Information System Design

Section 1: Software Development

What is Agile?

https://www.youtube.com/watch?v=AsFMHnSfI2I

Agile Manifesto

Values

1. Individuals and interactions over processes and tools

2. Working software over comprehensive documentation

3. Customer collaboration over contract negotiation

4. Responding to change over following a plan

12 Principles

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

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

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

Business people and developers must work together daily throughout the project

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

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

Working software is the primary measure of progress

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

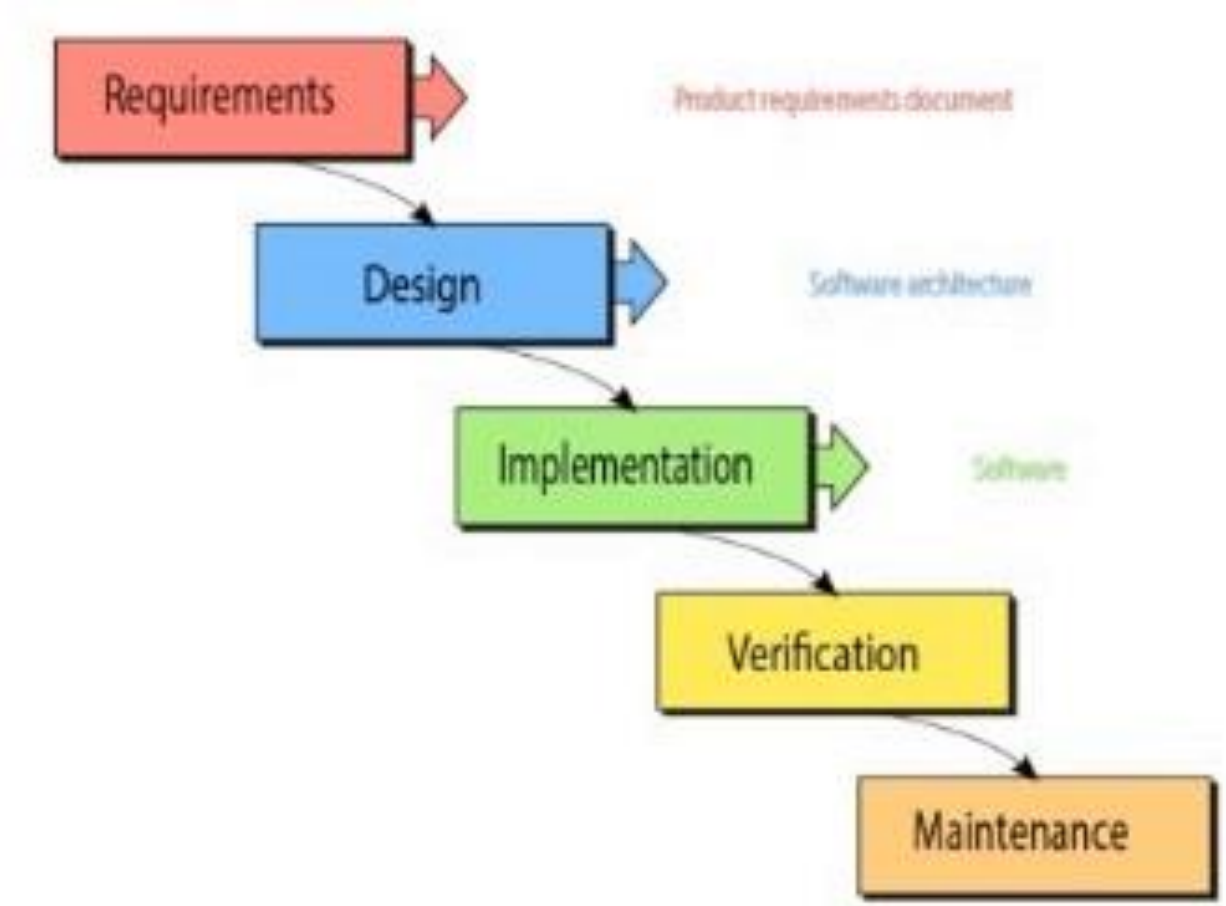
Continuous attention to technical excellence and good design enhances agility

