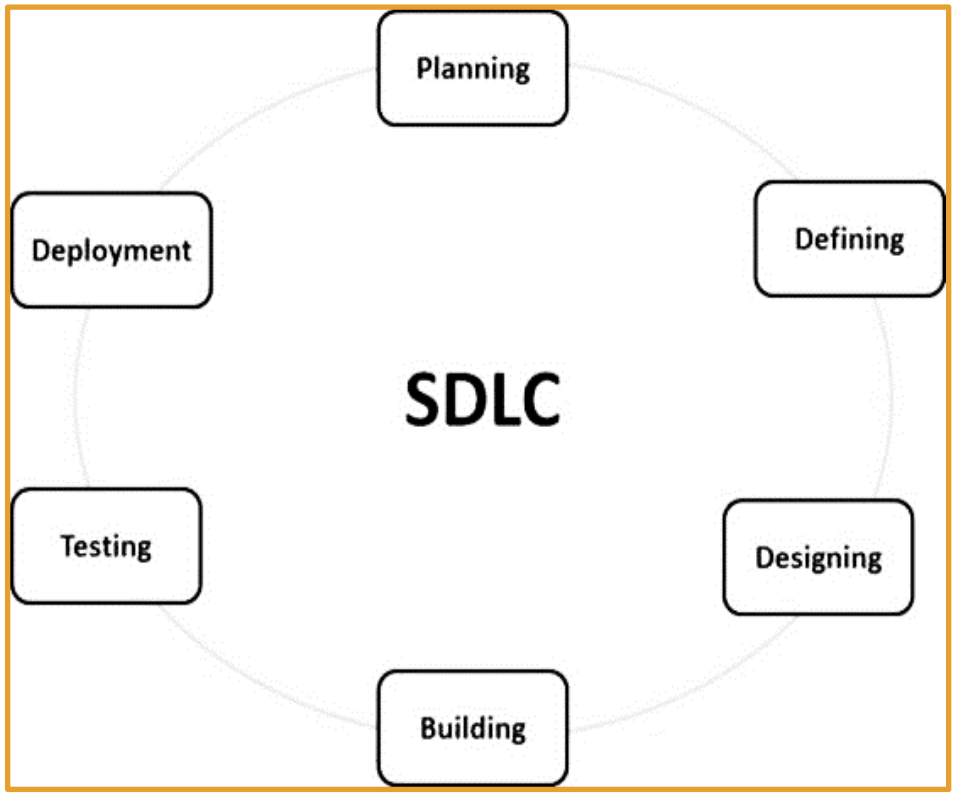
SDLC – Software Development Lifecycle

https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

The ***Software Development Life Cycle (SDLC)*** is a process used to design, develop, and test software.

The phases of the SDLC are:

1. Planning and Requirement Analysis.
2. Define the project requirements.
3. Design the Product Architecture
4. Build the product
5. Test
6. Deployment



SDLC – Stages Details

https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

Stage	Purpose
Planning and Requirement Analysis	All involved parties (Stakeholders) give input to plan the project and decide on the feasibility the various aspects of the project. Different approaches to solving various problems are explored.
Define Project Requirements	Stakeholders define and document the product requirements and get them approved by the customer. A final design approach is decided upon.
Design Product Architecture	Stakeholders decide on the best approach to implementing the design plan.
Build and Develop the Product	A specific programming language is chosen, and code is written following guidelines established by stakeholders.
Test the Product	Extensive testing and refactoring the product is conducted mostly after the product is complete.
Deployment and Maintenance	User Acceptance Testing. Formal (sometimes limited) release with continued evaluation. The code may be refactored based on user feedback.

SDLC – Waterfall Model

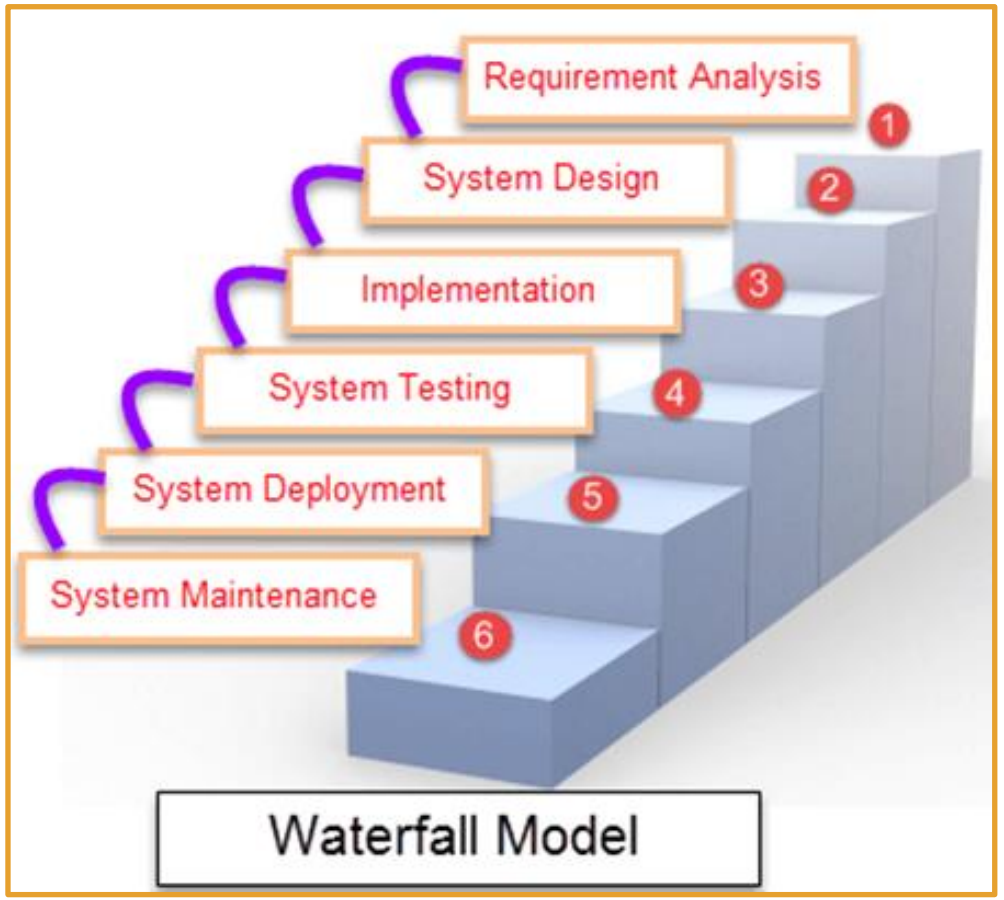
https://en.wikipedia.org/wiki/Waterfall_model

The ***Waterfall Model*** breaks down project activities into phases. Each depends on the deliverables of the previous phase.

Progress flows downward through each phase only when the preceding phase is verified to be complete.

Waterfall is less iterative and less flexible than other models and has high accountability and documentation requirements.

The ***Waterfall Model*** is too rigid for most situations.



