Software technology

04 – Agile Methodologies

Dr. Attila Gludovátz

Online catalog – every week

- https://catalog.inf.elte.hu/
- Log in
- Username: yourUsername (@inf.elte.hu)
- Password: your email password
- Captcha: I generate a number for you…
- Lecture attendance is ***not*** optional! Max 3 misses and you are out

Manifesto for Agile Software Development (2001)

- 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

Why Agile?

We go for …

- LWPs (Lightweight Processes)
- Flexible process
- Adaptive planning
- Fast response, short feedback loop
- Evolutionary development
- Early delivery and continuous improvement
- Self-organizing cross-functional teams

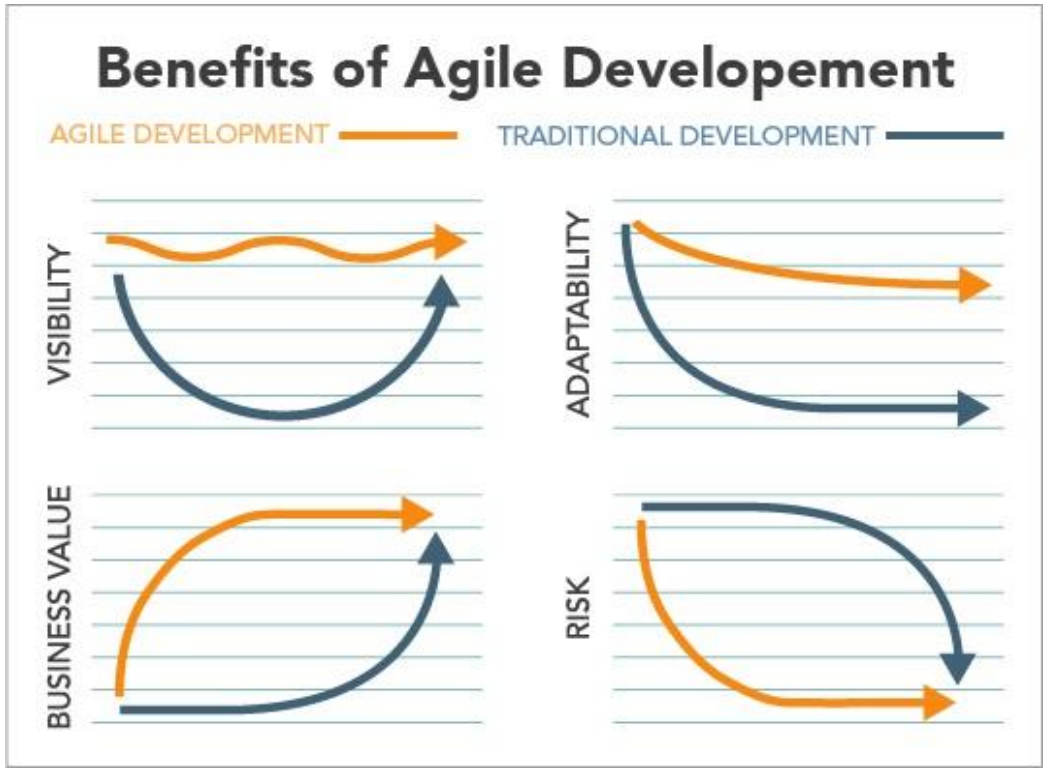
Basic principles of Agile SW Dev

12 AGILE PRINCIPLES BEHIND THE AGILE MANIFESTO

1 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	2 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	3 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4 Business people and developers must work together daily throughout the project.	5 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	6 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
7 Working software is the primary measure of progress.	8 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	9 Continuous attention to technical excellence and good design enhances agility.
10 Simplicity – the art of maximizing the amount of work not done – is essential.	11 The best architectures, requirements, and designs emerge from self-organizing teams.	12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Major Benefits of Agile Process

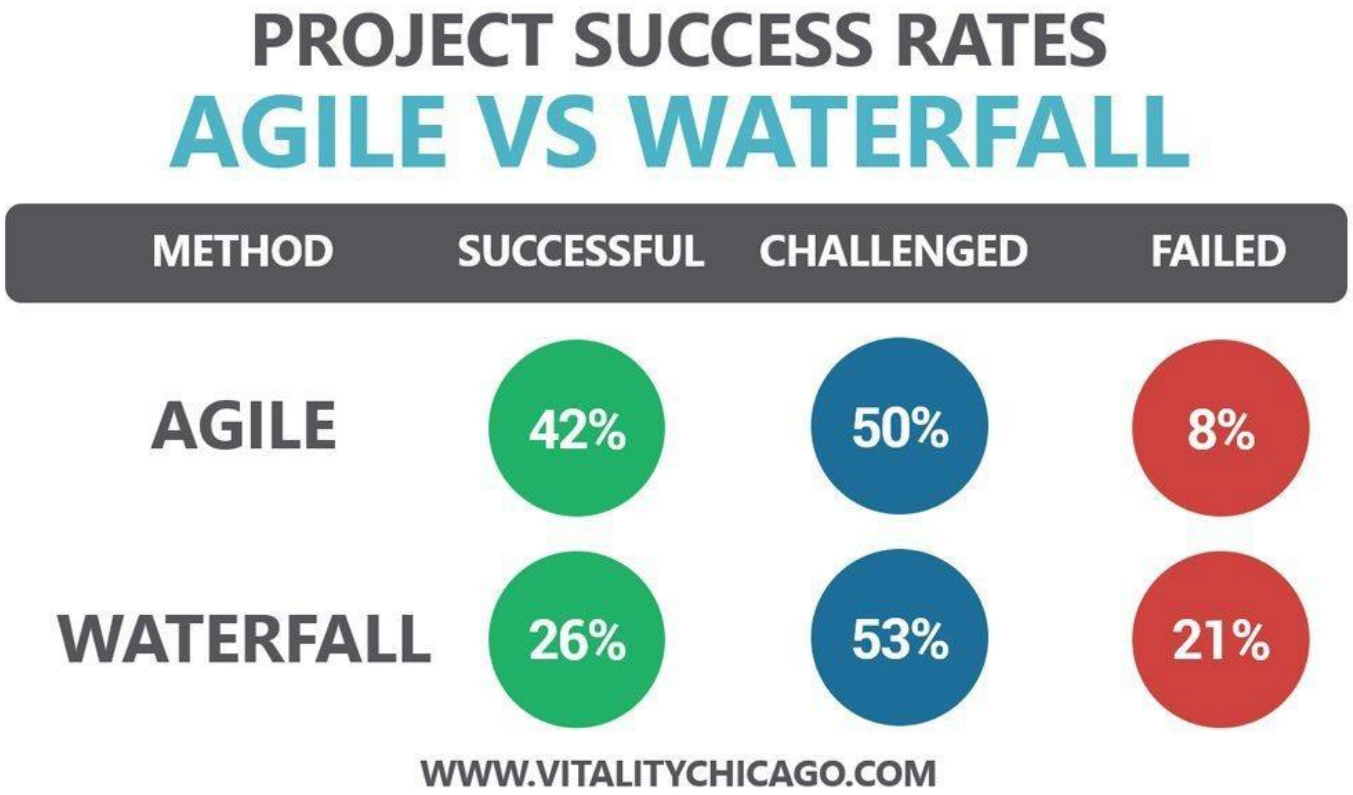
- Visibility of project details
- Increased team efficiency
- Ability to adapt to changes
- Ability to scale



Disadvantages of Agile Process

- The Agile process is mainly focused on the code work or development work but there is no consideration related to the documentation work required related to the development work
- In the case of large software development, it is quite impossible to estimate the total costs of the entire project in the initial stage of the development
- Rapid decisions can be taken by the Senior and experienced developers
  - So, it is quite impossible for an inexperienced or less experienced developer to make the decisions

Agile vs. Waterfall model



Scalability of Agile projects

- **Time and cost** are the main factors in determining whether a company will go forward with a project.
    - *How long will the project take?*
    - *What will it cost?*
    - *Will it be worth the initial investment to get this project done?*
    - *What else could be done with the same resources and team members that may hold more value?*

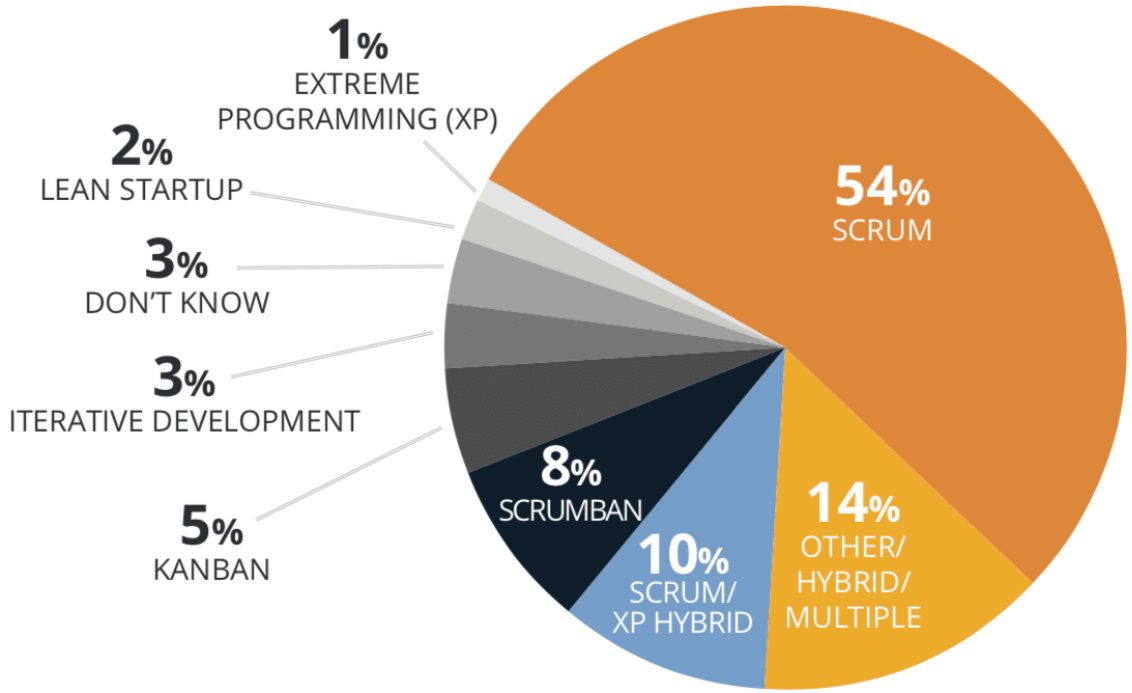
Summary

- Agile is a framework or methodology which helps us to plan or decide how the software development process needs to be carried out
- Agile is not a single process or mechanism. It is basically containing a group of methods and best practices according to the Agile Manifesto
- But the Agile process never promises to solve all the problems which are present in the current software industry
- This process actually establishes a culture and a mindset where the solution needs to merge

Agile has become an “Umbrella” Term

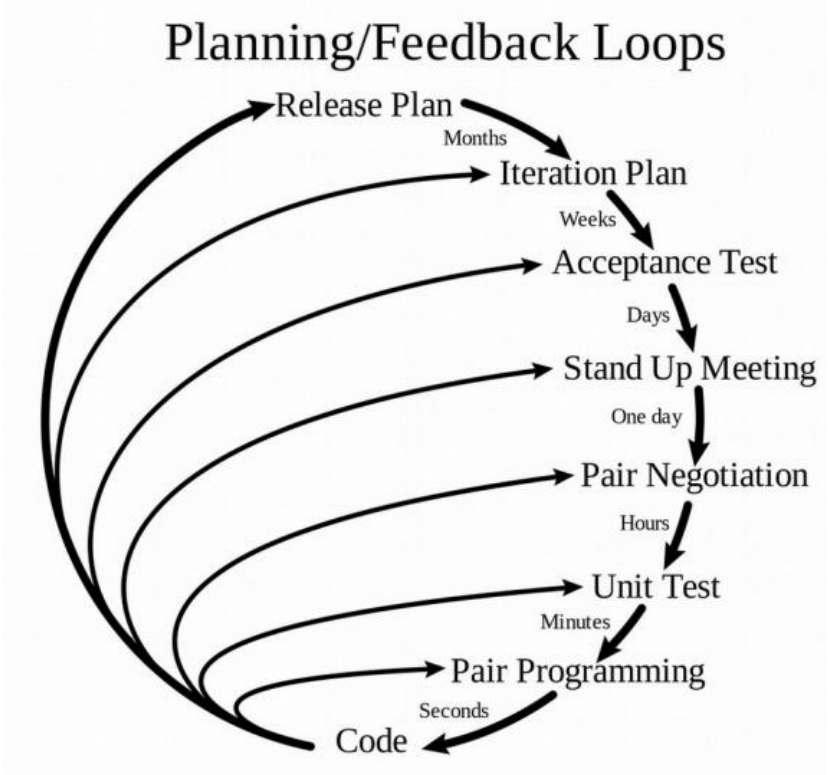
Agile Methodologies Used

Scrum and Scrum/XP Hybrid (64%) continue to be the most common agile methodologies used by respondents’ organizations.

