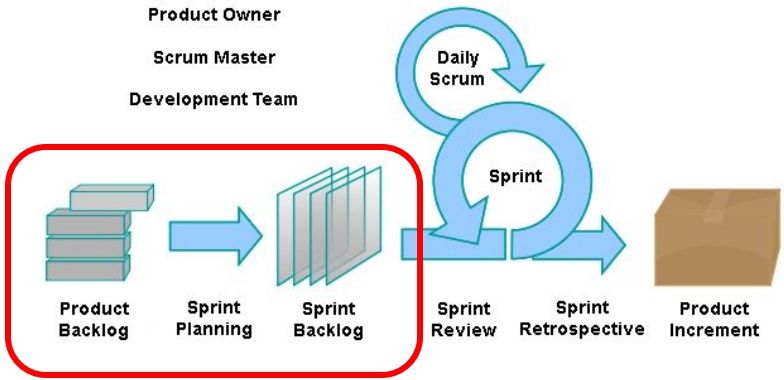
Working Agile

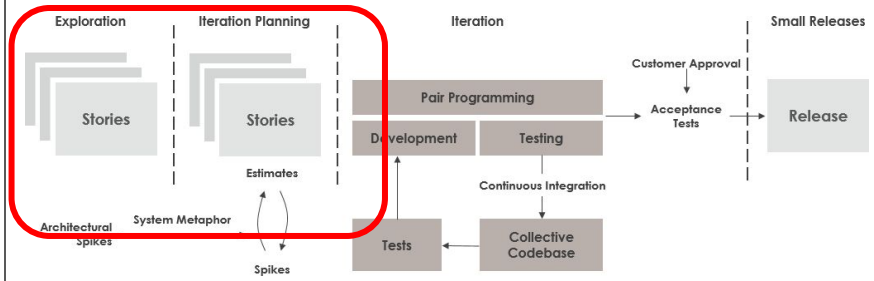
Repeating the Success

How to bring it to practice?

Scrum



XP

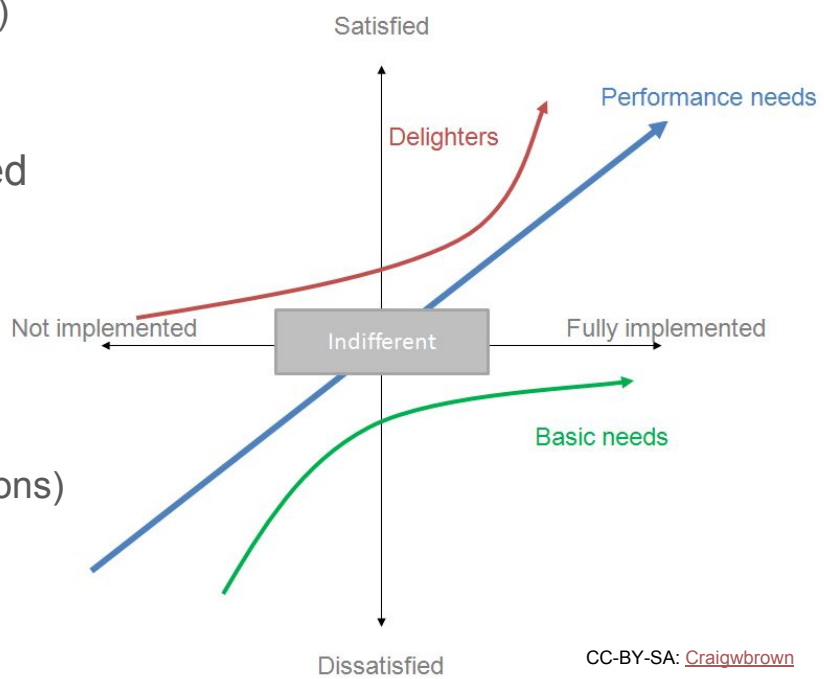


Know what you're going to do

- Product Vision
  - Create **Value fast**
  - Involves Users'/Stakeholders' needs
- **MVP** = Minimal Viable Product
  - Have a working + valuable product fast
  - Get better with every iteration
- Change will come
  - Be aware that **things will change** - embrace it
- Stay focused
  - Don't create Santa's Wishlist
  - Think "agile" - iterative
  - **Fail Fast**

