PRACTICAL WORKSHOP ON AGILE SOFTWARE DEVELOPMENT

Biniam Asnake
Lead Software Engineer

biniamasnake@gmail.com
https://github.com/PracticalSoftwareEngineeringWorkshop

EXECUTIVE SUMMARY

- The workshop will focus on **practical** experience sharing about **agile** software development methodology and detail day-to-day implementation of one of the agile methodologies i.e. SCRUM as well as step by step workshop on a **real-world implementation** of a system.
- At the end of the course, trainees will have understanding, experience and knowledge of how American and European IT companies develop applications and be able to develop one using the same methodology and technology stack.

GOAL

TO INCREASE PRACTICAL KNOWLEDGE OF PROFESSIONAL SOFTWARE DEVELOPMENT IN SHEBA VALLEY

BINIAM ASNAKE

- **Lead Software Engineer** at bonial.com in Berlin, Germany
- **MSc in Information Science**, Addis Ababa University
- **Java**, Python, **ReactJS**, NodeJS, **Flutter/Dart**, Android …
- **Trainer** of Agile and Software Engineering
- Blogs on biniamasnake.medium.com

- was **Senior Developer** at Apposit/Paga Tech solutions
- was **Lecturer** at Haramaya University and Lucy University College

OUTLINE

- *Agile Manifesto and Principles (Theory)*
- *Scrum vs Kanban Methodologies/Frameworks (Theory)*
- Git version control system
- Trello project management
- Developing a money transfer application
  - API development using Spring boot and Java
  - Front-end app development using ReactJS (and Javascript)
- Continuous Integration and Continuous Delivery (CI/CD - DevOps)
  - Travis CI
  - Heroku Deployment

> Organisations which design systems
> … are constrained to produce designs
> which are copies of the
> communication structures of these
> organisations.

-Melvin Conway

“

Team designs are the first draft of your architecture.

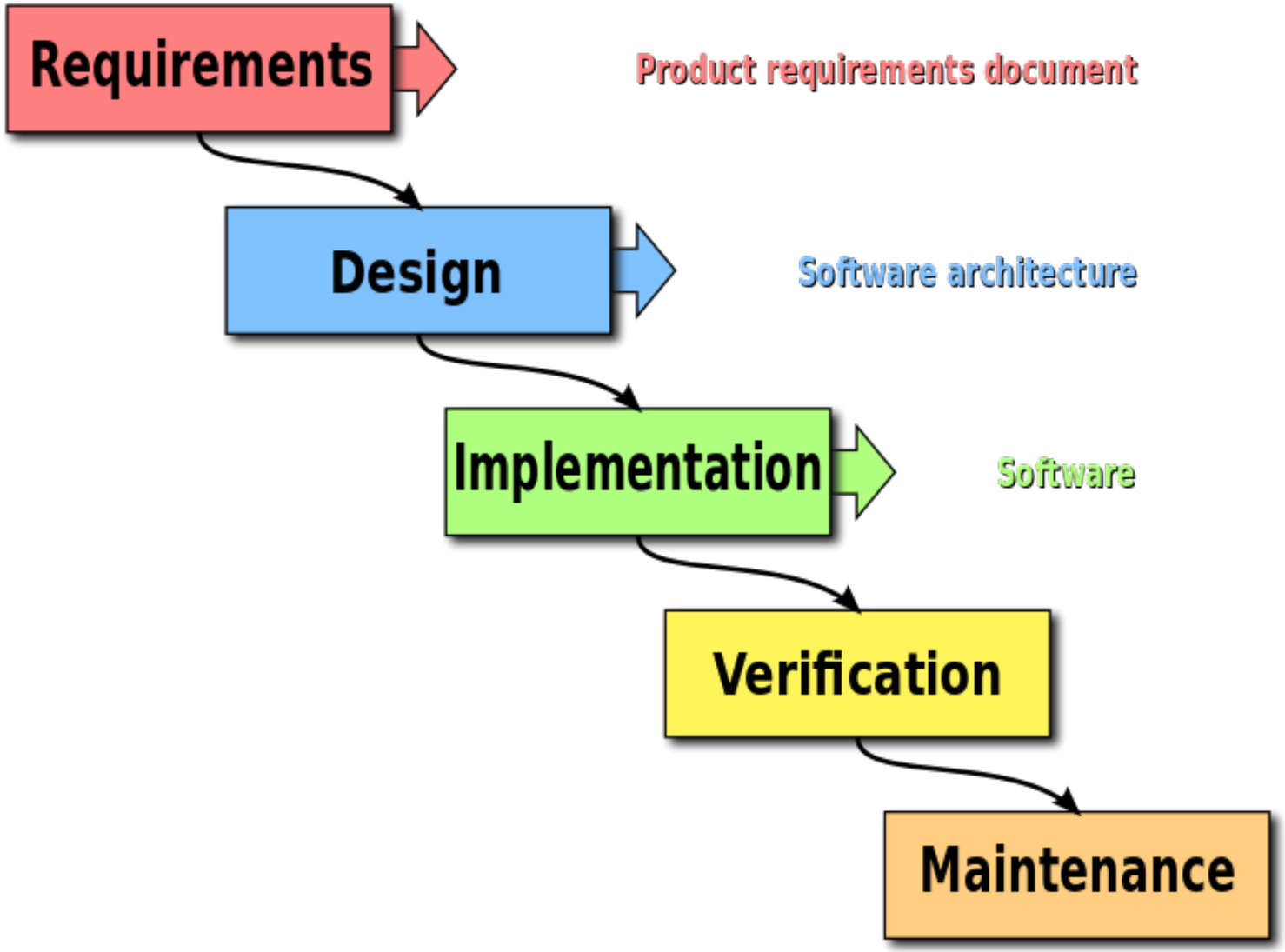
-Michael Nygard (author of RELEASE IT!)

WHAT IS AGILE?

THE ABILITY TO CREATE AND RESPOND TO CHANGE IN ORDER TO SUCCEED IN AN UNCERTAIN AND TURBULENT ENVIRONMENT.

https://www.agilealliance.org/agile101/what-is-agile/

WATERFALL MODEL



What is Agile Software Development?

Agile Software Development is an umbrella term for a **set of methods and practices** based on the values and principles expressed in the **Agile Manifesto**.

Solutions evolve through collaboration between self-organizing, cross-functional teams utilizing the appropriate practices for their context.

https://www.agilealliance.org/agile101/what-is-agile/

A LITTLE BIT OF HISTORY

- In the late 1990's, several methodologies began to gain increasing public attention, each having a different combination of old and new ideas. These methodologies emphasized close collaboration between the development team and business stakeholders; frequent delivery of business value, tight, self-organizing teams; and smart ways to craft, confirm, and deliver code.
- The term "**Agile**" was applied to this collection of methodologies in **early 2001** when **17** software development practitioners gathered in **Snowbird, Utah** to discuss their shared ideas and various approaches to software development. This joint collection of values and principles was expressed in the **Manifesto for Agile Software Development** and the corresponding **twelve principles.**
- The Agile Alliance was formed shortly after this gathering to encourage practitioners to further explore and share ideas and experiences.

AGILE MANIFESTO

www.agilemanifesto.org

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: …

www.agilemanifesto.org

“

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

www.agilemanifesto.org

“

That is, while there is value in the items on the right, we value the items on the left more.

www.agilemanifesto.org

PRINCIPLES BEHIND THE AGILE MANIFESTO

WE FOLLOW THESE 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 customer's 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.

WE FOLLOW THESE PRINCIPLES:

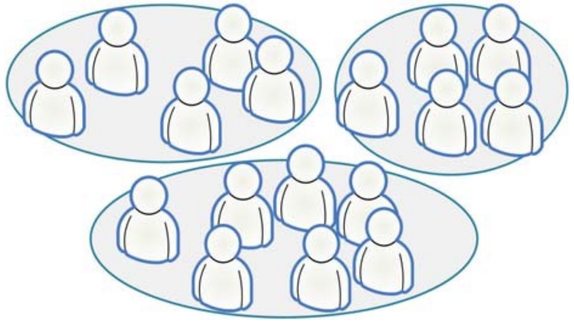
- Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.
- **Working software** is the primary measure of progress.
- Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

WE FOLLOW THESE PRINCIPLES:

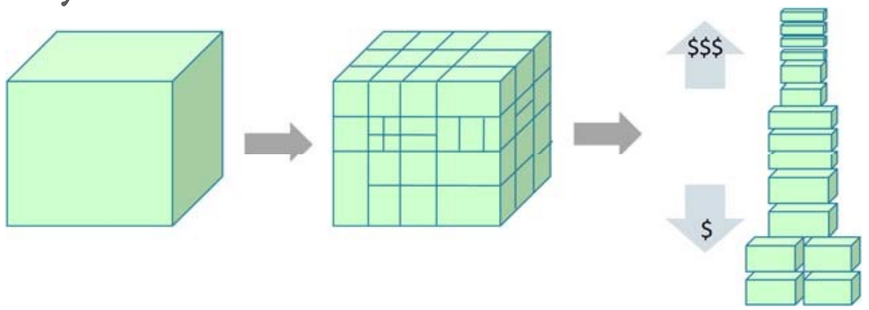
- Continuous attention to **technical excellence and good design** enhances agility.
- **Simplicity**--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from **self-organizing teams.**
- At regular intervals, **the team reflects** on how to become more effective, then tunes and adjusts its behavior accordingly.

SCRUM IN NUTSHELL

- Split your organization into small, cross-functional, self-organizing teams.

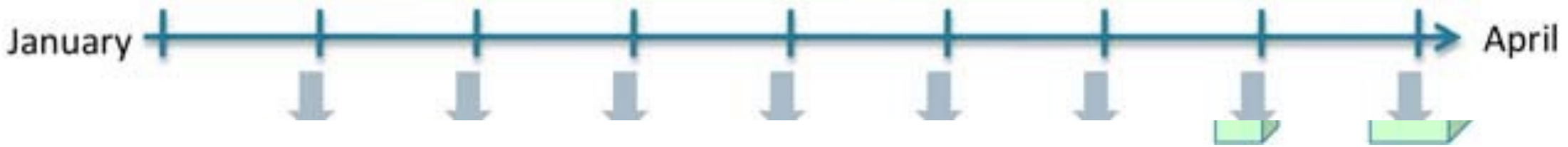


- Split your work into a list of small, concrete deliverables.
  - Sort the list by priority and estimate the relative effort of each item.



. . . SCRUM IN NUTSHELL

- Split time into short fixed-length iterations (usually 1 – 4 weeks), with potentially shippable code demonstrated after each iteration.

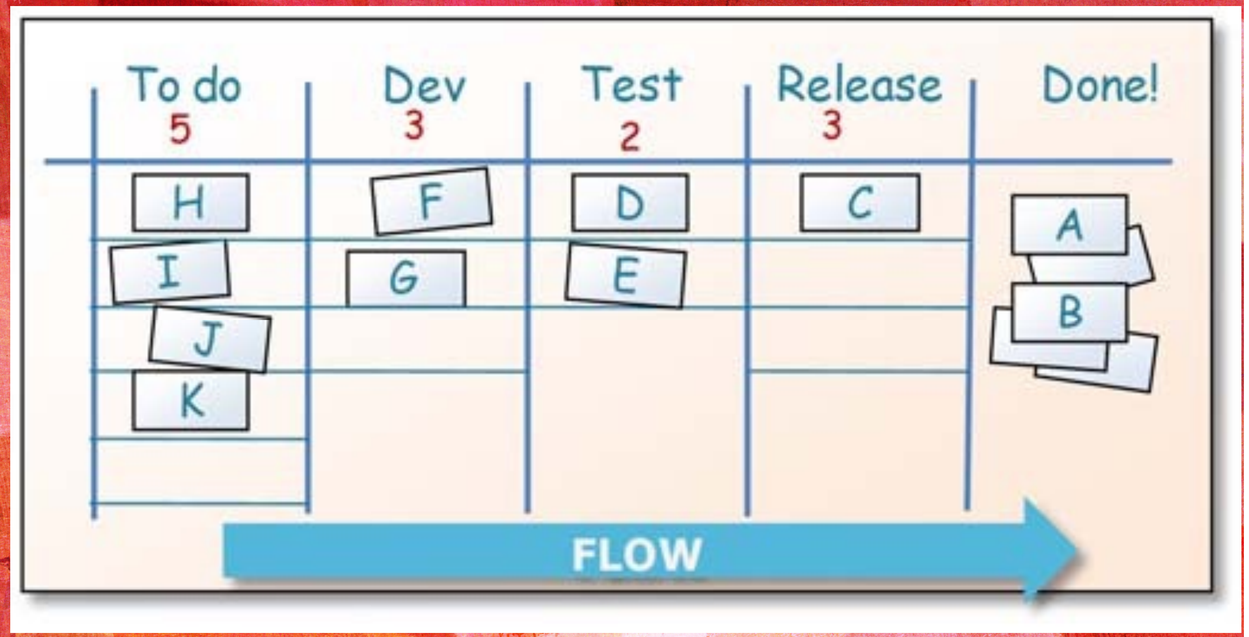


- Optimize the release plan and update priorities in collaboration with the customer, based on insights gained by inspecting the release after each iteration.
- Optimize the process by having a retrospective after each iteration.

“

Instead of a **large group** spending a **long time** building a **big thing**, we have a **small team** spending a **short time** building a **small thing**. But **integrating** regularly to see the whole.

- In Ken Schwaber's words, Scrum is not a methodology, it is a *framework*. What that means is that Scrum is **not really going to tell you exactly what to do.**
- The strength and pain of Scrum is that you are forced to adapt it to your specific situation.



KANBAN IN NUTSHELL

- **Visualize the workflow**
  - ***Split the work into pieces***, write each item on a card and put on the wall.
  - ***Use named columns*** to illustrate where each item is in the workflow.
- **Limit Work In Progress (WIP)** – assign explicit limits to how many items may be in progress at each workflow state.
- **Measure the lead time** (average time to complete one item, sometimes called "cycle time"), optimize the process to make lead time as small and predictable as possible.

SCRUM VS KANBAN

Both Scrum and Kanban are based on incremental development,

i.e. break the work into smaller pieces.

SCRUM VS KANBAN

Compare tools for understanding, not judgment.

Knife or fork – which tool is better?

Scrum and Kanban are neither perfect nor complete.

They **don't tell you everything** that you need to do, they just **provide certain constraints & guidelines.**

The value of a tool is that it limits your options.

E.g.: **Scrum** constrains you to have ***timeboxed iterations*** and ***cross-functional teams***, and

Kanban constrains you to use ***visible boards*** and ***limit the size of your queues***.

➤ Mix and match the tools as you need!

ROLES IN SCRUM

Scrum prescribes 3 roles:

1. Product Owner (sets product vision & priorities),
2. Team (implements the product) and
3. Scrum Master (removes impediments and provides process leadership).

*Kanban doesn't prescribe any roles at all. **Means**: you can but you don't have to.*

Additional Roles?

In both Scrum and Kanban, you are **free to add** whatever additional roles you need.

Be careful! when adding roles though, make sure the additional roles actually add value and don't conflict with other elements of the process.

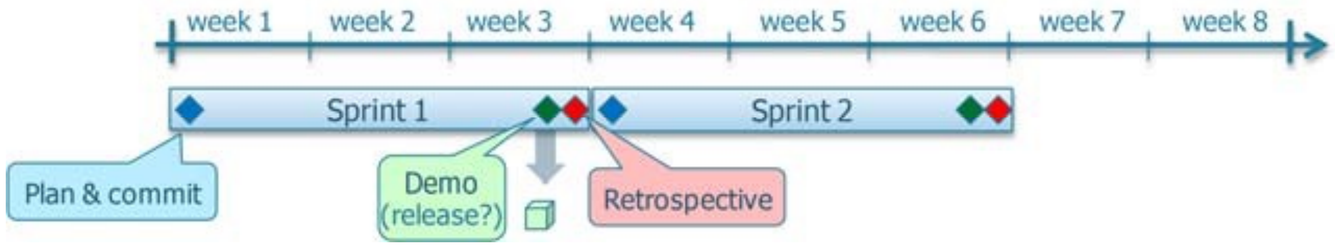
➤ The general mindset in both Scrum and Kanban is "**less is more**". So when in doubt, start with less.

SCRUM: TIMEBOXED ITERATIONS

A Scrum team will **only commit** to items that they can **complete within one iteration** (based on the definition of "Done").