SDLC – Spiral Model

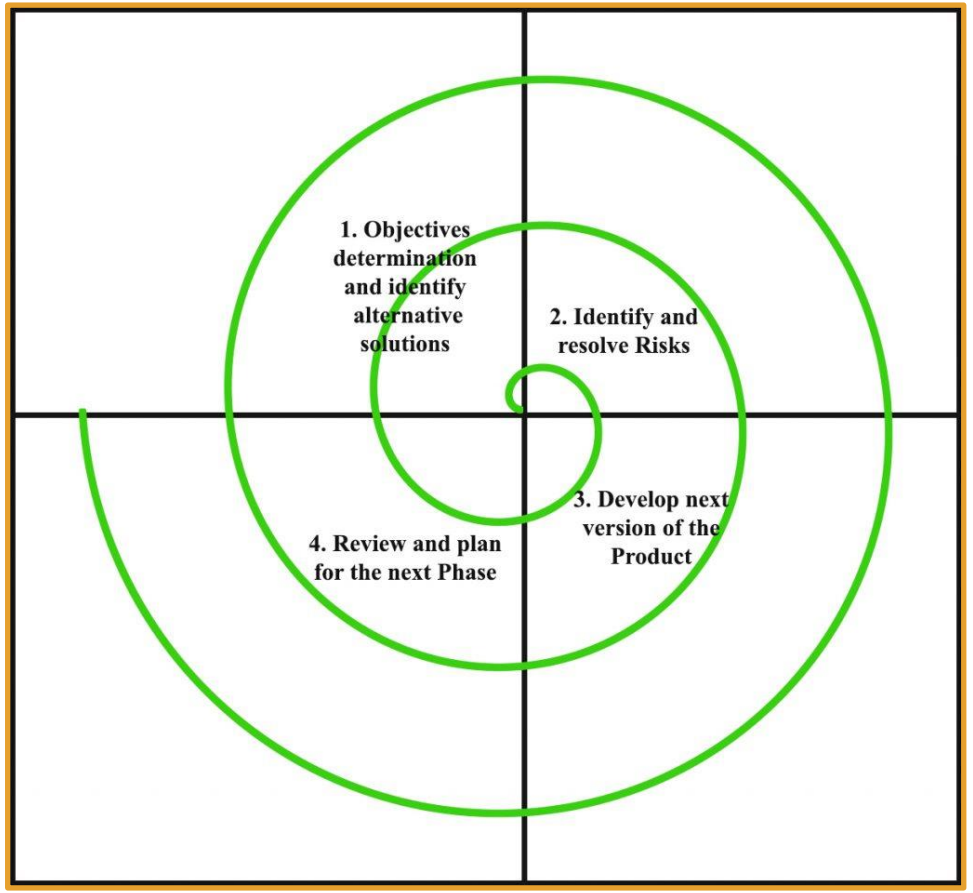
https://en.wikipedia.org/wiki/Spiral_model#The_six_invariants_of_spiral_model
https://www.geeksforgeeks.org/software-engineering-spiral-model/

The ***Spiral Model*** bases its processes on the unique risk patterns of the project. This approach guides a team to adopt elements of other models while implementing the ***Spiral Model*** design pattern.

Spiral often results in a series of mini-Waterfalls. A single loop of the ***Spiral*** represents one full ***Waterfall Model*** cycle.

The exact number of loops of the ***Spiral*** depends on the requirements of the project

Each loop of the ***Spiral*** is called a “phase” and each phase is divided into quadrants.



SDLC – Spiral Model – Pros and Cons

https://en.wikipedia.org/wiki/Spiral_model#The_six_invariants_of_spiral_model
https://www.geeksforgeeks.org/software-engineering-spiral-model/

Advantages	Disadvantages
Risk Handling. You can deal with new requirements/fixes during the next cycle.	Complexity
Good for large and complex projects.	Expense
Flexibility. Each cycle adjusts requirements based on the state of the project.	Requires expertise to determine the risk factors before each cycle.
Customer Satisfaction. Clients can see the results of each cycle immediately.	An unknown number of cycles in the completed project.

SDLC – Iterative Model

https://en.wikipedia.org/wiki/Iterative_and_incremental_development

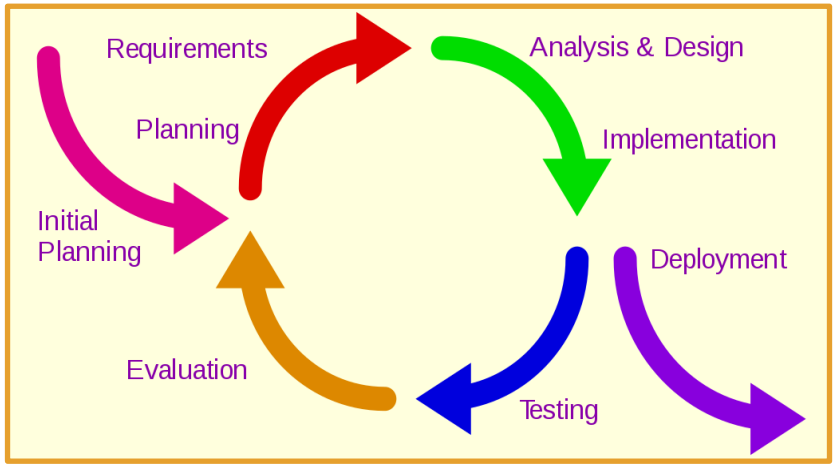
The ***Iterative Model*** develops a project through small, iterative development cycles.

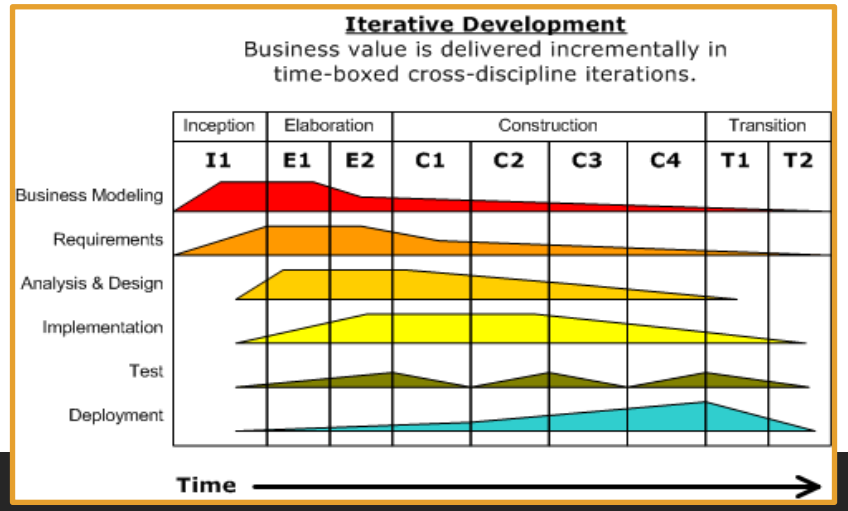
This allows developers to learn from earlier versions of the project. Each iteration adds features until the full application is implemented.

A ***Project Control List*** is updated after each iteration to focus developers on the specific functionality to implement in that iteration.

Analysis of each iteration is based on client and user feedback. The next round of changes are implemented in the next iteration.

Unlike with the ***Waterfall Model***, the ***Iterative Model*** allows backtracking.