What users expect

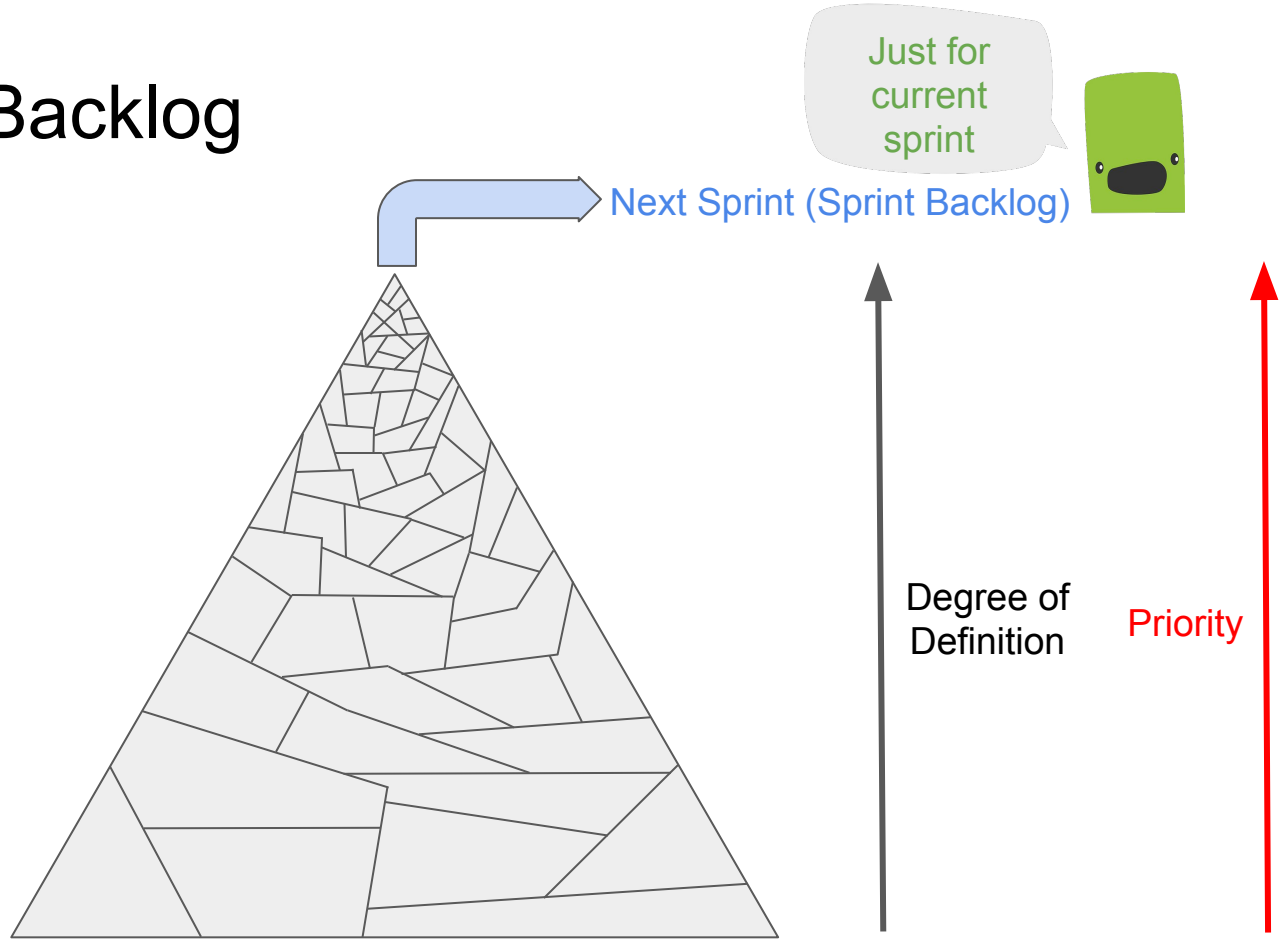
- Herzberg's Two Factor Theory
  - Motivators (satisfaction through their presence)
  - Hygiene Factors (dissatisfaction through absence)
- Kano Model (Customer Satisfaction)
  - Different Levels of "Quality" - mixture needed
    - Basic/Must needs
      - if missing: **dissatisfaction**
    - Performance needs
      - the more, the better
    - Delighters/Exciters:
      - Not required, but **satisfy** (e.g.innovations)
    - Indifferent
      - Neither positive nor negative impact
  - Delighters can get Basic Need over time