Team #1 (single cadence)

"We do Scrum iterations"

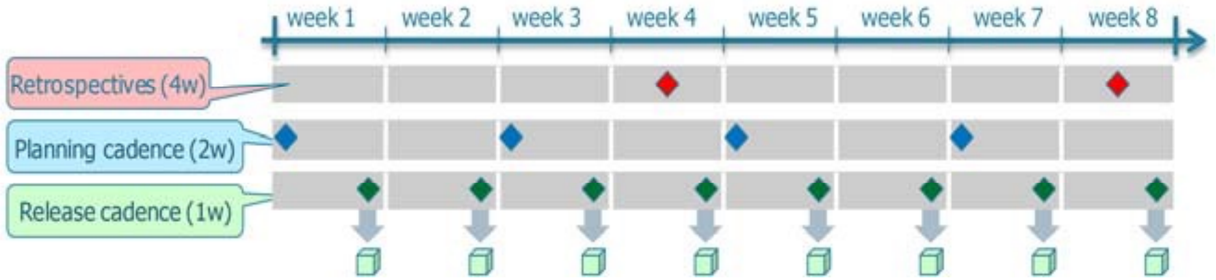


Team #2 (three cadences)

Everyweek: release

Every 2 weeks: planning

Every 4 weeks: retro



Team #3 (mostly event-driven)

Release: whenever there is MMFs

Planning: when run out of stuff to do

Retro: every 4 weeks

➤ Note: retrospective meeting to tweak and improve our process

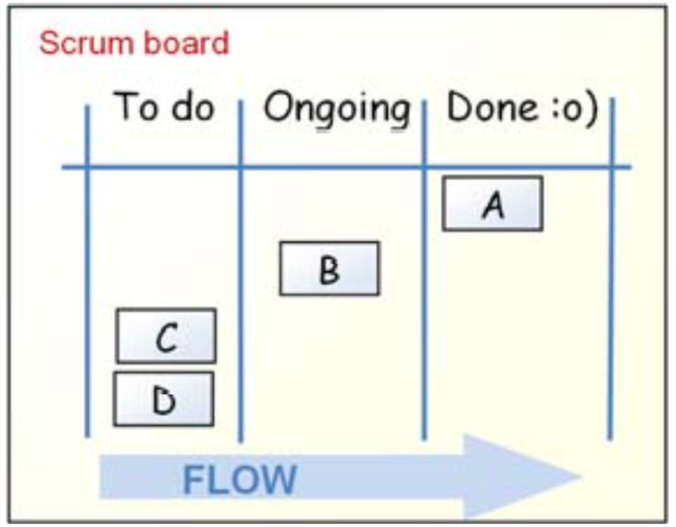
SCRUM VS KANBAN BOARDS

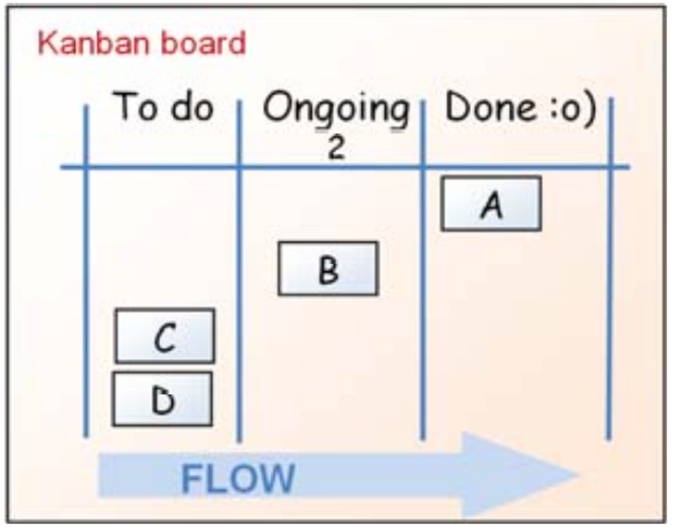
Scrum limits WIP per iteration

Kanban limits WIP per workflow state

(general idea is to

limit WIP of all workflow states)

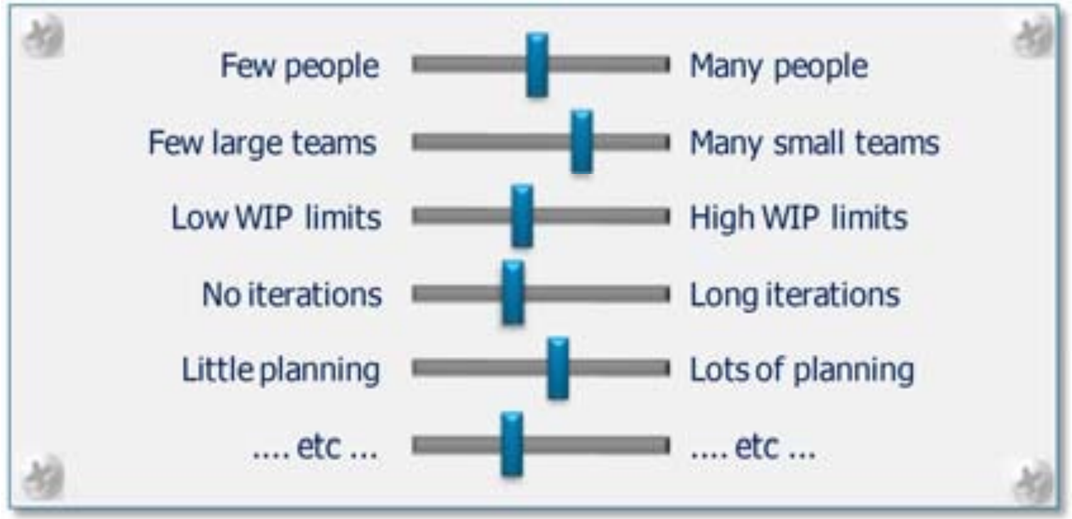




- Every iteration in SCRUM is called **Sprint**
- Scrum teams measure **velocity** – how many items (or corresponding units such as “story points”) get done per iteration.
- Kanban teams measure and predict **lead time**, i.e. the average time for an item to move all the way across the board.
  - Having predictable lead times allows us to commit to **SLAs** (service-level agreements) and make realistic release plans.
    - Too low kanban limit => idle people => bad productivity
    - Too high kanban limit => idle tasks => bad lead time

SCRUM AND KANBAN ARE EMPIRICAL

- You are expected to **experiment** with the process and customize it to your environment.
- Neither Scrum nor Kanban provide all the answers – they just give you a basic **set of constraints** to drive your own process improvement.

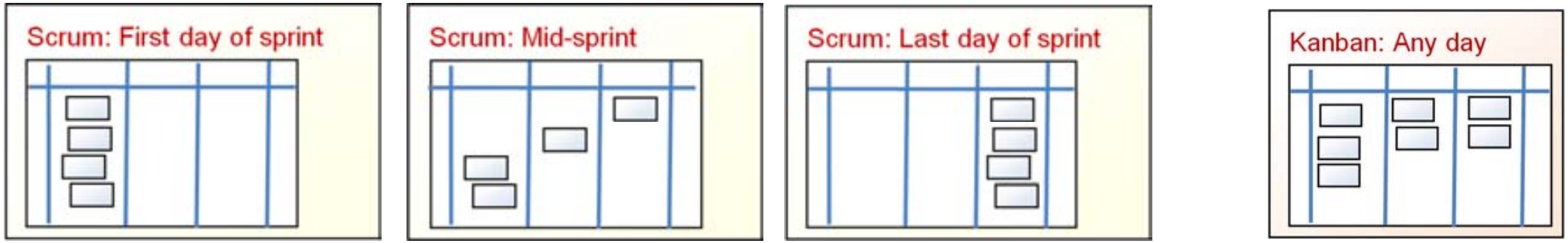


- The most critical element of this is the **feedback loop**.

  Change something => Find out how it went => Learn from it => Change something again.

- You want as short a feedback loop as possible, so you can adapt your process quickly.

THE BOARDS OVER TIME



A **Scrum board** is owned by exactly one Scrum team.

- A Scrum team is **cross-functional**, it contains all the skills needed to complete all the items in the iteration.

In **Kanban**, cross-functional teams are optional, and a board ***doesn't need to be owned by one specific team.***

- A board is related to one workflow, not necessarily one team.

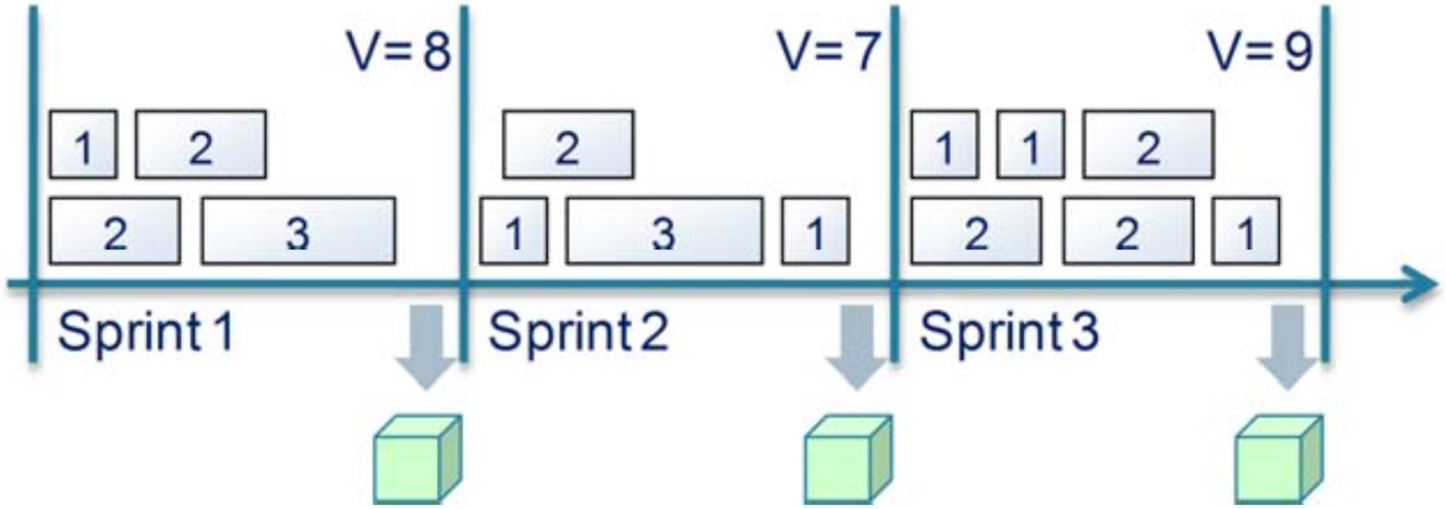
SCRUM PRESCRIBES ESTIMATION AND VELOCITY

Scrum teams estimate the relative size (= amount of work) of each item that they commit to.

Velocity is a measure of capacity – how much stuff we can deliver per sprint.

> *By adding up the size of each item completed at the end of each sprint, we get velocity.*

Knowing the average velocity is nice, because then we can make realistic predictions about which items we can complete in upcoming sprints, and therefore make realistic release plans.



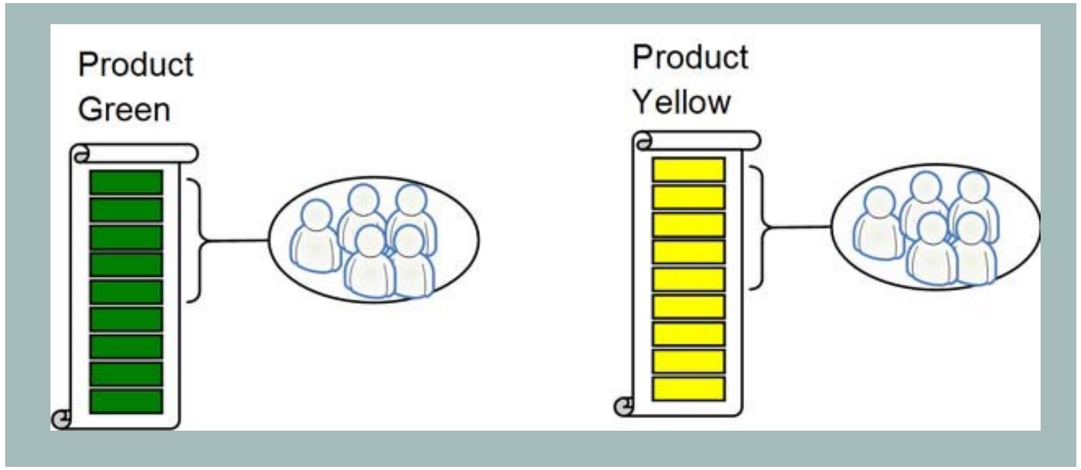
KANBAN DOESN'T PRESCRIBES ESTIMATION AND VELOCITY

In Kanban, estimation is not prescribed. Hence,

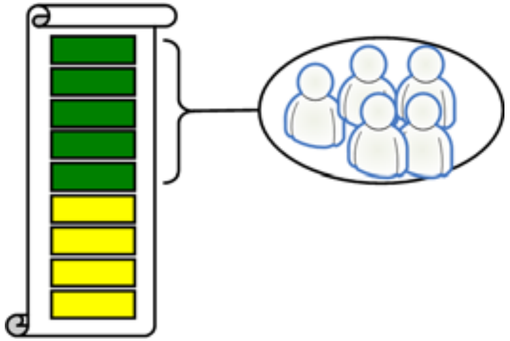
1. Some teams choose to make estimates and measure velocity just like in Scrum.
2. Other teams break items into pieces of roughly the same size and measure velocity simply in terms of how many items were completed per unit of time (for example features per week).
3. Some teams group items into MMFs and measure the average lead time per MMF, and use that to establish Service-Level Agreements (SLAs).

   E.g.: "when we commit to an MMF it will always be delivered within 15 days".

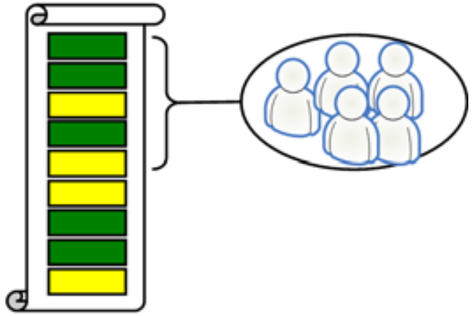
SCRUM AND KANBAN ALLOWS WORKING ON MULTIPLE PRODUCTS SIMULTANEOUSLY



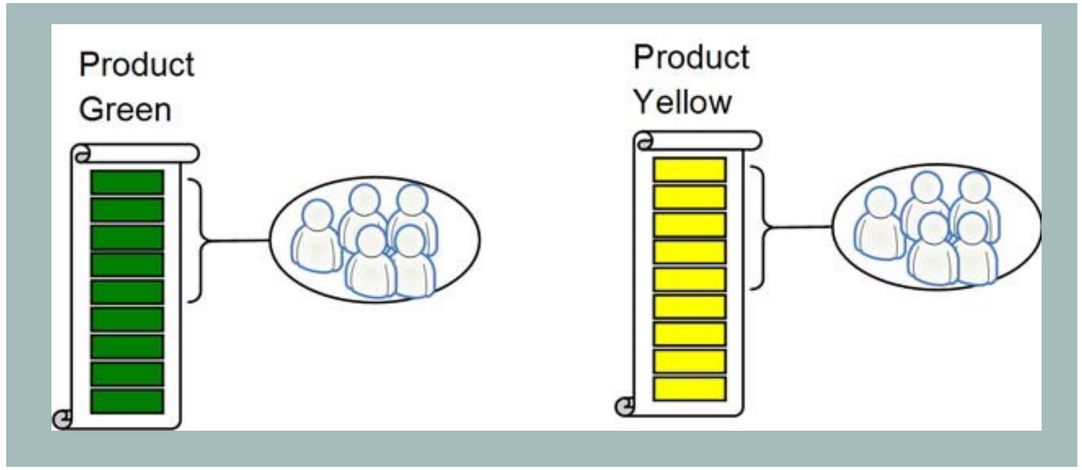
1. team focus on one product per sprint



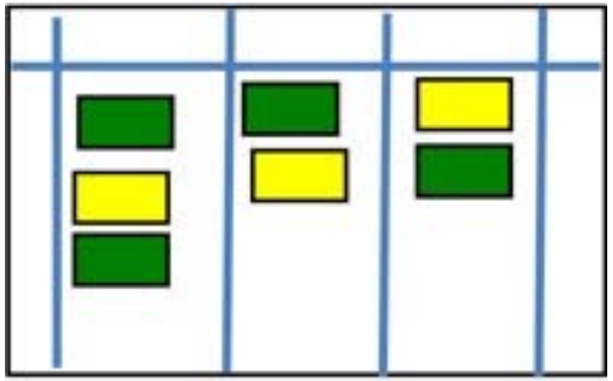
2. work on features from both products each sprint:



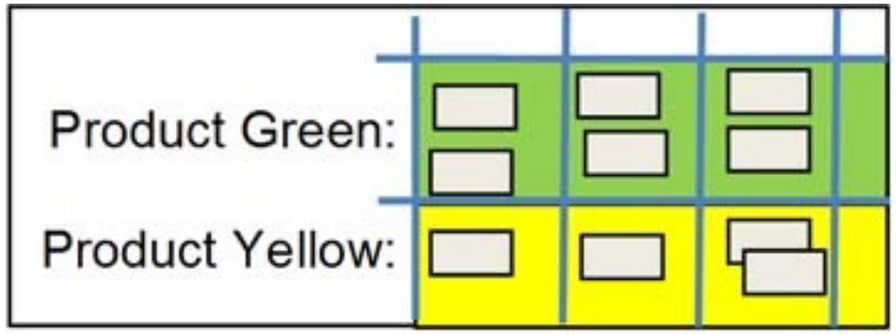
SCRUM AND KANBAN ALLOWS WORKING ON MULTIPLE PRODUCTS SIMULTANEOUSLY



1. coloured cards



2. swimlanes



SCRUM PRESCRIBES PRIORITISED PRODUCT BACKLOG

In Scrum, **prioritisation** is always done by sorting the product backlog, and *changes to priorities take effect in the next sprint* (not the current sprint).

In Kanban you can choose **any** prioritisation scheme (or even **none**), and *changes take effect as soon as capacity becomes available* (rather than at fixed times).

- There ***may or may not be a product backlog***, and it may or may not be prioritised.
- The most left column typically fulfills the same purpose as a Scrum product backlog.

SCRUM PRESCRIBES DAILY STANDUP

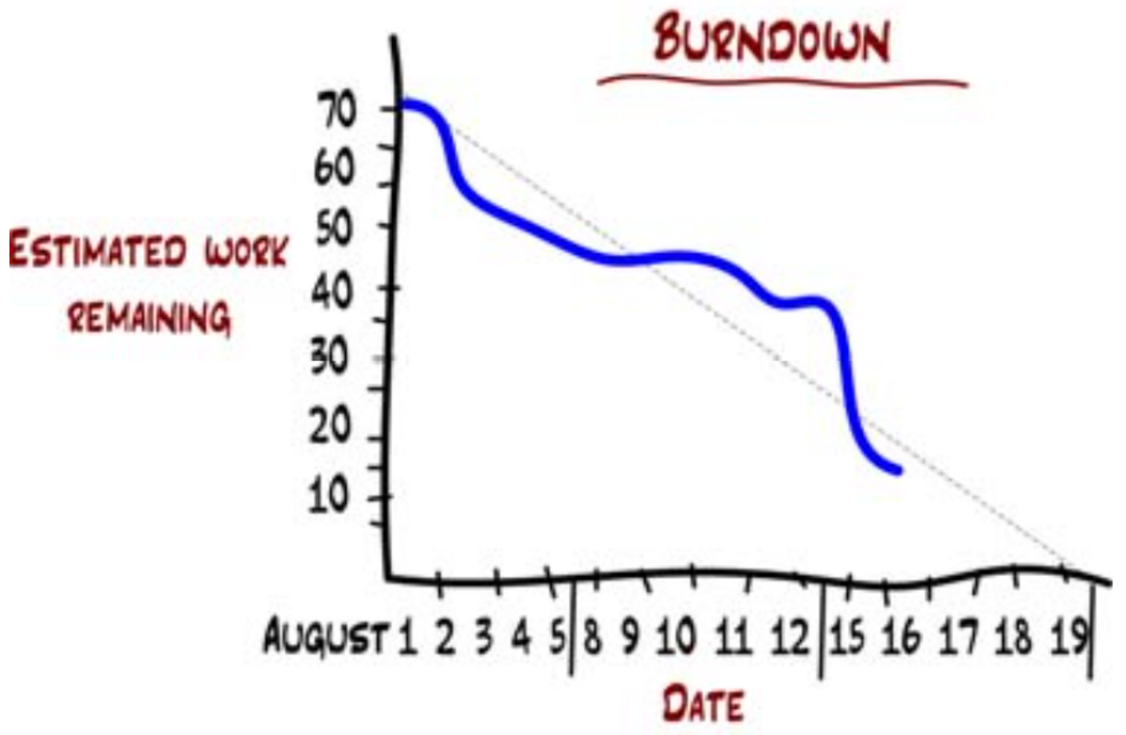
- A Scrum team has a short meeting (at most 15 minutes) every day at the same time & same place.
- The purpose of the meeting is to spread information about:
  - what is going on,
  - plan the current day's work, and
  - identify any significant problems.
  - it is usually done standing (to keep it short & maintain a high energy level).
- Daily standups are not prescribed in Kanban, but most Kanban teams seem to do it anyway. It's a great technique, regardless of which process you use.

