



# Teamwork

in IT Development Projects

# About the presenters

- Pascal Furer (38y)
- Bachelor Computer Science (FHNW)
- > 15y in Software Development and IT Consulting
- COO at INGTES Development in Cluj



# About the presenters

- Mihai Ibanescu(39)
- Bachelor Computer Science (UBB)
- > 15y in Software Development and IT Consulting
- Tech Lead at INGTES Development in Cluj



# Agenda:

1. Agile Development
2. Scrum
3. Kanban
4. Tools
5. GIT and Workflows
6. Hands On GIT
7. Questions



# Manifesto for Agile Software Development

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.



<https://agilemanifesto.org/>

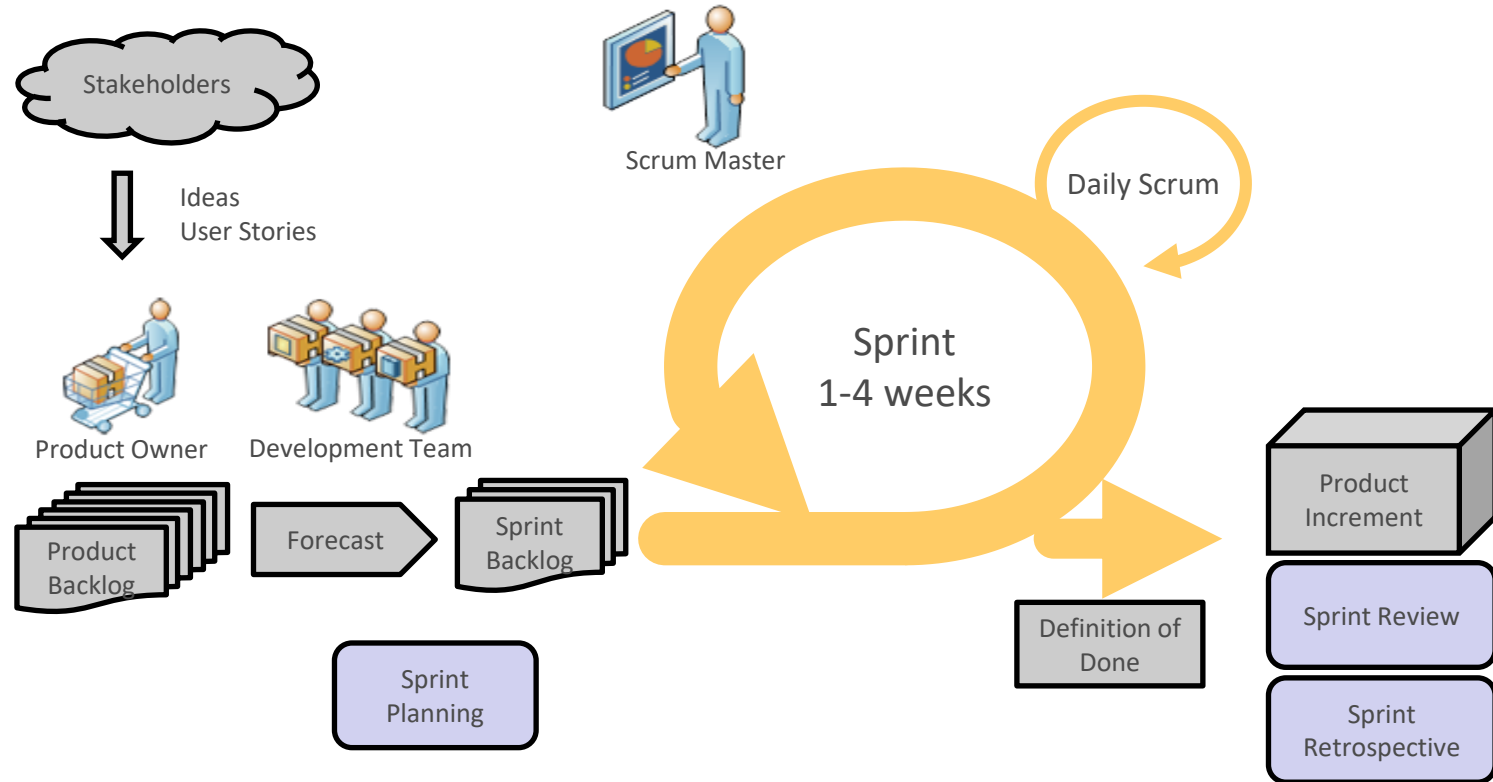


# Scrum

How 90% of teams work in Cluj

# Scrum

## Defining Roles, Events and Artifacts



# Scrum

## Roles



### Development Team

“Deliverer” of the Product.

Implements incremental parts of the product, for the Product Owner.

Plans sprints in personal responsibility



### Product Owner (PO)

Represents the customer side.

Responsible to maximise the value of the product and the work of the development team

Defines and prioritises the requirements in the backlog.

This is a single person, not a committee. The organisation has to respect his decisions.



### Scrum Master (SM)

Is responsible for the knowhow and execution of Scrum

Helps the team in their self-organisation and reflection.

Removes impediments in the team

Supports the PO managing the Backlog



# Scrum

## Events



### **Sprint**

A Timebox (max 1 month), in which a product increment is produced.

All Sprints have the same length. The new Sprint starts immediately after finishing the last.

During a sprint a sprint goal and a quality expectations will not be adapted.



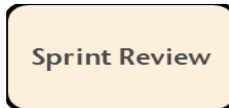
### **Sprint Planning**

During the Sprint Planning the Scrum Team plans the new Sprint together. The Scrum Team is not only the development team!!!