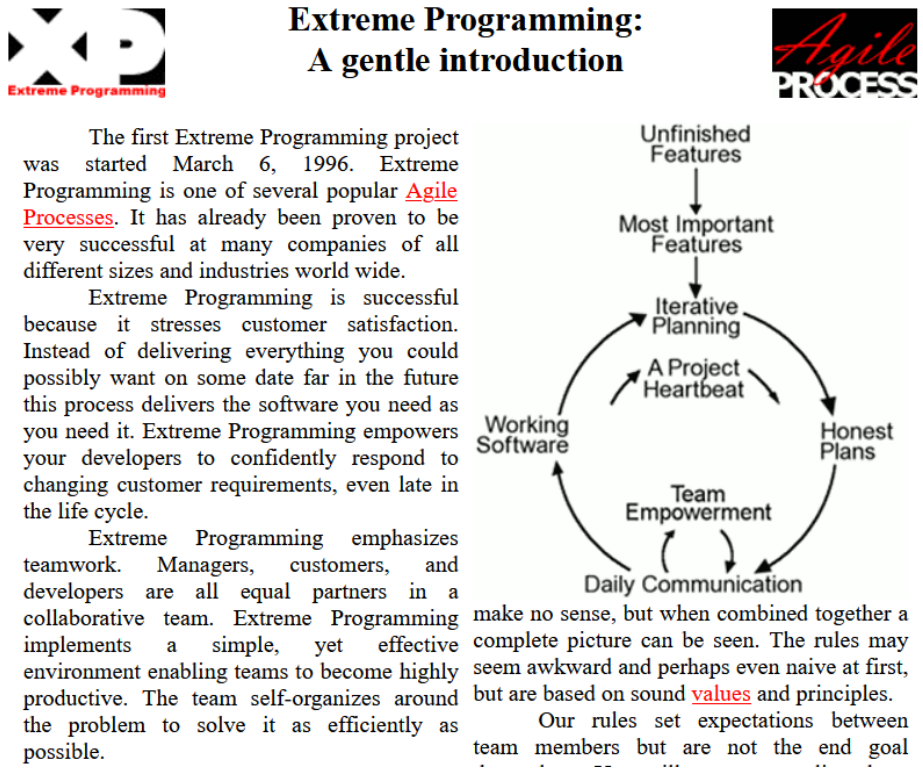


XP – eXtreme Programming

- Goals:
  - Be responsive to changing requirements
  - Improve productivity
  - Improve quality
  - Introduce fine-grained monitoring (frequent checkpoints)



Extreme Programming: A gentle introduction



Extreme Programming: A gentle introduction

The first Extreme Programming project was started March 6, 1996. Extreme Programming is one of several popular Agile Processes. It has already been proven to be very successful at many companies of all different sizes and industries world wide.

Extreme Programming is successful because it stresses customer satisfaction. Instead of delivering everything you could possibly want on some date far in the future this process delivers the software you need as you need it. Extreme Programming empowers your developers to confidently respond to changing customer requirements, even late in the life cycle.

Extreme Programming emphasizes teamwork. Managers, customers, and developers are all equal partners in a collaborative team. Extreme Programming implements a simple, yet effective environment enabling teams to become highly productive. The team self-organizes around the problem to solve it as efficiently as possible.

make no sense, but when combined together a complete picture can be seen. The rules may seem awkward and perhaps even naive at first, but are based on sound values and principles.

Our rules set expectations between team members but are not the end goal

- For more see:
  - http://www.extremeprogramming.org/index.html

XP Coding

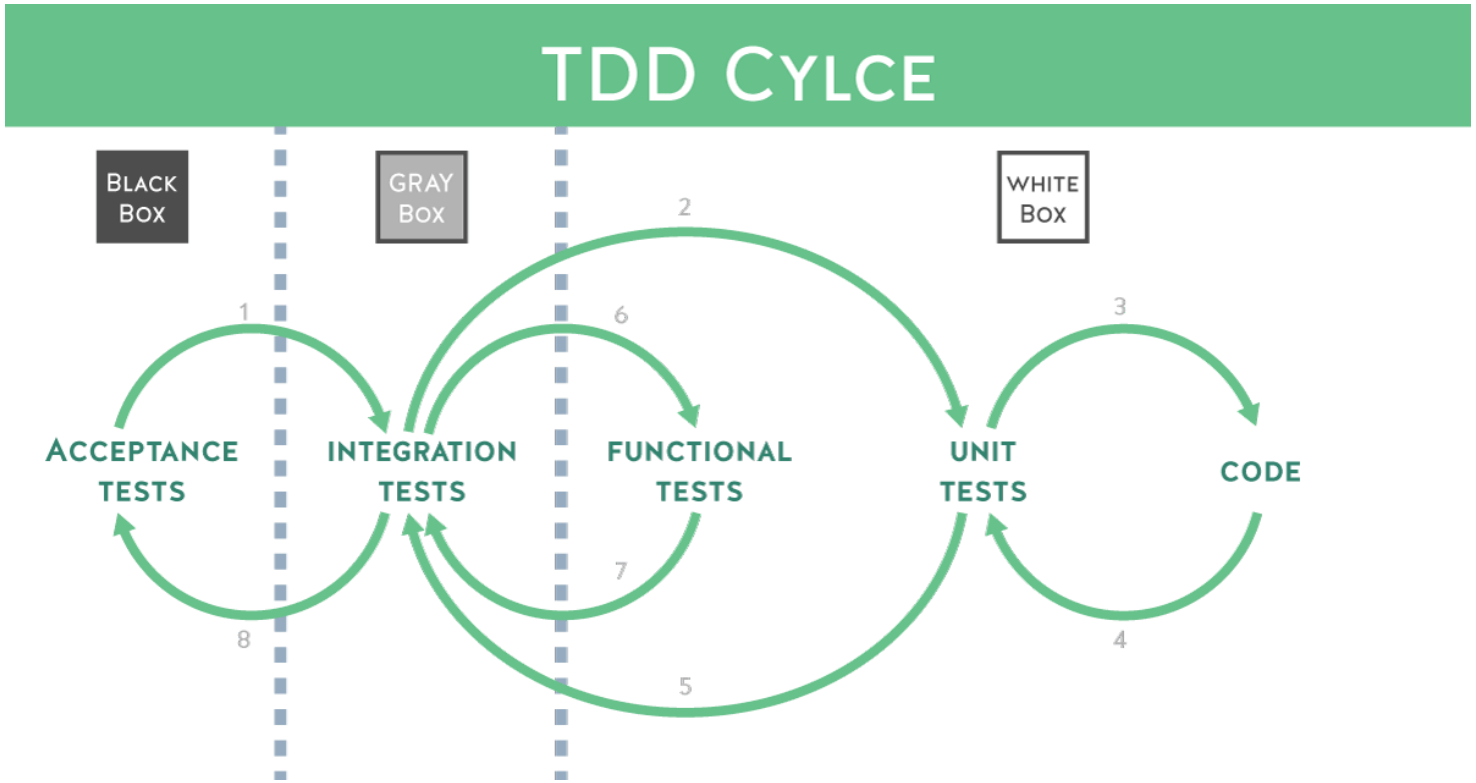
- Extensive **Code Reviews** or
  - May spot problems (of various levels)
  - Information sharing!!!
- **Pair Programming**
  - Knowledge sharing!!!
  - Higher quality, lower error rate
  - Expensive (probably)
  - Pairs: expert-expert, expert-novice
  - Can be non-functioning (disengaged pair, no communication)

XP Testing = TDD

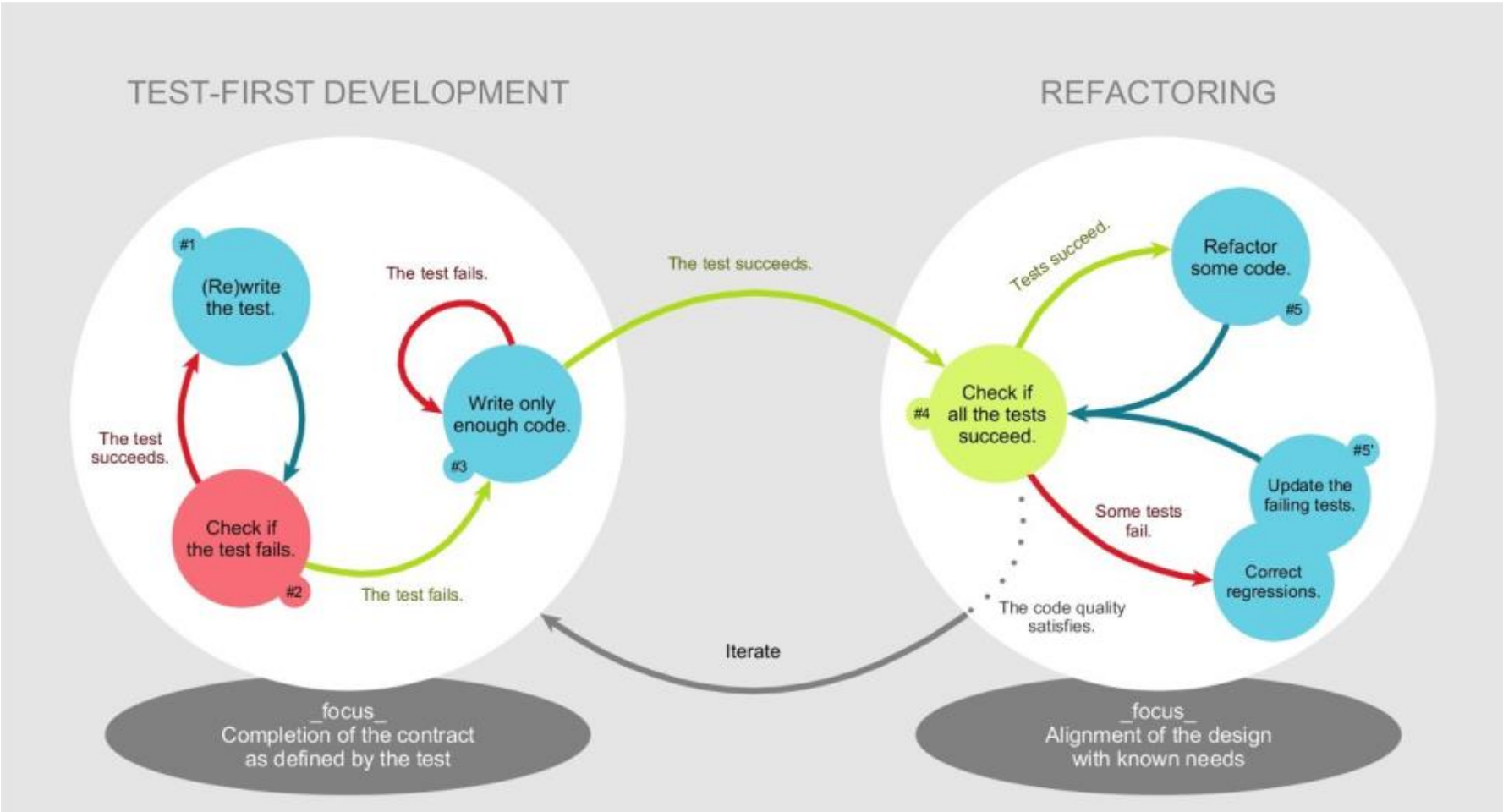
Test Driven Development

- Test-first approach
- Test is Documentation (no comments)
- Refactor at end (clean up mess)
- You aren't gonna need it (KISS)
- Testing levels
- Dummy, Stub, Spy, Mock, Simulator
- Bad tests
- ATDD (Acceptance Test Driven Development)

Test-driven development (TDD)



Test-driven development (TDD)



Principles of TDD

- Test-driven development is a software-development methodology which essentially states that for each unit of software, a software developer must:
  - define a test set for the unit first
  - make the tests fail
  - then implement the unit
  - finally verify that the implementation of the unit makes the tests succeed

Behavior-driven Development (BDD)

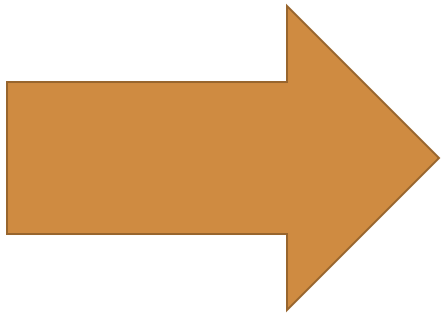
- BDD = ATDD + DSL (Domain-Specific Language)
- **Tooling:** uses a natural language-like-DSL and auto-generates test stubs
- **Test:** Acceptance Criterias in different Scenarios
- **Good practice:** Name tests on expected results, normal behavior (not errors or submodule names)
- **Tools:** Cucumber, JBehave...

Behavioral specifications – user story

- **Title**
  - An explicit title.
- **Narrative**
  - A short introductory section with the following structure:
    - **As a:** the person or role who will benefit from the feature;
    - **I want:** the feature;
    - **so that:** the benefit or value of the feature.
- **Acceptance criteria**
  - A description of each specific scenario of the narrative with the following structure:
    - **Given:** the initial context at the beginning of the scenario, in one or more clauses;
    - **When:** the event that triggers the scenario;
    - **Then:** the expected outcome, in one or more clauses.