*In Scrum the format of the meeting is **people-oriented** - every person reports one by one.*

*Many Kanban teams use a more **board-oriented** format, focusing on bottlenecks and other visible problems.*

SCRUM: BURNDOWN CHARTS

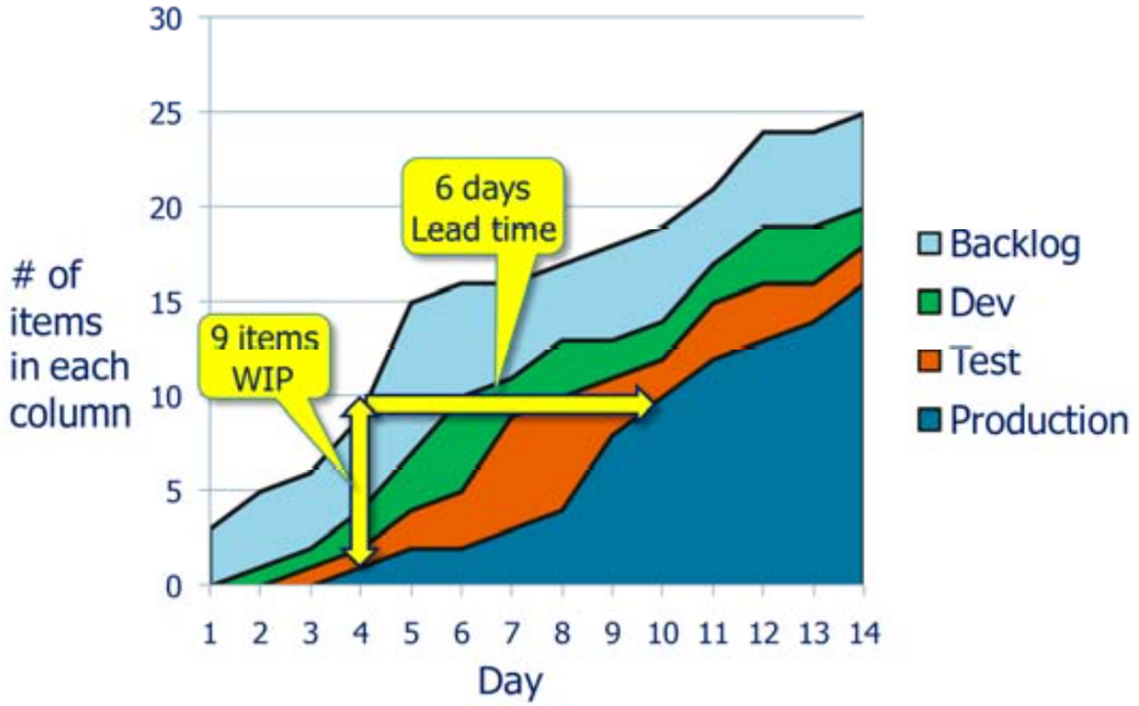
- A sprint burndown chart shows, on a daily basis, **how much work remains in the current iteration.**
- In Scrum, sprint burndown charts are used as one of the **primary tools** for tracking the progress of an iteration.
- The main purpose of a burndown chart is to easily find out as early as possible if we are behind or ahead of schedule, so that we can adapt.



KANBAN CHARTS

In Kanban, **no particular type of chart is prescribed.** But you are of course allowed to use any type of chart you like (including burndowns).

Here's an example of a Cumulative Flow diagram. This type of chart illustrates nicely how smooth your flow is and how WIP affects your lead time.



Here's how it works. Every day, total up the number of items in each column on the Kanban board and stack these on the Y axis. So on day 4 there were 9 items in the board. Starting from the right-most column there was 1 item in Production, 1 item in Test, 2 items in Dev, and 5 items in the Backlog. If we plot these points every day and connect the dots we get a nice diagram like the one above. The vertical and horizontal arrows illustrate the relationship between WIP and lead time.

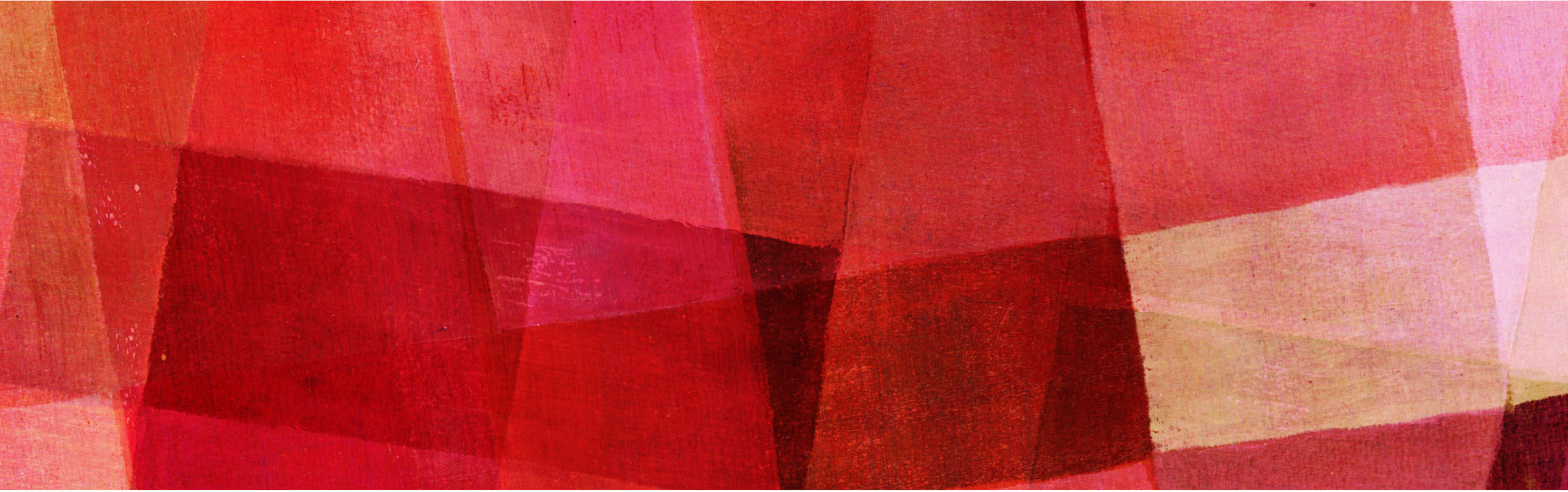
The horizontal arrow shows us that items added to the backlog on day 4 took on average 6 days to reach production. About half of that time was Test. We can see that if were to limit the WIP in Test and Backlog we would significantly reduce the total lead time.

SUMMARY: SIMILARITIES

- Both are Lean and Agile.
- Both use pull scheduling.
- Both limit WIP.
- Both use transparency to drive process improvement.
- Both focus on delivering releasable software early and often.
- Both are based on self-organizing teams.
- Both require breaking the work into pieces.
- In both, the release plan is continuously optimized based on empirical data (velocity / lead time).

SUMMARY: DIFFERENCES

Scrum	Kanban
Timeboxed iterations prescribed.	Timeboxed iterations optional. Can have separate cadences for planning, release, and process improvement. Can be event- driven instead of timeboxed.
Team commits to a specific amount of work for this iteration.	Commitment optional.
Uses Velocity as default metric for planning and process improvement.	Uses Lead time as default metric for planning and process improvement.
Cross-functional teams prescribed.	Cross-functional teams optional. Specialist teams allowed.
Items must be broken down so they can be completed within 1 sprint.	No particular item size is prescribed.
Burndown chart prescribed	No particular type of diagram is prescribed
WIP limited indirectly (per sprint)	WIP limited directly (per workflow state)
Estimation prescribed	Estimation optional
Cannot add items to ongoing iteration.	Can add new items whenever capacity is available
A sprint backlog is owned by one specific team	A kanban board may be shared by multiple teams or individuals
Prescribes 3 roles (PO/SM/Team)	Doesn’t prescribe any roles
A Scrum board is reset between each sprint	A kanban board is persistent
Prescribes a prioritized product backlog	Prioritization is optional.



//TODO

what are we going to do today in the workshop?

TASKS

- Setup cross-functional team(s)
  - **Developers** (backend, frontend, mobile, api)
  - Database Developers/Engineers
  - UI/UX Designers
  - **Testers** (manual, automated test engineers/software engineers in Test)
  - DevOps
  - Architects (software arc., database arc., system arc., enterprise arc., integration arc. …)
  - **Others:** ***Product Owner, Scrum master****, Team Lead, Lead Developer*
- Product Kickoff Meeting
  - Presentation of the product we are going to develop by the **Product Owner** to the team

TASKS

- Setup Tools
  - Project management tools: **Trello**, JIRA, YouTrack, …
  - Communication tools: **Slack,** HipChat, Skype for Business, …
  - Source code repositories: **GitHub**, GitLab, BitBucket
- Product feature identification and Breakdown (***@Team***)
- Architecture design
- Technology comparison and selection (***@Team***)
- Sprint Planning and Estimation meeting (***@scrum master*** *and* ***@team***)
- *Coding Time! yeah! (**@Team**)*
- Sprint Review (***@Team***)
- Sprint Retrospective (***@scrum master*** *and* ***@team***)

SETUP CROSS-FUNCTIONAL TEAM(S)

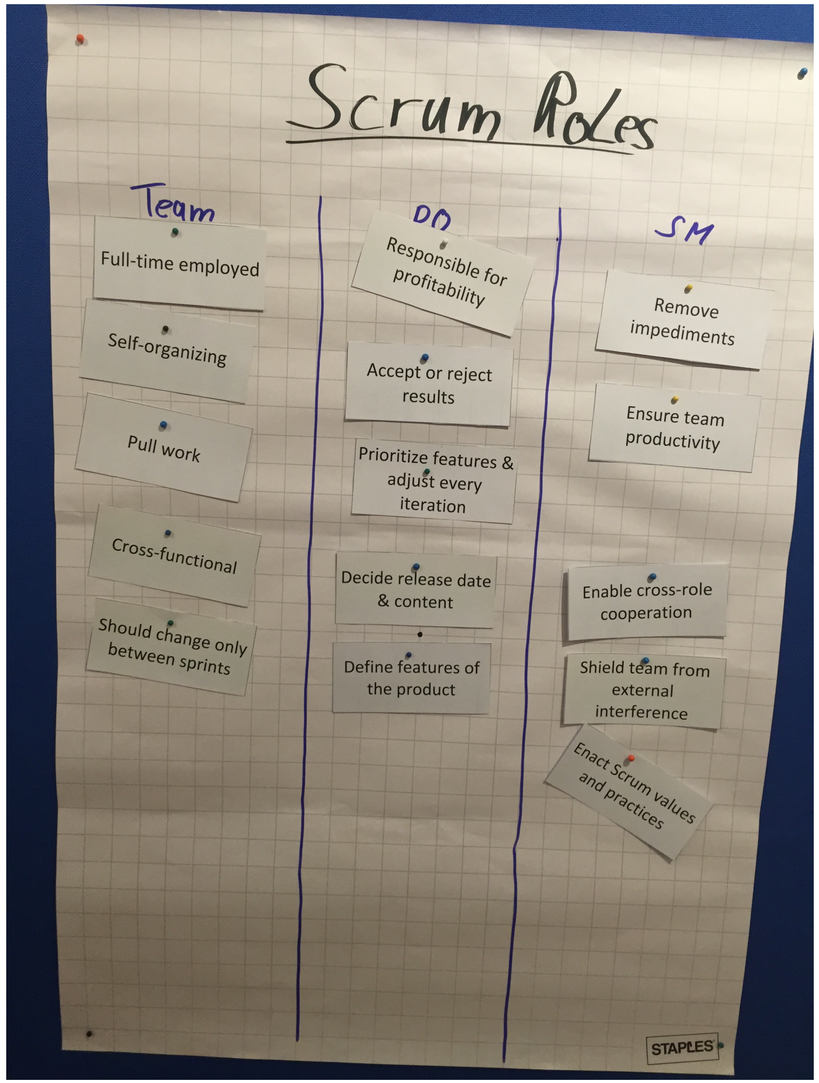
GIVE IT A NAME - FIRST EXERCISE AS A TEAM

CLIENT EXPLAINS THE PRODUCT FEATURES TO THE PRODUCT OWNERS

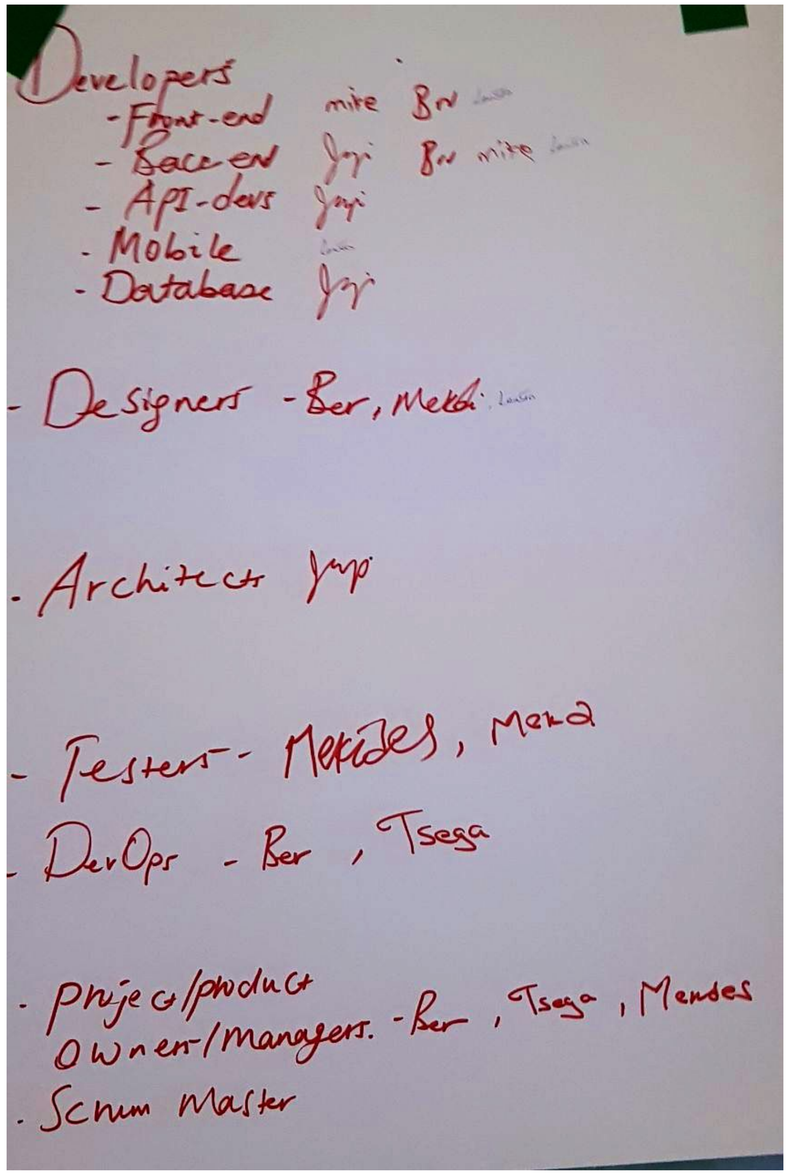
CREATE ACCOUNT
ON
TRELLO.COM AND GITHUB.COM

THE 3 MAIN SCRUM ROLES:

TEAM, PRODUCT OWNER AND SCRUM MASTER

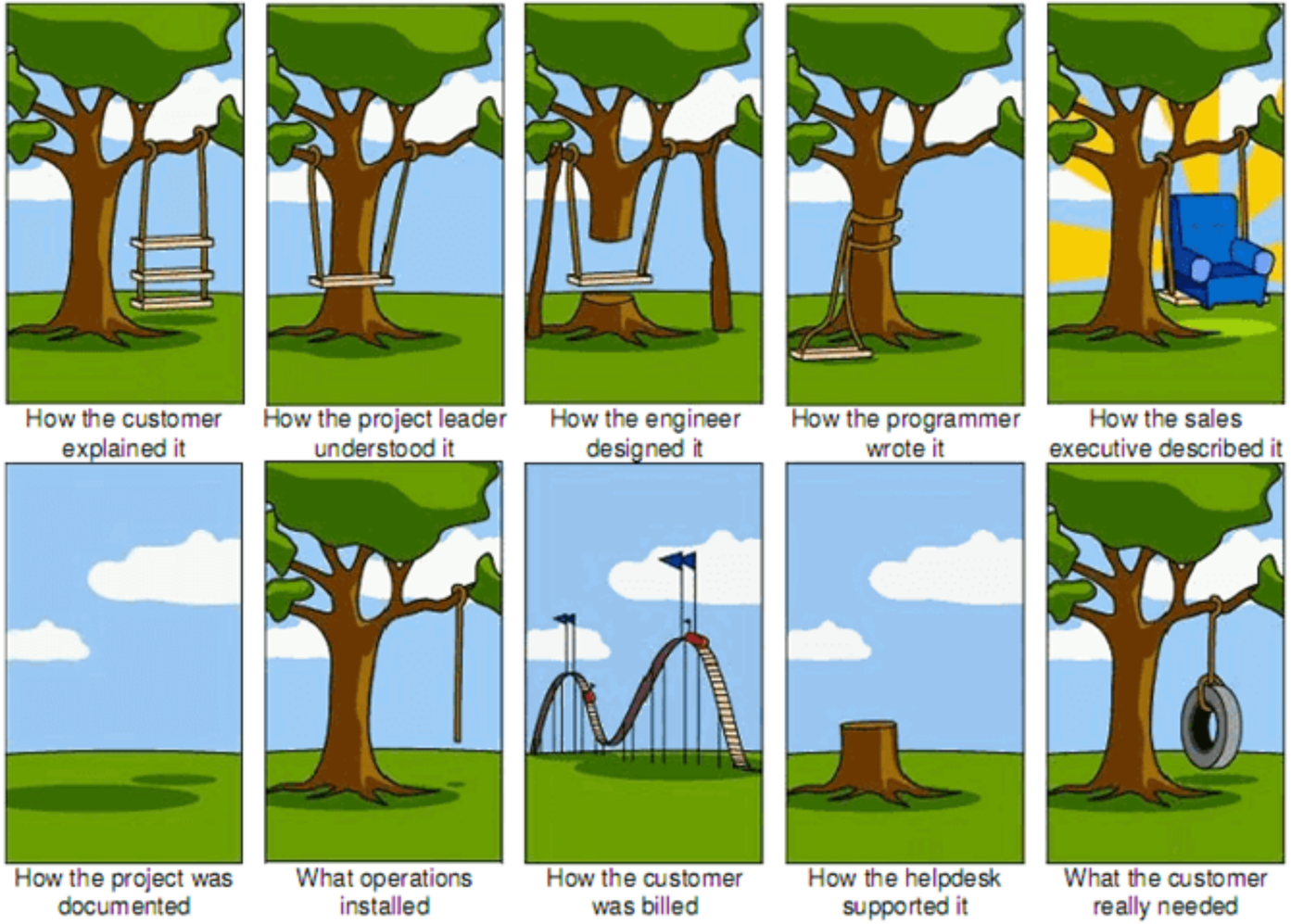


SELF-ORGANISE A CROSS-FUNCTIONAL TEAM

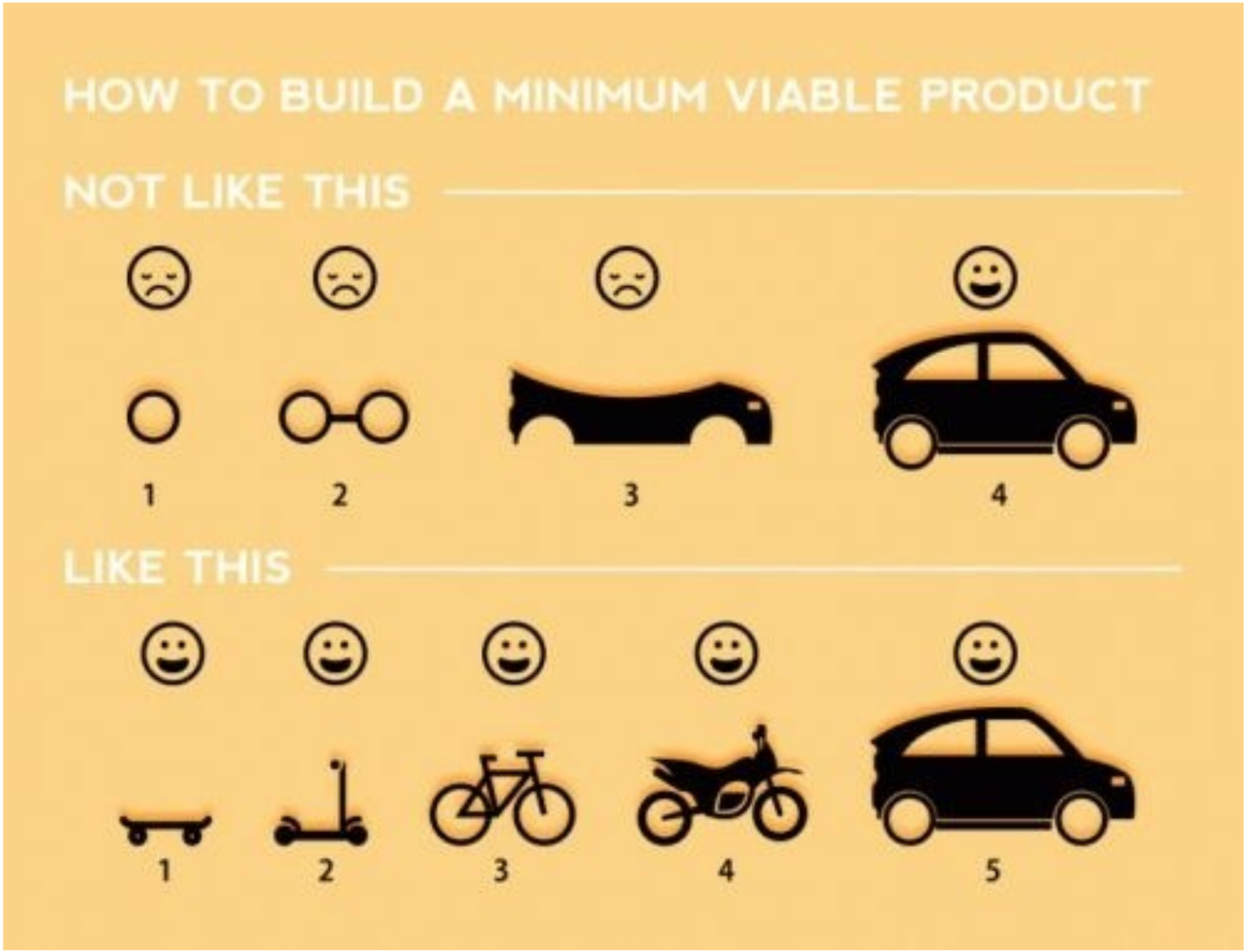


PRODUCT KICKOFF

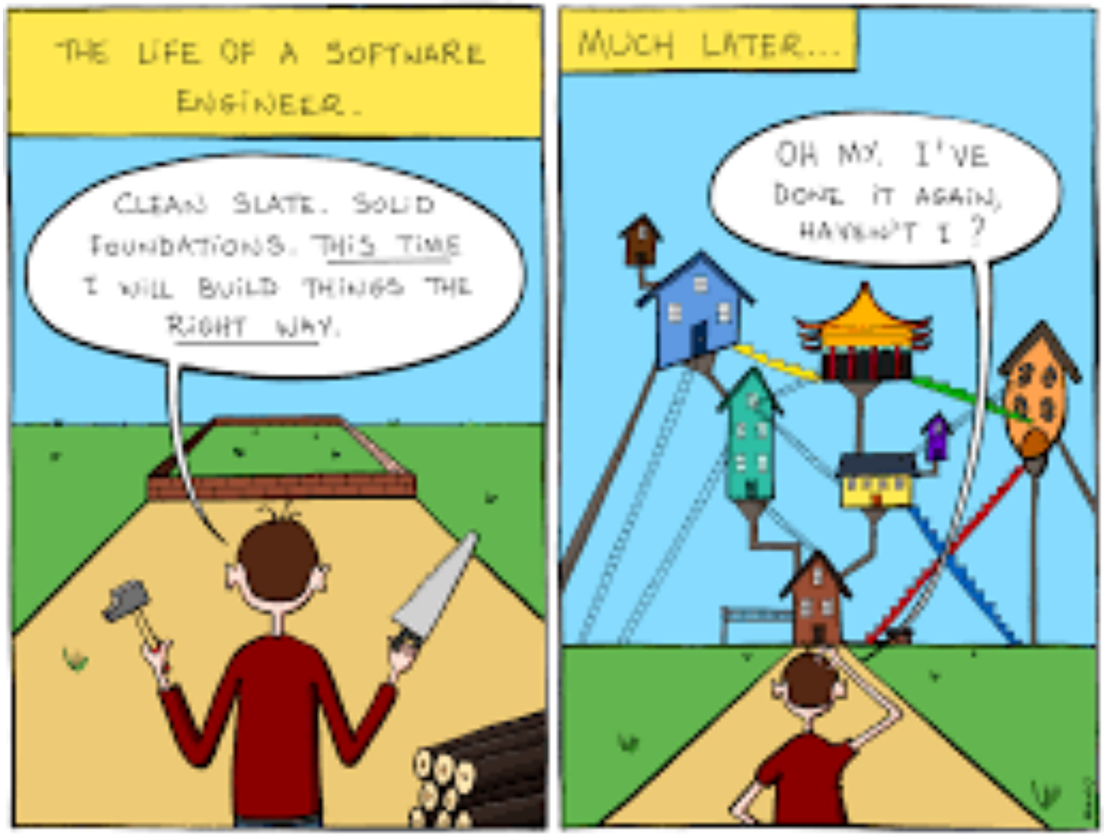
TRY TO UNDERSTAND THE CLIENT REQUIREMENT AND MAKE SURE THE DELIVERABLE FITS THE REQUIREMENT



MAKE THE CLIENT HAPPY AT ALL PHASES OF DELIVERY



ALWAYS DEVELOP CLEAN ARCHITECTURE AND CODE



PRODUCT SPECIFICATION

- We need a money transfer application that works on both web and mobile platforms.
- The actors of the system are **Users** who send money to each other and **Admins** who can manage the whole system
- The application data should be stored in a persistence store/database.
- The application should be able to perform the following tasks:
  - Create user Account
  - List all accounts
  - Show individual account
  - Update account
  - Delete account
  - Make money transfer
  - List all transfers
  - Show individual transfer
  - Update transfer
  - Delete transfer

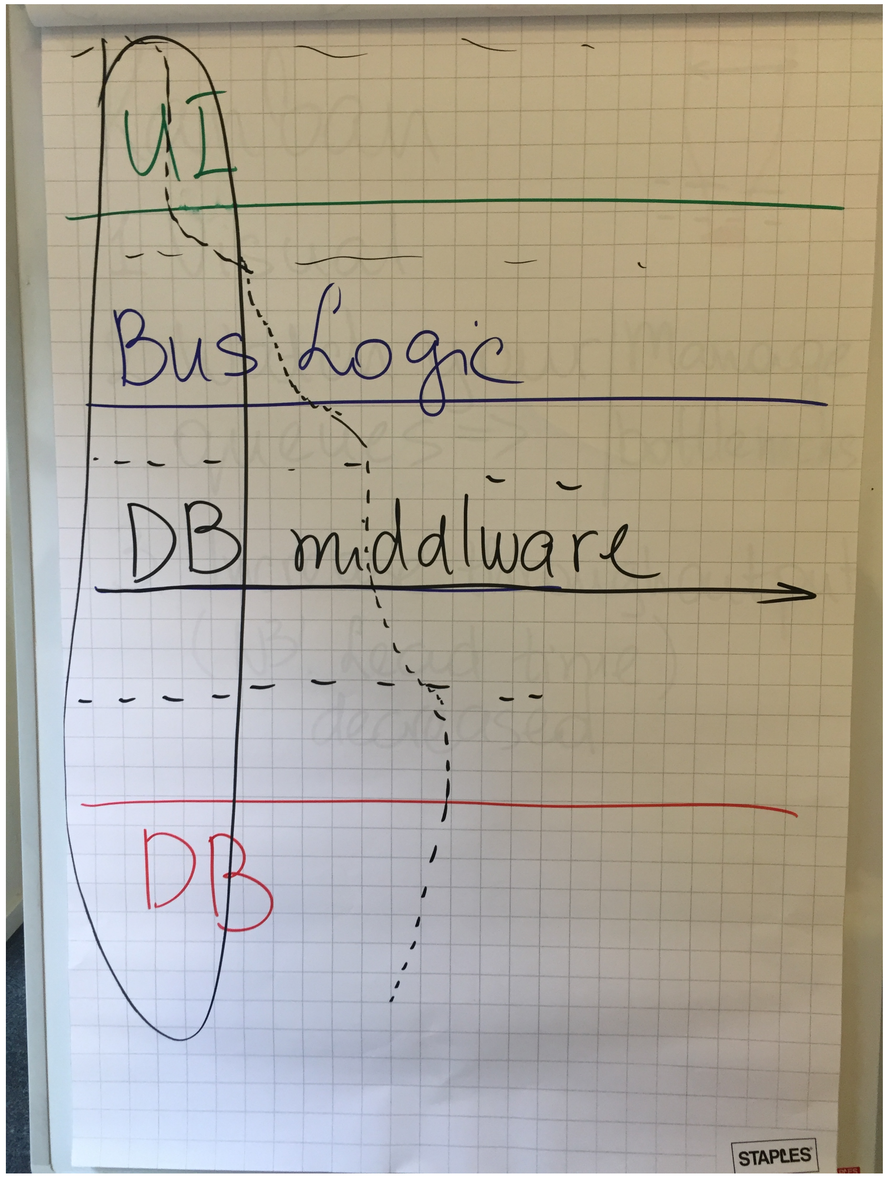
SETUP TOOLS (DEMO)

Create a Trello Board

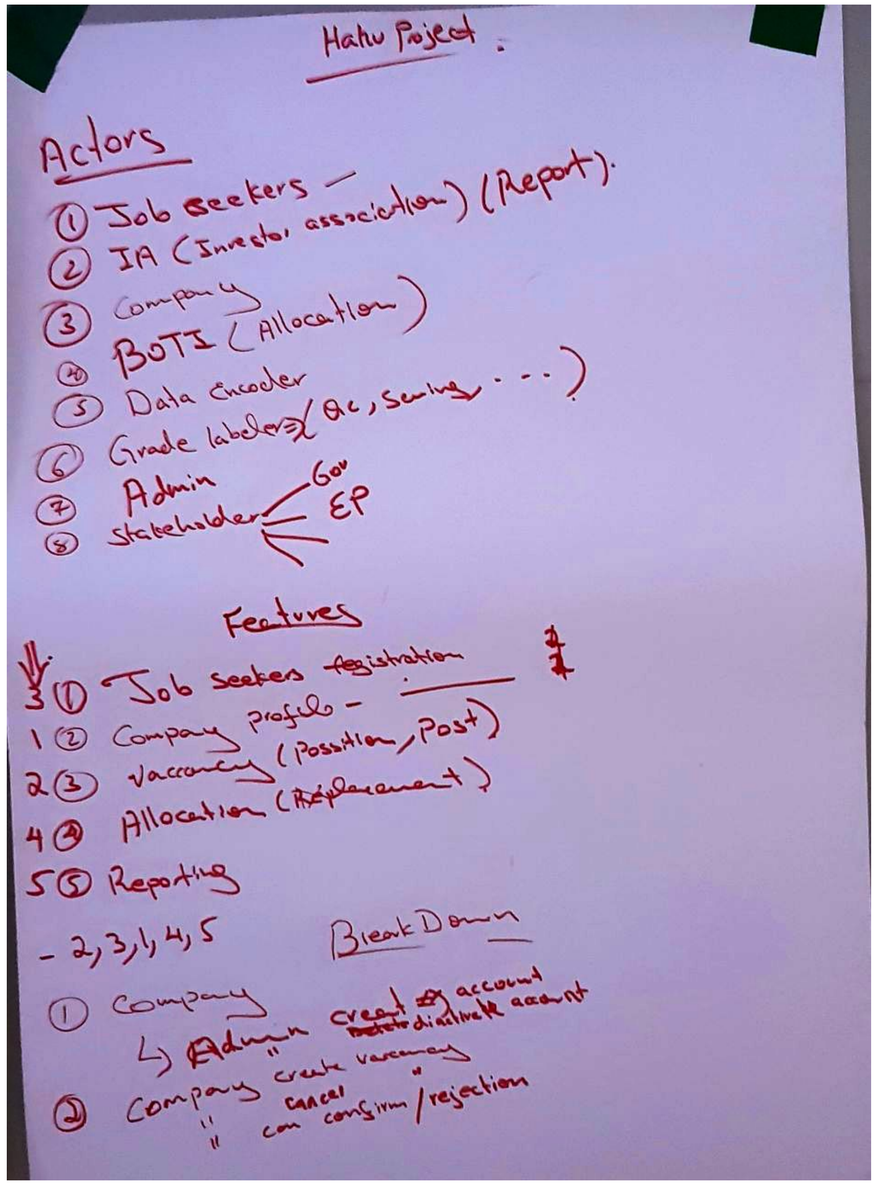
Create a Slack workspace

PRODUCT FEATURE IDENTIFICATION AND BREAKDOWN

INSTEAD OF WORKING ON ONE COMPONENT AFTER ANOTHER, TRY DELIVERING VALUE BY DEVELOPING USABLE AND UTTERABLE COMPONENTS



YOUR WORK: FEATURE IDENTIFICATION AND BREAKDOWN

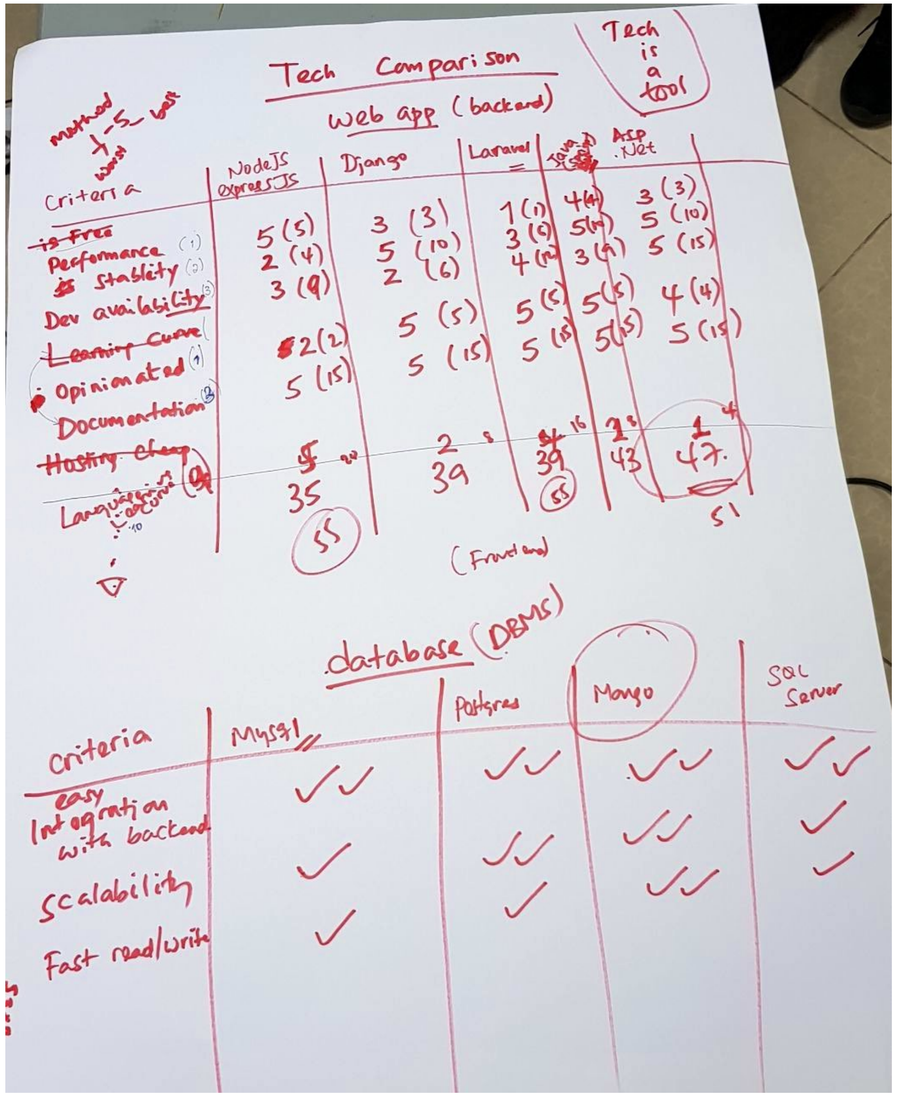


Steps:

1. *Identify the actors in the system*
2. *Identify features that can be released at once using the MVP principle*
3. *Break each feature down into items that can be given to a developer to work on*

TECHNOLOGY COMPARISON AND SELECTION

COMPARE THE TECHNOLOGIES/FRAMEWORKS USING MULTIPLE CRITERIA



- Comparison methods -

  USE ONLY ONE METHOD

  - *Numbers*
    - e.g. 1 up to 5

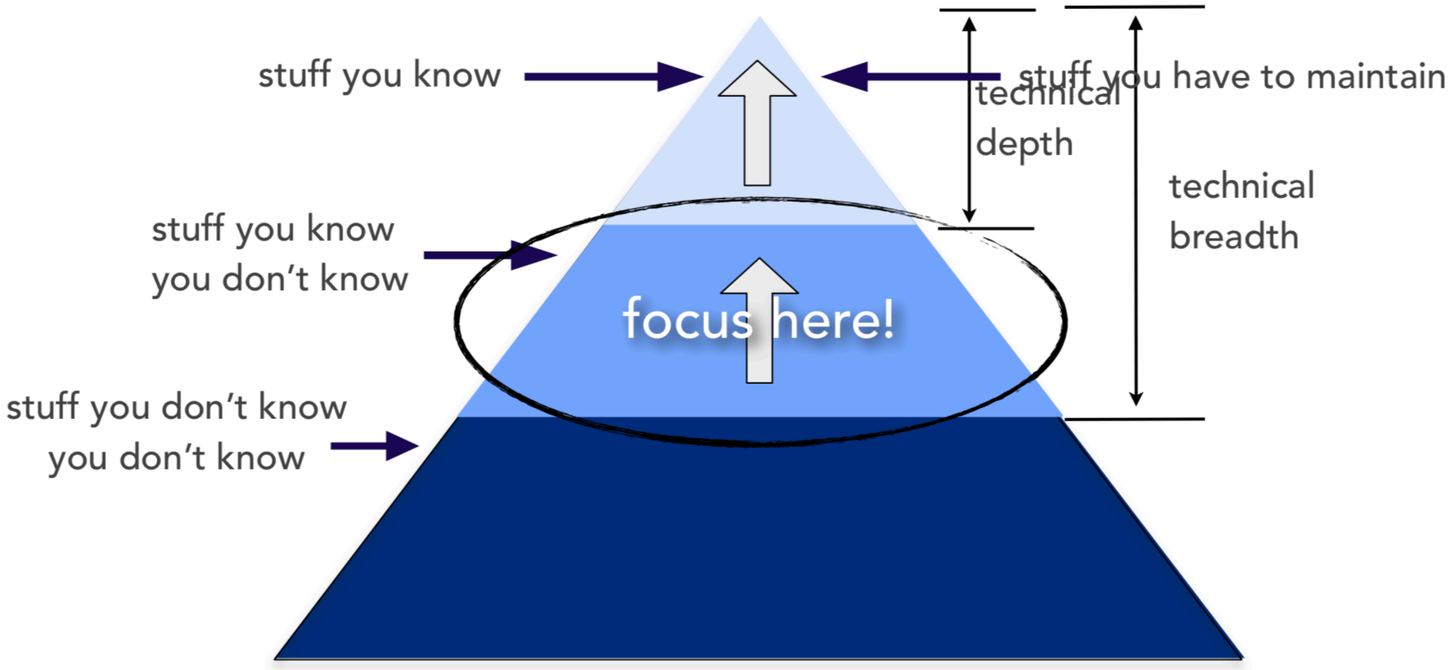
    (5 is best and 1 is worst)

  - *Yes or No*
  - *few Thumbs up or Down*
- *Assign weight to each criteria for unbiased judgement*
- **DON'T BE BAISED!!!**

GET DEEPER KNOWLEDGE OF THE STUFF YOU ALREADY KNOW

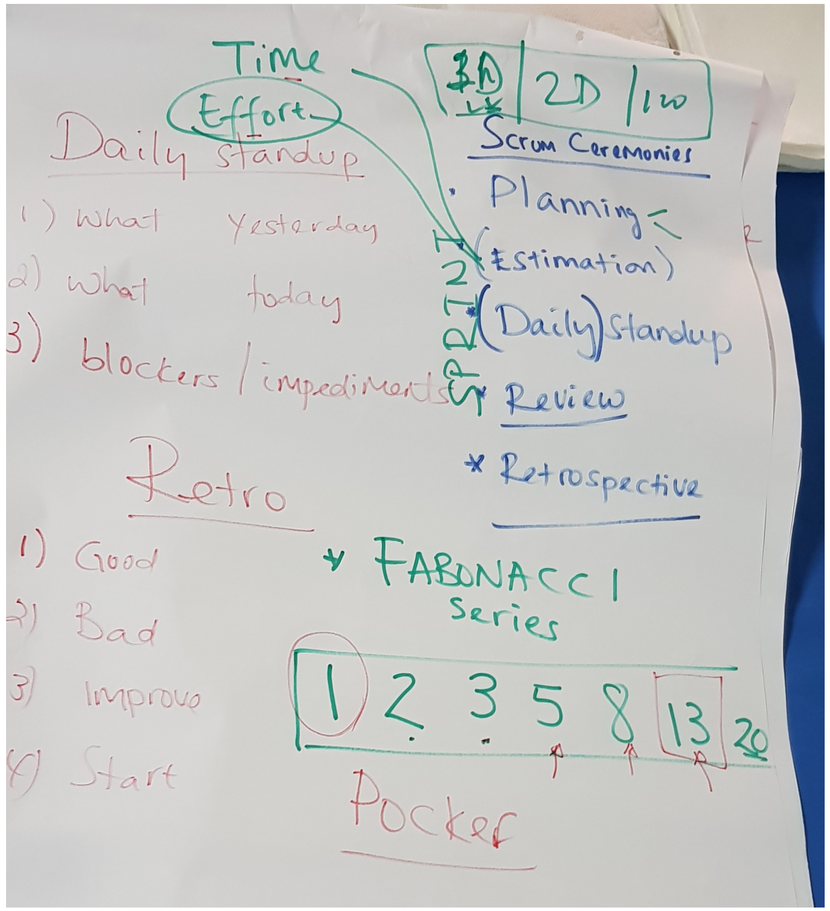
GET FAMILIAR WITH THE STUFF YOU KNOW YOU DON'T KNOW

technical knowledge



SPRINT PLANNING AND ESTIMATION MEETING

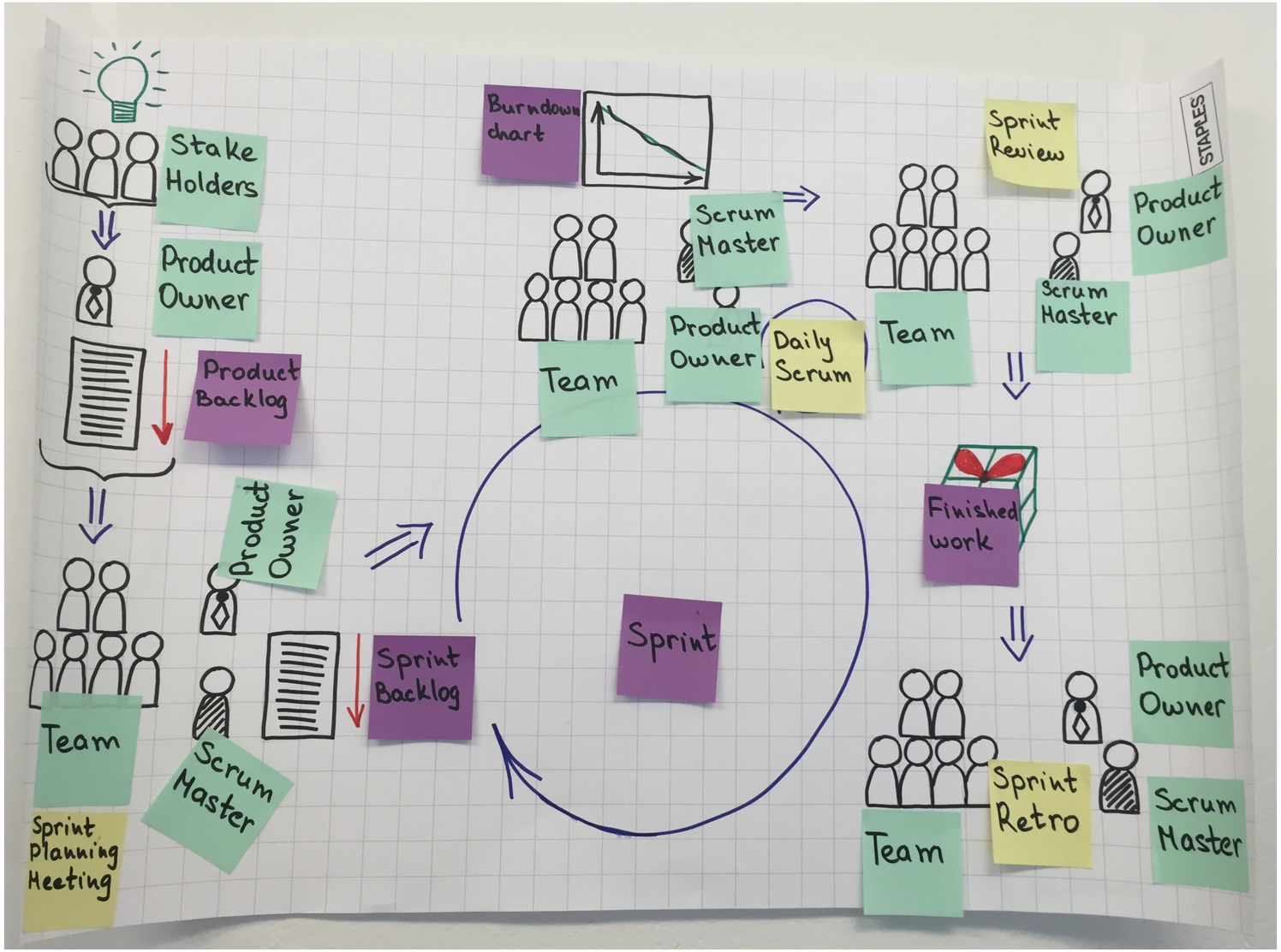
FOLLOW SCRUM AND CONDUCT THE CEREMONIES



SCRUM CEREMONIES

1. sprint planning and estimation **using fibonacci series**
2. daily standup
3. sprint review (for the team + PO)
4. sprint demo (for stakeholders)
5. Sprint retrospective

A TYPICAL SCRUM PROCESS FLOW



TIME FOR

ARCHITECTURE DESIGN
DATABASE DESIGN
USER INTERFACE DESIGN
CODING
TEST CASE DESIGN

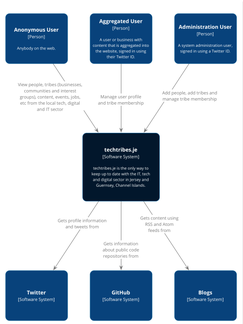
USE C4MODEL.COM FOR DESIGNING ARCHITECTURAL DIAGRAMS

Visualising software architecture

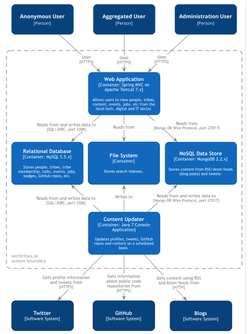
1 As a team, agree upon a set of abstractions you will use to communicate software architecture. The "C4 model" is a hierarchical way to think about the static structures of a software system in terms of containers, components and classes (or code):

> A **software system** is made up of one or more **containers** (web applications, mobile apps, desktop applications, databases, file systems, etc), each of which contains one or more **components**, which in turn are implemented by one or more **classes** (or **code**).

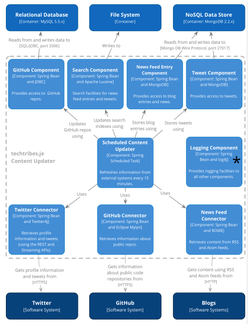
2 Visualise this hierarchy by creating a collection of System Context, Container, Component and (optionally) UML class diagrams. Think about these diagrams as maps of your software, showing different levels of detail.



[System Context] techtribes.je



[Containers] techtribes.je



[Components] techtribes.je - Content Updater

Level 1: System Context

A System Context diagram is a good starting point for diagramming and documenting a software system, allowing you to step back and see the big picture. Draw a diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interacts with. Detail isn't important here as this is your zoomed out view showing a big picture of the system landscape. The focus should be on people (actors, roles, personas, etc) and software systems rather than technologies, protocols and other low-level details.

Level 2: Containers

The next step is to illustrate the high-level technology choices with a Container diagram. A "container" is something like a web application, mobile app, desktop application, database, file system, etc. Essentially, a container is a separately deployable unit that executes code or stores data. The Container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another.

Level 3: Components

Next you can zoom in to each container further to visualise the major structural building blocks and their interactions. The Component diagram shows how a container is made up of a number of components, what each of those components are, their responsibilities and the technology/implementation details. If your components don't all fit on a single diagram, create multiple versions showing different portions of the container.

3 A common set of abstractions is more important than a common notation, but do ensure that your notation (shapes, colours, line styles, acronyms, etc) makes sense. If in doubt, add a diagram key/legend, even when using UML.

4 Use the elements in your model of the static structure to create additional supplementary diagrams in order to communicate runtime behaviour and deployment (the mapping of containers to infrastructure).

"**Software Architecture for Developers**"
A developer-friendly, practical and pragmatic guide to lightweight software architecture, technical leadership and the balance with agility.

Structurizr
Visualise, document and explore your software architecture

A collection of tooling to help you visualise, document and explore your software architecture.