### **Daily Scrum**

Timebox of 15 min, coordination of the development team.



### **Sprint Review**

Check of the product increment, adaptation of the backlog if needed. Goal is to get as much feedback as possible.



### **Sprint Retrospective**

Überprüfung und Verbesserung der Zusammenarbeit

# Scrum

## Artifacts



### Product Backlog

A ordered List of all things what should be part of the product.

Ist the only source of requirements for the whole application.

The Product Owner is responsible to manage the Backlog.



### Sprint Backlog

The total amount of entries selected from the Product Backlog for the next sprint which will be created for the next product increment.

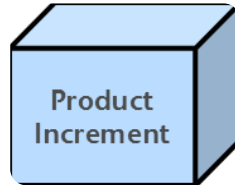
It's an estimation of the development team how much work they can finish.



### Definition of Done

Defines what "Done« means, if a Backlog entry or a Product Increment is called «Done».

Increases the transparency through a clear definition, which needs to be followed by the team.

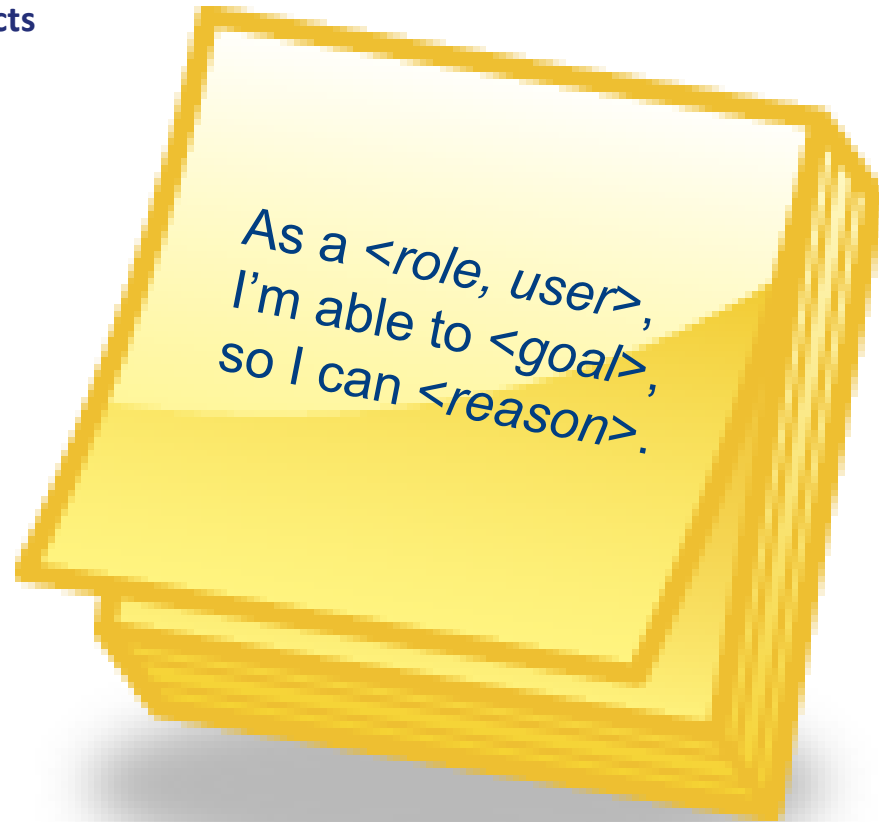


### Product Increment

The result of all the entries which are done in the last sprint + all the results from previous sprints.

# Scrum

## Artifacts



[User Story Template Design from Mike Cohen]