Tooling examples

```
Given a 5 by 5 game
When I toggle the cell at (3, 2)
Then the grid should look like
.....
.....
.....
..X..
.....
When I toggle the cell at (3, 1)
Then the grid should look like
.....
.....
.....
..X..
..X..
When I toggle the cell at (3, 2)
Then the grid should look like
.....
.....
.....
.....
..X..
```



```
private Game game;
private StringRenderer renderer;

@Given("a $width by $height game")
public void theGameIsRunning(int width, int height) {
    game = new Game(width, height);
    renderer = new StringRenderer();
    game.setObserver(renderer);
}

@When("I toggle the cell at ($column, $row)")
public void iToggleTheCellAt(int column, int row) {
    game.toggleCellAt(column, row);
}

@Then("the grid should look like $grid")
public void theGridShouldLookLike(String grid) {
    assertThat(renderer.asString(), equalTo(grid));
}
```

XP Planning Game

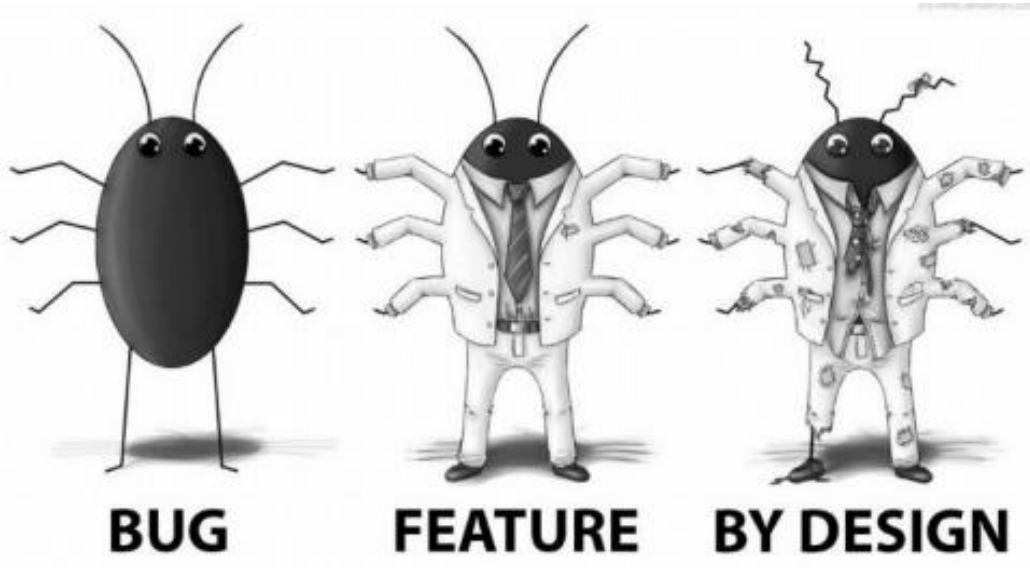
- **Release Planning** (~month) (with Customer) - Whole Team approach
  1. Exploration phase (User Stories from Customer, estimate, split)
  2. Commitment phase (Scope of next release, sort by value, risk)
  3. Steering phase (plan adjustments)
- **Iteration Planning** (~week) (with or without Customer)
  1. Exploration phase (requirements → Tasks, combine, split)
  2. Commitment phase (assignment to developer)
  3. Steering phase (testing, coding)

Other XP Practices

- **Shared Responsibility**
  - Information and Knowledge Sharing at all costs
  - Collective Code Ownership
  - Coding Standards (and enforcing tools: pep8, clang-format…)
- Continuously
  - **Continuous Integration**
    - Frequent code merges
    - Automated, always running tests
    - Constant refactoring
  - Design Improvements
  - Small Releases (Release simple, release often)
- **Sustainability**
  - Unexpected problems minimized
  - No overtime
  - Fresh mindset (escape through refactoring)

Feature Driven Development (FDD)

- Feature-driven development (FDD) is an iterative and incremental software development process
- It is a lightweight or Agile method for developing software
- FDD blends a number of industry-recognized best practices into a cohesive whole
- These practices are driven from a client-valued functionality (feature) perspective
- Its main purpose is to deliver working software repeatedly in a timely manner in accordance with the Principles behind the Agile Manifesto



Why use feature-driven development?

- You may want to consider using FDD methodology **if your project grows too large and complex** for smaller scrum teams to effectively handle the amount of work
- This agile feature-driven methodology is well-suited for **long-term projects that continually change** and add features in regular, predictable iterations
- FDD is **very scalable** from small teams to large cross-functional teams because it is designed to always focus on what the **customer needs** and wants

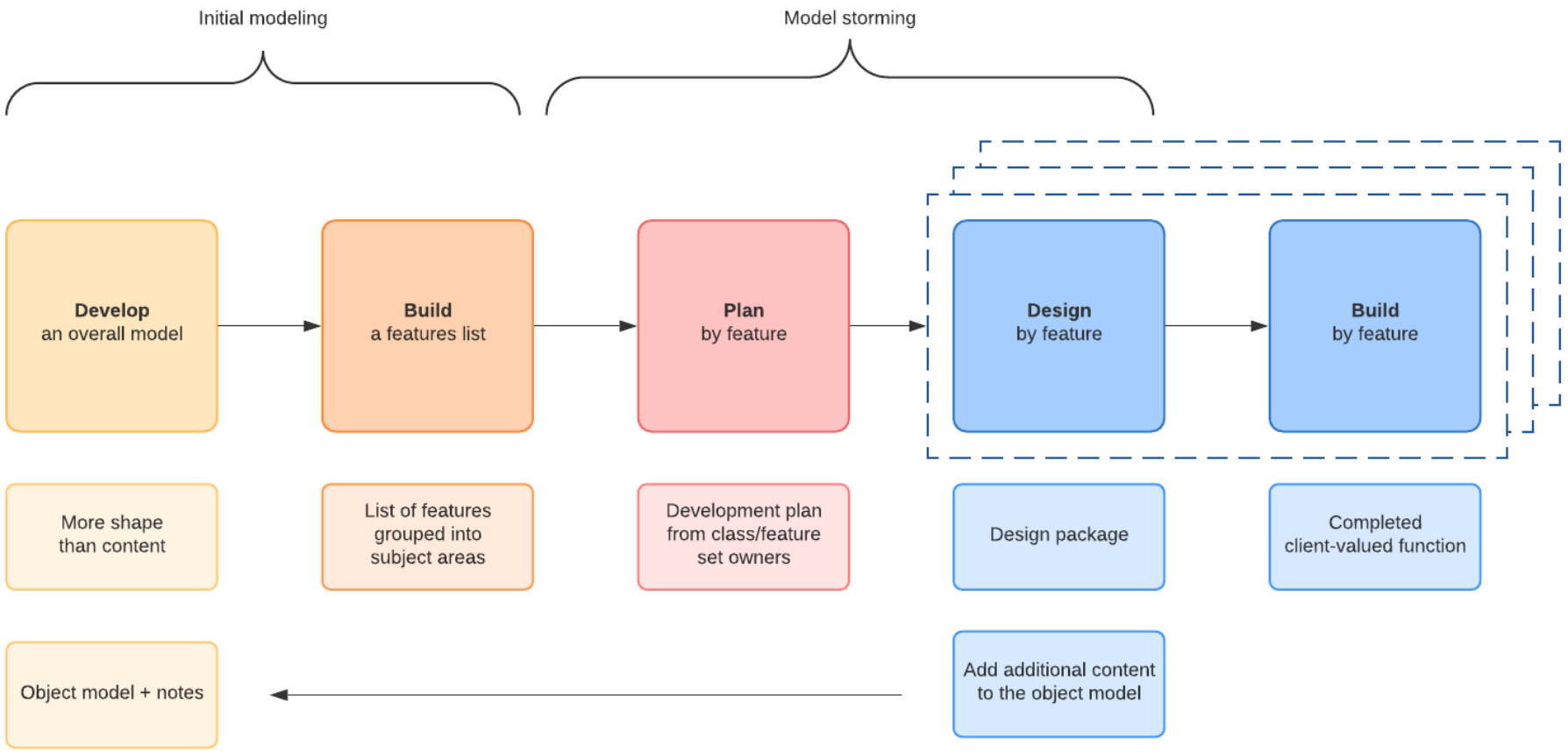
Best practices for feature-driven development

- Domain object modeling
- Developing by feature
- Individual class (code) ownership
- Feature teams
- Inspections
- Configuration management
- Regular build schedule
- Progress reports

Members of the FDD team

- Project manager
- Chief architect
- Development manager
- Chief programmer
- Class owner
- Domain expert

Feature Driven Development (FDD)



Strengths and Weakness of FDD

- **FDD's strengths include:**
  - Simple five-step process allows for more rapid development
  - Allows larger teams to move products forward with continuous success
  - Leverages pre-defined development standards, so teams are able to move quickly
- **FDD's weaknesses include:**
  - Does not work efficiently for smaller projects
  - Less written documentation, which can lead to confusion
  - Highly dependent on lead developers or programmers

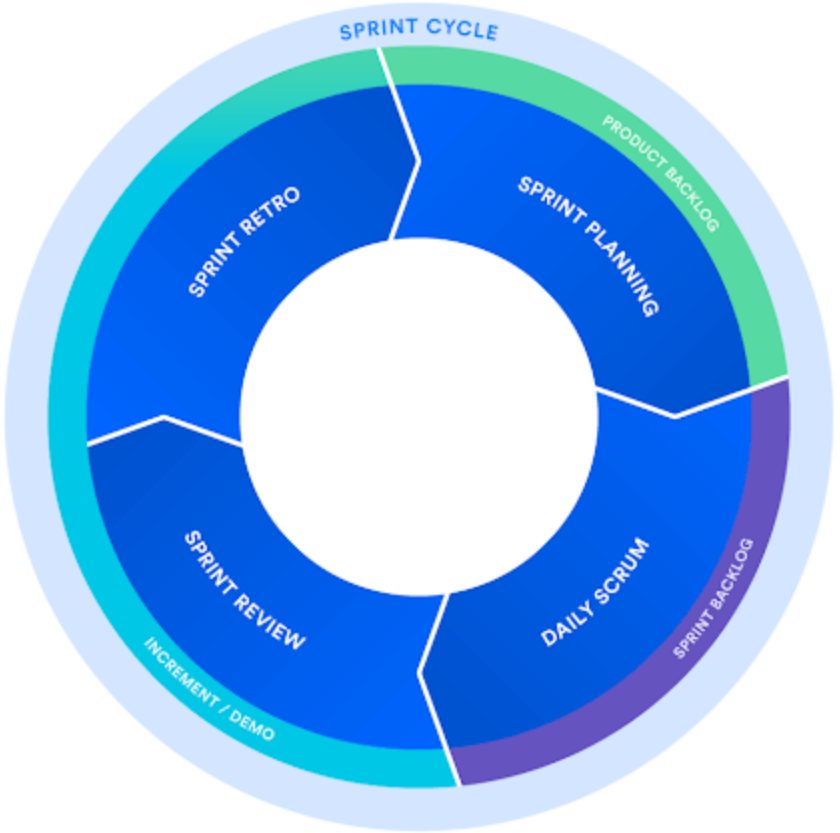
What is scrum?

- In short, scrum refers to a framework that makes for effective collaborations among teams that are working on complex products
- Although it is most often used by software development teams, scrum can essentially be beneficial to any team that is working toward a common goal
- In particular, scrum is a collection of meetings, roles and tools that work together to help teams to better structure and manage their workload

The framework

- People often think Scrum and agile are the same thing because Scrum is centered around continuous improvement, which is a core principle of agile
- However, Scrum is a framework for getting work done, where agile is a mindset
- You can't really *"go agile"*, as it takes dedication from the whole team to change the way they think about delivering value to your customers
- But you can use a framework like Scrum to help you start thinking that way and to practice building agile principles into your everyday communication and work

The framework



Scrum artifacts

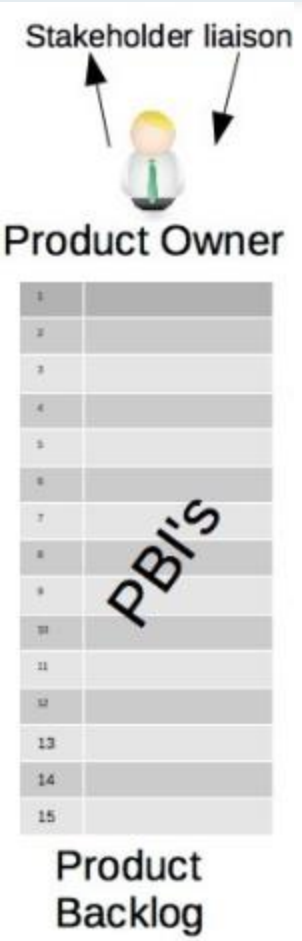
1. Product Backlog
2. Sprint Backlog
3. Increment (or Sprint goal)

Scrum artifacts: Product Backlog

- **Product Backlog** is the master list of work that needs to get done 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
- It is, essentially, the team's *"To Do"* list

- The product backlog is constantly revisited, re-prioritized and maintained by the Product Owner because, as we learn more or as the market changes, items may no longer be relevant, or problems may get solved in other ways



Scrum artifacts: Sprint Backlog

- **Sprint Backlog** 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 the team chooses which items it will work on for the sprint from the product backlog
- A sprint backlog may be flexible and can evolve during a sprint
- However, the fundamental sprint goal – what the team wants to achieve from the current sprint – cannot be compromised