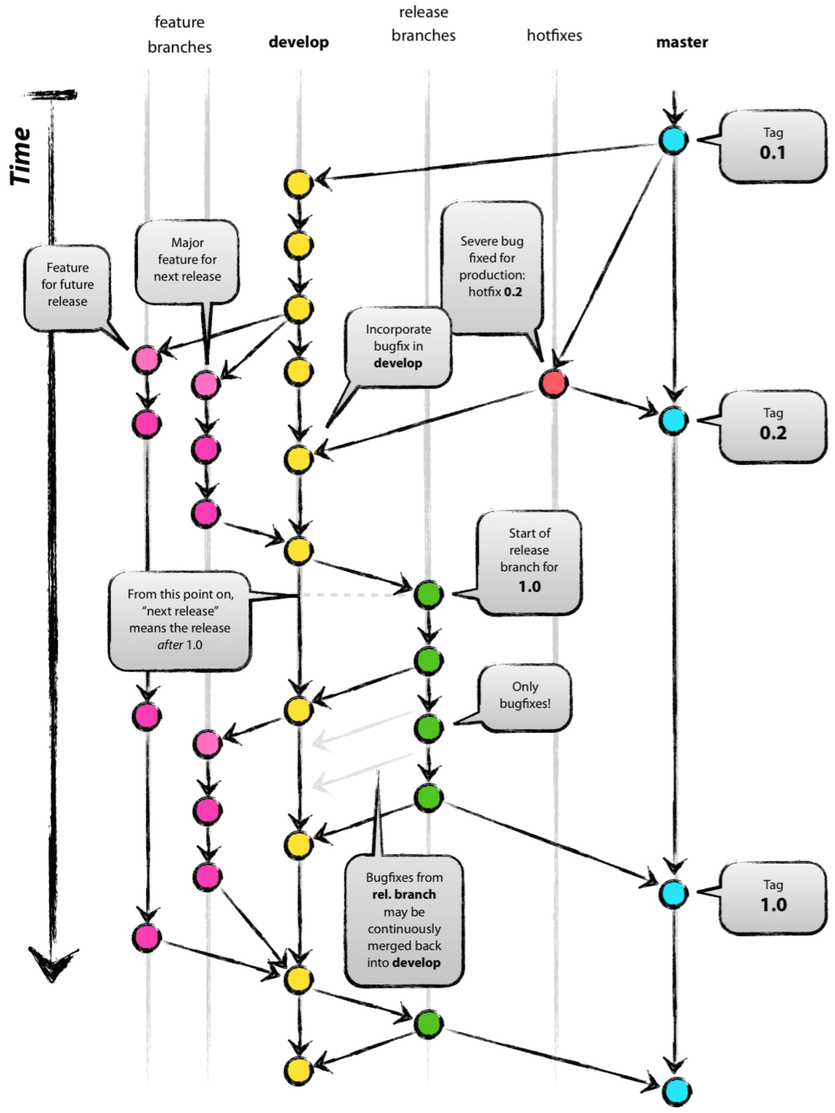
@simonbrown

INTRO TO GIT

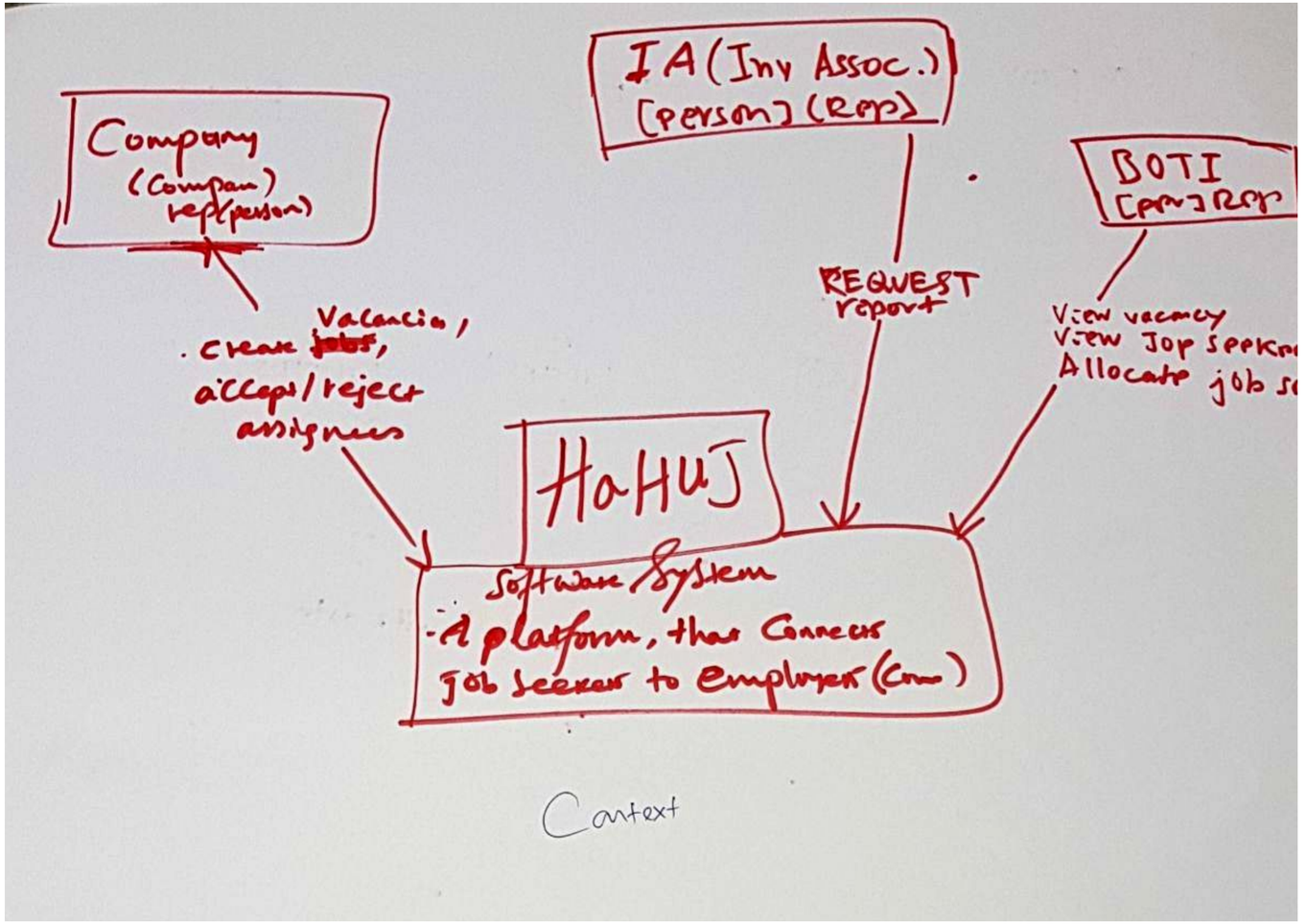
https://github.com/biniama/git-101

FOLLOW GIT WORKFLOW

[IT MIGHT SEEM COMPLEX BUT ONCE YOU LEARN HOW TO DO IT, IT MAKES YOUR LIFE EASIER]

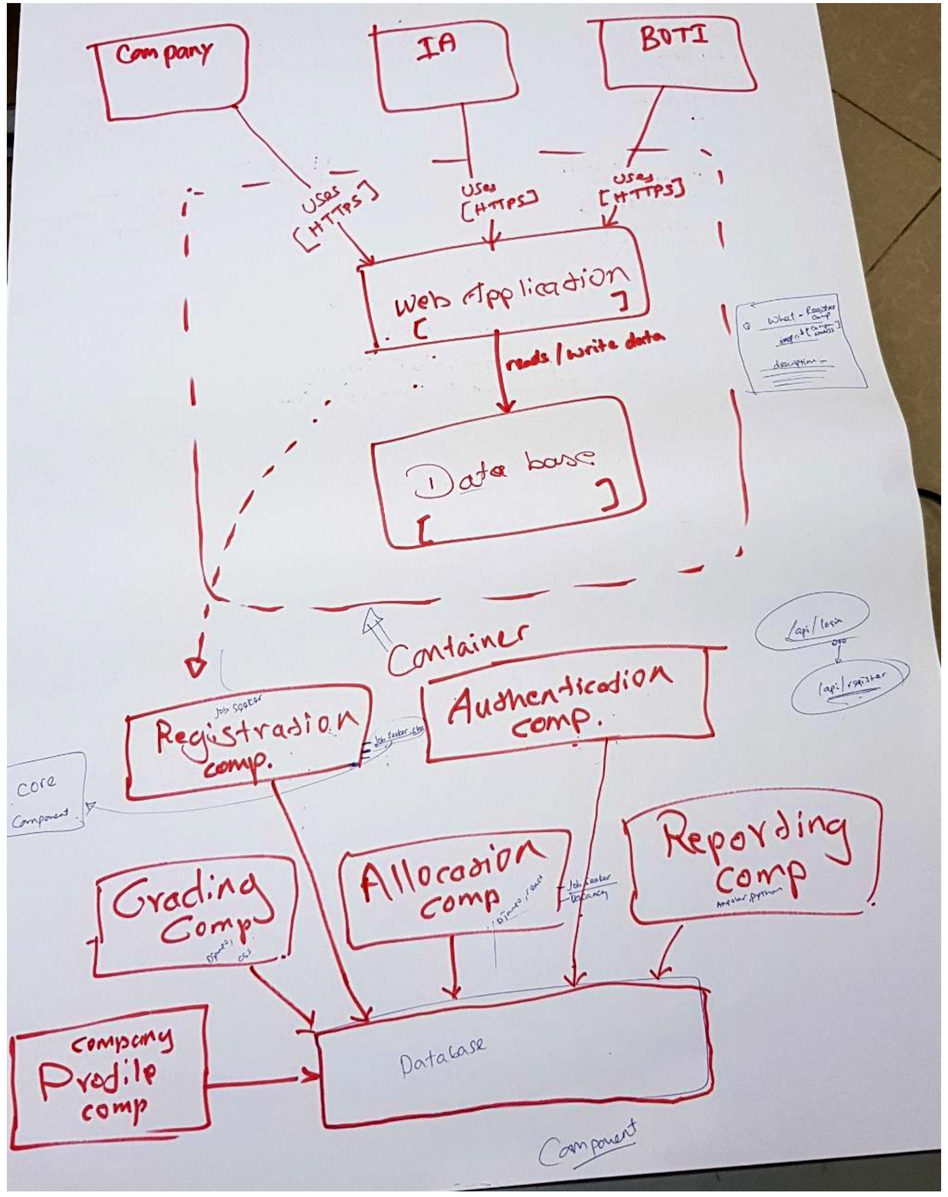


THE **ARCHITECTURE** YOU DESIGNED



C4 Model: System Context Diagram

THE **ARCHITECTURE** YOU DESIGNED



C4 Model: Container Diagram

and

Component Diagram

p.s. the diagrams here doesn't actually follow the C4 model principles and should be corrected in the future.

SPRINT REVIEW

This is a presentation of all the works that were done in the sprint to THE TEAM AND THE PRODUCT OWNER.

* *be prepared for the presentation*
* *make sure your demo works on all machines and pushed to the repository*
* *make use of your time*

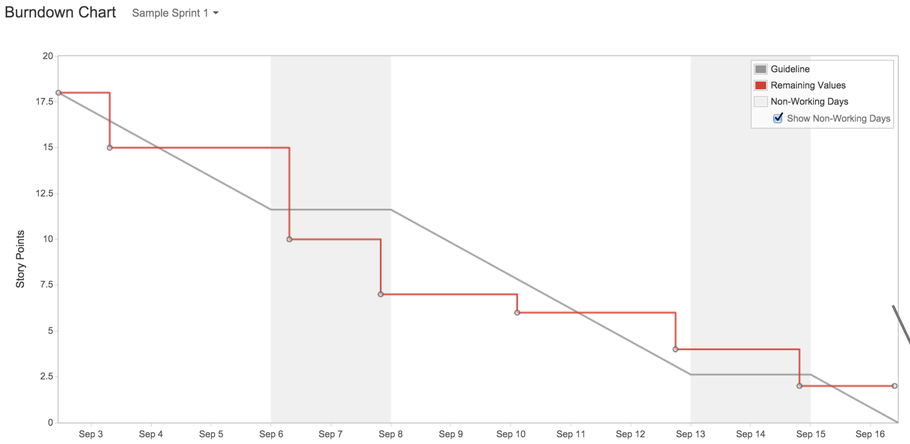
SPRINT RETROSPECTIVE

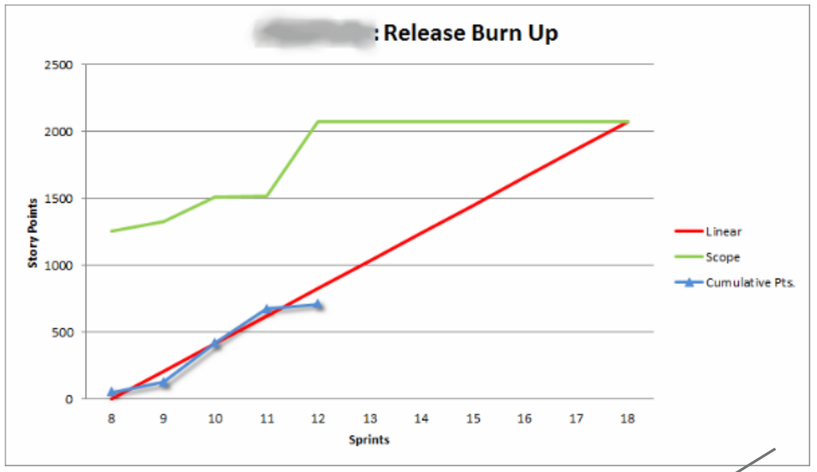
GOOD, BAD, IMPROVE, START

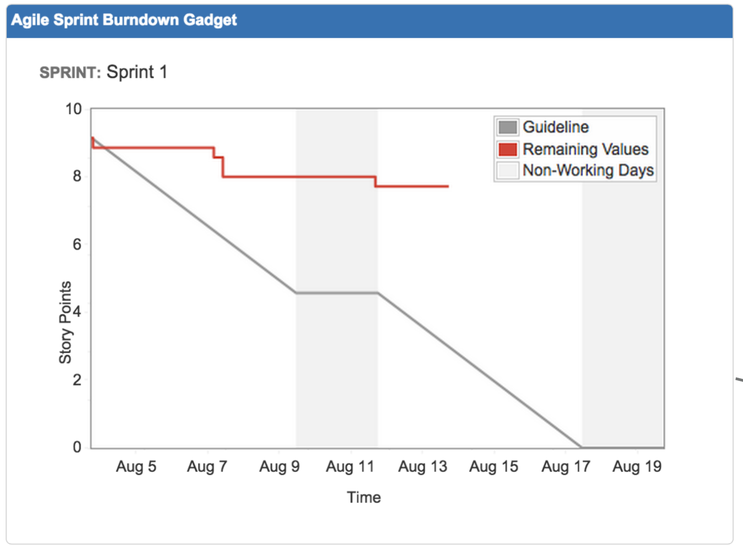
In this meeting, we talk about

* ***good*** *things that we did and happened in the sprint related to work, the team and the company*
* ***bad*** *things*
* *things to* ***improve*** *or continue*
* *things we should* ***start*** *doing which we haven't practiced yet*

WHEN SPRINT ENDS, WE DRAW A BURNDOWN CHART THAT SHOWS HOW MANY TICKETS WE WERE ABLE TO COMPLETE IN THAT SPRINT

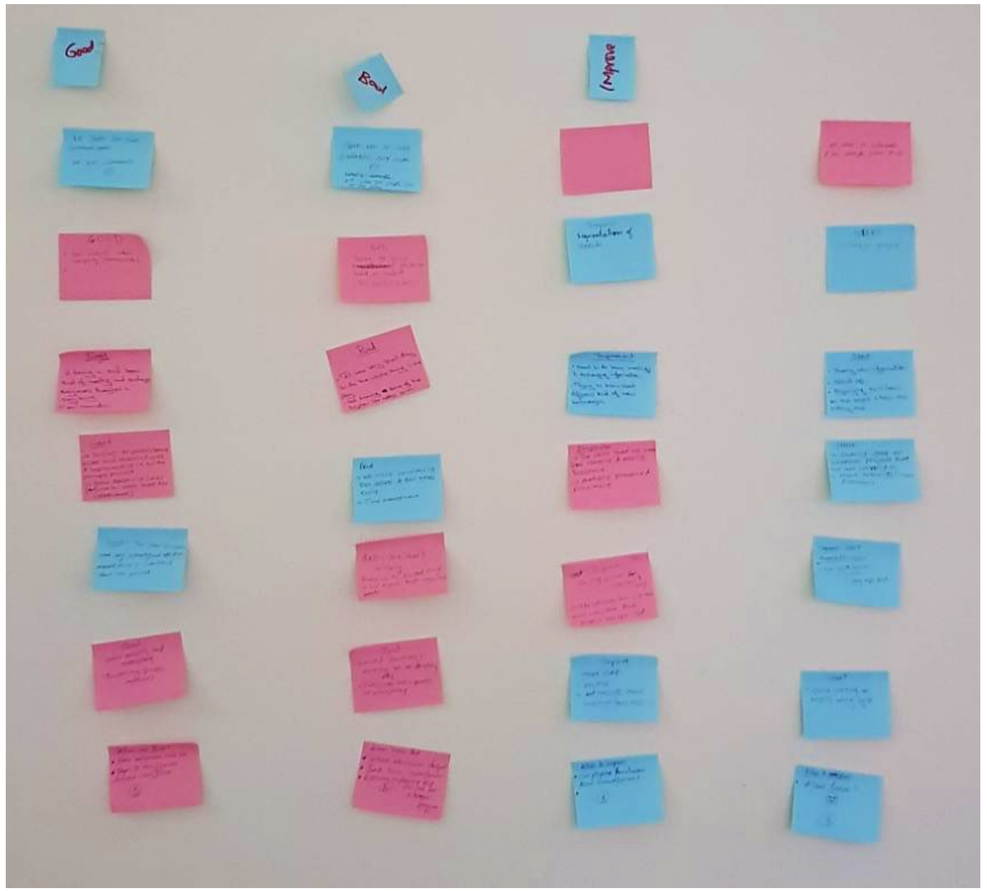






- **burn-down chart is great** because the team achieved all planned and committed tickets/features.
- **burn-up chart is the worst** because the team didn't accomplished what they committed and even added more commitment.
- **flat chart is not good** because the team is still working on the committed tasks/tickets.