The Backlog

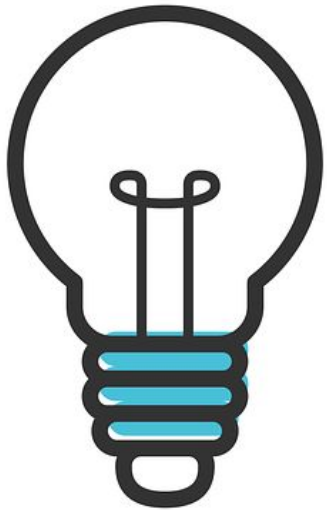
- Represents what needs to get done
  - Create value fast
  - Involves Users'/Stakeholders' needs
- Always Prioritized = the higher, the faster implemented
  - **In relation** to other stories (not equally important - not absolutely defined)
  - Can Change/re-prioritization: **Backlog Refinement/Grooming**
- Maintained by Product Owner
  - But involves developers, stakeholders,...
- Prioritization
  - Depending on Business Value, consequences if not implemented, dependencies,..
  - But: create value **and** fail fast

Building the Backlog



Product Vision

- Answering the questions: Why are we here and what are we working for?
- Guiding the development and involved people towards a shared goal
- A product vision should give insights into…
  - The intended target group
  - What values the product creates/needs it fulfills
  - What will the product roughly look like to create this value
  - How it compares and differentiate to competitors
  - How the product makes money
  - If it is realizable/feasible for the project



Release Goals (Planning)

We're agile: no Waterfall definitions! ;)

But, especially in longer projects (several months):

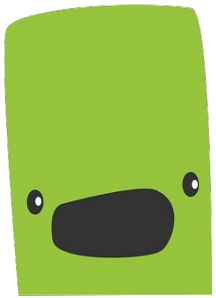
- Release Planning often needed in industry for Stakeholder, Deadlines,...
- Release plan not required, but not conflicting to Scrum
  - However, it might change/is not fixed - should be flexible
  - Prediction with Burn-Down Charts, Velocity, etc. possible (no force - get commitment!)
- Therefore: Define MVP
- Roadmap of the Product, represented by Backlog
- Benefit of Agility: Product is potentially shippable after every iteration

User Stories - What Users want

- **Why do we need them?**
- **Who can write/define them?** Everyone who is involved
- **Who manages them?** Product Owner is responsible for the Backlog
- **When?** Before the sprint, depending on Backlog
- **How?** Write them down (Best: A5 landscape)
- **What should they look like?** Be brief and focused
- **How many?** As many as needed in near future (YAGNI)

A Simple User Story

Be Aware if not given: developers tend to take themselves as reference for their work, not the users ;)
Solution: Personas



As a _____ I want to _____ so that _____

- Keep them short and focus on the essential stuff needed
  - If too long: split (-> Epic)
- Define them when needed
  - i.e. don’t waste time on unneeded stories due to low prioritization
- Need to can get finished (no “open” stories)

APP-05
[Story ID]

13
[Estimated Story Points]

As a student I want to see the lectures for my studies so that I can plan my semester

[Meaningful Title]

May also include:
- Relevant Dates (creation,...)
- UX/Security person in team
- Accepted by...

MM
[Story-Creator]

CS
[Developer]

Backside = the dark side of the moon

- User Story should be **self-explaining** (no further details needed)
- However, ambiguity, exact requirements, etc. are common in industry
- Solution: Further information on the Backside (just if really needed!)
  - Acceptance Criteria
    - **Requirements** that must be met
    - May be functional or non-functional
  - Conditions of Satisfaction
    - From **user view**
    - Must be met to fulfil the story

Please note..