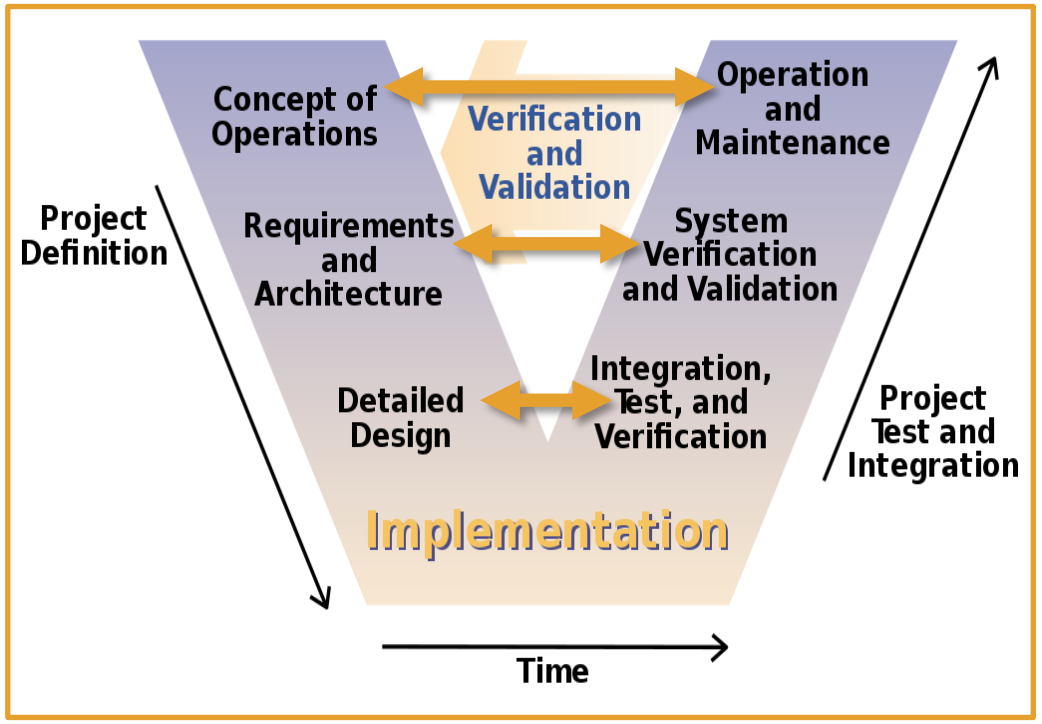
SDLC – V Model

https://en.wikipedia.org/wiki/V-Model_(software_development)

In the ***V Model***, the process steps flow downward during the development phase and back upwards during the testing phase in the shape of a 'V'.

The relationships between each SDLC phase and testing phase is represented by the horizontal pairs of actions.

The horizontal and vertical axes represent time, project completeness, and level of abstraction.



SDLC - Big Bang Model

https://www.tutorialspoint.com/sdlc/sdlc_bigbang_model.htm

The ***Big Bang Model*** does not follow any specific process. Requirements are implemented without much analysis.

Development starts as soon as the required funds are acquired and developers are ready.

The ***Big Bang Model*** is meant for small projects with small teams. It's higher risk than other models with misunderstandings and failure a likely outcome.

For small projects, the ***Big Bang Model*** offers easy management, rapid prototyping, more flexibility, and requires few resources.

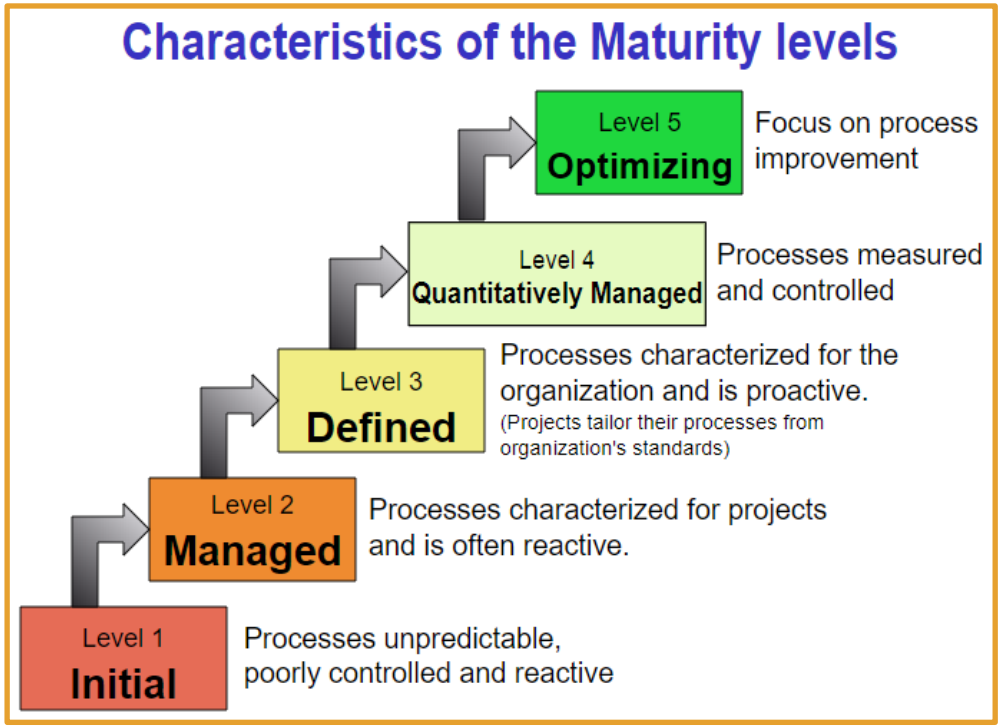
Agile – CMMI (Capability Maturity Model Integration)

https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration
http://dthomas-software.co.uk/resources/frequently-asked-questions/what-is-cmmi-2/

CMMI is a process level improvement, training, and appraisal program. Its main sponsors originally included the Office of the Secretary of Defense (OSD) and the National Defense Industrial Association.

CMMI was developed by Carnegie Mellon University and ***The CMMI Project***. The aim was to improve existing software development processes.

The ***CMMI*** principal is “The quality of a system or product is highly influenced by the process used to develop and maintain it.”. ***CMMI*** can be used to guide process improvement across a project, a division, or an entire organization.