# Scrum

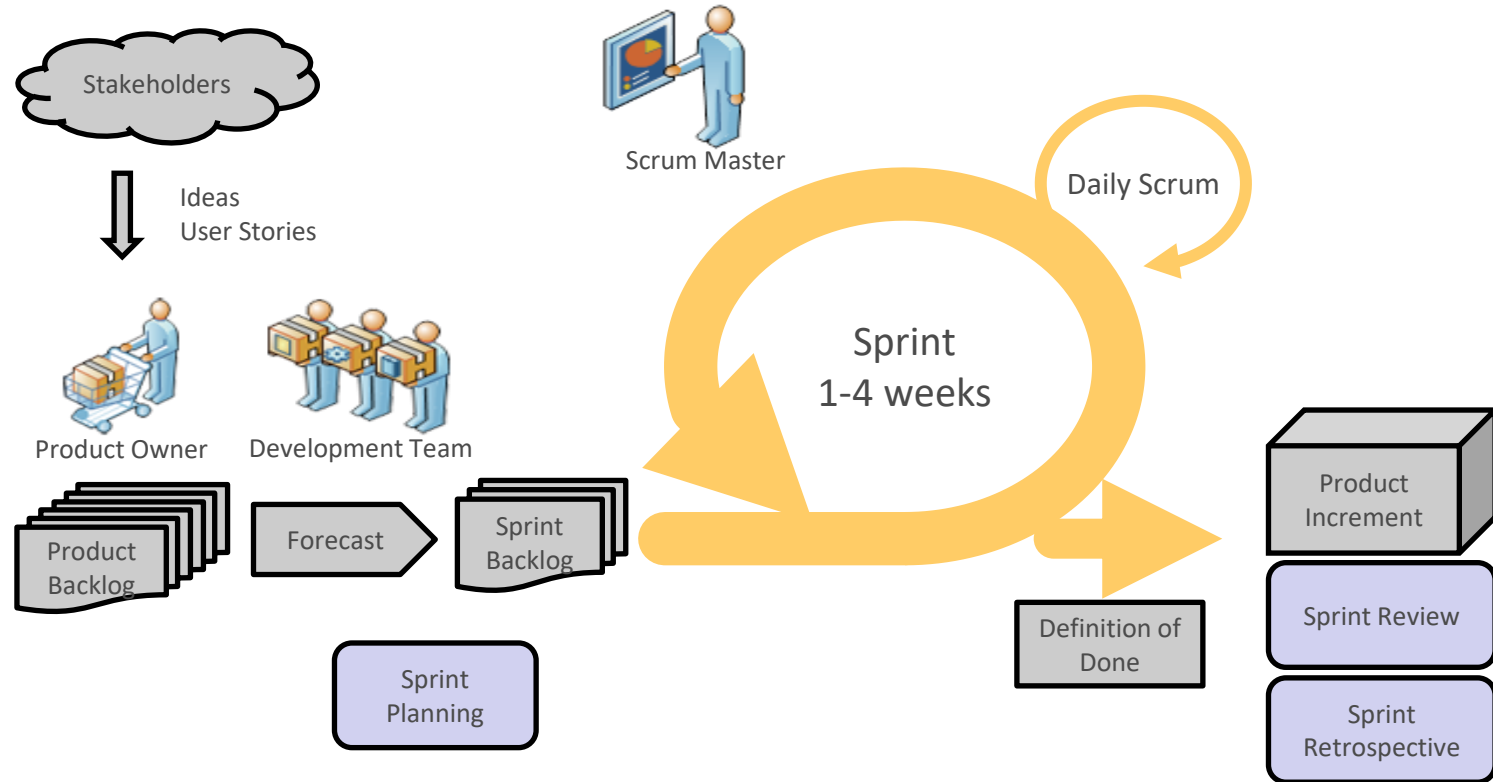
## Artifacts



[User Story Template Design from Mike Cohen]

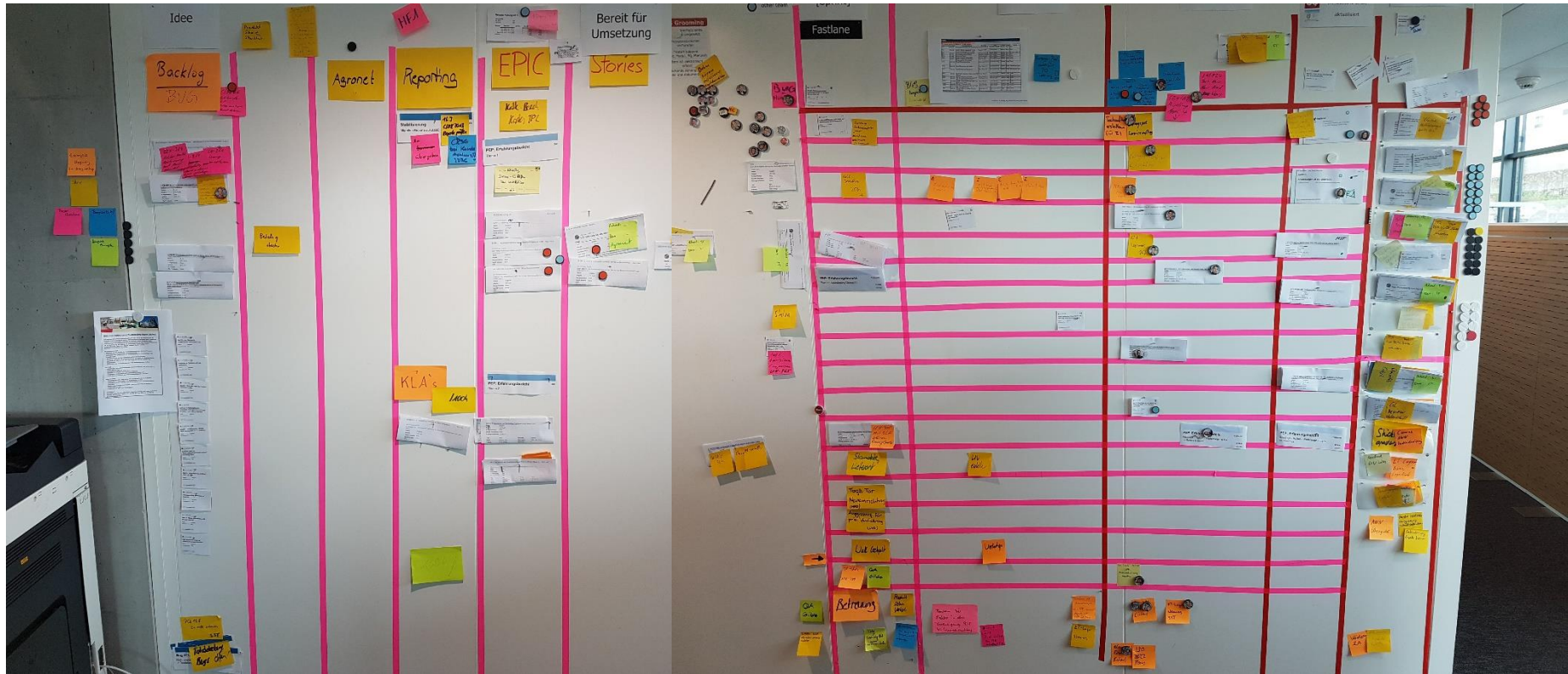
# Scrum

## Defining Roles, Events and Artifacts



# Scrum

## Examples





# Kanban

Might be the better option for the student project

# What is Kanban?

Creating a culture of continuous improvement

Kan

看

signal

Ban

板

card



# Kanban overview

Pull principle with visual representation via cards

## kanban

- Toyota Production systems, kanban is the core piece of visualisation of their production management
- Signal cards show downstream production steps, that a task is done, and that new material and components are needed
- Pull principle: reducing storage and visualize problems in the process