- Refactoring
  - If it creates **direct value to the users**: It's a story (e.g., new Android Version)
  - If just indirect value: Do it when you touch the code anyway **through a story**
  - *when there is enough time (e.g. improve architecture, replace libraries,...)*
- Non-functional Features
  - May be in **Definition of Done** or as **Acceptance Criteria**
- **YAGNI** = You aren't gonna need it
  - From XP: Implement it just when it is necessary
  - Same for Stories: Define them in time (the higher in Backlog, the better defined)
    - Story might change very often
    - Hard to maintain too many small stories
    - Not sure if this feature will ever get implemented

Epic Stories

- Avoid complex Monster-Stories to get into a sprint
  - Especially if they are likely to not get finished (too many criteria, unclear,...)
  - However, might be useful to get feedback from stakeholders, users,.. In advance

- Instead: Having an epic-story, consisting of several user stories
  - Each should fit into a sprint alone
  - Epic can last over several sprints
  - May be depending on each other (order in which they get implemented)
  - Each creates value on its own/doesn’t make the product worse
    - i.e., still have a shippable product at any time independent if epic is fully implemented (e.g., does not crash, unfinished features not visible,...)

Definition of Done

- Criteria when a Story is finished and ready to release
- Followed by the whole team
  - kind of part of a team’s culture
  - Different from team to team/project to project
- Ensure sustainable development, quality and value
- Examples:
  - Non-functional requirements (standards, performance,...)
  - Coding Standard
  - Testing
  - Usability test for specific stories
  - …. - depending on team and project

Planning Event

- Participants
  - PM/PO/Manager,...
  - Development Team
- Motivation
  - Same Understanding
  - Discussing upcoming stories
  - Getting commitment
- Outcome
  - The work for the next sprint

Planning Event for the next Sprint

- Go through stories **with the team**
  - If not already done: estimate them (-> Planning Poker)
- Repeat until developer team says stop (= **limit they think they can deliver**)
- Team commits to the stories and moves it from Backlog **into Sprint Backlog**

Can the scope of the sprint change?

- Shouldn’t, but may happen in practice (should not be common)
- How?
  - PO/manager/.. proposes new story that is urgently required (*possible next sprint?*)
  - Commitment by team to add story in current sprint
  - If needed: remove other stories from Sprint-Backlog and postpone them

Playing the “Planning Poker”

- Discuss and estimate issues from prioritized Backlog
  - = Usually at start of a sprint, but can also happen independently
- Discussion
  - All having the same understanding (Devs and PO)
  - Are clear and ready to get implemented
- Estimation
  - Individual for every developer
  - Discuss results - lowest and highest estimation defend
- Ends when team commits to have enough tickets
  - Not predefined by PO or velocity (but should usually align)

But, how to Estimate?

- Estimation is more than just time, size,...
- Combines several aspects (also risk, effort,...)
- Therefore: Relational estimation (i.e. #1 > #2 or #1 = #2)
- Everyone in the (Development) team is included
  - Heavily depending on experience
- How to measure?
  - Fibonacci, numeral sequence, T-Shirt sizes,...
  - How NOT to estimate: hours, LoC,...

Example Poker Cards

- Different Designs/Estimation Methods
- Either offline or online (apps)
- Idea: Pictures as “guide”
- If estimation high: split story?
  - E.g. if one can’t do it alone

Source cards (hard copy): https://ulassa.com/tool/planning-poker-cards/

Playing the game

1. Story gets introduced
2. Story gets discussed till it is clear to everyone
3. Every developer estimates the story on its own (secretly)
4. Poker card for the estimation gets laid on the table (face down)
5. All Developers turn around their card, showing it to the others
6. The one with the highest & lowest estimation argue their point
7. The team finds consensus for an estimation and state it

Velocity

- Over time, story-points per sprint will get constant (more or less)
  - Velocity = SUM(story points)/COUNT(sprints)
  - However, velocity influenced by: vacation, sickness, new members/projects,...
  - Velocity is no guarantee to get as many story points finished in a sprint (just

- Less story points at project beginning
  - Especially if it’s new team - needs to find common ground in estimation
  - New product = time for introduction in product needed
  - High chance of over-estimation in the beginning
  - E.g. in 2nd sprint probably double the story-points of the first sprint will get implemented
  - Over a couple of sprint, velocity will get steady