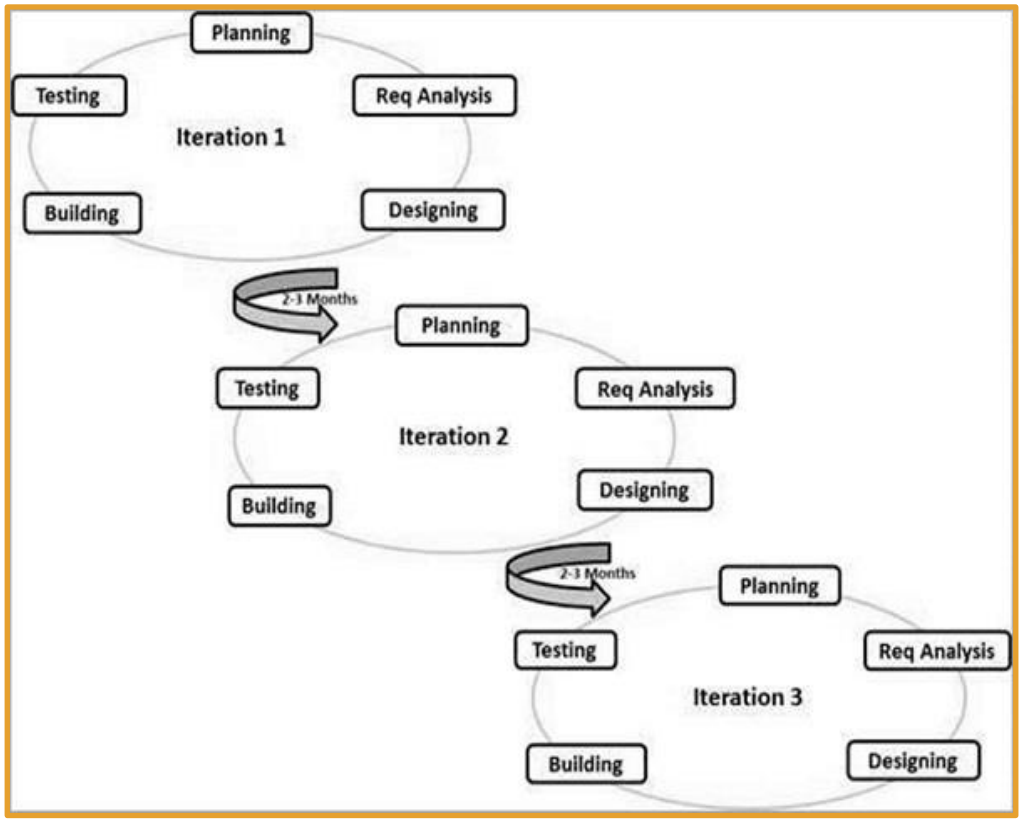
Agile – Overview

https://agilemanifesto.org/principles.html

There are multiple ***Agile*** styles. Together, they are referred to as ***Agile Methodologies***.

In ***Agile:***

- Interactions are prioritized over processes.
- Working software is more important than extensive documentation.
- Collaboration with the client is prioritized above negotiation.
- Quick response to new requirements is the most important aspect of the whole process.

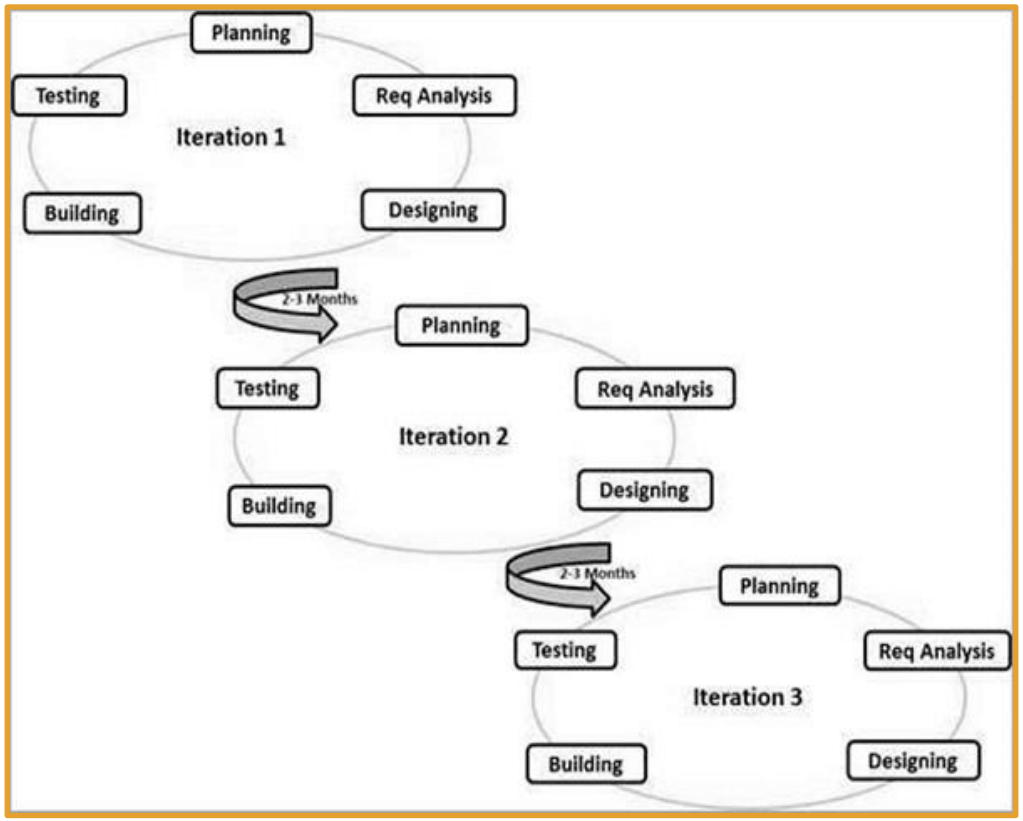


Agile – Overview

https://agilemanifesto.org/principles.html

Agile Priorities:

- Early and continuous delivery (CI/CD).
- Quick acceptance of and adjustment to changing requirements.
- Quick delivery of a working product.
- Sustainable development process.
- Regularly scheduled sessions of introspection (Daily Stand-up's)
- Constant adjustment to a dynamic development environment.



Agile - Characteristics

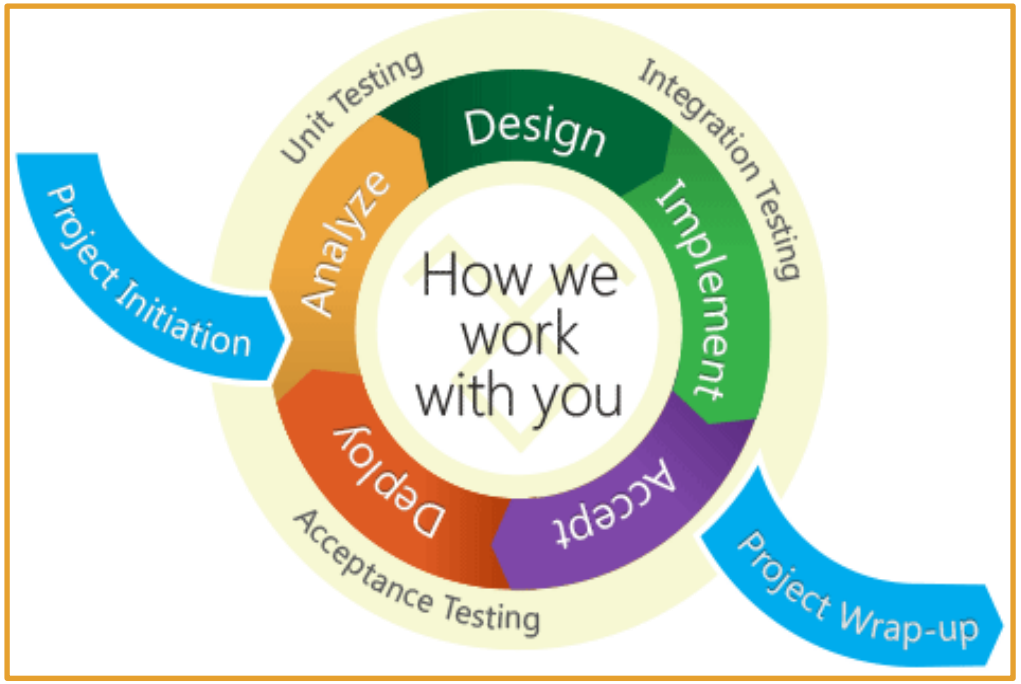
https://en.wikipedia.org/wiki/Agile_software_development
https://www.agilealliance.org/glossary/daily-meeting
https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm