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

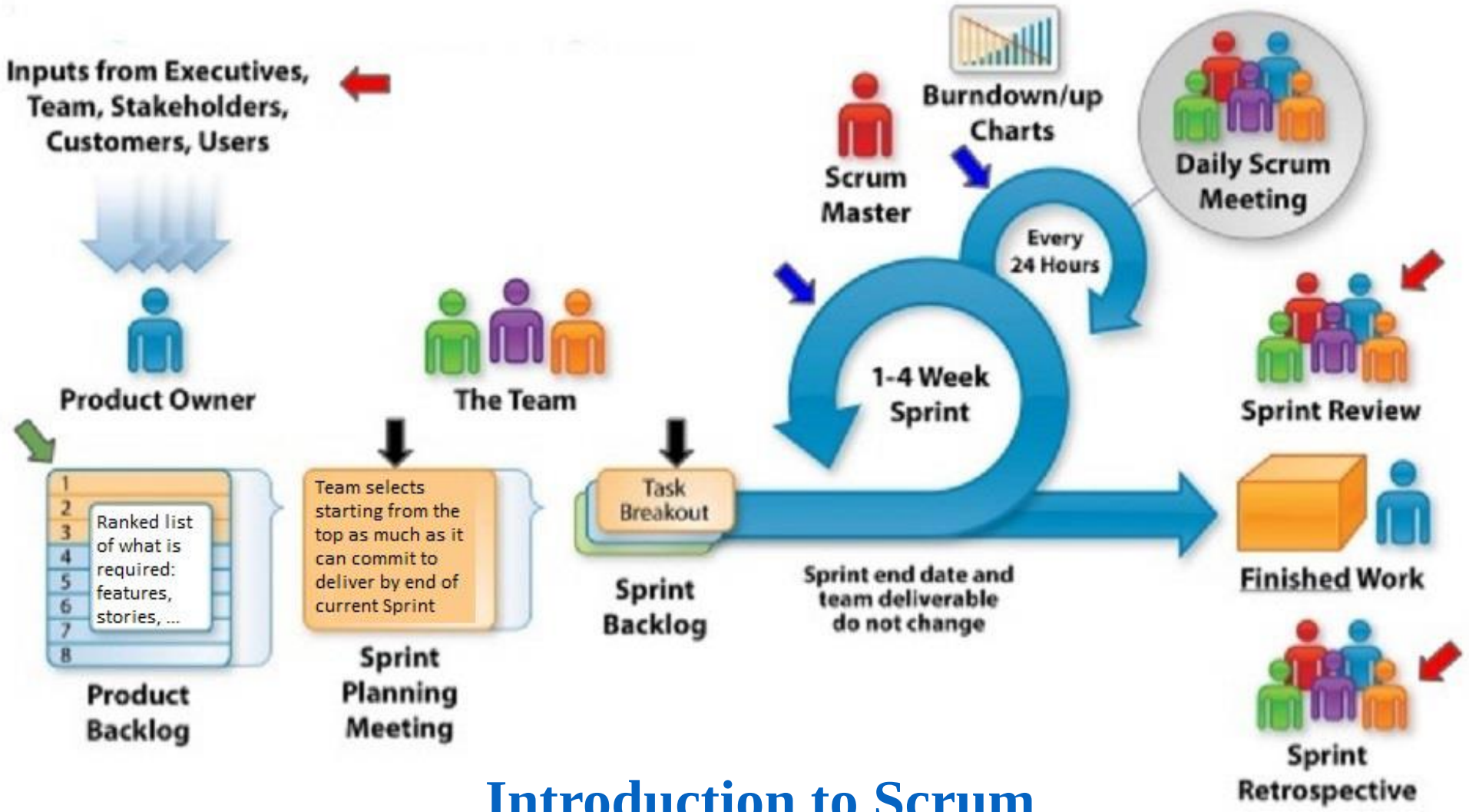
The best architectures, requirements, and designs emerge from self-organizing teams