## Kanban in IT

- Adapted system in software development
- Supports change evolutionary by optimizing processes successively
- Kanban in IT is based on lean (efficient design of a value chain)
- David Anderson is the creator of Kanban in IT
- Works especially well in Support and Maintenance

# Kanban overview

Stop starting, start finishing

## Kanban Base Principles

- Start with what you do already
- Pursue incremental, evolutionary change
- Respect the initial process, roles and responsibilities
- Support leadership in all levels of an organisation

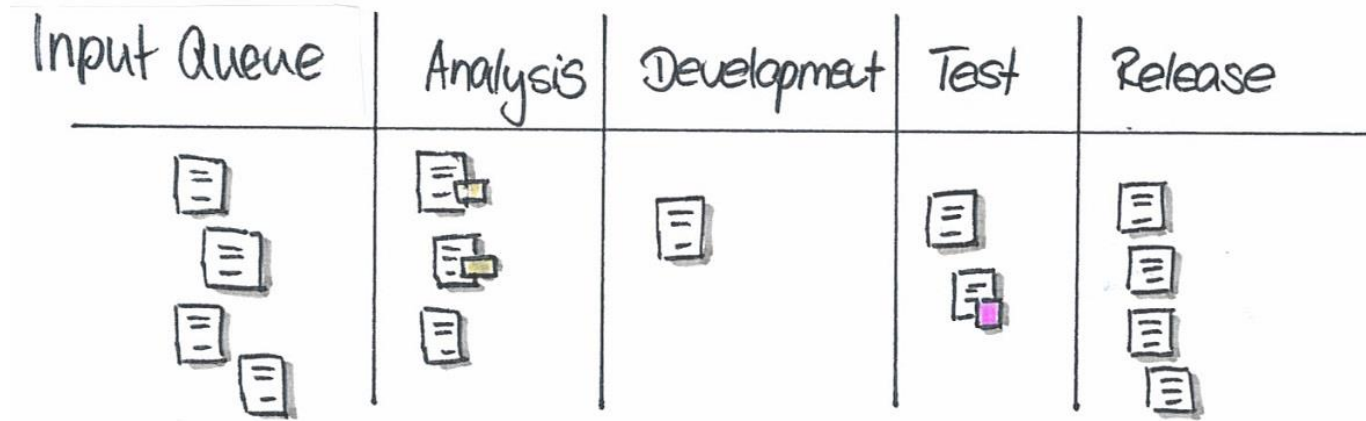
## Kanban Core Practices

- Visualize work
- Limit the work in progress (WIP)
- Manage flow
- Make process steps explicit
- Implement fast feedback mechanisms
- Execute communal improvements

# Visualize work

## Visualize the value chain

Work will not be pushed into the next step. The workers from the next step pull the work once they have the necessary capacity. (Pull principle)

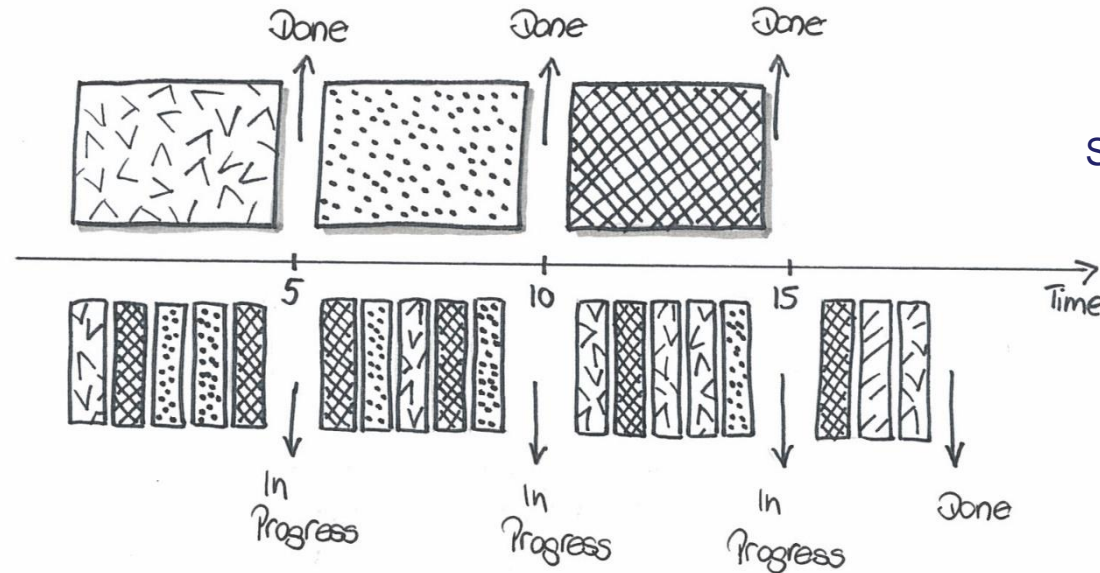


The Kanban Team identifies in which position of the value chain it's located, how the process works and defines the borders.

# Limit the work in progress (WIP)

## Difference between sequential and parallel work

A limit on the amount of work, which is executed in one step, reduces the time and improves the workflow.