The ***Agile*** Process is a series of ***Sprints*** where each team has a specific ***User Story***. A ***Sprint*** has a predetermined start and stop date and lasts 2-4 weeks. After each ***Sprint***, teams demonstrate what they've accomplished to each other and to the client. Feedback is then taken to inform the next ***Sprint***. A ***Sprint*** also includes short, daily "Standup" meetings.

Daily Standup – Each member reports:

1. What I did yesterday.
2. What I intend to do today.
3. What "blockers" I have.

Information Radiator (Kanban) - A board presenting an up-to-date summary of the status of the product.
Continuous Integration – The practice of merging all developers' working copies to a shared master. This is done several times a day.
Test Driven Development – Relies on the repetition of a very short development cycle: requirements are turned into very specific test cases. Then the code is improved until the tests pass.



Agile – Scrum

https://www.agilealliance.org/glossary/scrum
https://www.scrum.org/resources/what-is-scrum

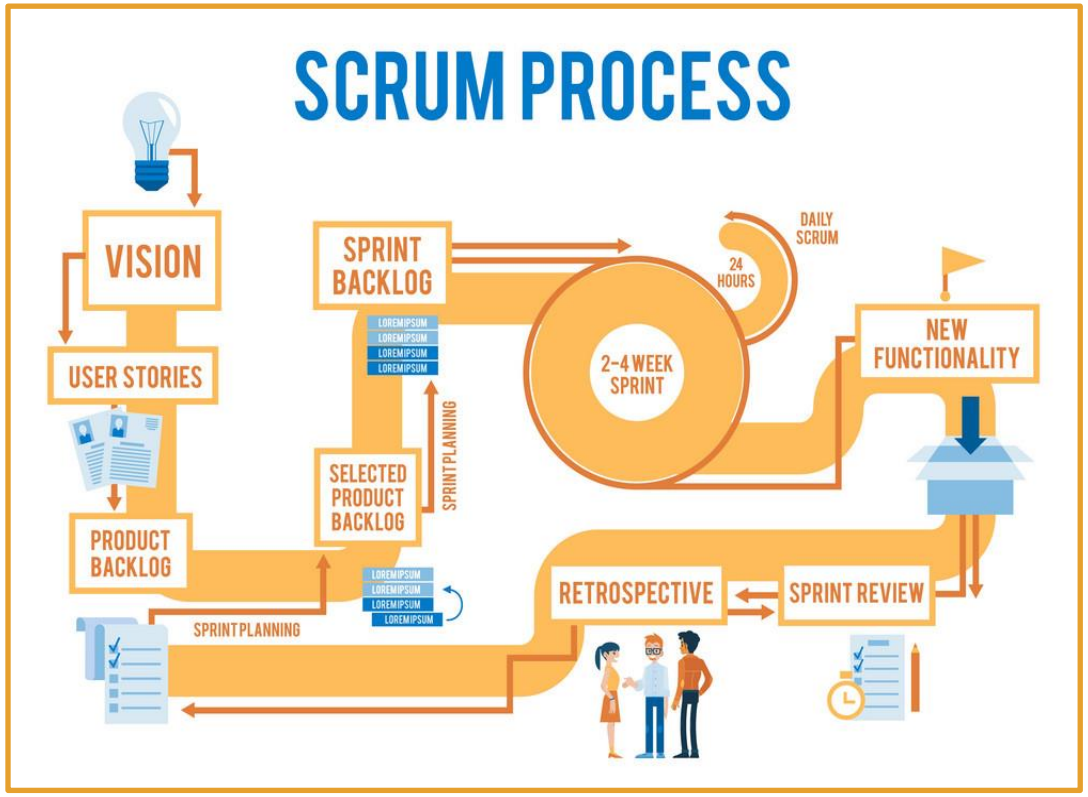
Scrum is a framework that facilitates addressing complex problems while delivering products of the highest possible value.

In a *Scrum*, teams work iteratively to:

1. hypothesize about how the final project should work,
2. try to implement the idea,
3. reflect on the experience,
4. Make necessary adjustments,
5. repeat the cycle.

Scrum is best used when the amount of work required can be split in to more than one *Sprint*.

Daily *Stand-ups* allow each person to transparently share the difficulties (*Blockers*) they are having. Adjustments are made to focus the following days efforts, based on the *Stand-up*.



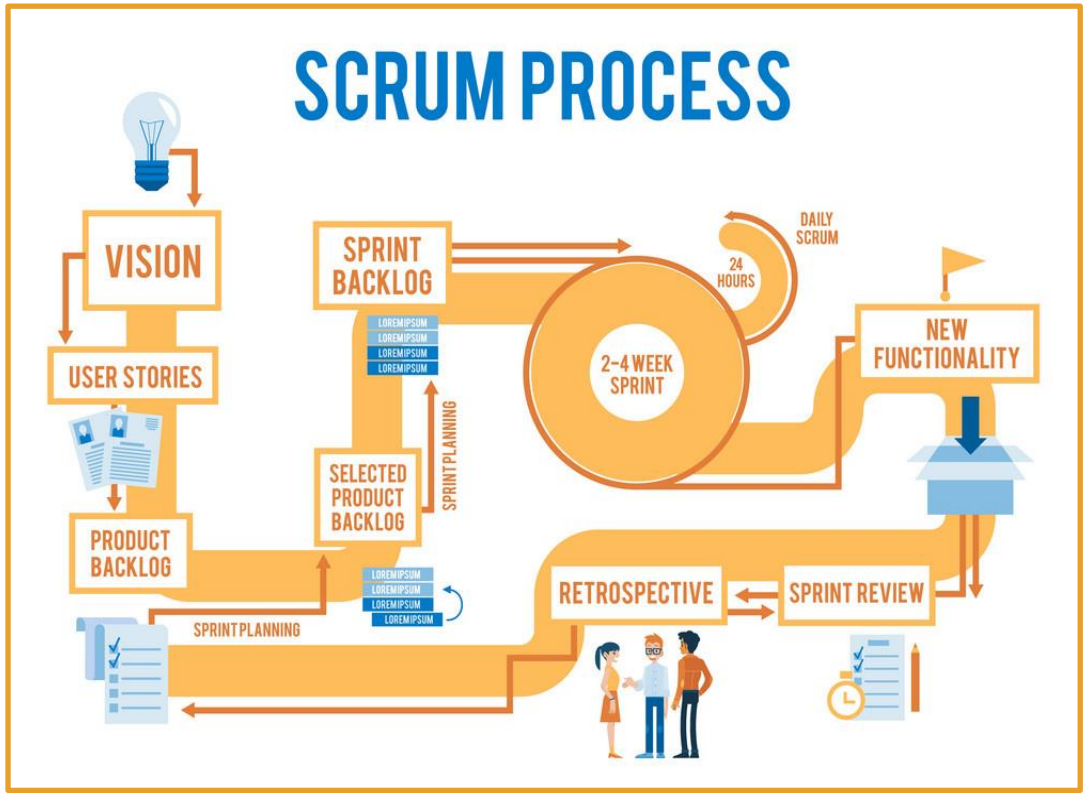
Agile – Scrum Details

https://www.agilealliance.org/glossary/scrum

A ***Scrum*** starts with a ***Sprint Planning Meeting*** with the client. In a ***Sprint Planning Meeting***, ***User Stories*** that form the ***Sprint Backlog*** are created. ***Backlog*** items are added to the ***Scrum Board (Kanban)***.