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

Waterfall model



Scrum Framework



Introduction to Scrum

https://www.youtube.com/watch?v=9TycLR0TqFA

Scrum Framework

3 Roles

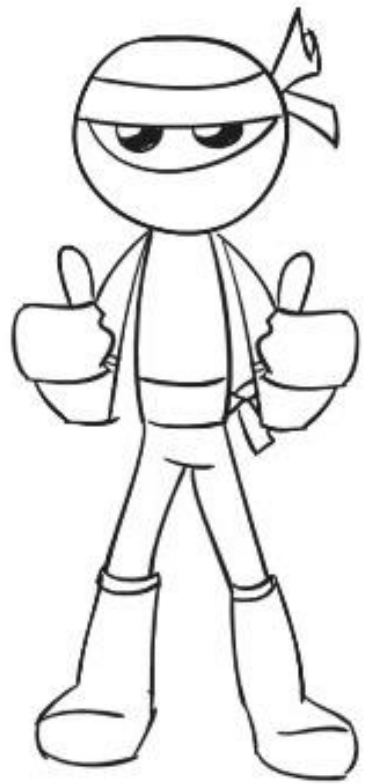
3 Artifacts

3 Ceremonies

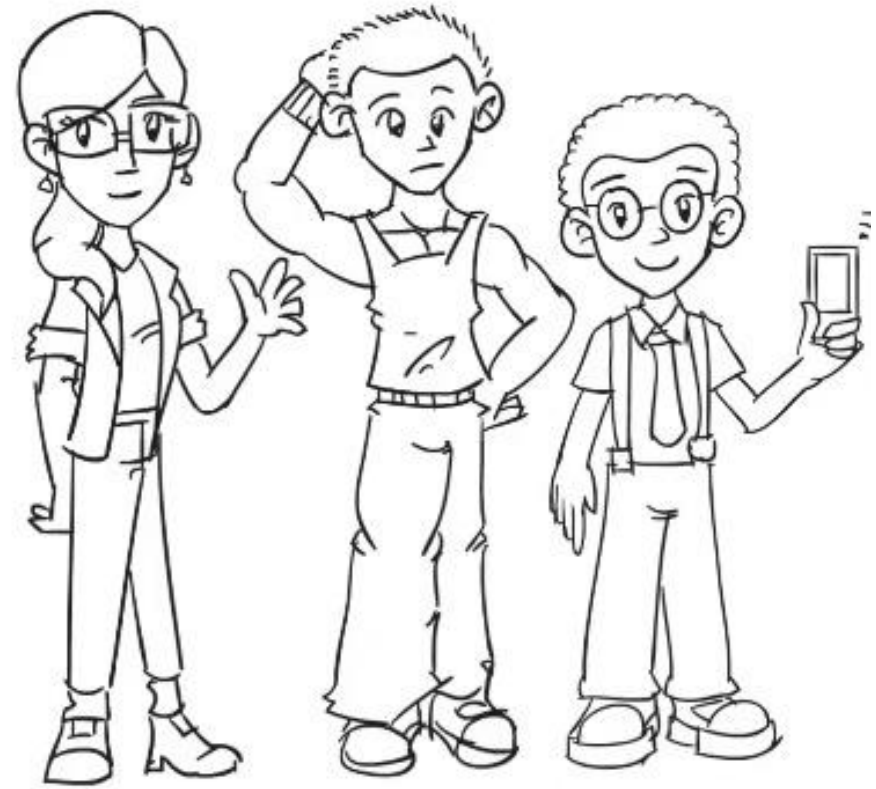
Scrum Framework

3 Roles

Product Owner



Scrum Master



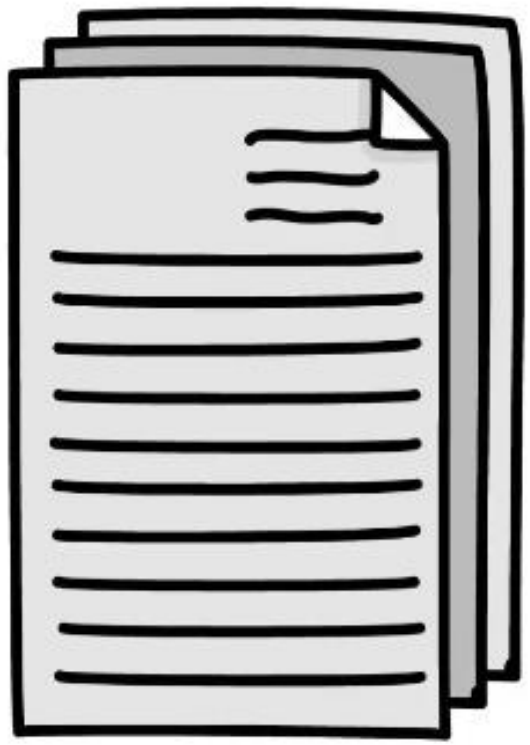
Team

Scrum Framework

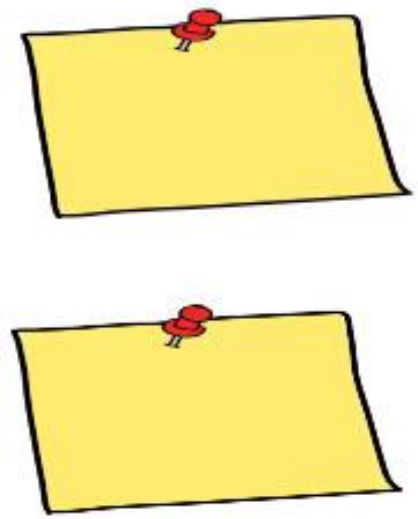
Scrum Framework

3 Artifacts

Product Backlog

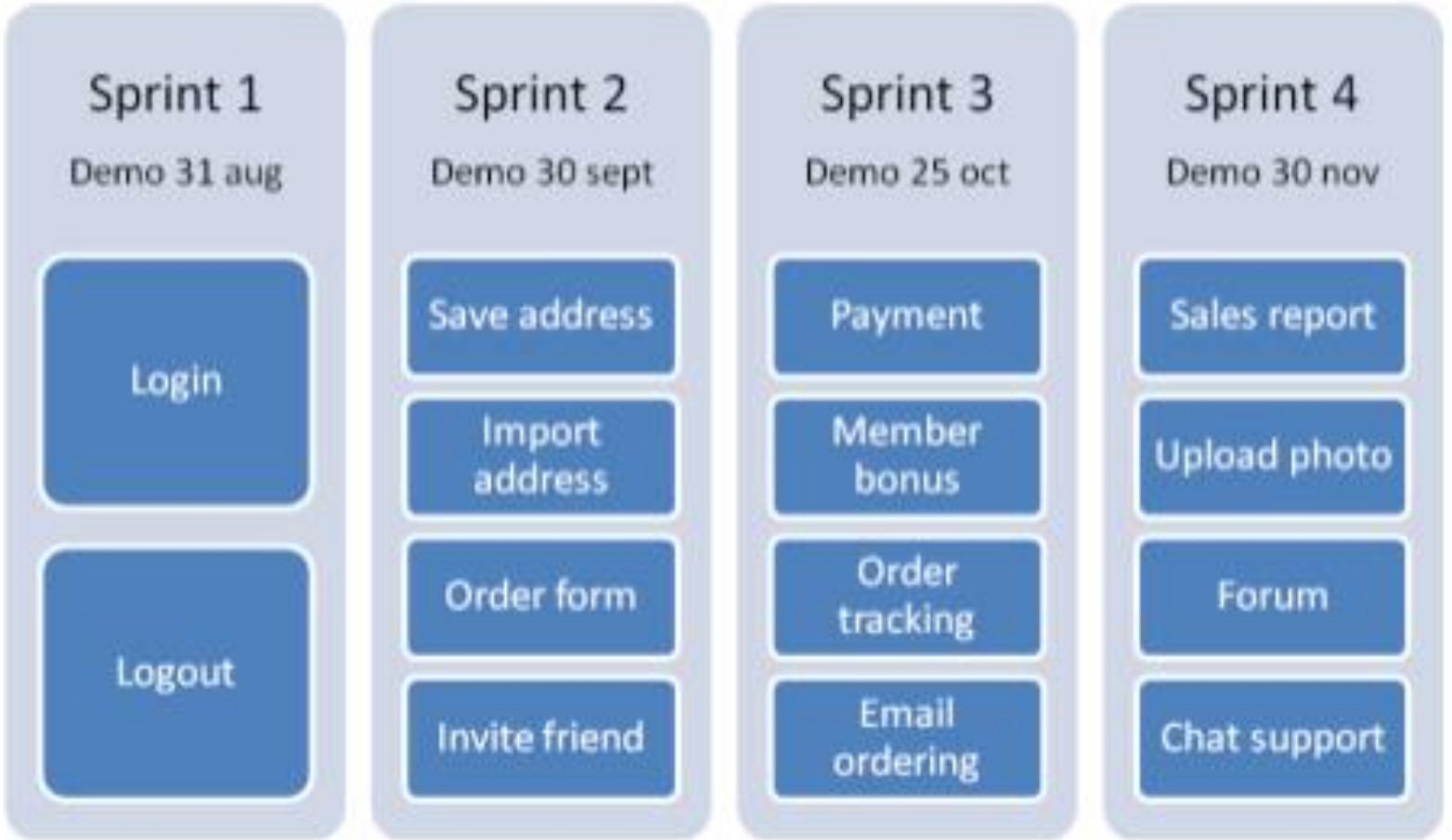


Sprint Backlog



Burndown Chart

Scrum Framework



