Software Development

⇒ Typical Software development related activities:

1. Collect and analyse requirements
2. Validate Ideas, feasibility study.
3. Design and Prototype {for proof of concept}
4. Write code and tests
5. Write documents and guides
   - → Design documents
   - → Test Scenario description
   - → User guides
   - → API Documentation etc...
6. Release, Support & maintenance.

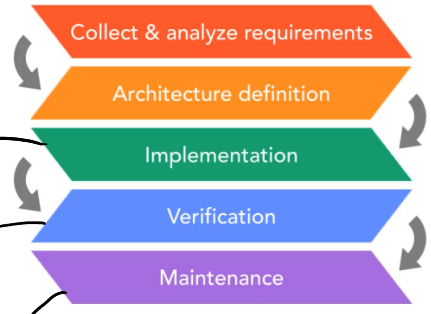
★ Software development Methodologies

⇒ Describes how to organize the activities involved in the software development process.

① The Waterfall model

⇒ It is a linear model, it defines development in steps or phases.

⇒ Each development phase is built upon the previous phase.



Implementation phase is usually devided into smaller units. Each unit is then implemented and tested.

The software is evaluated based on predefined criteria. We test for function, performance, security and usability.

Maintenance phase is about fixing smaller bugs, functional enhansments.

⇒ Use the waterfall if the requirements are clear and wont change frequently.

Pros

1. Well-defined specification and design.
2. Early problem identification.
3. Knowledge preservation.

Cons

1. Inflexible (Based on the assumption design can be defined at early stages)
2. Requirements must be well defined early on.
3. Time consuming architecture defination phase.

⇒ When to use waterfall model:

- Fixed scope and requirements.
- Mature/Legacy projects.
- Technology is reliable.

② Agile Software development

Core Values

Individual and Interactions over Processes and Tools

Customer collaboration over Contract negotiation

Working Software over Comprehensive Documentation

Responding to change over following a plan

⇒ The main idea behind Agile is that we can deliver functional software iteratively, instead of delivering the entire project all at once.

⇒ The work is broken up into smaller chunks called sprints.

Sprint

- The sprint is usually two to four week long.
- At the end of each sprint the team should provide something thats an inprovement over previous sprint outcome.
- This interactive approach provides an opportunity to frequently review the product thats being developed.
- Stakeholders have a chance to evaluate the software and provide there feedback early on, rather then waiting for the final product to be delivered.
- These frequent checkpoints are super useful as they ensures that project evolve in right direction.

⇒ Agile methodology do not seperate testing from development.

⇒ This model works best in situations where requirements cant be defined upfront.

⇒ Agile is not a methodology but rather a way of thinking defined by the agile manifesto values and principles.

⇒ Methodologies that implement agile approach

- Scrum
- Kanban

Pros

1. Quick results
2. Adaptiveness
3. Customer satisfaction
4. Less waste

Cons

1. Inaccurate estimation
2. Collaboration is time-consuming.
3. Issues caused by lack of documentation

⇒ When to use Agile?

- Vague requirements
- Moving target
- Need to involve the client
- Technology is unknown

★ Scrum

⇒ Came from rugby. Based on collaboration.

⇒ Simple framwork for complex project.

Scrum

- A way to organize team work around agile principles.
- The work is organized in short, sustainable burst of activity called sprints.
- Sprint is usually two week long.
  - Keeping the sprint short ensures that the project is evolving in right direction.

Scrum team

- Typically consists of around five to nine people.
- **Roles**
  - Product Owner
  - Scrum master
  - Team member

1. Product Owner

⇒ Product owner represents the customer. Duty is to clarify wht the customer wants.

⇒ Communicates a lot with the clients.

- Documents the client need and expectation about the software product.
- Product owner holds the vision of the product.
- Acts as the bridge between the customer and the team.

② Scrum master

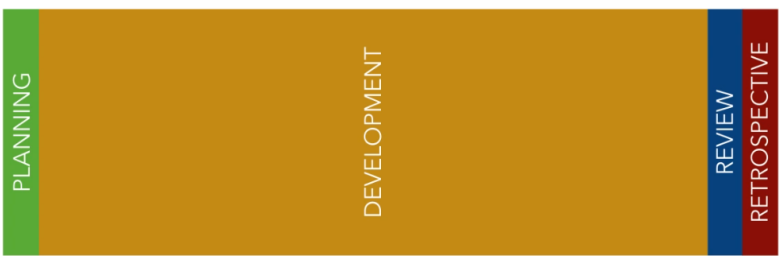
- Acts as a coach and coordinates work within the team.
- Helps in removing any blockers in the flow of work.
- Protects the team from unimportant, disturbing events.
- Guide the team.
- Helps the team in applying scrum and agile principles.