Teams then decide how they will successfully deliver a working ***iteration*** at the end of the first ***Sprint***. The ***Sprint Backlog*** is then locked and no more ***User Stories*** can be added.

During the ***Sprint***, the ***Scrum Master*** conducts a short (15-minute) ***Daily Standup*** in which members discuss ***blockers*** and coordinate efforts for the day.

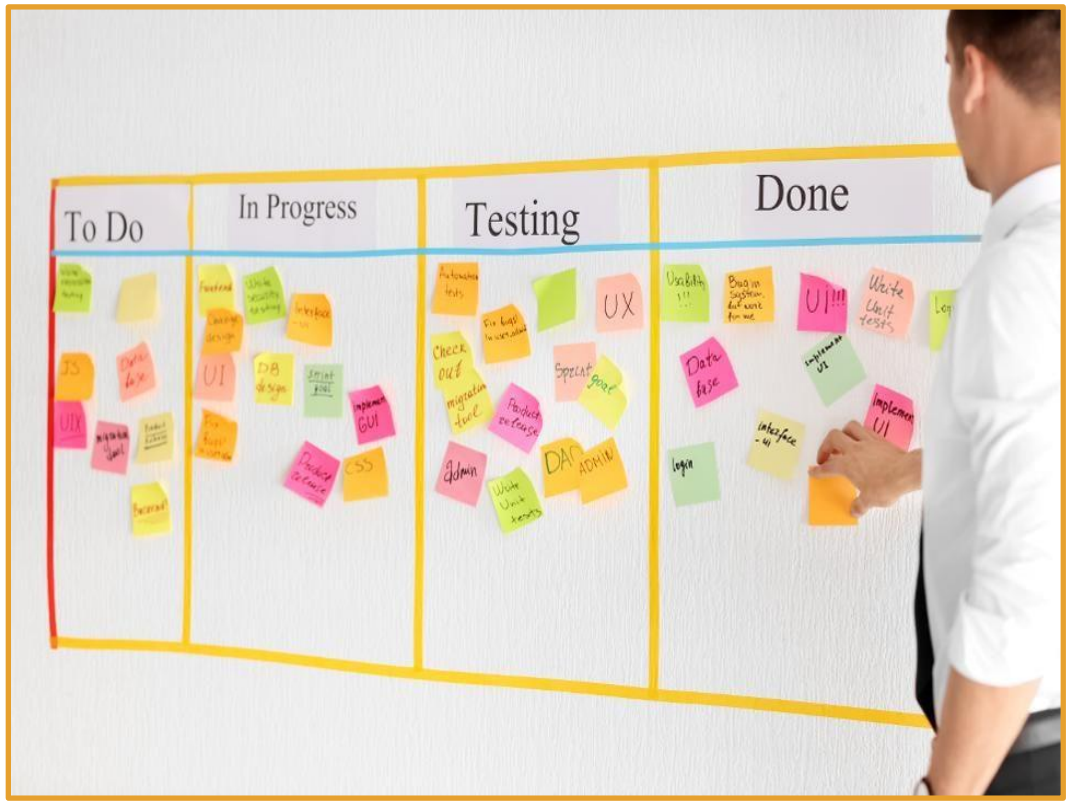


Agile – Kanban

https://www.atlassian.com/agile/kanban/wip-limits
https://www.agilealliance.org/glossary/kanban

The ***Kanban Method*** gets its name from the use of "***Kanban"***, a Japanese word for visual signaling mechanisms representing work-in-progress.

A general term for systems using the ***Kanban Method*** is "***flow***". ***Flow*** shows that work ***flows*** continuously through the system instead of being constrained to distinct timeboxes.

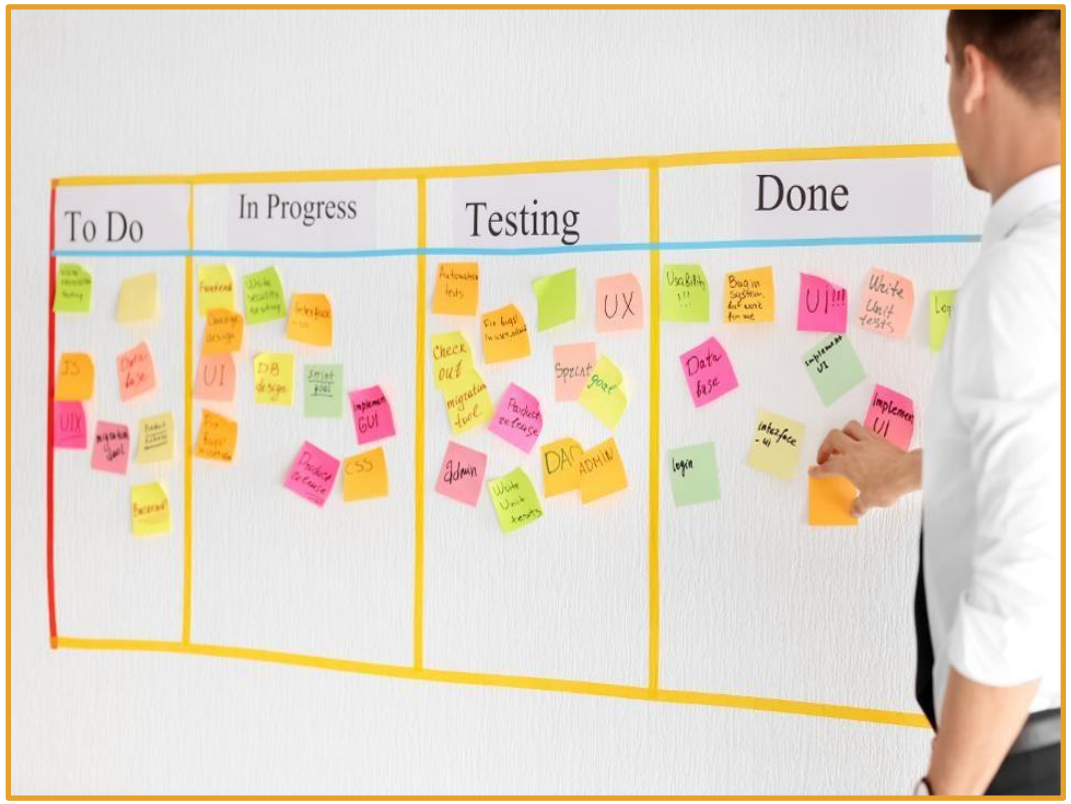


Agile – Kanban Board

https://www.atlassian.com/agile/kanban/wip-limits
https://www.agilealliance.org/glossary/kanban

Kanban systems use a ***Kanban Board*** to visualize work and the process it goes through.
In the ***Kanban*** system,

- programmers are aided in maintaining a narrow focus on tasks.
- Bottlenecks and blockers are identified and addressed.
- Feedback loops are used to make small changes to increase efficiency.
- Balance of effort across services and response to risks and inefficiencies is emphasized.
- Kanban starts with the process as it currently exists and applies continuous and incremental improvement.