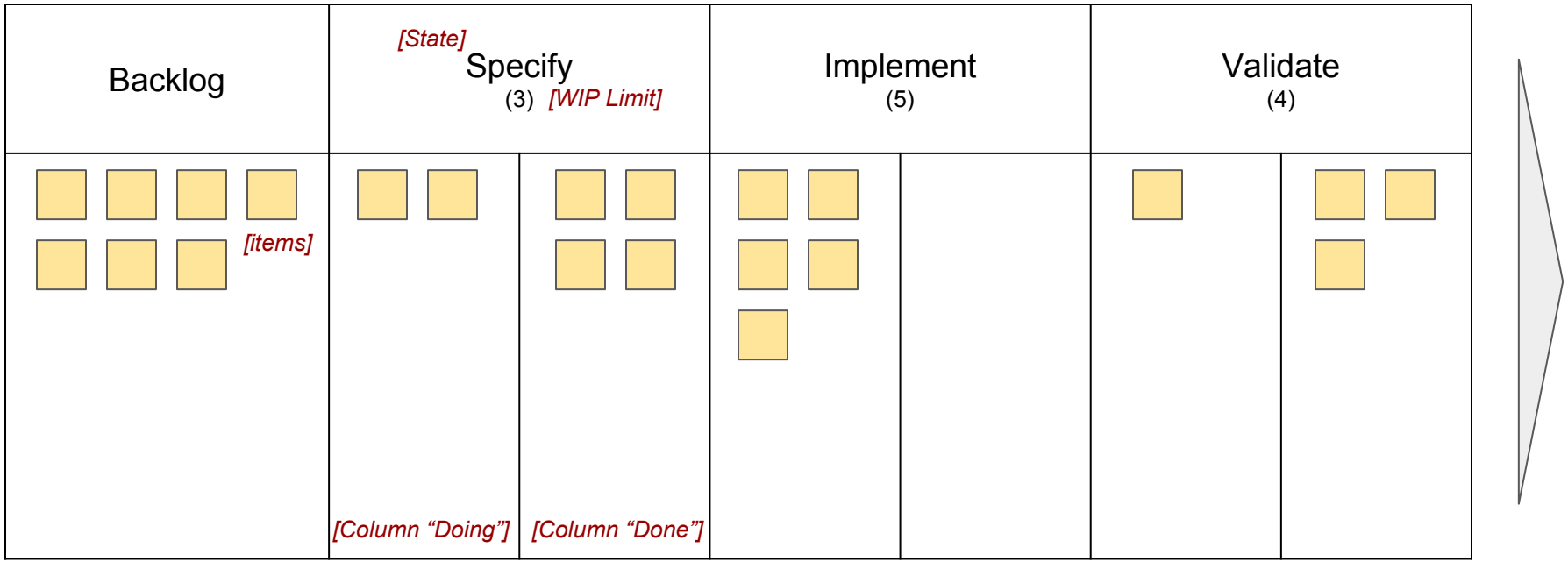
How to manage User Stories and the Backlog

- Aims:
  - Visible for everyone
  - Easy to use
  - Always up to date
  - Allows to response to change
  - Fostering Discussion and Communication
- Requirements:
  - Accessible for everyone
- Solution:
  - Using “Boards” to organize stories
  - Either on wall in office (offline) or special software (Jira,...)

Kanban - the next agile Framework

- Similar agile approach - create value and have a working product
- Not time boxed (no sprints, fixed time spans, etc.)
- No pre-defined roles
- But: also Backlog and is built on communication/teamwork
- Certain set of rules of working together (WIP, Definition of Done,...)
- Team defines together how to organize their work

That’s Kanban’s signboard



How it works

- Items (stories) are placed depending on their state in a workflow on a board
- Common Steps/States: **Specify -> Implement -> Validate** *-> Deliver*
  - *Delivery: Usually in bulk (more than one item at once) and not tracked on board anymore*
- Two columns for every state:
  - Doing: The item is currently in progress
  - Done: The item is ready for the next state *(Definition of Done!)*
- WIP (Work in Progress) Limits
  - Maximum number of items in a specific “doing” column of a state
- Backlog constantly groomed/maintained (also by the team)
- Team (~ its members) organizes itself

Conclusion

Let’s create value

… and be successfull

- Know your product
  - And what your customers need
- Take care of your backlog
  - It lives and adopts
- Use User Stories
  - Don’t be a code monkey
- Communication
  - Planning
  - Using accessible boards

source: https://knowyourmeme.com/photos/789208-gif

Take Aways for practical work

- Have user value in mind (don’t dissatisfy!)
- Prioritize your work (in relation to others)
- Definition of Done
- Work with User Stories and split them if necessary
- Estimate stories (in relative values)
- Be organized and open -> use boards