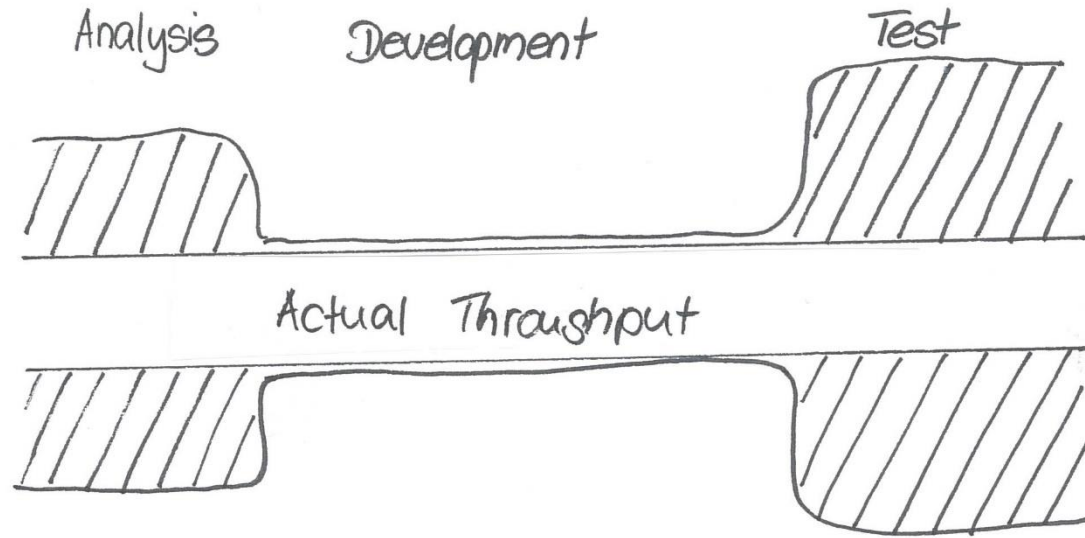


Sequential work → smaller worktime

parallel work → higher worktime

# Manage Flow

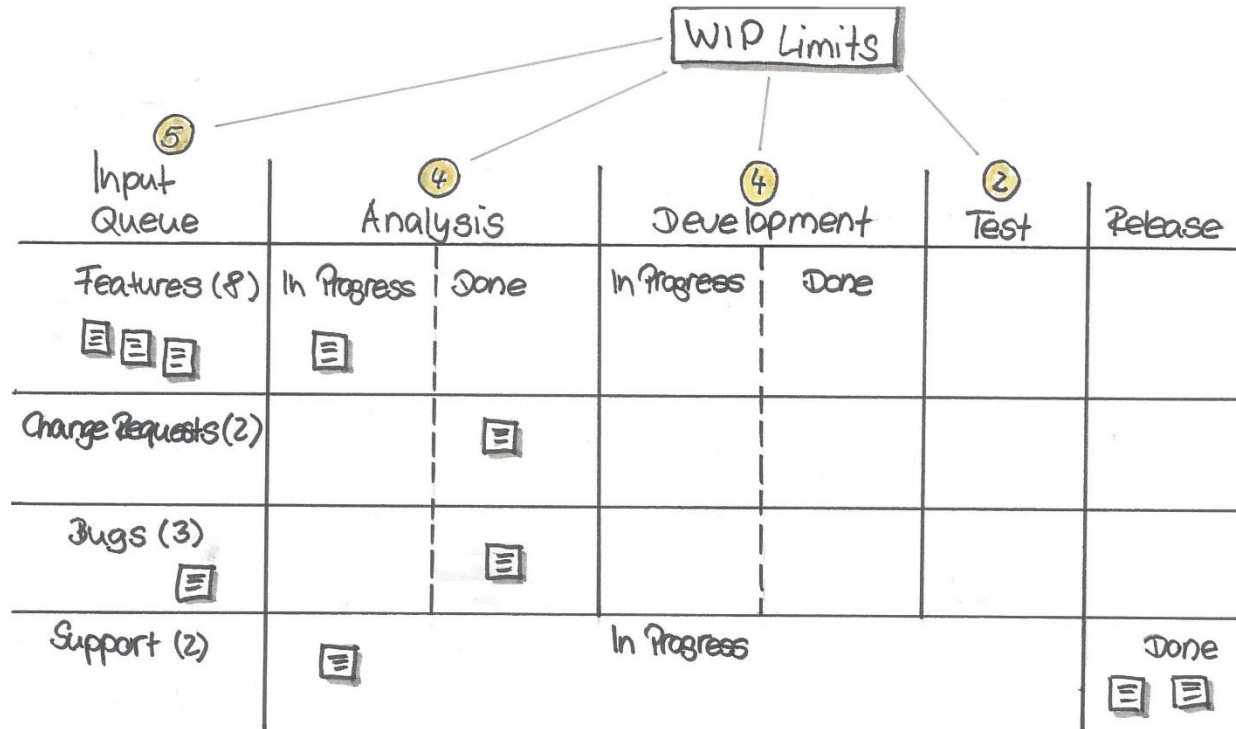
The total amount done is defined by the bottleneck of the system



Don't let more Tasks into the system than the bottleneck can handle. WIP limits are managed so that the bottleneck is not overwhelmed.