Scrum Framework

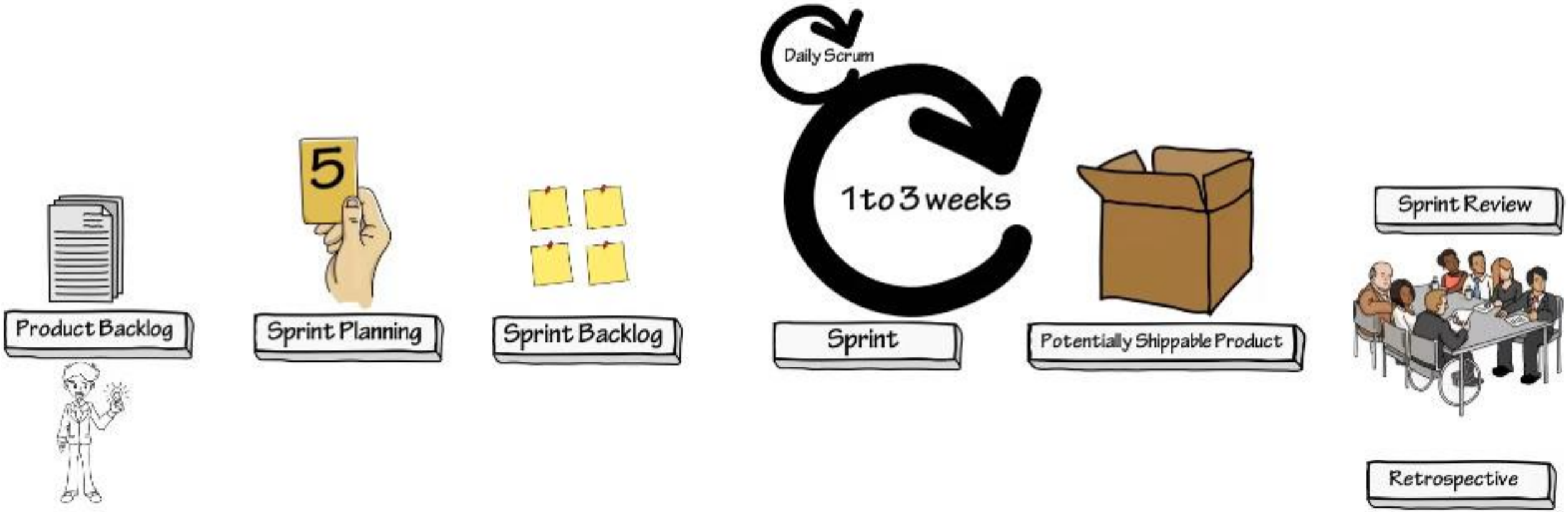
3 Ceremonies

Sprint Planning

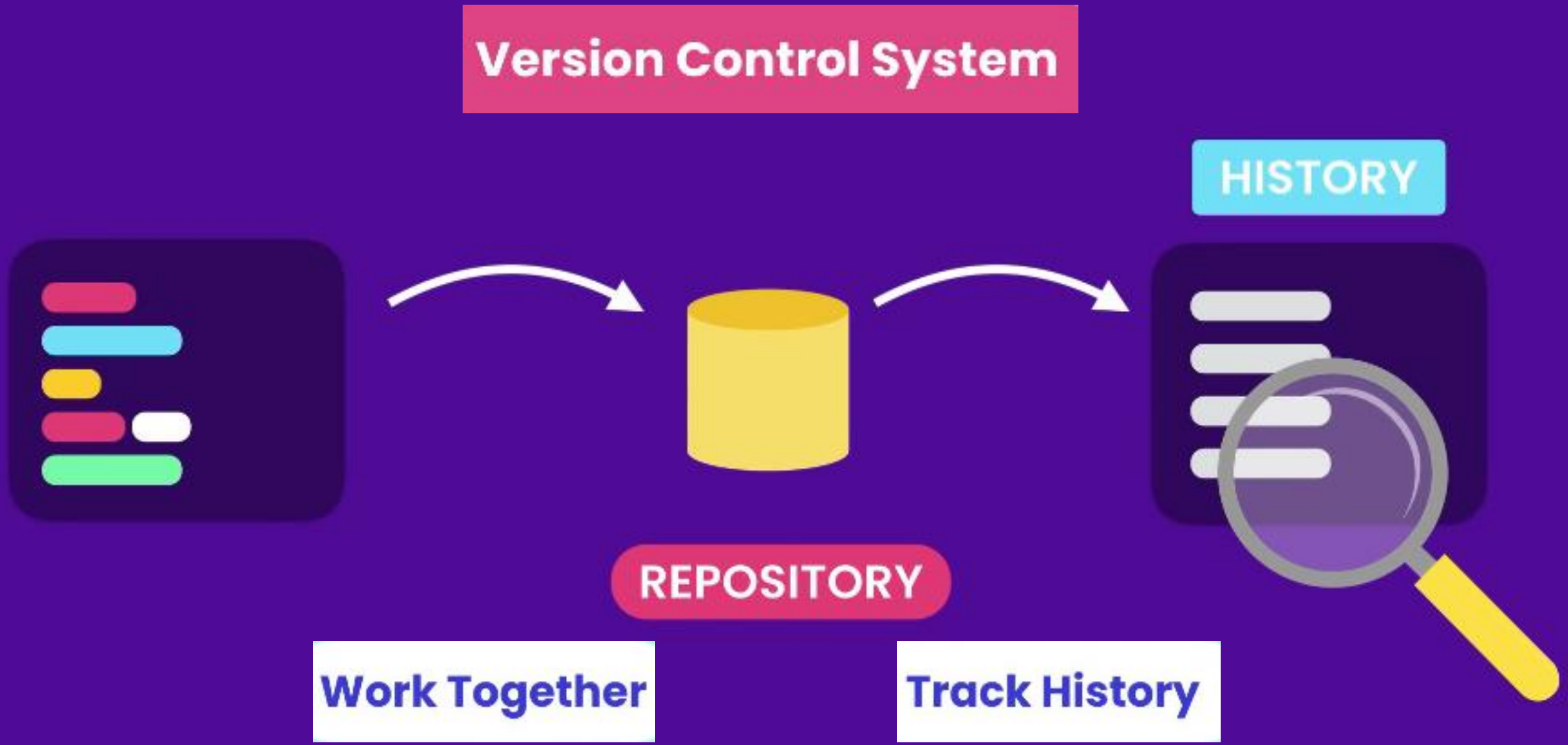
Daily Scrum

Sprint Review

Scrum Framework

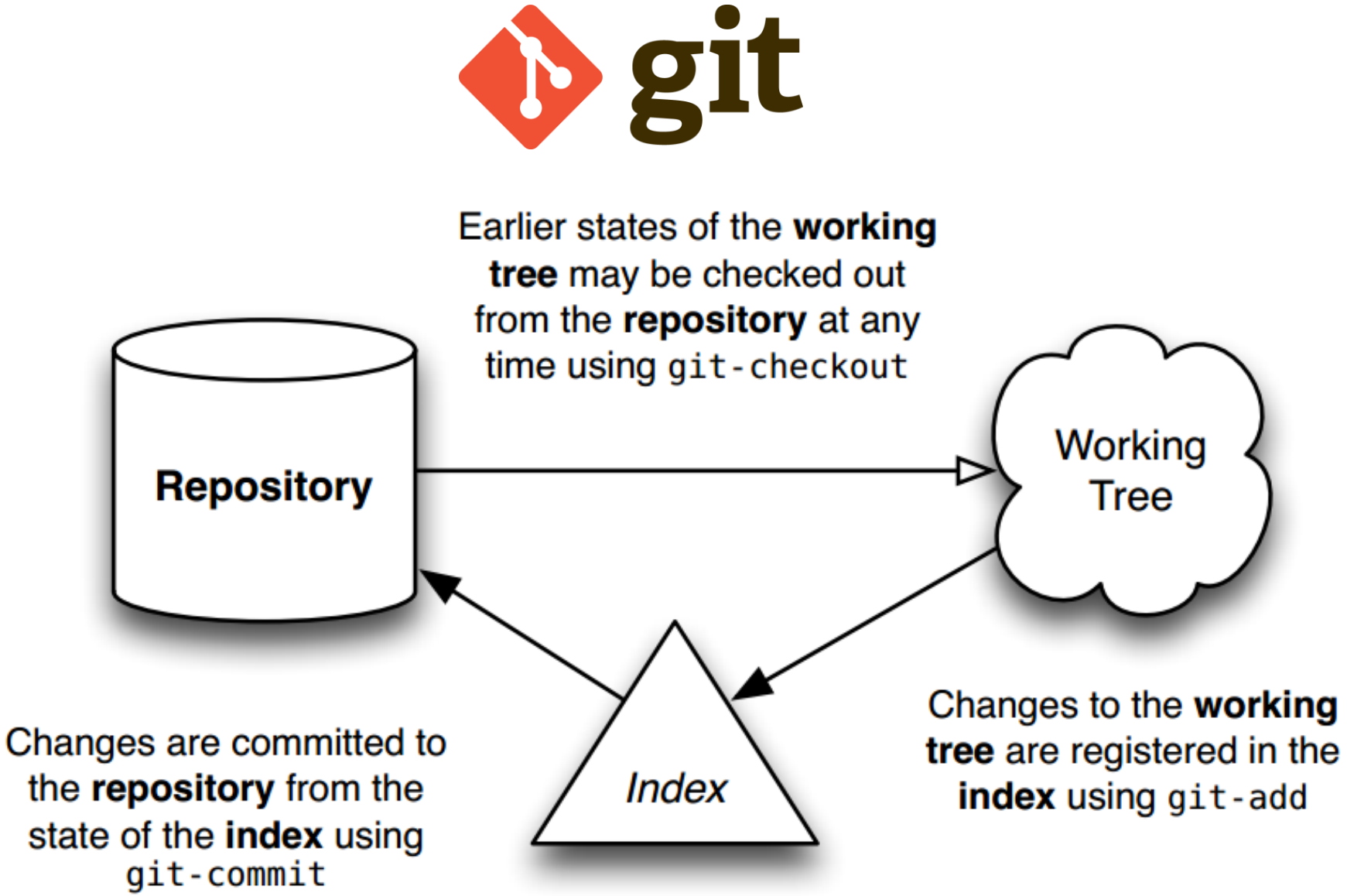


Repeat this workflow for each sprint



Version control system

https://www.youtube.com/watch?v=2ReR1YJrNOM



Learn Git In 15 Minutes

https://www.youtube.com/watch?v=USjZcfj8yxE

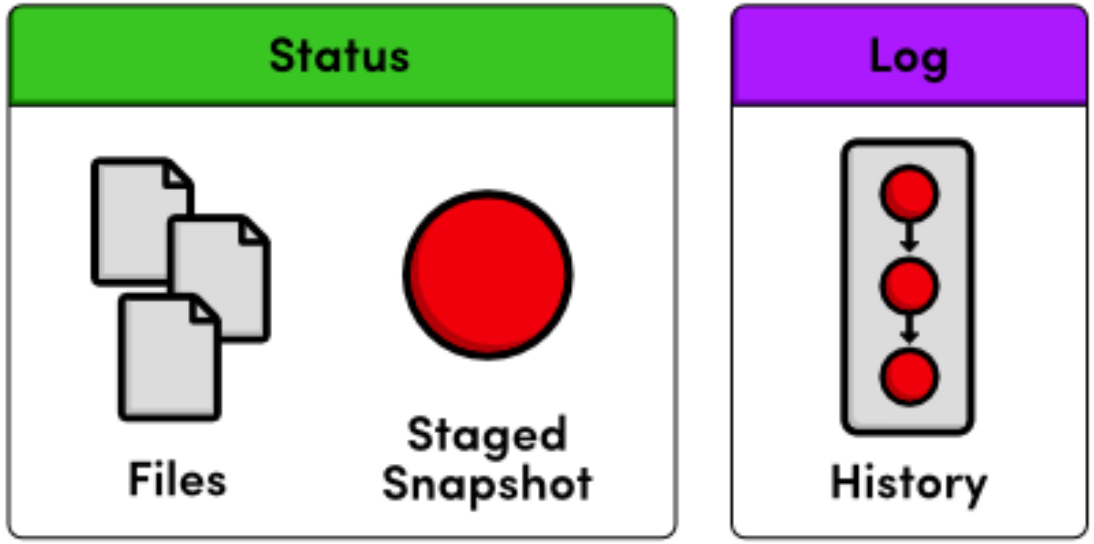
- **git config --global user.name "<name>“**
  - Define the author name to be used in all repositories.
- **git config --global user.email "<email>“**