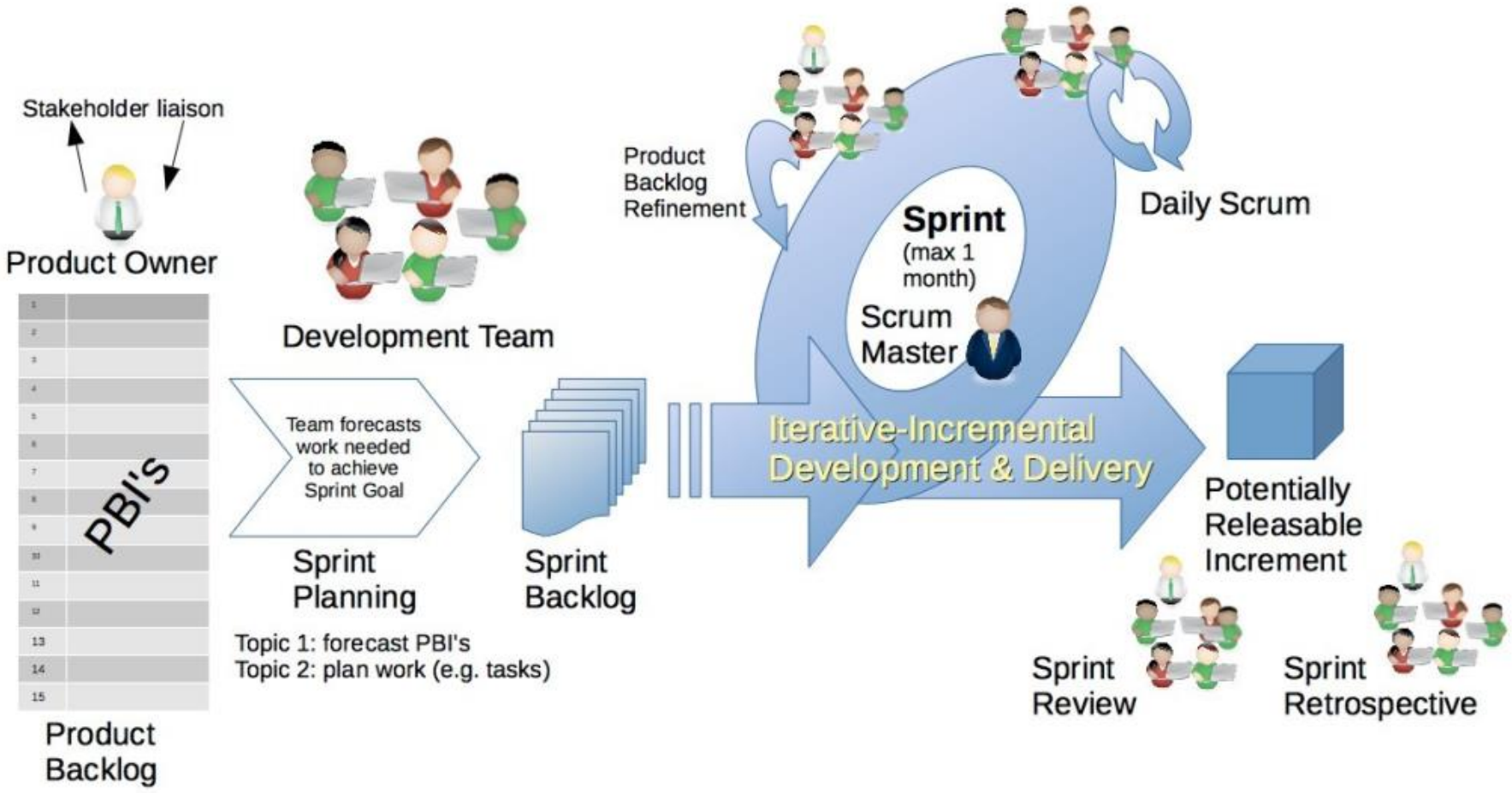
Scrum artifacts: Increment (or Sprint goal)

- **Increment (or Sprint Goal)** is the usable end-product from a sprint
- We usually demonstrate the *"increment"* during the end-of-sprint demo, where the team shows what was completed in the 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

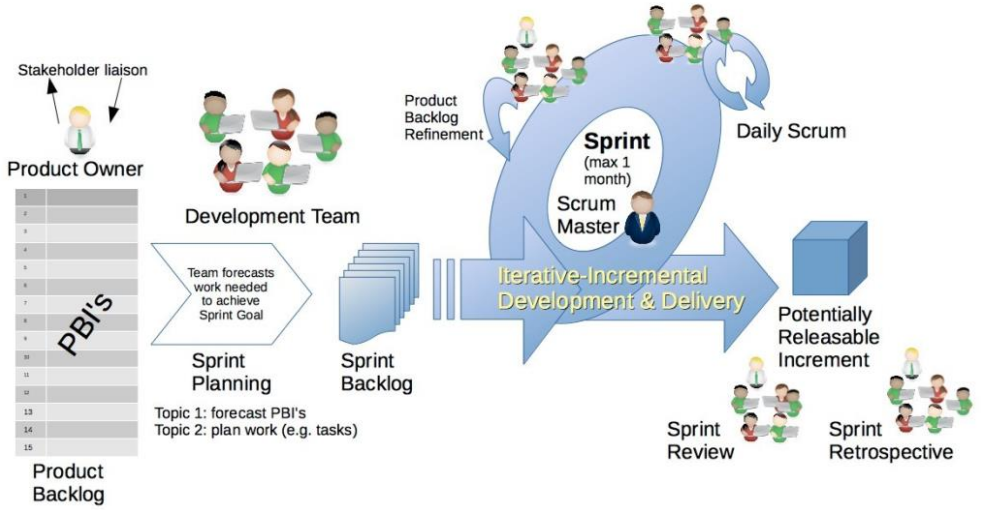
Scrum artifacts: Increment (or Sprint goal)

- For example, some teams choose to release something to their customers at the end of every sprint
- So their definition of *'done'* would be *'shipped'*
- However, this may not be realistic of other types of teams
- Say you work on a server-based product that can only ship to your customers every quarter
- You may still choose to work in 2-week sprints, but your definition of *'done'* may be finishing part of a larger version that you plan to ship together
- But of course, the longer it takes to release software, the higher the risk that software will miss the mark

Scrum Process, ceremonies and events

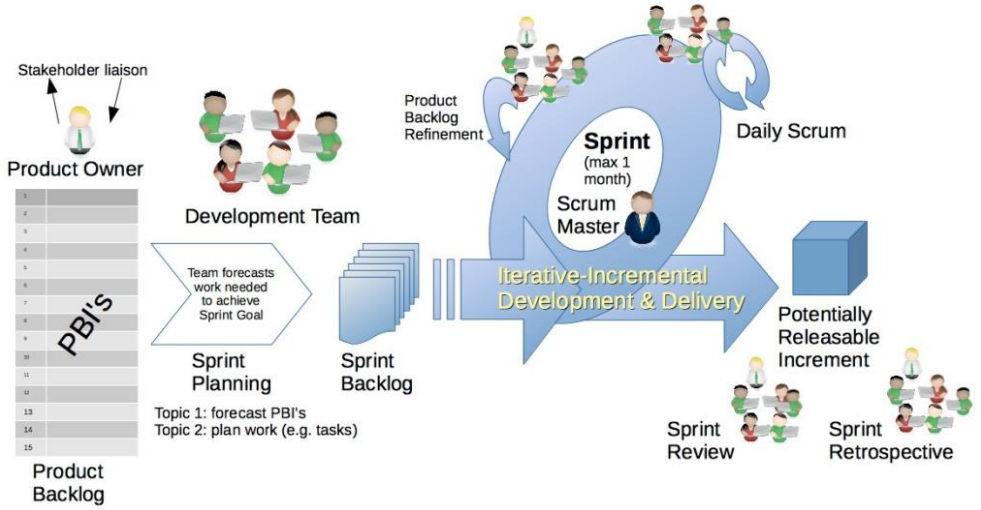


Scrum Process:
1. Organize the backlog



- Sometimes known as backlog grooming, this event is the responsibility of the **product owner**
- The product owner's main jobs are to drive the product towards its product vision and have a constant pulse on the market and the customer
- Therefore, he/she maintains this list using feedback from users and the development team to help prioritize and keep the list clean and ready to be worked on at any given time

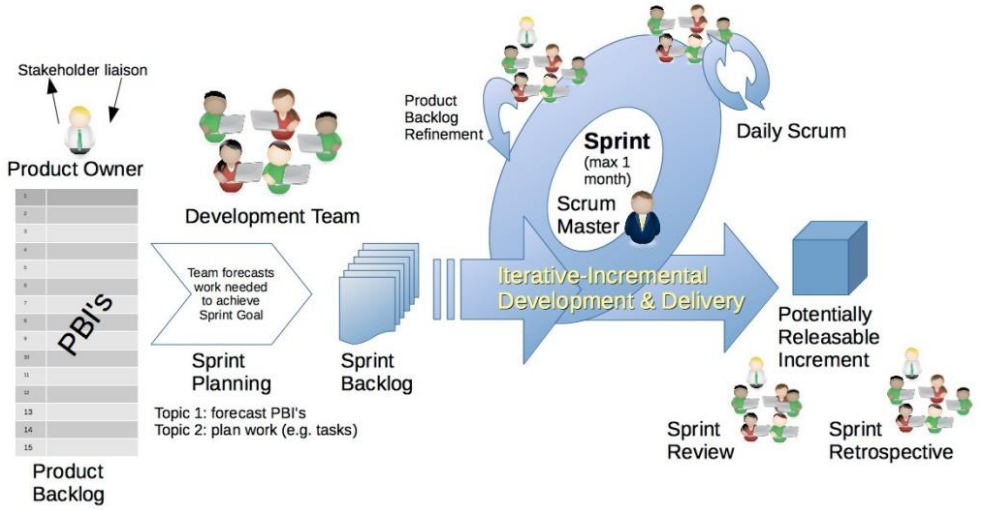
Scrum Process:
2. Sprint planning



- The work to be performed (scope) during the current sprint is planned during this meeting by the entire development team
- This meeting is led by the **scrum master** and is where the team decides on the sprint goal
- Specific use stories are then added to the sprint from the product backlog
- These stories always align with the goal and are also agreed upon by the scrum team to be feasible to implement during the sprint
- At the end of the planning meeting, every scrum member needs to be clear on what can be delivered in the sprint and how the increment can be delivered

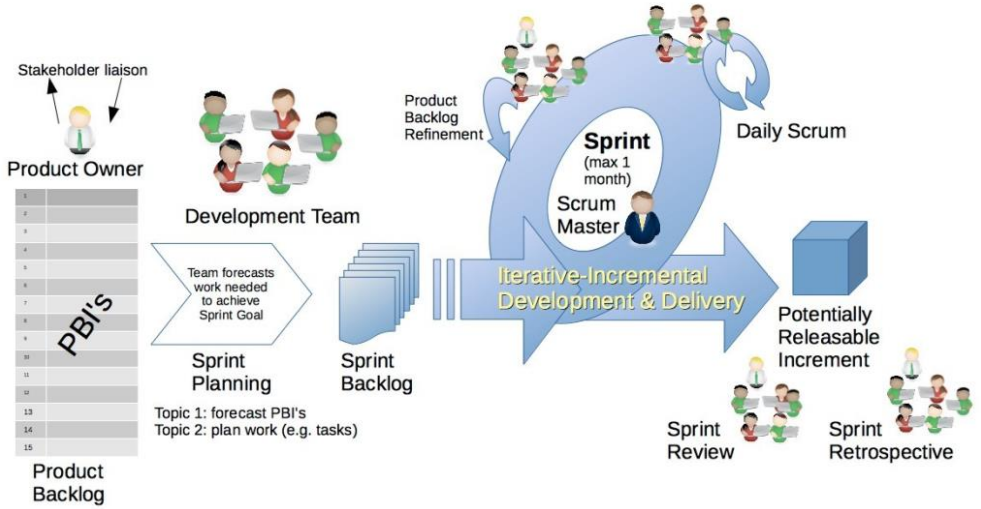
Timebox 2-4 hours
PBIs are prioritized, selected
PBIs are decomposed
Commitment is made on filled Sprint Backlog

Scrum Process:
3. Sprint



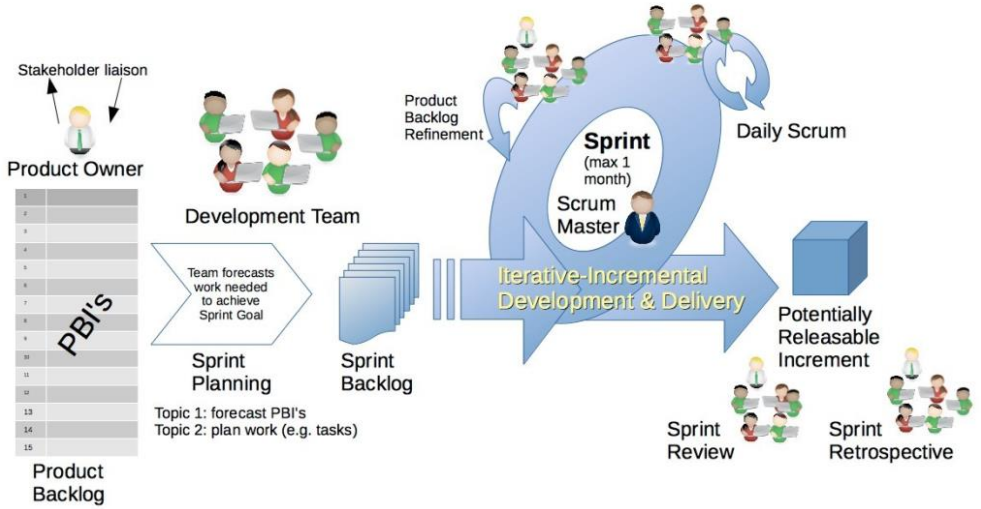
- A sprint is the actual time period when the **scrum team** works together to finish an increment
- Two weeks is a pretty typical length for a sprint, though some teams find a week to be easier to scope or a month to be easier to deliver a valuable increment
  - Dave West, from Scrum.org advises that the more complex the work and the more unknowns, the shorter the sprint should be
  - But it's really up to your team, and you shouldn't be afraid to change it if it's not working!
- During this period, the scope can be re-negotiated between the product owner and the development team if necessary
- This forms the crux of the empirical nature of scrum