RETROSPECTIVE YOU HAD



ALWAYS TAKE ACTION POINTS IN SPRINT RETROSPECTIVE AND ASSIGN RESPONSIBILITY AND MONITOR IN THE NEXT MEETING

Action Points are in the Next slide

ACTION POINTS

- Use agile methodology at AhadooTec ***@Tsega***
- organise knowledge sharing sessions ***@Mekides and @Jupi***
- organise team event ***@Tsega***
- organise similar trainings by external trainers/consultants ***@Mekdes***
- AhadooTec Project intro and status presentations ***@Mekdes***
- Find ways to develop technical kill of PO and Testers ***@Mekdes***
- Get/buy white board (ask and make sure it is bought by AhadooTec) ***@Bruk***
- Use Slack at AhadooTec ***@Bruk***

REFERENCES

- “Agile Software Development, Principles, Patterns, and Practices” By Robert C. Martin
  - http://www.amazon.com/gp/product/0135974445?ie=UTF8&tag=lstasd01-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=0135974445
- “Agile Estimating and Planning” by Mike Cohn
  - http://www.amazon.com/gp/product/0131479415?ie=UTF8&tag=lstasd01-20&linkCode=as2&camp=1789&creative=9325&creativeASIN=0131479415

- Other agile books:
  - https://www.crisp.se/bocker-och-produkter/scrum-and-xp-from-the-trenches
  - http://www.infoq.com/minibooks/kanban-scrum-minibook
  - http://noop.nl/2008/06/top-20-best-agile-development-books-ever.html

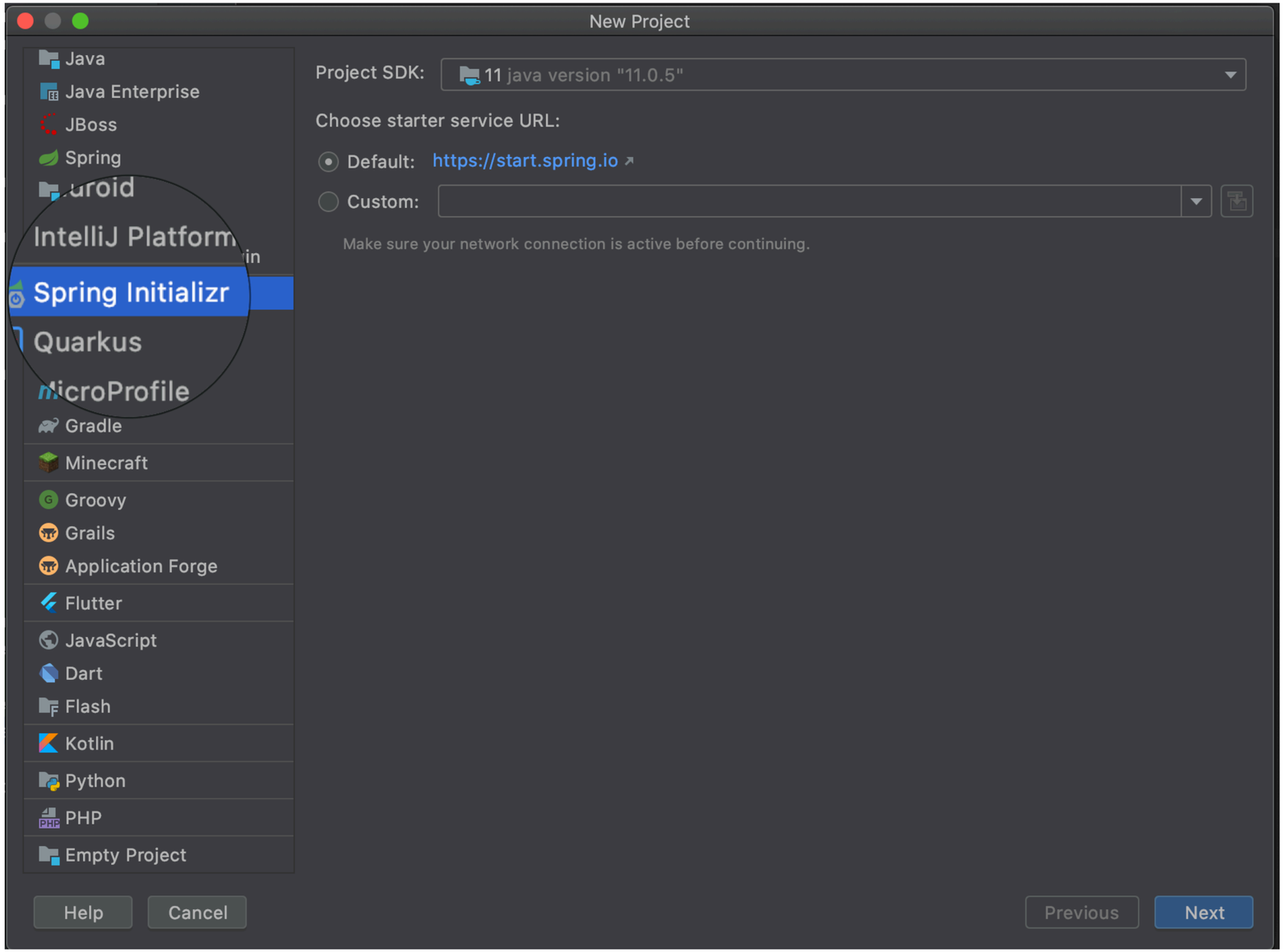
CODING TIME

Let's get practical!

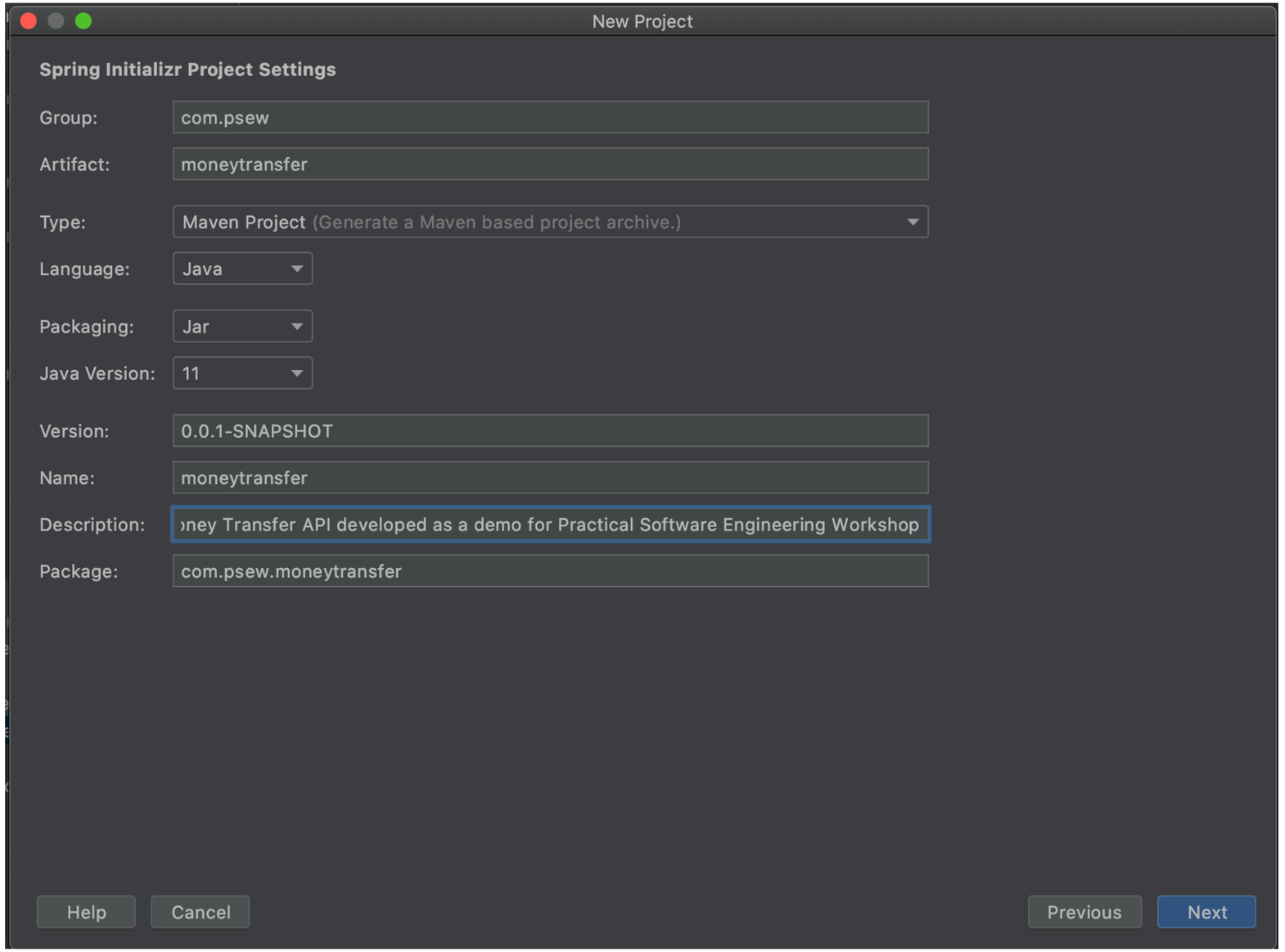
REQUIRED RESOURCES

- Every trainee should have a computer (preferably a high performance laptop) with high speed internet connection to install the required application packages. These are:
  - Installing SDKMAN! **https://sdkman.io/install**
  - Spring Boot with **sdk install springboot 2.4.3**
  - Java version 11 with **sdk install java 11.0.3.hs-adpt**
  - IntelliJ IDEA Ultimate https://www.jetbrains.com/idea/download/#section=windows
  - PostgreSQL database
  - Downloads Node.js Latest LTS Version: 14.16.0 (includes npm 6.14.11) **https://nodejs.org/en/download/**
  - Download Postman - a REST API Client (tester) **https://www.postman.com/downloads/**
  - ~~Flutter SDK https://flutter.dev/docs/get-started/install/windows~~
  - ~~Download and install Android Studio https://developer.android.com/studio~~
  - ~~Set up the Android emulator https://flutter.dev/docs/get-started/install/windows#set-up-the-android-emulator~~