Agile – Sprint

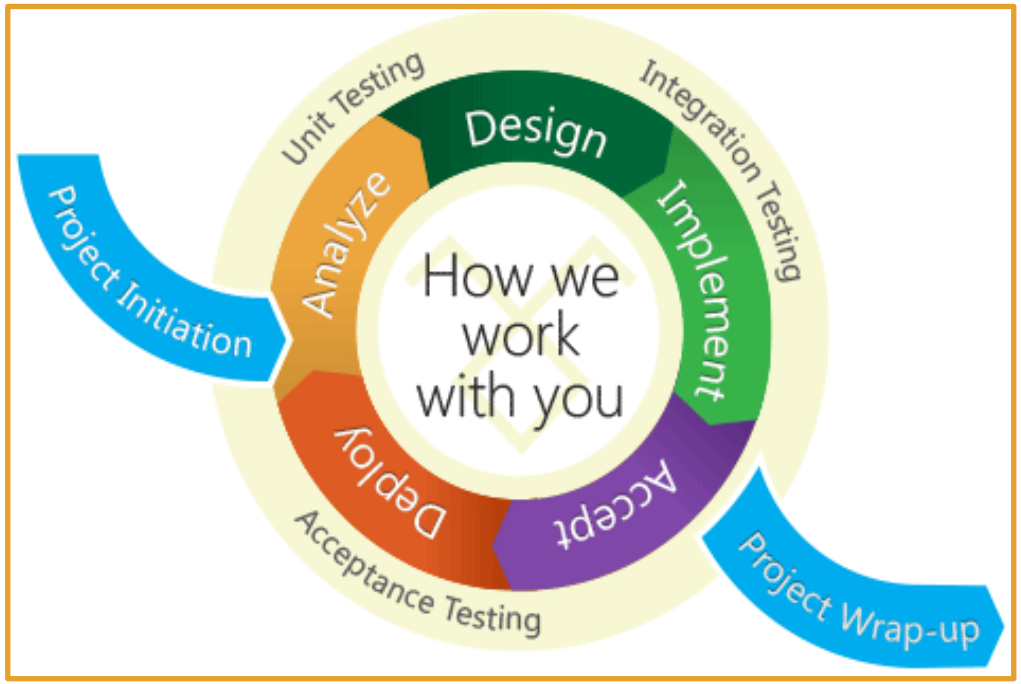
https://www.agilealliance.org/glossary/user-story-template/

A ***Sprint*** is a designated period of time in which teams work to complete a ***User Story***. All ***User Stories*** for a project form what's called the ***Backlog***. The Backlog is a record of all the ***User Stories***.
Each ***User Story*** is assigned ***points*** representing the expected effort needed to complete it.

After the ***Sprint***, Team members add up effort ***points*** for ***User Stories*** completed during the ***Sprint***.

Points completed represent the ***Velocity***.

Knowing their ***Velocity***, the team can estimate how many ***Sprints*** will be required to complete the project. Teams can use their ***Velocity*** to modulate how much work they take on in each ***Sprint***.



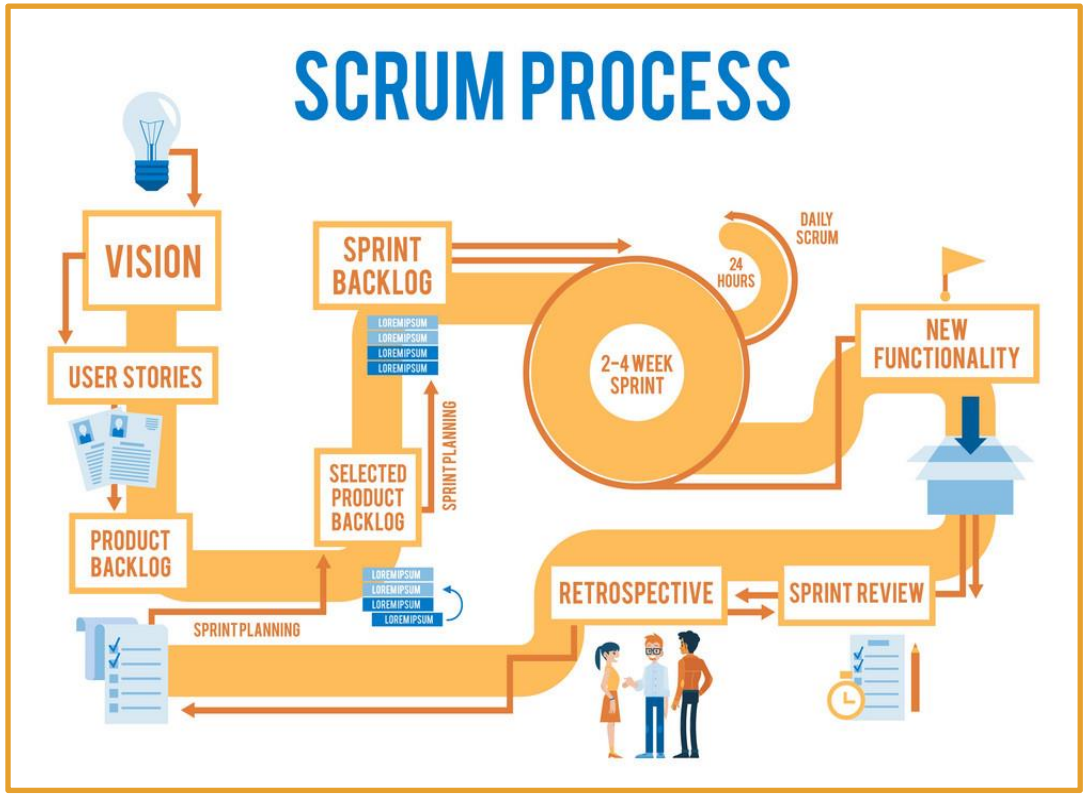
Agile – Sprint Review

https://www.agilealliance.org/glossary/scrum

Sprint Review - After the ***Sprint***, teams, client, and stakeholders meet to:

- demonstrate the in-progress product,
- review the results of the ***Sprint***,
- place feedback into the ***Product Backlog*** for future iterations.

Sprint Retrospective – After the ***Sprint Review***, the team meets to make adjustments going forward and decide on which ***User Stories*** will be included in the next ***Sprint***.