Scrum Process:
4. Daily Scrum or Stand Up



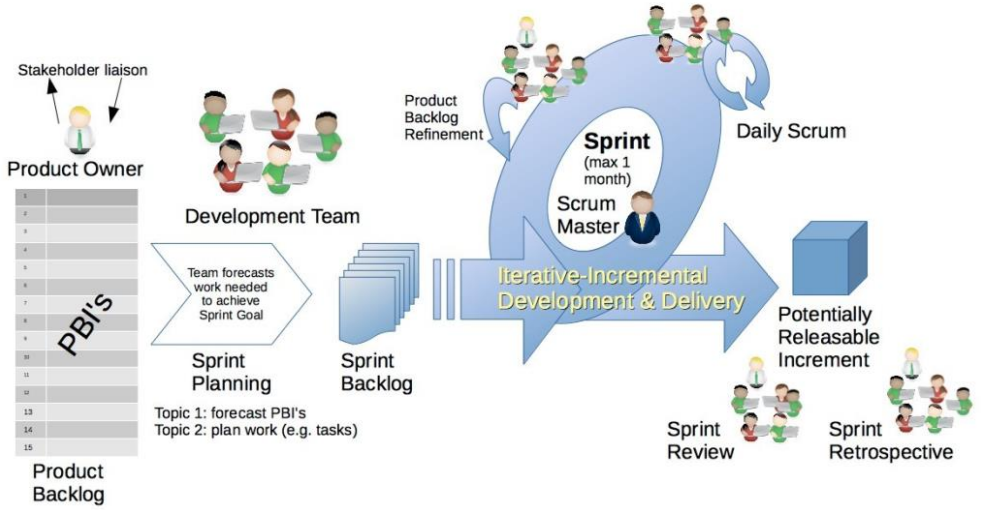
- This is a daily super-short meeting that happens at the same time (usually mornings) and place to keep it simple
- Many teams try to complete the meeting in 15 minutes, but that's just a guideline
- This meeting is also called a *'daily stand-up'* emphasizing that it needs to be a quick one
- The goal of the daily scrum is for everyone on the team to be on the same page, aligned with the sprint goal, and to get a plan out for the next 24 hours
- **The stand up is the time to voice any concerns you have with meeting the sprint goal or any blockers**

Scrum Process:
4. Daily Scrum or Stand Up



- A common way to conduct a stand up is for every team member to answers three questions in the context of achieving the sprint goal:
  - **What did I do yesterday?**
  - **What will I do today?**
  - **Are there any problems, blocking issues?** → job for Scrum Master

Scrum Process:
5. Sprint review



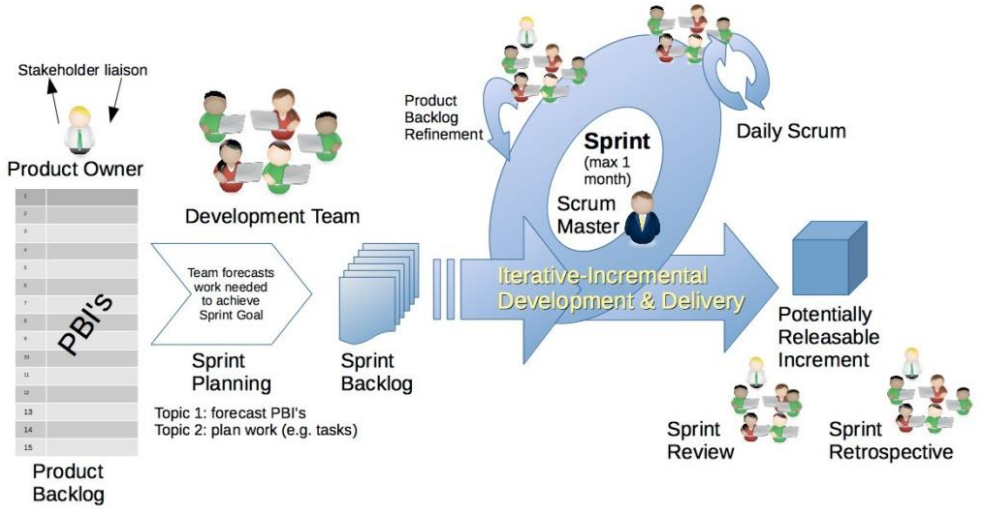
- At the end of the sprint, the team gets together for an informal session to view a demo of, or inspect, the increment
- The development team showcases the backlog items that are now *'Done'* to stakeholders and teammates for feedback
- The product owner can decide whether or not to release the increment, although in most cases the increment is released

Timebox 1 hour

What was completed?

Demo (Incomplete work doesn't count)

Scrum Process:
6. Sprint retrospective



- The retrospective is where the team comes together to document and discuss what worked and what didn't work in a sprint, a project, people or relationships, tools, or even for certain ceremonies
- The idea is to create a place where the team can focus on what went well and what needs to be improved for the next time, and less about what went wrong

Timebox 1 hour

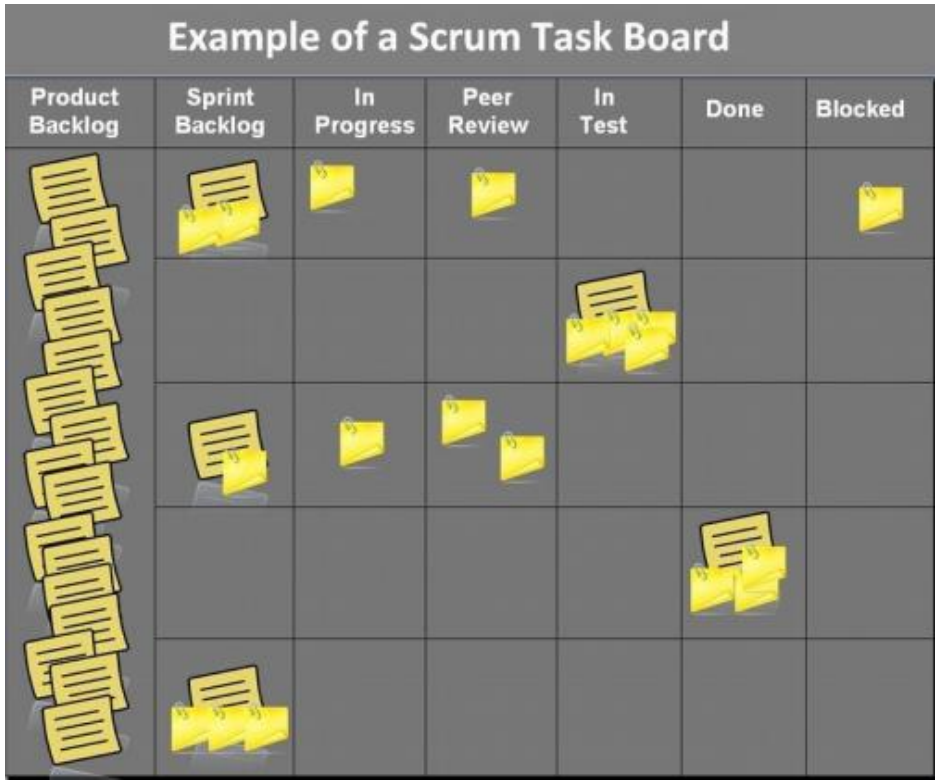
Questions:

- What went well?
- What went wrong?

Agreement on team process improvements

Essential roles for scrum success

1. Product owner
2. Scrum master
3. Development team



Essential roles for scrum success
1. 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

Essential roles for scrum success
2. The scrum master

- Scrum masters are the champions for 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
- *An effective scrum master deeply understands the work being done by the team and can help the team optimize their transparency and delivery flow*
- As the facilitator-in-chief, he/she schedules the needed resources (both human and logistical) for sprint planning, stand-up, sprint review, and the sprint retrospective

Essential roles for scrum success
3. The scrum development team

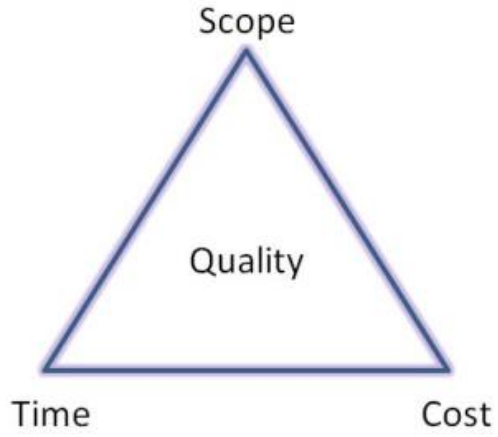
- They are the champions for sustainable development practices
- The most effective scrum teams are tight-knit, co-located, and usually five to seven members
- One way to work out the team size is to use the famous ‘two pizza rule’ coined by Jeff Bezos, the CEO of Amazon (the team should be small enough 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
- Strong scrum teams are self-organizing and approach their projects with a clear ‘we’ attitude
- All members of the team help one another to ensure a successful sprint completion

Why scrum?

- The scrum framework itself is **simple**
  - The rules, artifacts, events, and roles are **easy to understand**
- Its semi-prescriptive approach actually helps remove the ambiguities in the development process, while giving sufficient space for companies to introduce their individual flavor to it
- The organization of complex tasks into manageable user stories makes it ideal for difficult projects
- Also, the clear demarcation of roles and planned events ensure that there is transparency and collective ownership throughout the development cycle
- **Quick releases keep the team motivated and the users happy as they can see progress in a short amount of time**

Scrum – critical approach

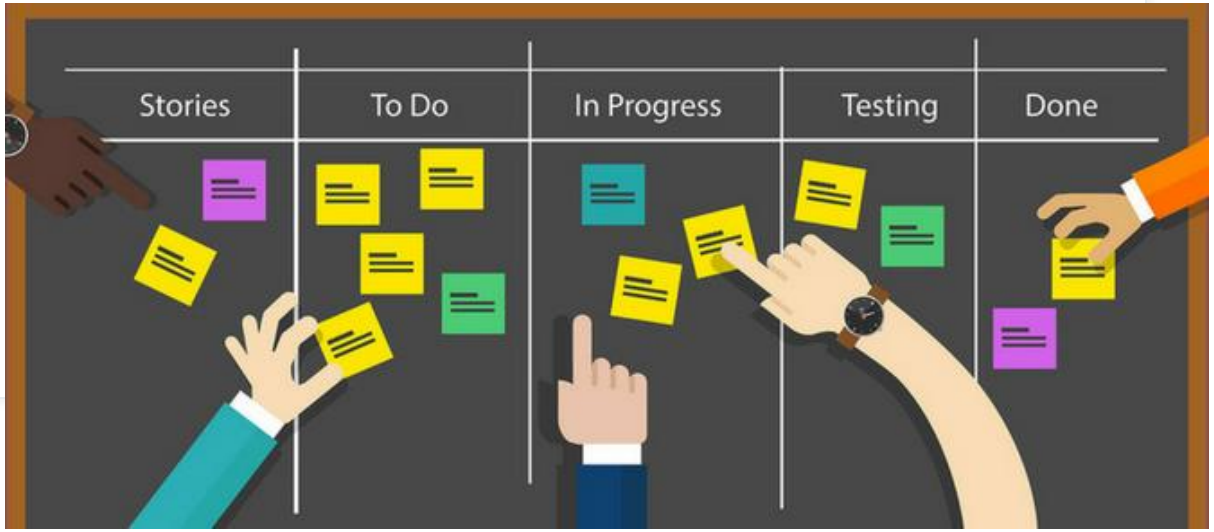
- Co-location
- Team responsibility: anyone can do anything, but cross-functional teams have specialized members
- Triple Constraint (Project Management Triangle) – all of them cannot be fixed
- Measure Velocity (Agility)?
  - Statistics (commit number, fixed tickets… highly debatable)
  - Surveys (better for moral detection)



Kanban (billboard)

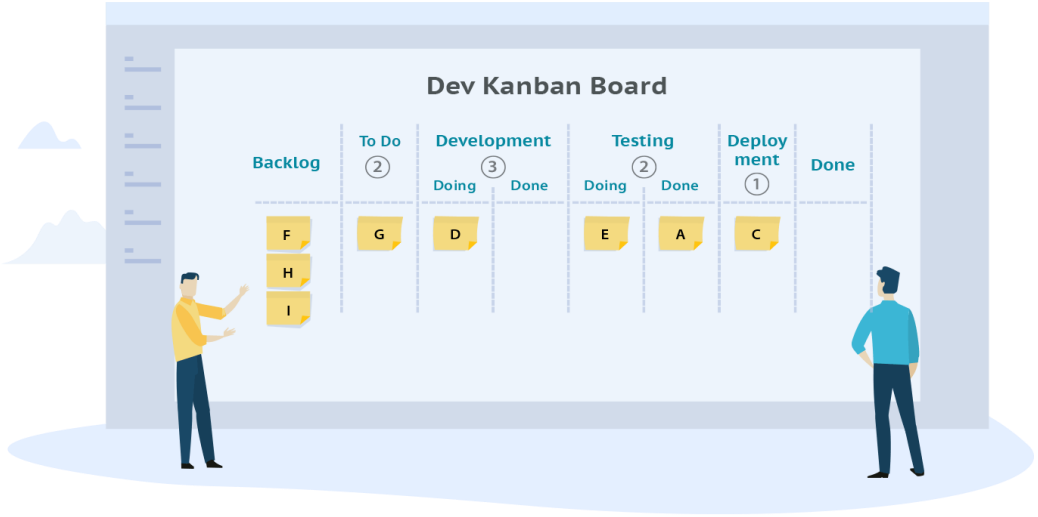
- Kanban is not as structured as scrum
- Other than the Work In Progress limit, it is fairly open to interpretation
- Scrum, however, has several categorical concepts enforced as part of its implementation such as sprint review, retrospective, daily scrum, etc.
- It also insists on cross-functionality, which is the ability of a scrum team to not depend on external members to achieve their goals
- Putting together a cross-functional team is not straightforward
- In that sense, Kanban is easier to adapt whereas scrum can be considered as a fundamental shift in the thought process and functioning of a development team