  - Define the author email to be used in all repositories.

- **git init**
  - Create a Git repository in the current folder.

- **git add <file>**
  - Add a specific file to the staging area
- **git add .**
  - Adds all the files and folders inside the project folder to the staging area
- **git commit -m "Commit message“**
  - Commit the files from the staging area to the repository
    The commit message should be a descriptive summary of the changes that you are committing to the repository
- **git revert**
  - Reverts a commit. Does not delete the commit object, just applies a patch
    Reverts can themselves be reverted!

- **git status**
  - It shows us which files have been staged (tracked) and which files are untracked

- **git log**
  - View a repository's commit history.

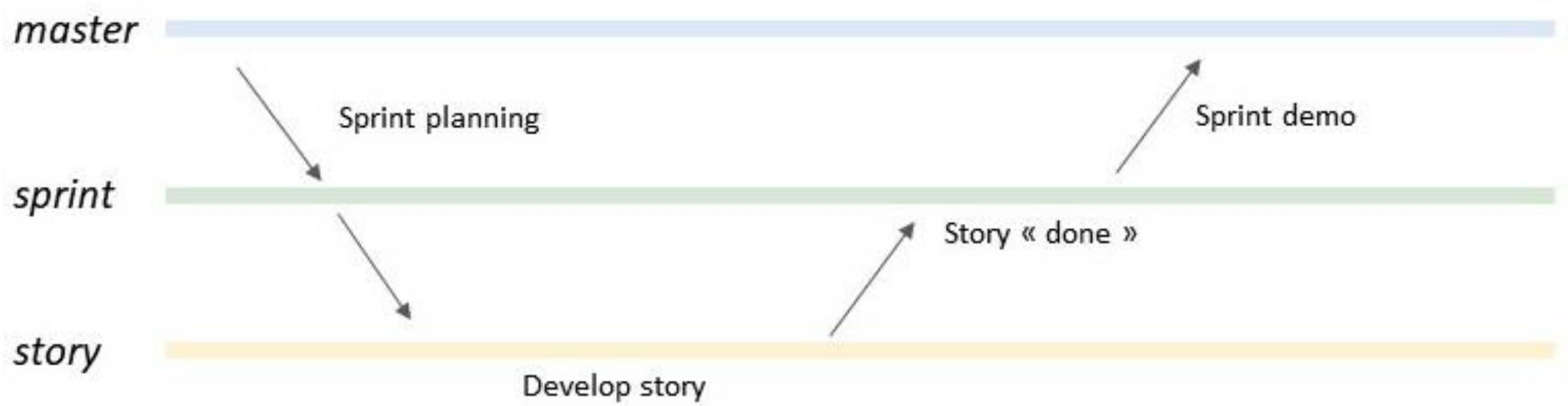


- **git branch <new-branch-name>**
  - Create a new branch (different isolated timeline of your project)
- **git branch**
  - List all the branches for your project
- **git checkout <branch-name>**
  - Switch to a different branch
    That branch could be a previous state that you committed
    (type <**commit-hash**> instead of <**branch-name**>)
- **git checkout -b <new-branch-name>**
  - Create a new branch and change to it at the same time
- **git merge <branch-name>**
  - Merge the changes from a different branch into your current branch

- **git clone**
  - Creates a local copy of a project from a remote repository

- **git push**
  - Updates the remote repository with any commits made locally

- **git pull**
  - Updates the local repository with the latest updates from the remote repository



Branching model for Scrum using Git

https://www.stenusys.com/scrum-branching-strategy/

The master branch

reflects the level of code approved by the Product Owner as being ready for the production.