Agile – Extreme Programming

https://en.wikipedia.org/wiki/Extreme_programming
https://www.agilealliance.org/glossary
http://www.extremeprogramming.org/when.html

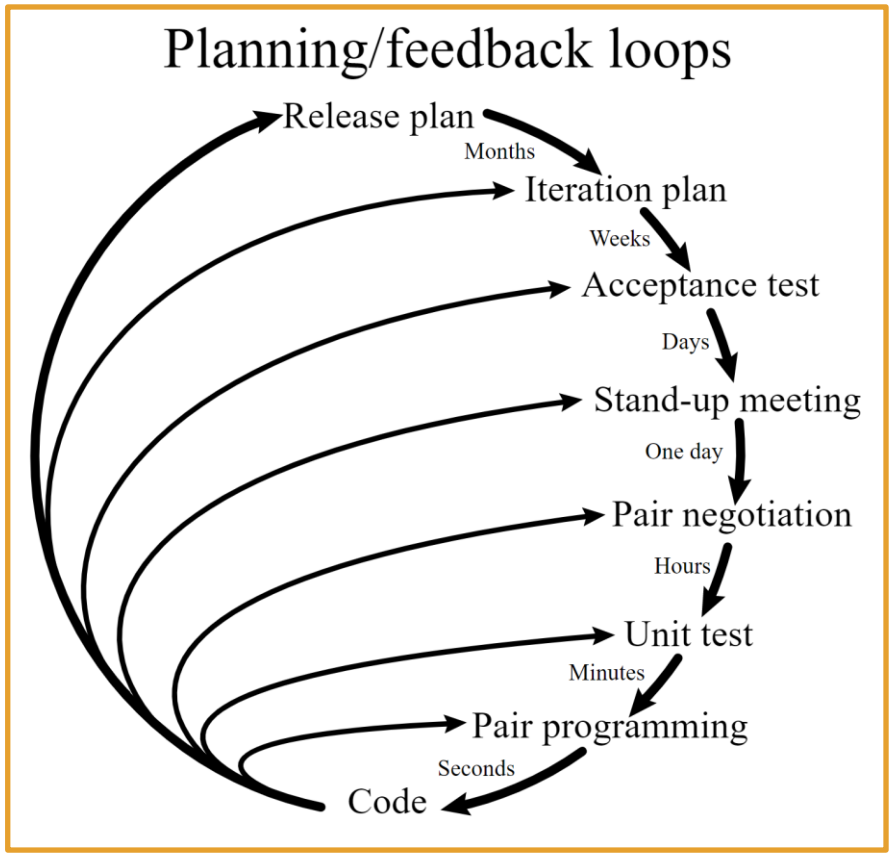
Extreme Programming (XP) is an ***Agile*** software development framework that aims to produce higher quality software, and higher quality of life for the development team in programming environments where the requirements constantly change.

XP is the most specific of the ***Agile*** frameworks regarding appropriate engineering practices for software development.

XP is set up for small groups of between 2 and 12 programmers including developers, managers, and customers.

Automated unit and functional tests must be created for the project using TDD.

XP advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

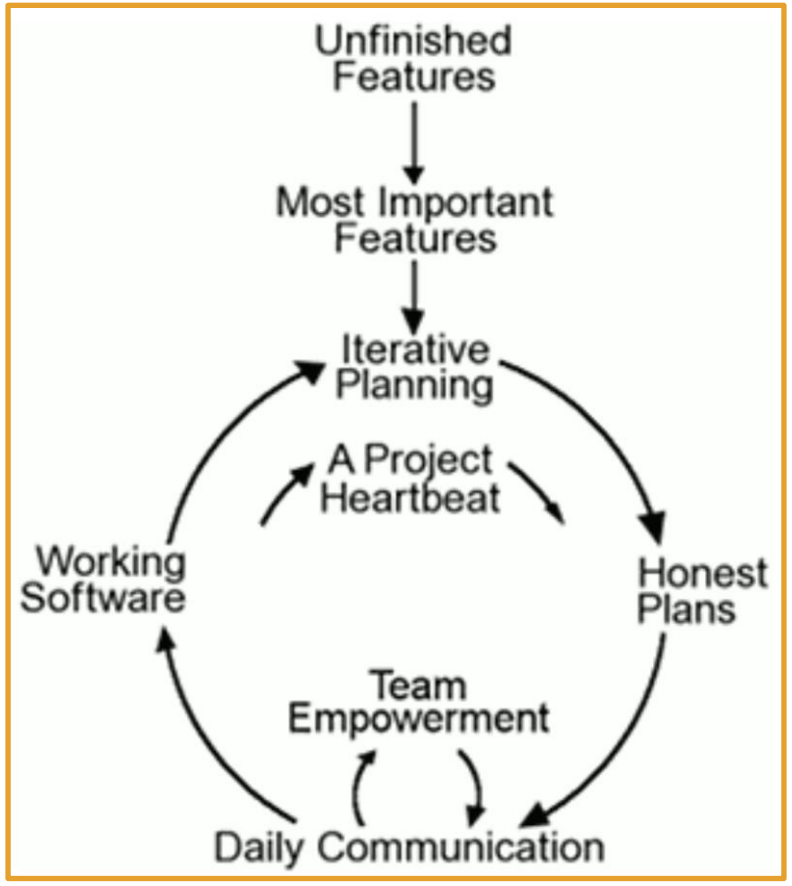


Agile – EP Best Practices (1/2)

https://en.wikipedia.org/wiki/Extreme_programming
https://www.agilealliance.org/glossary
http://www.extremeprogramming.org/when.html

The defining practices of XP are:

- ***Sit Together*** – The team sits together in the same place to better allow efficient, effective communication
- ***Whole Team*** – Client and all people who play any part in the development work together daily.
- ***Informative Workspace*** – The workspace is set up to allow face-to-face communication. Information Radiators are used to clearly and quickly communicate information.
- ***Energized Work*** - Make sure SE's are able physically and mentally focus on the work.
- ***Pair Programming*** – This provides a continuous code review as one SE codes while another monitors and provides input.
- ***Stories*** - Short descriptions of things users want to be able to do with the product and that can be used for planning.



Agile – EP Best Practices (2/2)

https://en.wikipedia.org/wiki/Extreme_programming
https://www.agilealliance.org/glossary
http://www.extremeprogramming.org/when.html

The defining practices of XP are:

- *Weekly Cycle* – An Iteration that starts with a Scrum Meeting where the client picks stories to focus on during that week and SE's decide how to accomplish them. It ends with unit tested functionality.
- *Quarterly Cycle* – A release of the product. The purpose is to keep the detailed work of each weekly cycle in context of the overall project.
- *Slack* – Include some stories in the weekly cycle that can be postponed if unexpected delays arise.
- *10-minute Build* – Keep the overall build time under 10 minutes. This promotes more frequent build triggers.
- *Continuous Integration* – Immediately include tested code changes when they are merged to the primary code.
- *Test-Driven Development* – TDD is required so that bugs are found and fixed earlier.
- *Incremental Design* – Create the framework of the project first, then dive into more detailed work later.