Kanban

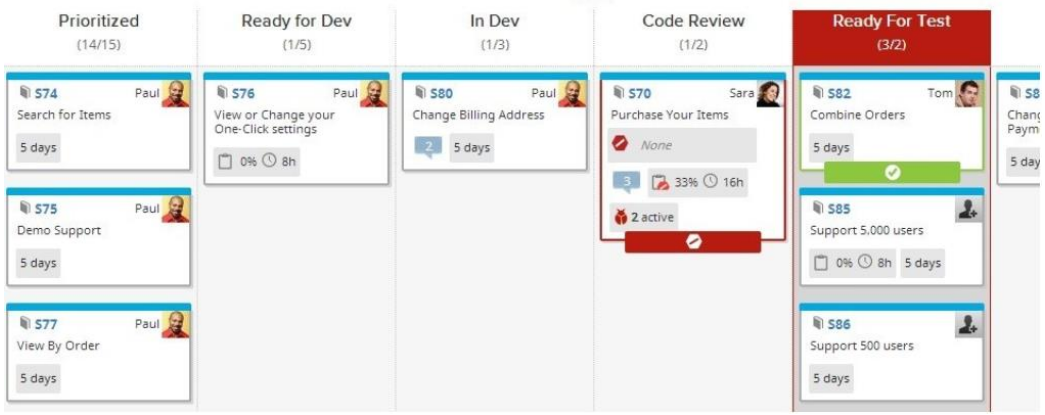


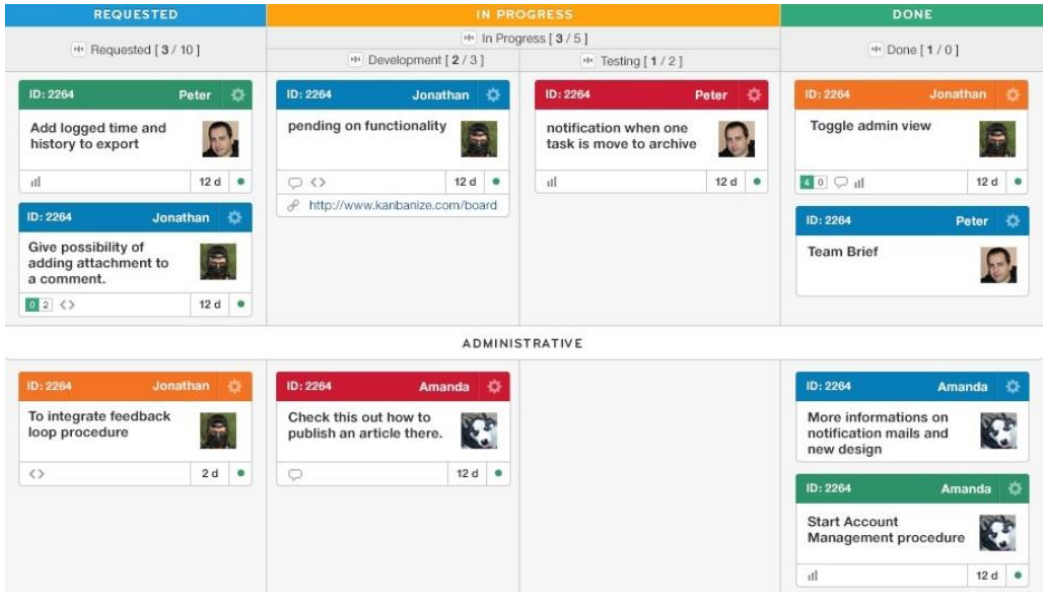
- Lean manufacturing or Just-In-Time manufacturing from Toyota
- A visual inventory control system for supply chain management
- Idea: software development management = supply chain management (flow management)

Kanban board examples

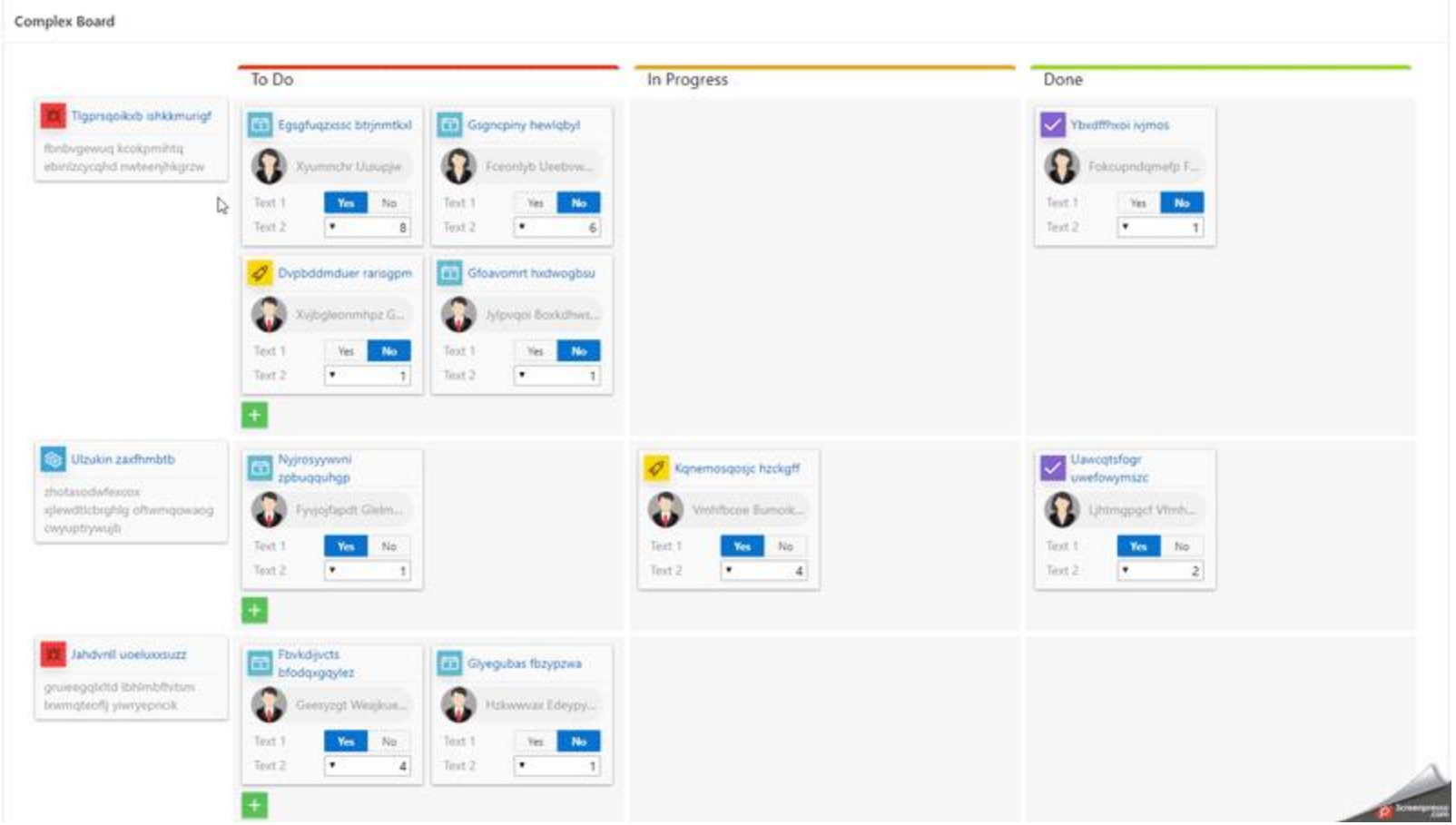


Kanban board examples

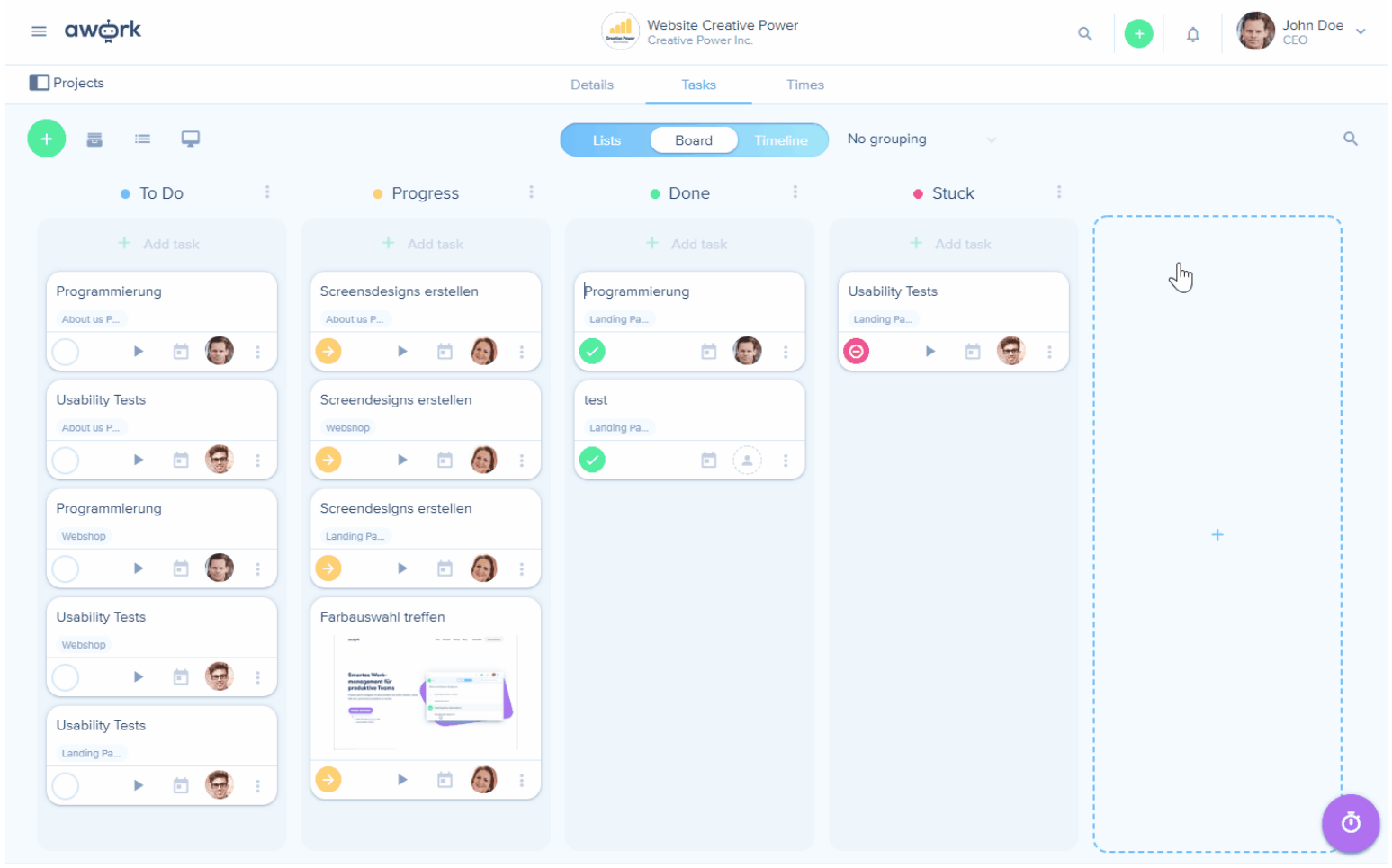




Kanban board in production

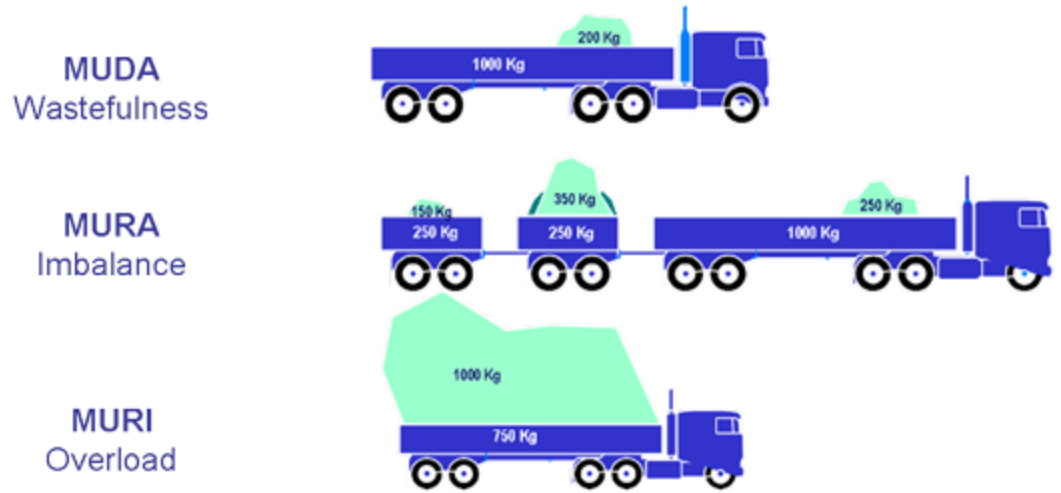


Kanban board in production



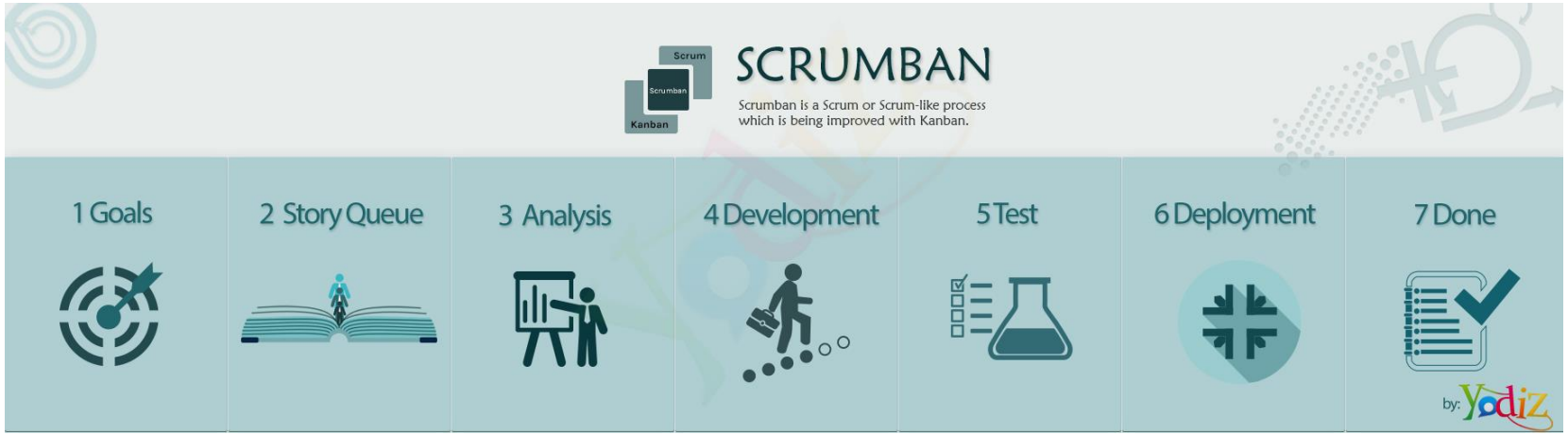
Lean / Kanban Practices

- Eliminate Waste (“Kaizen”)
  - Useless work – “Muda”
  - Unevenness – “Mura”
  - Overwork – “Muri”
- Previous include
  - Task switching overhead
  - Unfinished work
  - Waiting
- Team members pull work (not pushed onto them)
- Quality is fixed (not Scope or Schedule)
- WIP Limit = Work In Progress Limit