# Manage Flow

Create a value chain with WIP limits per task type and process step





# Tools

What you should know and what you should use

# Tools

## Physical vs. online Boards

### Physical boards have benefits, but only work for local teams

- Haptic feedback
- Fast overview
- Easy to adjust to the needs

### Online Boards

- Atlassian Trello for beginners: <https://trello.com/b/UOhltZa/testboard>
- Atlassian Jira is most commonly used in Scrum and DEV Teams
- Microsoft TFS in .NET oriented companies





# GIT

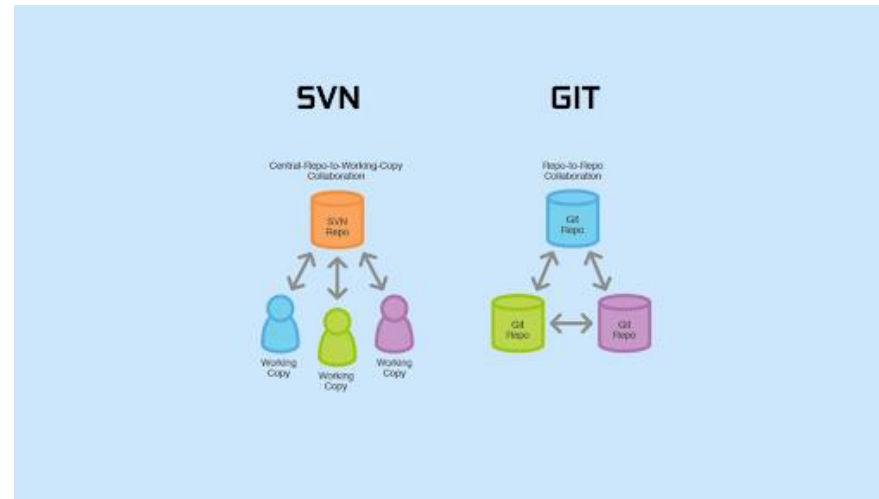
Git is a distributed version control system (DVCS) used for tracking changes in source code during software development.

# Version Control System (VCS)

- Git allows multiple developers to collaborate on the same project.
- It tracks changes to files, allowing you to revert to previous stages, compare changes over time, and see who last modified something.

# Distributed

- Git is a distributed version control system, meaning that each developer has a complete copy of the repository (including its entire history).
- This is different from centralized version control systems (like SVN), where developers must be online to access the repository.



# Key Concepts

- **Repository:** The collection of all files and folders in your project, along with the history of changes.
- **Commit:** A snapshot of your repository at a point in time.
- **Branches:** Independent lines of development. You can create branches to work on new features without affecting the main codebase.
- **Merge:** Combining changes from one branch into another.
- **Pull Request (PR):** In platforms like GitHub, GitLab, etc., this is a way to propose changes and request that someone else review and merge your changes into their branch.

# Basic Workflow

1. **Clone:** Create a local copy of a repository.
2. **Modify:** Make changes to files.
3. **Add:** Stage your changes for commit.
4. **Commit:** Save your changes with a message describing the changes.
5. **Push:** Send your changes to a remote repository (like GitHub, GitLab, etc.).
6. **Pull:** Get the latest changes from the remote repository to your local repository.
7. **Merge:** Combine changes from different branches.

# Popular Platforms for Git Hosting

- GitHub: A web-based platform for hosting Git repositories and collaboration. Widely used in open-source projects.
- GitLab: Similar to GitHub but provides both cloud-hosted and self-hosted options.
- Bitbucket: Offers both Git and Mercurial repositories. Owned by Atlassian, it's often used for private repositories.
- DevOps (Azure): Azure DevOps is a set of cloud services provided by Microsoft that offers a full-featured platform for managing the entire software development lifecycle. It includes a range of tools for agile planning, development, testing, and deployment.
- other (many)



# Why Use Git?

- **Collaboration:** Multiple developers can work on the same project simultaneously.
- **History:** Detailed history of changes, who made them, and why.
- **Branching and Merging:** Safe environment for trying out new features without affecting the main codebase.
- **Backup:** Having a remote repository acts as a backup for your code.
- **Open Source:** Many open-source projects use Git, making it easier to contribute to them.

Git is powerful but can have a steep learning curve, especially for beginners. However, once you get the hang of it, it becomes an invaluable tool for managing and collaborating on software development projects.



# GitHub

GitHub is a web-based platform built around Git, the distributed version control system. It provides a hosting service for software development and version control using Git. GitHub offers a variety of features to help developers and teams collaborate on projects, manage code repositories, track issues, and more.






# Hosting Git Repositories

- **Repository Hosting:** GitHub allows users to host their Git repositories online. This means developers can push their local Git repositories to GitHub to make them accessible from anywhere with an internet connection.
- **Remote Collaboration:** Multiple developers can work on the same project by cloning repositories, making changes, and pushing those changes back to GitHub.

## Top Repositories



Find a repository...

-  mihai-ibanescu/ubb-calculator-demo
-  mihai-ibanescu/travisci-test
-  mihai-ibanescu/Test

 **ubb-calculator-demo** Private

 main  2 Branches  0 Tags

Switch branches/tags

Find or create a branch...