The sprint branch

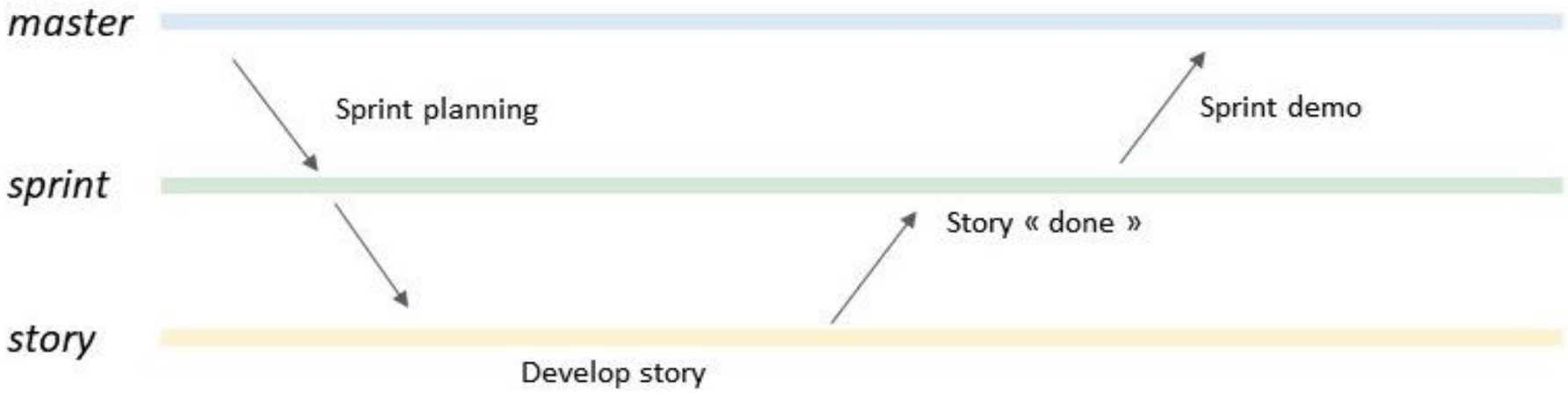
At the beginning of each new sprint, Sprint branch is created from the master branch.

This operation can be managed by the Scrum Master.

Sprint branch will be presented to the product owner during the sprint review

The story branch

A dedicated branch for each story will be created from the sprint branch, Even if it is a short-lived branch , commits that will be pushed to the story branch should be relevant, atomic and clean.



When the story finally meets the done criteria, it can then be merged in the sprint branch. All tests (unit tests and integration tests) should be rerun at this stage. This will ensure that no story broke another one.

During the sprint review, the product owner will decide whether each story passes or fails.

If all stories have passed

The sprint can be merged back to the master branch.
This operation is called the delivery.

If the Product Owner has rejected at least one story, the whole sprint is failed.
The way to deal with failed stories may vary a lot depending of the failure type and the Product Owner’s point of view

Failed stories

If the failed story can be fixed easily and quickly, the Product Owner can agree on keeping the story for the delivery. The dev team will fix the sprint branch and will rerun all tests. The Product Owner should accept the fix based on the test results or a short demo.

Otherwise, the failed story has to be removed from the delivery.
That’s the touchy case that should be handled carefully.

If the failed story has to be removed from the delivery.
Here are different kinds of approaches:

- Commit(s) of failed stories are **reverted** in the sprint branch.
  This is an easy operation if story's commits are atomic. But if this kind of operation is often done, this will lead to a dirty code base (dirty commit history).
- A **new sprint branch** is created from the original one. All story branches of approved stories are merged into this new branch. This is a tedious operation but the code history will be cleaner
- Interactively rebased upon the original sprint branch (Rewriting History). That way, only accepted stories are picked. This is the safer method, but we will lose the merge information of the story branch.
- Product Owner may decide that this is too risky and reject the product increment as a whole. The stories will then be integrated into a new sprint.

Useful Links

- Ry's Git Tutorial
  https://web.archive.org/web/20161121145226/http://rypress.com:80/tutorials/git/index
- Git Tutorial
  https://learngitbranching.js.org/
- Git interactive-rebase
  https://blog.axosoft.com/learning-git-interactive-rebase/
- Learn Github in 20 Minutes
  https://www.youtube.com/watch?v=nhNq2kIvi9s

Useful Links

- The 4 Values of Agile Manifesto Explained
  https://www.youtube.com/watch?v=gf7pBZxOCtY
- Agile Principles Explained
  https://www.youtube.com/watch?v=jSayJF0epJs
- Intro to the Scrum Framework
  https://www.youtube.com/watch?v=ZiEcq9uvi4Y
- Intro to Scrum in Under 10 Minutes
  https://www.youtube.com/watch?v=XU0llRltyFM
- Agile Product Ownership in a Nutshell
  https://www.youtube.com/watch?v=502ILHjX9EE
- Introduction to Agile And Scrum (بالعربي)
  https://www.youtube.com/watch?v=1Jd1cBn8tW4