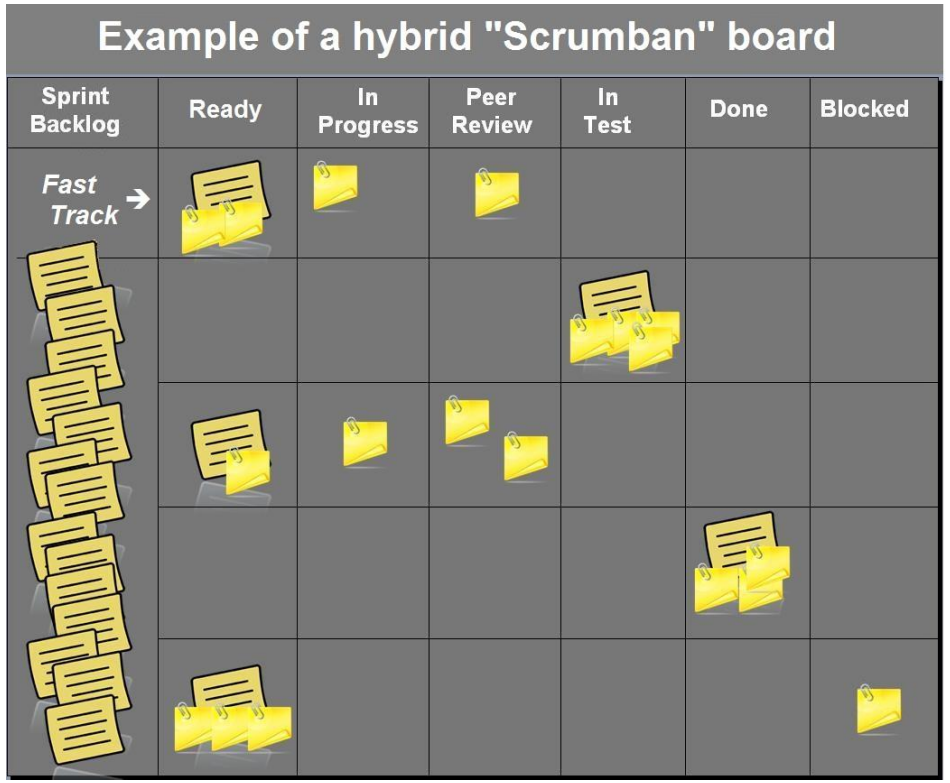
Scrum – "Scrumban" / "Kanplan" – Kanban

- **Scrum** is such a popular agile framework that scrum and agile are often misunderstood to be the same thing
- But there are other frameworks, like **Kanban**, which is a popular alternative
- Some companies even choose to follow a **hybrid model of Scrum and Kanban**, which has acquired the name of *'Scrumban'* or *'Kanplan'*, which is Kanban with a backlog



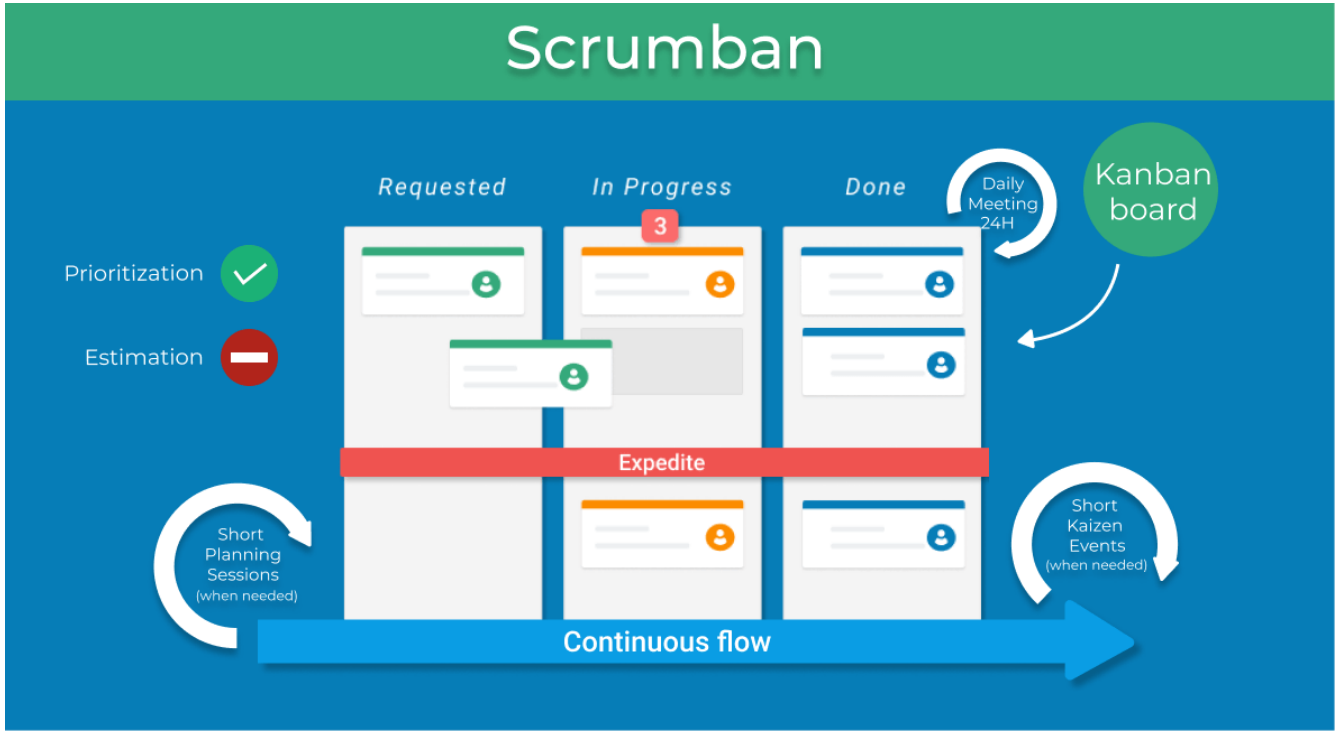
Scrum – “Scrumban” / “Kanplan” – Kanban

- Both Scrum and Kanban use visual methods such as the **Scrum board** or **Kanban board** to track the progress of work
- Both emphasize efficiency and splitting complex tasks into smaller chunks of manageable work, but their approaches towards that goal is different

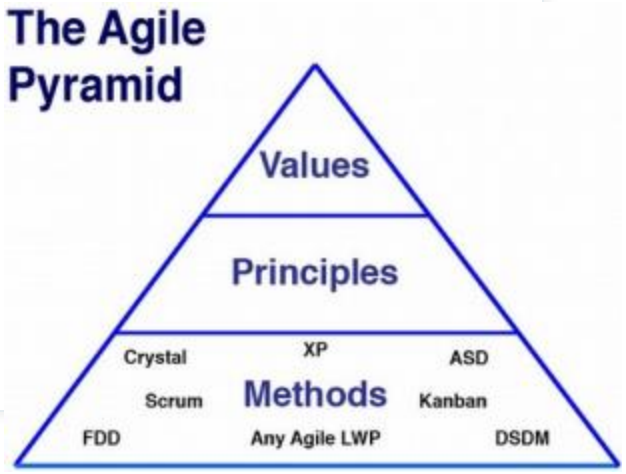


Scrum – “Scrumban” / “Kanplan” – Kanban

- Scrum focuses on smaller, fixed-length iterations
- In Kanban, however, the number of tasks or the work in progress to be implemented in the current cycle is fixed at first



Agile Thoughts



- Communication is important (co-located or not)
- Trust towards individuals, motivate
- Working software instead of documents
- Involve customer
- Be flexible
- Short feedback, frequent releases
- Self-organizing teams are best
- Simplicity – the art of maximizing the amount of work NOT done
- Evolve everything
- Sustainable development

Agile Criticism

- Lack of Plan vs Too Much Preparation
- Fixed time, scope, cost, quality = mission impossible
- Bad management
  - Manager(s) does not understand process (scrum master is a developer)
  - No support from other departments
  - No product owner (just a “clever” developer)
  - Assign tasks early or from outside
- No sufficient automation
- No escape from technical debt
- Too much iterations vs Lost focus (other work)

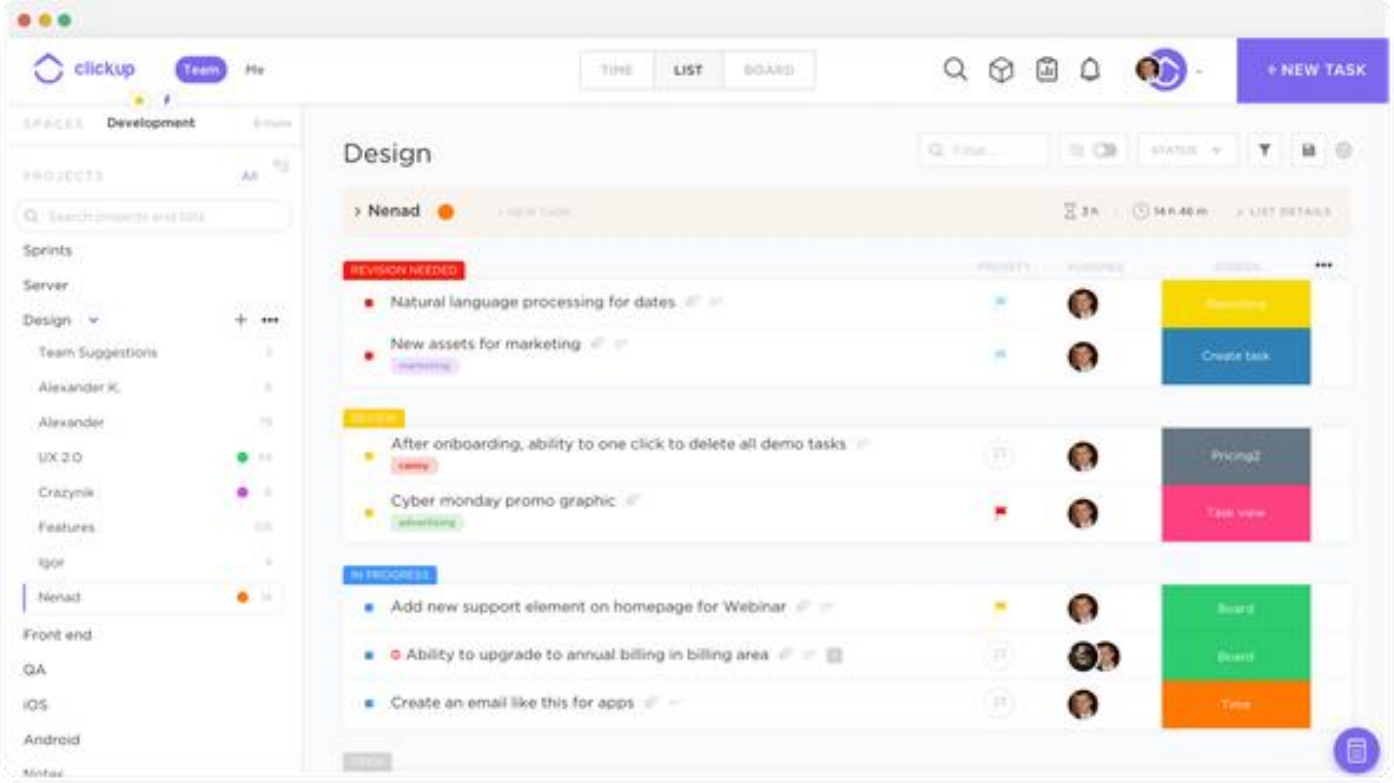
Questions?