Branches

Tags

✓ main

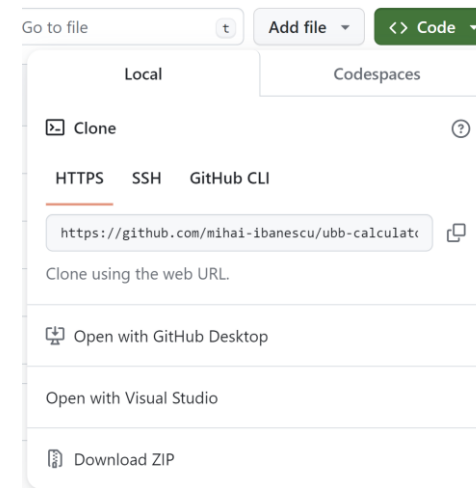
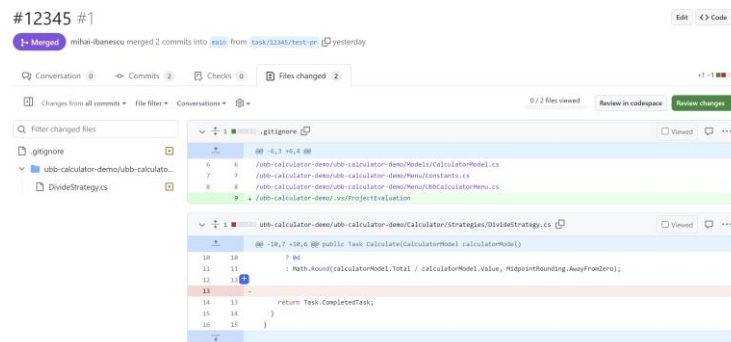
default

task/12345/test-pr

View all branches

# Collaboration and Code Review

- **Pull Requests (PRs):** A fundamental feature of GitHub, PRs allow developers to propose changes to a repository and request that someone else review and merge their changes. This is widely used for code review and collaboration.
- **Branching:** GitHub supports creating branches, which enables developers to work on features or fixes without impacting the main codebase until they're ready to merge changes.



# Issue Tracking and Project Management

- **Issues:** GitHub's issue tracker allows teams to track bugs, tasks, enhancements, and other items related to the project.
- **Project Boards:** GitHub provides project boards to help teams organize and prioritize their work. These boards can be customized with columns like "To Do," "In Progress," and "Done."

<> Code Issues 1 Pull requests Actions Projects

## Some new issue #2

Open mihai-ibanescu opened this issue now · 1 comment



mihai-ibanescu commented now

there is an issue



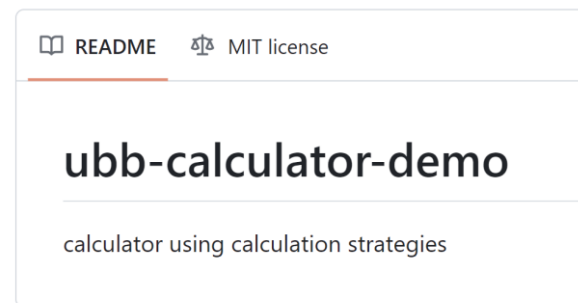
mihai-ibanescu commented now

need more details..



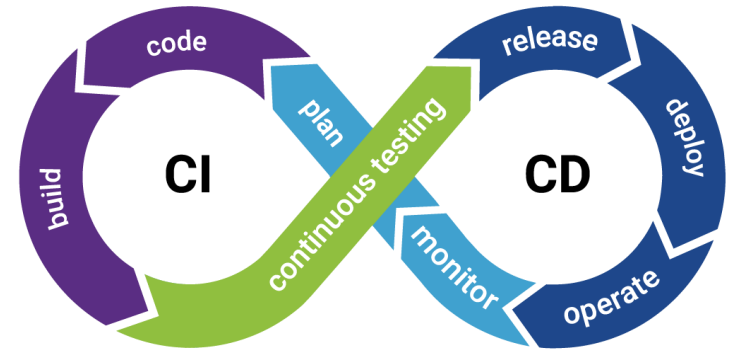
# Documentation and Wikis

- **Wiki Pages:** GitHub allows repositories to have their wikis for documentation purposes.
- **README Files:** Every repository can have a README.md file at the root, which typically serves as the project's front page and can include essential information like installation instructions, usage, etc.
- **License Management:** GitHub helps manage open-source licenses, making it easier for developers to understand the licensing requirements of projects.
- **Contributor License Agreements (CLAs):** Projects can require contributors to sign CLAs to clarify the intellectual property rights and licensing for contributed code.



# Continuous Integration and Deployment (CI/CD)

- **GitHub Actions:** A powerful feature that allows users to automate workflows such as building, testing, and deploying code directly from GitHub.
- **Integrations:** GitHub can be integrated with various CI/CD tools like Travis CI, Circle CI, and Jenkins for automating development workflows.



# Why Use GitHub?

- **Centralized Repository:** A single place for hosting code, documentation, and collaboration.
- **Collaboration:** Multiple developers can work on the same project with ease.
- **Visibility:** Public repositories can be accessed by anyone, making it great for open-source projects.
- **Code Review:** PRs facilitate code review and quality control.
- **Community:** Access to a vast community of developers, projects, and resources.

GitHub has become the standard platform for hosting and collaborating on Git repositories, particularly in the open-source community. Many companies also use GitHub for their private repositories, leveraging its features for team collaboration and project management.



# Branching

Branching in Git is a fundamental operation that allows developers to create divergent lines of development, work on new features, bug fixes, or experiments without affecting the main codebase.