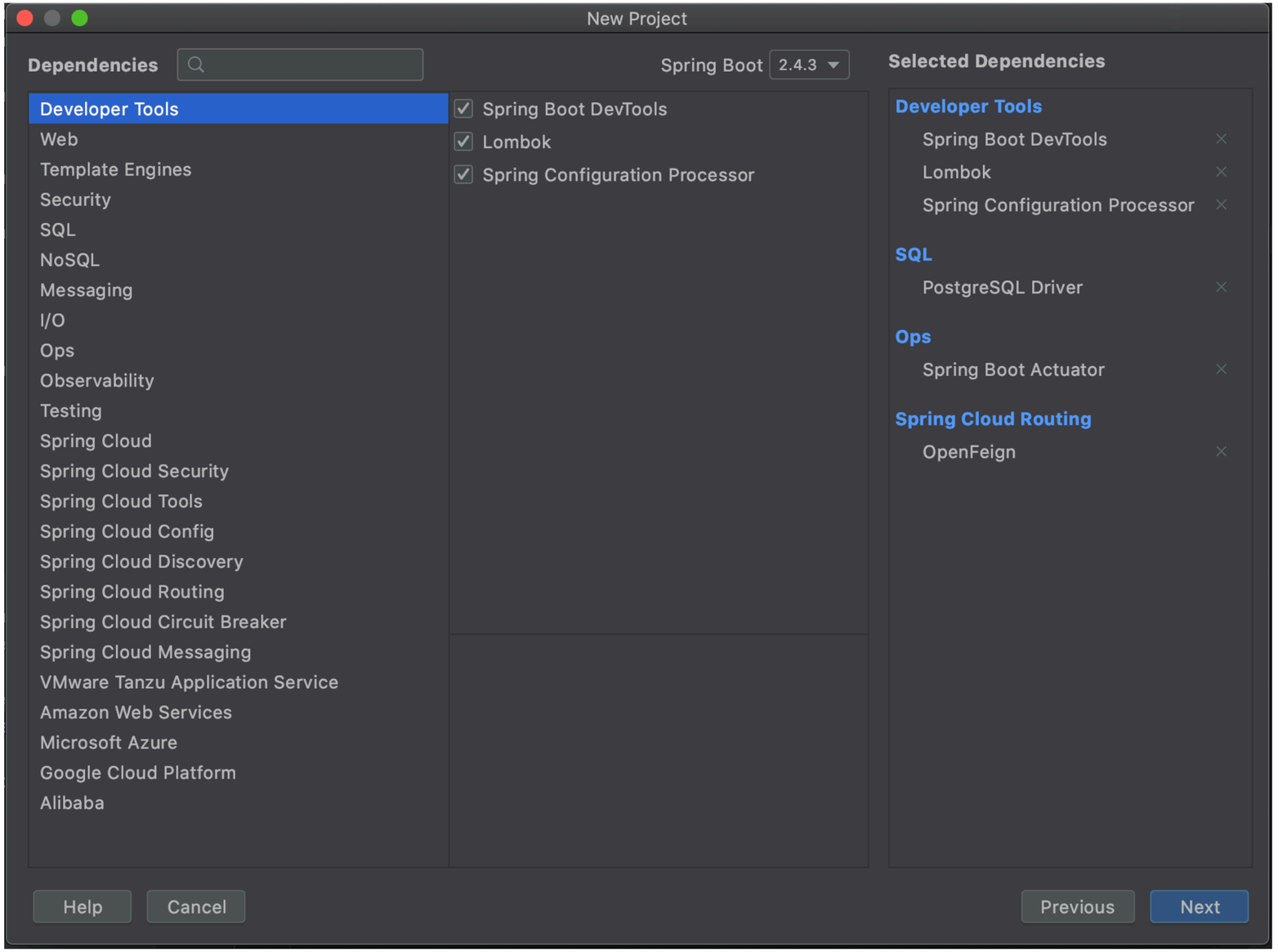
SELECT THE TYPE OF PROJECT



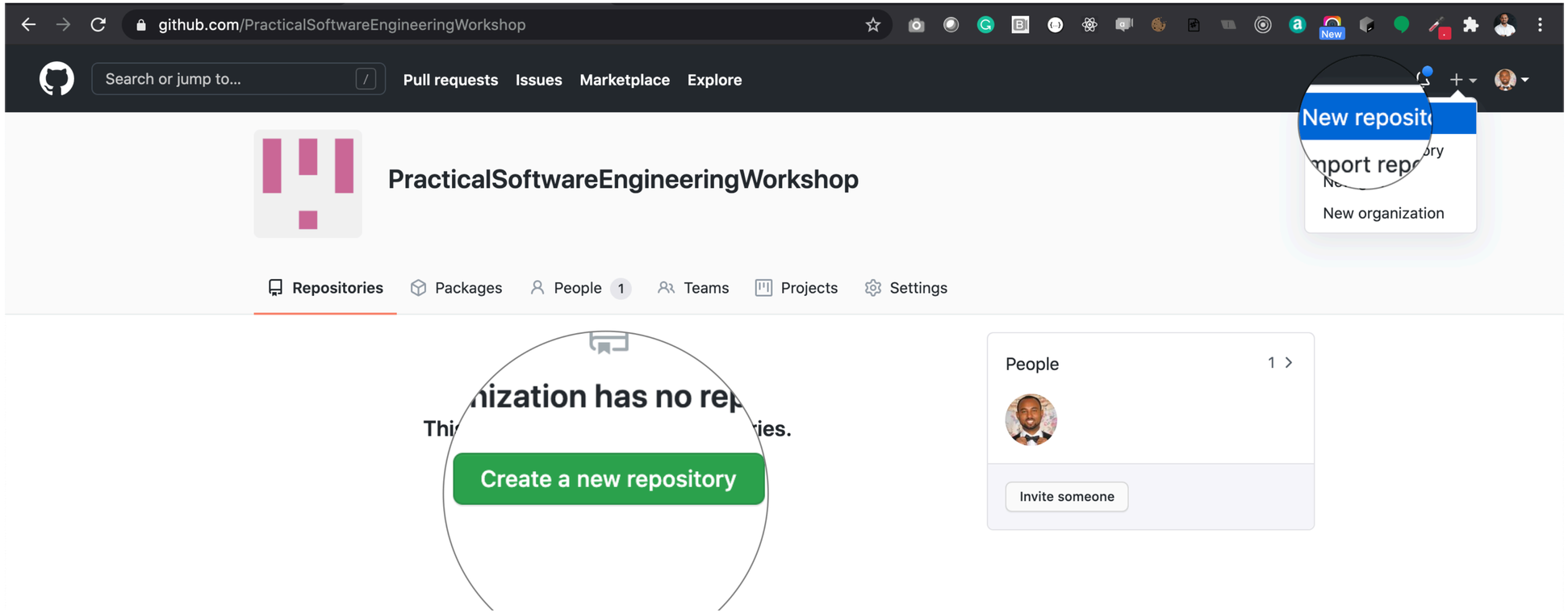
INITIALISE THE PROJECT



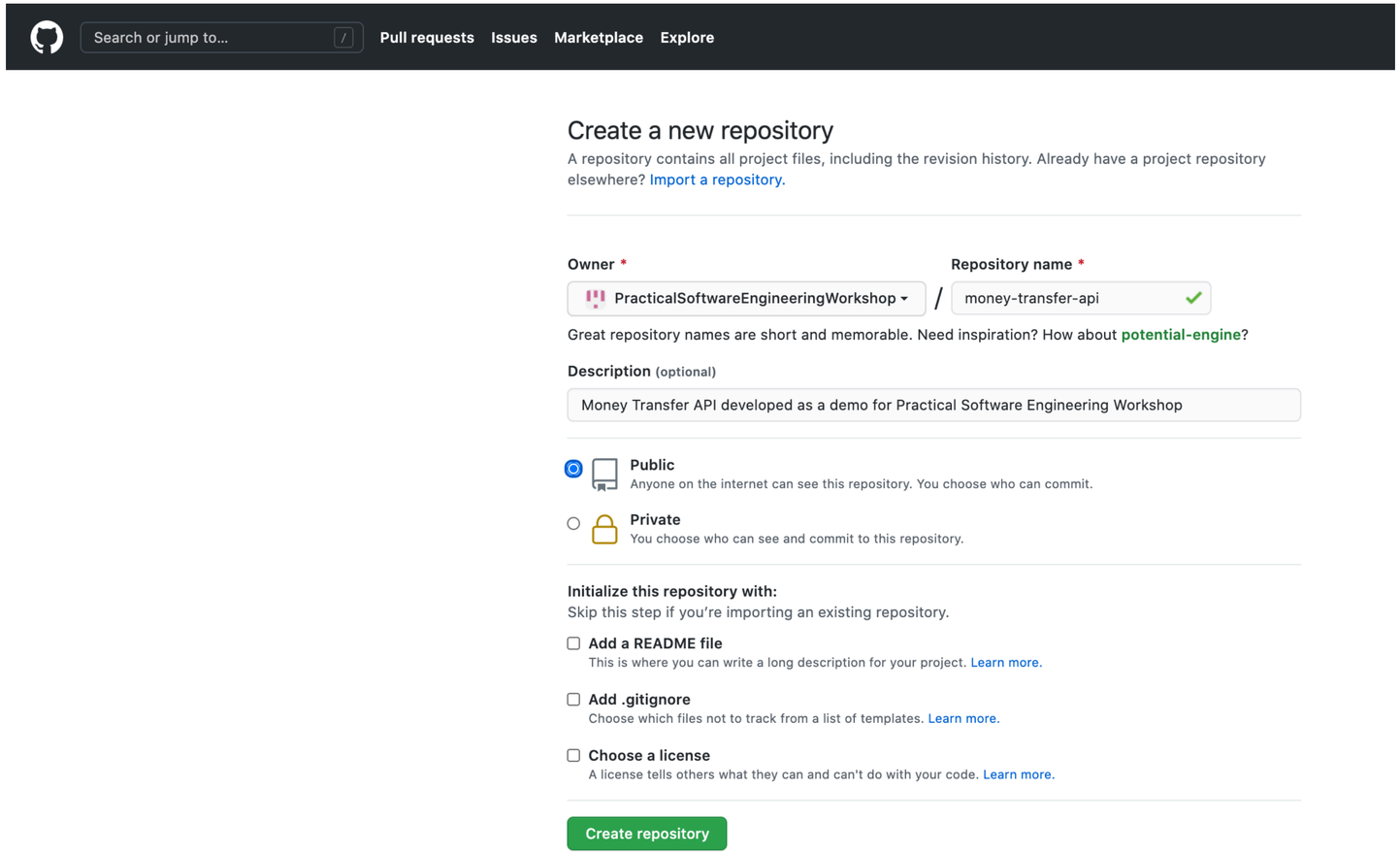
SELECT DEPENDENCIES



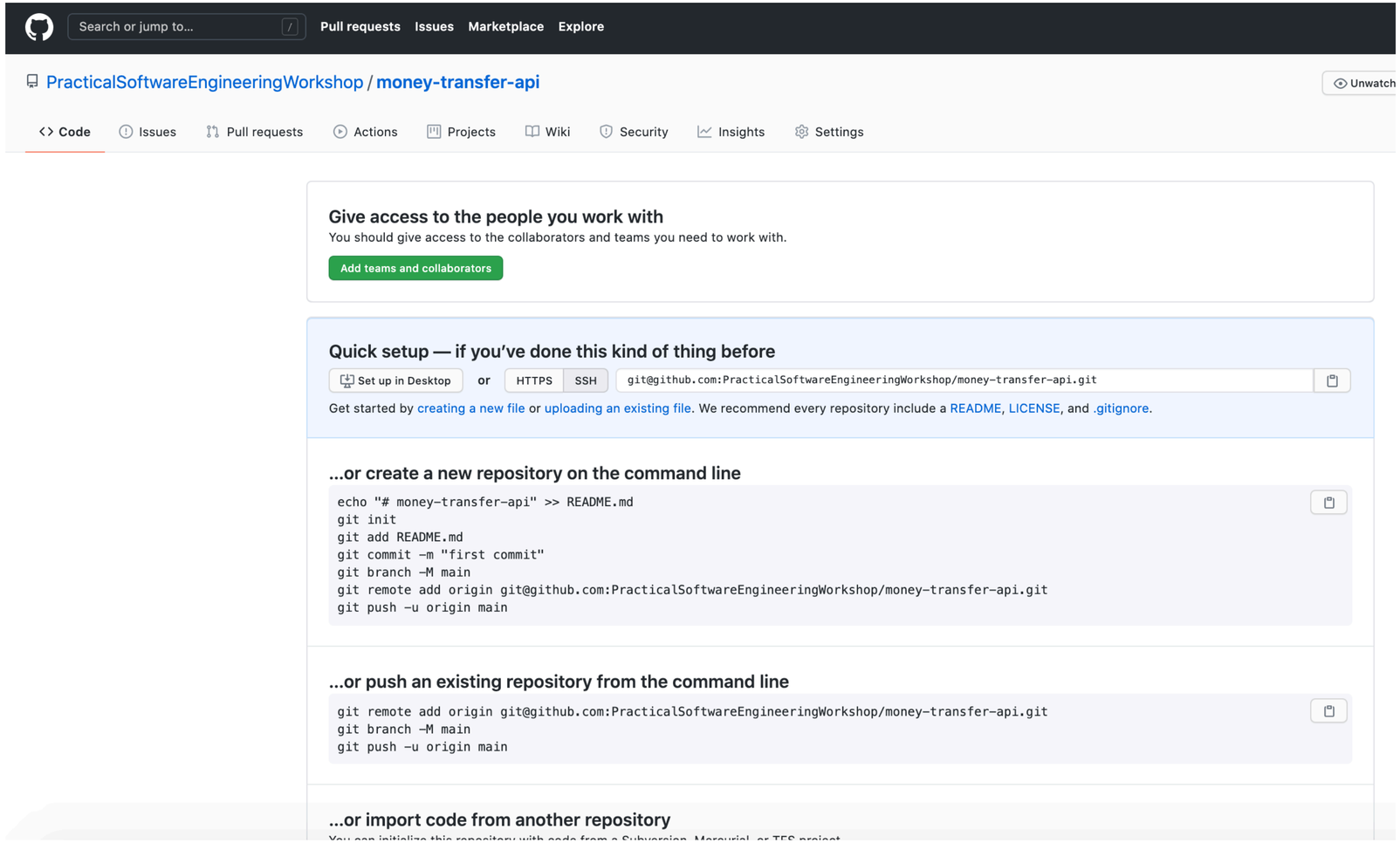
CREATE A REPOSITORY ON GITHUB.COM



STEP 1: CREATE A REPOSITORY AND FILL INFORMATION



STEP 2: REPOSITORY CREATED AND TIME TO PUSH YOUR CODE



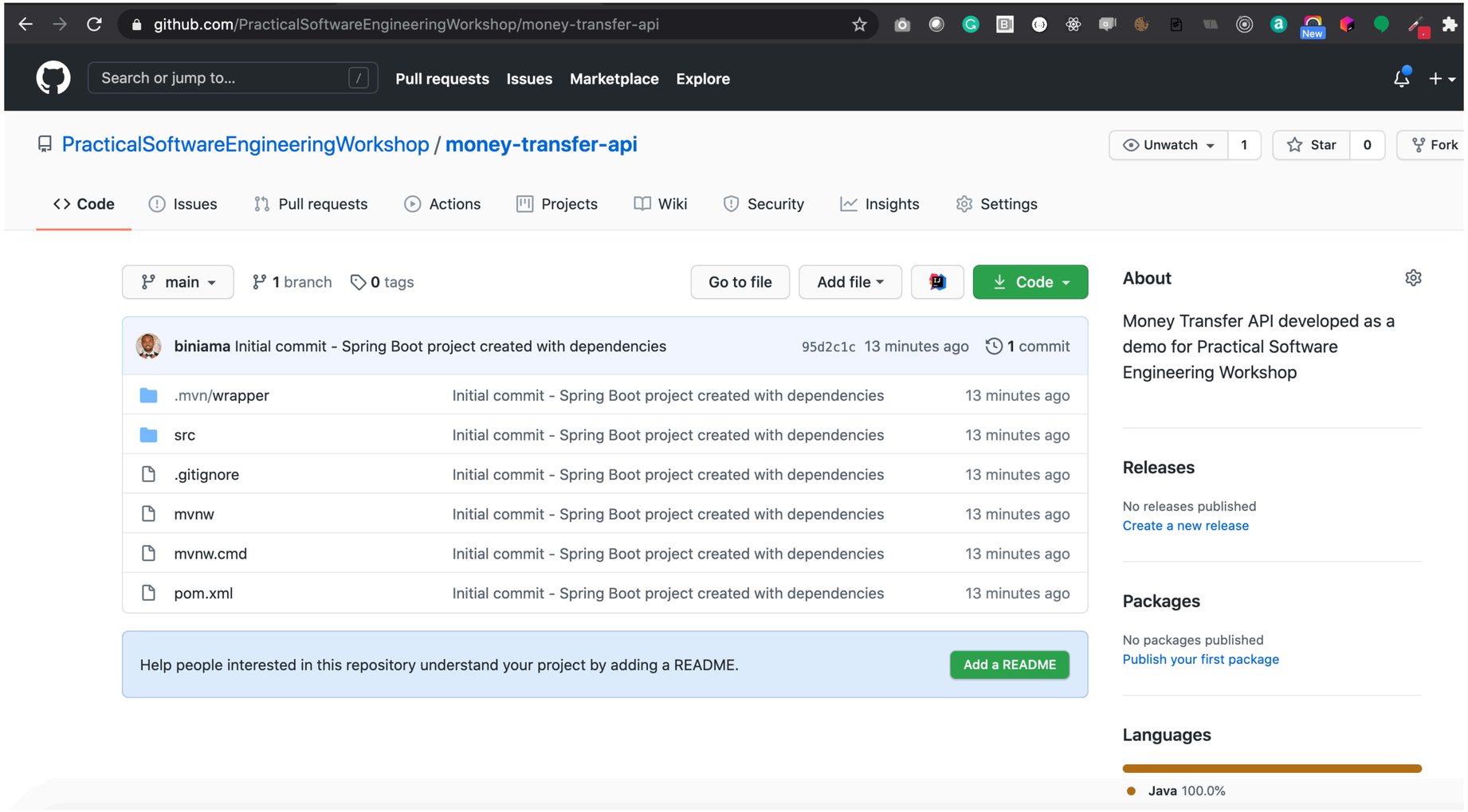
```
› git init
Initialized empty Git repository in /Users/biniam.asnake/github/moneytransfer/.git/

› git status
On branch master

No commits yet


Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        .mvn/
        mvnw
        mvnw.cmd
        pom.xml
        src/
nothing added to commit but untracked files present (use "git add" to track)
```

```
› git add .
```

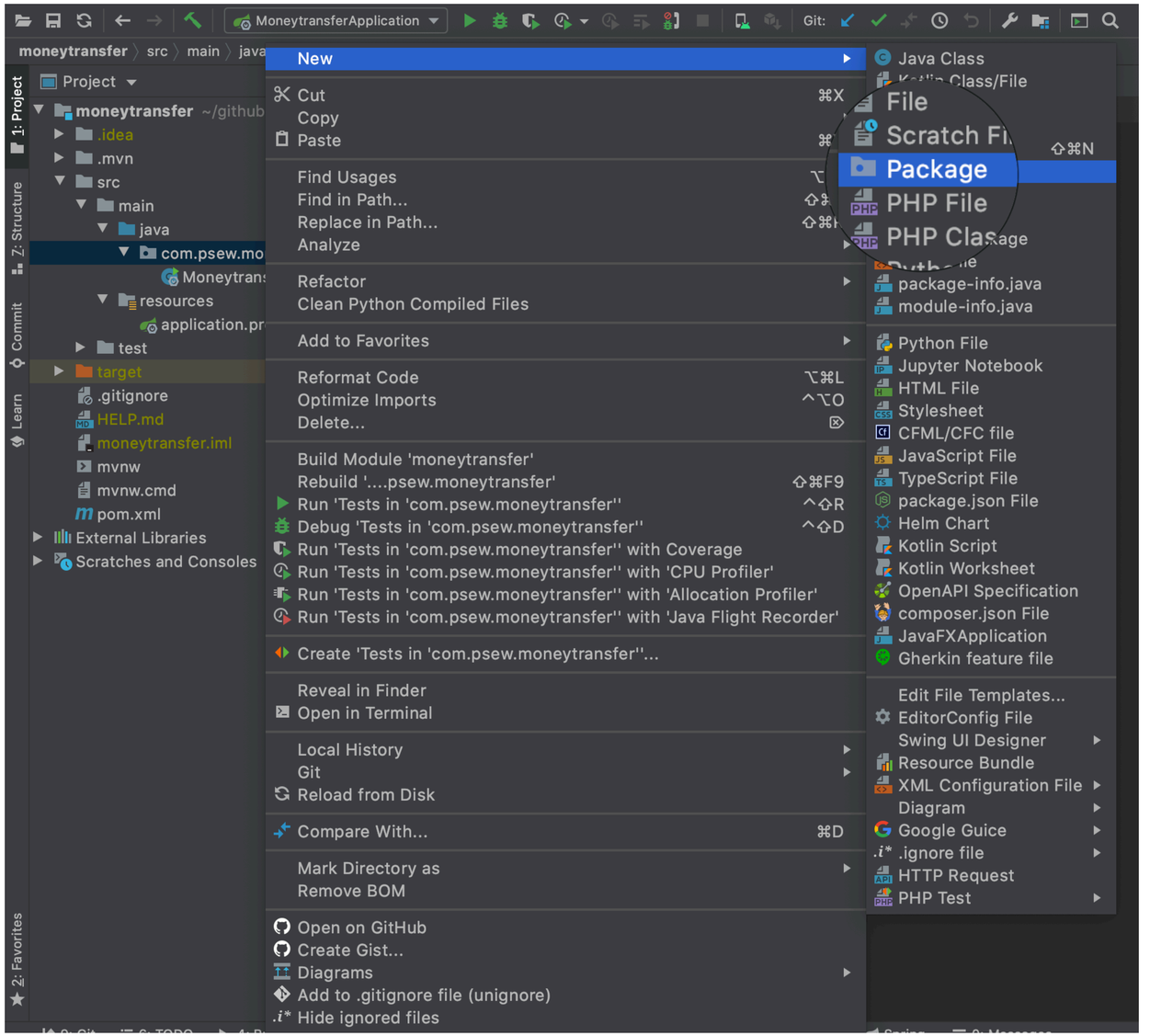
```
› git commit -m "Initial commit - Spring Boot project created with dependencies"
  [master (root-commit) 95d2c1c] Initial commit - Spring Boot project created with dependencies
   10 files changed, 758 insertions(+)
   create mode 100644 .gitignore
   create mode 100644 .mvn/wrapper/MavenWrapperDownloader.java
   create mode 100644 .mvn/wrapper/maven-wrapper.jar
   create mode 100644 .mvn/wrapper/maven-wrapper.properties
   create mode 100755 mvnw
   create mode 100644 mvnw.cmd
   create mode 100644 pom.xml
   create mode 100644 src/main/java/com/psew/moneytransfer/MoneytransferApplication.java
   create mode 100644 src/main/resources/application.properties
   create mode 100644 src/test/java/com/psew/moneytransfer/MoneytransferApplicationTests.java
```

```
› git remote add origin git@github.com:PracticalSoftwareEngineeringWorkshop/money-transfer-api.git

› git branch -M main

› git push -u origin main
  Enter passphrase for key '/Users/biniam.asnake/.ssh/id_rsa': *********
  Counting objects: 26, done.
  Delta compression using up to 8 threads.
  Compressing objects: 100% (17/17), done.
  Writing objects: 100% (26/26), 52.66 KiB | 13.16 MiB/s, done.
  Total 26 (delta 0), reused 0 (delta 0)
  To github.com:PracticalSoftwareEngineeringWorkshop/money-transfer-api.git
   * [new branch]      main -> main
  Branch 'main' set up to track remote branch 'main' from 'origin' by rebasing.
```

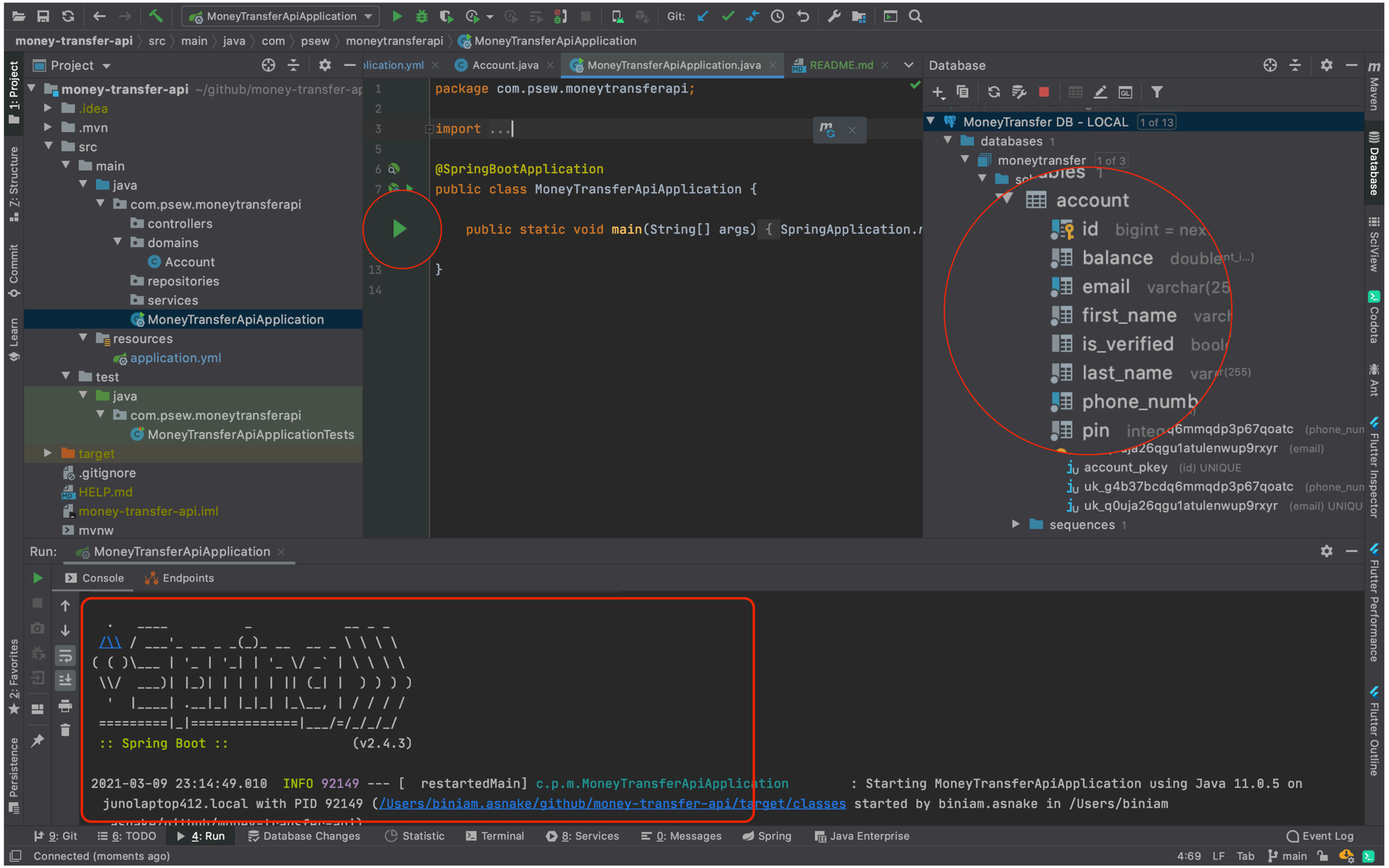
! YOUR CODE IS NOW ON VERSION CONTROL SYSTEM !



CREATING A PACKAGE ON INTELLIJ IDEA



APP – RUNNING AND TABLE CREATED



HTTPS://GITHUB.COM/BINIAMA/MONEYTRANSFERWORKSHOP

HTTPS://GITHUB.COM/PRACTICALSOFTWAREENGINEERINGWORKSHOP