- …
- Or write me an email to gla@inf.elte.hu

Scrum tools
Kanban tools

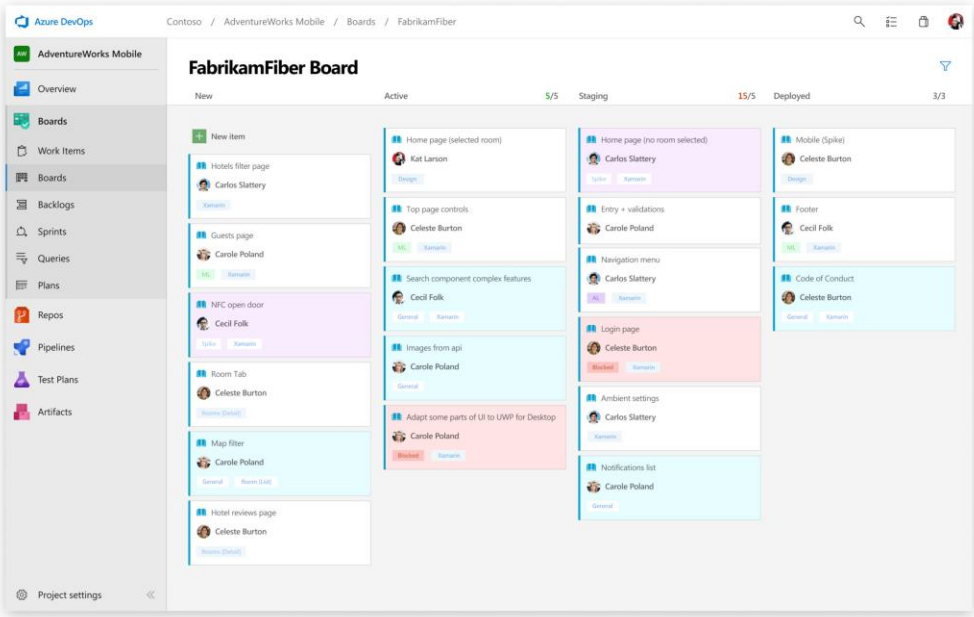
PivotalTracker

Clickup


clickup
Team
Me
TIME
LIST
BOARD
+ NEW TASK
SPACES
Development
PROJECTS
Sprints
Server
Design
Team Suggestions
Alexander K.
Alexander
UX 2.0
Crazynik
Features
Igor
Nenad
Front end
QA
iOS
Android
Design
Nenad
REVISION NEEDED
Natural language processing for dates
New assets for marketing
marketing
After onboarding, ability to one click to delete all demo tasks
Cyber monday promo graphic
advertising
IN PROGRESS
Add new support element on homepage for Webinar
Ability to upgrade to annual billing in billing area
Create an email like this for apps
Create task
Pricing2
Board
Board
Time

Azure Boards (Scrum + Kanban)

- Free for ELTE students (login with your username@inf.elte.hu)
- https://azure.microsoft.com/en-us/services/devops/boards/
- Try it now!



Yodiz (Scrum) – Intro

- **Yodiz** is one of the most comprehensive *Scrum tools* you can opt for. The tool comes with a rich feature set fitting perfectly with your team needs. The elaborate toolkit allows you to manage your epics, backlog, sprints, and releases without getting into the hassle of third-party add-ons.
- A module for epic management lets you track your feature requests in a better way. You can set priorities to define urgency for your team members, so they have a better picture of what needs to be done in due course of time.
- Burndown charts let you evaluate the amount of work that remains during a project lifecycle. Furthermore, these charts and other project metrics allow you to effectively track your releases and manage sprints accordingly.

Yodiz – Key Features

- Sprint management
- Product Backlog management
- Swimlane diagrams for viewing user-stories
- Issue tracker for timely issue management
- Due dates and custom fields
- Burndown charts for monitoring the workflow
- Comments and tagging
- Android and iOS apps
- Third-party application integrations
- **Starter** – free for up to 3 users, unlimited projects, and other core features

https://yodiz.com/

Pivotal Tracker (Scrum) – Intro

- As the name suggests, **Pivotal Tracker** is an agile project management tool which lets you track your project performance as effectively as possible
- The tool comes with all the essential features you need to manage your agile team successfully throughout the project lifecycle
- Through a shared and prioritized backlog, you can collaborate with your team on what needs to be done in due course of time
- This also helps with understanding each and everyone's role in achieving project goals
- Other noticeable features this *Scrum tool* comes with are user-story tracking, iteration management, multiple performance tracking reports, and third-party application integrations

Pivotal Tracker – Key Features

- User story mapping
- Backlog management
- User mentions and story follows
- Searchable task labels
- Burnup charts and cumulative flow diagrams
- Workspaces for multiple projects
- Project history for tracking decisions
- Third-party application integrations
- **Free** – for up to 3 users, 2 projects, and 2GB storage

https://www.pivotaltracker.com/

Preparation for the Implementation

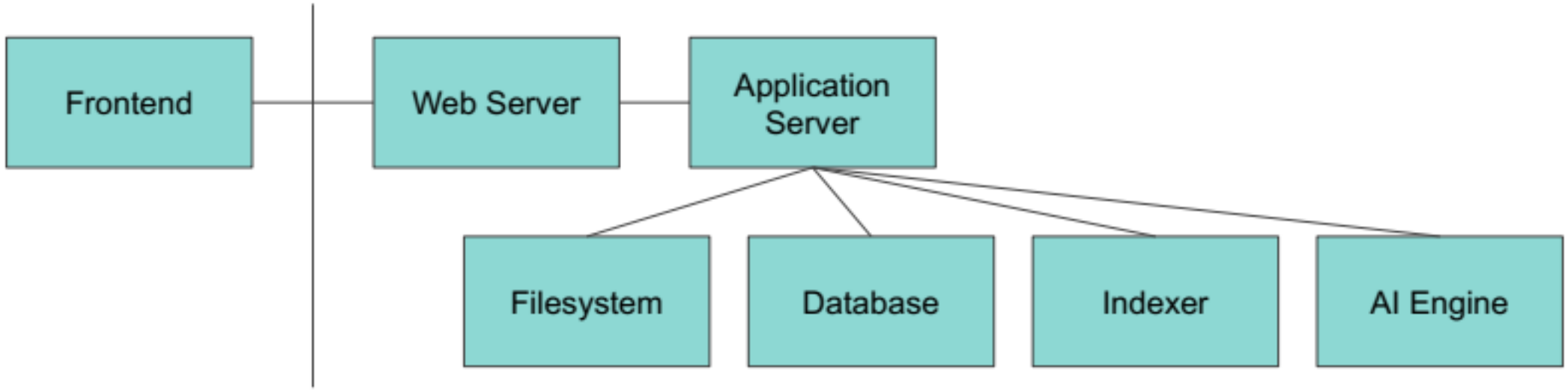
Toolchain

- Communication - Slack (www.slack.com)
- Code repo – Azure DevOps (or Git www.github.com)
- Documents – OneDrive (or Google Drive http://drive.google.com)
- Task management - Azure DevOps (or Trello www.trello.com)

Architecture

- We will do the API based approach
- The advantages:
  - secure
  - cleaner code (the backend is isolated from the frontend)
  - less cost when porting (WEB → Android)
- It also means: **MVC (and similar) is forbidden**

System Components



Infrastructure

- You need to set up the infrastructure that runs the backend
- The open source alternatives (you can choose any technology)
  - OS – Windows or Ubuntu, Debian (www.ubuntu.com)
  - Web server – Apache (https://httpd.apache.org)
  - Application server – Tomcat, WildFly (http://tomcat.apache.org, https://wildfly.org)
  - Database – MySQL, Cassandra (https://www.mysql.com, http://cassandra.apache.org)
  - Queues – Kafka, RabbitMQ (https://www.rabbitmq.com, https://kafka.apache.org)

Development

- You need tools to conduct the development phase in a team
- The open source alternatives (you can choose any technology)
  - IDE – Eclipse (https://www.eclipse.org)
  - Dependency Control – Azure DevOps (or Maven (https://maven.apache.org)
  - Version Control – Azure DevOps (or Git https://git-scm.com)

Maven

- A project management tool, basically a standard. See: https://maven.apache.org
- Some IDE supports it. However, you can do it also command line (CLI).
- Defines a folder structure of your project
  - Based on an artifact (template)
  - A pom.xml file is in the root of the project
- Dependency control
  - Central repository of jar files. Resolves files automatically.
  - Recursive resolution

Git

- A version control that supports teamwork
  - Not trivial, do some testing / trial
  - See: https://git-scm.com
- Version control - automatic versioning of commits
- Collaboration
  - Merges the projects
  - Detects the collisions and tries to merge them
- Distributed repository
  - The repo is cloned - redundant
  - You can change the master repo easily

Backend

- You need a technology to implement the backend

- The open source alternatives *(you can choose any technology)*
  - Java
  - J2EE
  - C# / ASP.NET WEB API

Apache

- See: https://httpd.apache.org
- The most popular web server
- Highly configurable, stable
- Manages virtual servers
- Designed to serve static files
- Put it in the front of your application server to protect it against vulnerabilities

WildFly

- See: https://wildfly.org
- Application server, you can implement a logic to serve the HTTP calls
- Runs the Java code
- J2EE container, supports an abstraction layer
  - To define database connection
  - To upload web applications
  - To configure modules (logging, ...)

J2EE

- JDK based
- JDBC support
- EJB support (transactions, scheduling, JPA)
- Java Servlet API support
- Websocket API
- Dependency Injection (Inversion of Control)
- For a more complete list, see:
  - https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

ASP.NET WEB API

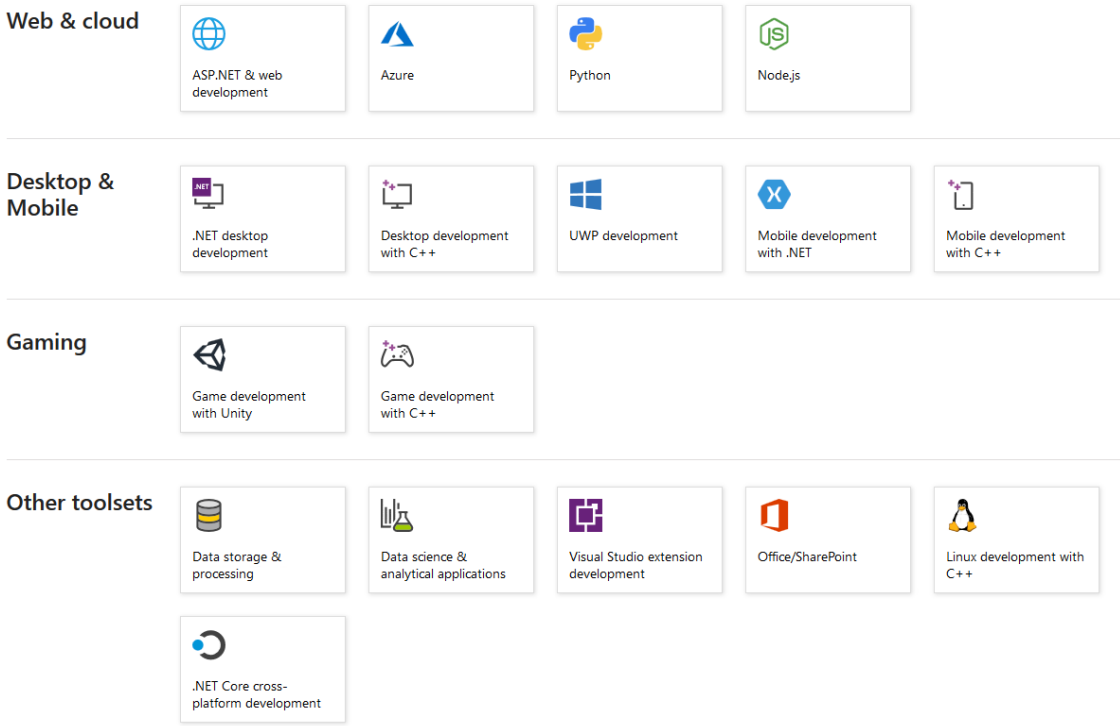
- Build secure REST APIs on any platform with C#
- Simple serialization
- Authentication and authorization
- Routing alongside your code
- HTTPS by default
- https://dotnet.microsoft.com/apps/aspnet/apis

Eclipse

- See: https://www.eclipse.org
- Java based IDE for various programming languages
- Several plugins, extensible (also with a marketplace)
- Supports Maven
- Excellent for J2EE development
- Supports refactoring
  - https://martinfowler.com/books/refactoring.html
- A bit instable Java debugger
- Can be a good choice for a web-based project

Visual Studio

- Develop with the entire toolset from initial design to final deployment
- Improved IntelliSense performance for C++ files
- Local development with many common emulators
- Simplified test access in Solution Explorer
- Git management and repo creation in the IDE
- Kubernetes support now included in Microsoft Azure workload



MySQL

- See: https://www.mysql.com/
- The most popular open source + free database server
- Stable, high performance
- Has to be fine tuned
- Clusterable
- Admin interface: WorkBench, PhpMyAdmin

Backend hints

- Find / develop a server-side framework for back-end development
- Should rely on
  - CDI (IoC)
  - JAX-WS
  - Generics would be nice
- Dao – Service – Access architecture
- J2EE compliant

Frontend

- You need a technology to implement the frontend.
- The open source alternatives (you can choose any technology)
  - Angular – https://angular.io
  - Bootstrap – https://getbootstrap.com
  - Android Studio – https://developer.android.com/studio
  - React Native – https://reactnative.dev/

Homework

- The homework is
- Design the API - list of methods - Canvas submission (API design)
  - URL
  - Signature (name, parameters, return value, exceptions)
- Develop a workflow of your choice on the frontend - Prepare for demo
  - No need to connect it to the backend
  - For example: registration + login
- The deadline is next Monday, 14:00.