# Create a New Branch

To create a new branch in Git, you use the **git branch** command followed by the name of the new branch. Then, you switch to the new branch using **git checkout** or **git switch**. In newer versions of Git, you can also create and switch to a new branch in a single command with **git switch -c** or **git checkout -b**.

```
# Create a new branch
git branch new-feature

# Or, create and switch to the new branch in one command
git checkout -b new-feature

# Or, with newer Git versions
git switch -c new-feature
```



# Work on the Branch

After creating the branch, you can start making changes to your code. These changes will be isolated to the new branch.

```
# Edit files  
vim myfile.js
```

```
# Stage changes  
git add .  
  
# Commit changes  
git commit -m "Implemented new feature"
```

# Switch Between Branches

You can switch between branches using **git checkout** or **git switch**. This allows you to move between different branches in your repository.

```
# Switch to an existing branch
git checkout main

# Or, with newer Git versions
git switch main
```

# Merge Changes (Optional)

When you're done with the changes in your new branch and you want to incorporate them back into the main branch (like **main**), you can merge the changes.

```
# Switch to the main branch
git checkout main

# Merge the new-feature branch into main
git merge new-feature
```

# Delete a Branch (Optional)

Once you've merged the changes from a feature branch, you might want to delete the branch to keep your repository clean. Be cautious and ensure you've merged all necessary changes before deleting.

```
# Delete a local branch  
git branch -d new-feature  
  
# To force delete (if not fully merged)  
git branch -D new-feature
```

# Push a Branch to Remote (Optional)

If you've created a new branch locally and want to share it with others or work on it from a different machine, you can push the branch to a remote repository.

```
# Push the new-feature branch to remote (origin)  
git push origin new-feature
```

# Here are some common mistakes when working with Git



## Committing Everything at Once

- **Mistake:** Committing all changes with `git add .` or `git add -A`.
- **Issue:** Creates large, messy commits that are hard to review and understand.
- **Solution:** Stage and commit related changes in smaller, logical chunks. Use `git add <file>` or `git add -p` for selective staging.

## Not Pulling Before Pushing

- **Mistake:** Pushing changes without pulling latest changes from the remote repository.
- **Issue:** Can cause conflicts with others' changes, leading to merge conflicts.
- **Solution:** Always pull (`git pull`) before pushing (`git push`) to ensure your local branch is up to date.

## Ignoring Code Reviews

- **Mistake:** Merging branches without proper code reviews.
- **Issue:** Missed bugs, quality issues, and lack of knowledge sharing.
- **Solution:** Always conduct code reviews (via pull requests) to ensure quality, readability, and knowledge transfer.

## Not Using Branches

- **Mistake:** Making changes directly on main or master branch.
- **Issue:** Risk of breaking the main codebase, especially in team environments.
- **Solution:** Use feature branches for development, keeping main or master stable and deployable.

## Poor Commit Messages

- **Mistake:** Writing vague, non-descriptive commit messages.
- **Issue:** Difficult to understand changes later, especially during code reviews or when looking through history.
- **Solution:** Write clear, concise commit messages that describe what was changed and why.



# Branching Strategies

Branching strategies in Git are methodologies or patterns that teams use to manage branches in a repository. These strategies help organize code development, coordinate teamwork, and facilitate the integration of changes.

# GitHub Flow

**Description:** Lightweight workflow designed around GitHub's pull requests.

• **Workflow:**

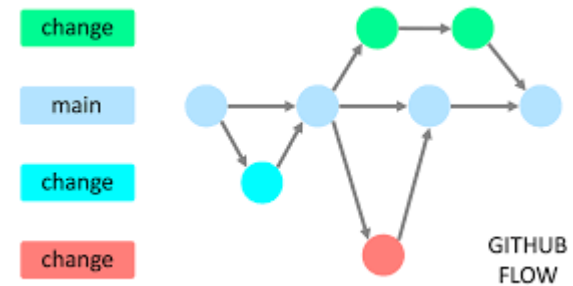
1. Create a new branch (**main** is the default)
2. Develop the feature in the branch
3. Open a pull request (PR) to merge changes into **main**
4. Review code, discuss, and make changes as necessary
5. Merge the PR into **main**

• **Advantages:**

- Simple and easy to understand.
- Encourages code review and collaboration.

• **Disadvantages:**

- Less suited for larger, more complex projects.
- May result in a fast-moving **main** branch





# Feature Branching

•**Description:** Each new feature or task is developed in its own branch.

•**Workflow:**

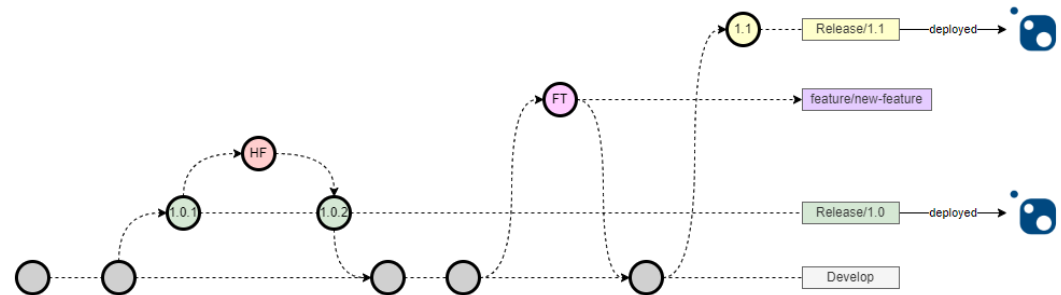
1. Create a new branch for the feature/task (**feature/<feature-name>**, **task/<task-name>**)
2. Develop the feature in the branch
3. Merge the branch back into the main branch (**main**, **develop**) when the feature is complete

•**Advantages:**

- Isolates work on a specific feature, preventing conflicts with other changes.
- Allows for code review of individual features.

•**Disadvantages:**

- If many features are being developed simultaneously, it can lead to a large number of open branches.
- May require frequent merging.





# Workshop

<https://github.com/mihai-ibanescu/ubb-calculator-demo.git>

# Questions



Next Generation

*Thank You!*