③ Team Members

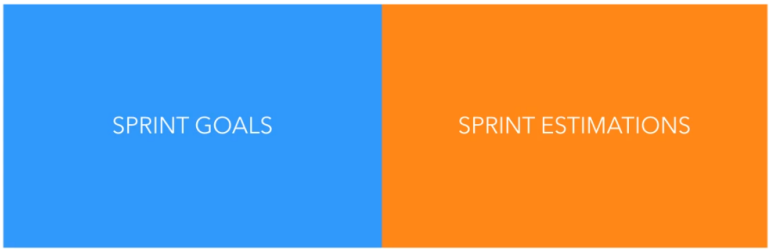
- Have complete authority and responsibility.
- Provides and own estimates.
  - We must provide estimates. (Sometime based on our gut-feelings which usually dosent end well)
- Desides how to do the work.
- The team compleates user stories.
  - Involves Development, writing unit test, testing, fixing bugs, writing documentation
- Self organize to get the work done.
- If they need to talk, they need to organize ad-hoc meetings.
- A scrum team does not need a managet to organize meetings.

Sprint

1. Each sprint starts with a planning phase.
2. Followed by development related activities.
3. At the end of the sprint there is a sprint review.
   - Team shows what they have achived during the sprint. And the customer can provide feedback about the product.
4. Finally there is a retrospective meeting.
   - Here team discusses what went wrong and what went well during the sprint.



Planning phase



⇛ Planning actually consists of two parts

- → Understanding the sprint goals.
- → Breaking the sprint goles to managable unit of work and providing estimations.

Sprint goal meeting

→ During the sprint goal meeting, the product owner discusses the sprint goles with the scrum team.

- → The product owner tells what should be done by the end of sprint. (based on prioritized list of user stories)
- → The team member and scrum master can ask questions and collaborate with the product owner to come to a good understanding of the sprint goals.

Sprint Estimations meeting

⇛ The product owner does not have to join this meeting.

⇛ During the effort estimation meeting, the team breakes down the user stores into smaller units of work called backlog itemes.

{ A backlog item must be small enough to be completed within a sprint. }

⇛ Then each backlog item is decomposed into smaller task. → { Smallest unit of work }

⇛ Any arbitrary team member can pick up a task and work on it independently.

⇛ The tasks of a backlog item may or may not depend on each other.

- → It is important to highlight dependencies between tasks.

➔ After estamating the prioritized list of backlog item, it should be clear whether all or only a subset of the backlogs fit in the given sprint.

➔ The team should only commit to the amount of work that can be compleated without compromising the quality of the end result.

{ Example }

Backlog items

Add logging capabilities	5 PD	PRIO 1
Implement user authentication	2 PD	PRIO 1
Offline persistence for sales orders	1 PD	PRIO 2
Implement Settings UI	2 PD	PRIO 2
Increase unit test coverage to 75%	3 PD	PRIO 2
Create iPad UI mockups	2 PD	PRIO 3
Add automated UI tests	4 PD	PRIO 3

Current Sprint

→ Line of Commitment

- Only the people who do the actual work know the effort it takes to complete what they committed to.

- Sprint planning should be time boxed:
  - Sprint goal meeting should not exceed 1hr.
  - Sprint Estimation meeting also should not exceed 1hr.

Development Phase

Daily Scrum

- Each day of the sprint starts with a brief meeting called daily scrum.
- Every one should stand during meeting, which contribute to keeping this meeting short.
- Max duration of this meeting 15 min.
- In this meeting, each team member should answer three questions:
  - What did I work on yesterday
  - What am I going to work on today
  - Where there any issues or blockers that prevented me from doing my work.

Review Phase (Demo)

- Team demos sprint achievements.
- Stackholders have the chance to inspect the product, and see how it evolves.
- Customers can provide direct feedback about the product being developed.
- The product owner collects the findings of the review meeting and creates new product backlog if needed.
- The sprint review shall not be longer than one hour per sprint week.

Retrospective Meeting

- The aim of the retrospective is to improve the performance of the team.
- Team members address the following questions:
  - What went well during the sprint
  - What could have gone better
  - What could be improved in the next sprint

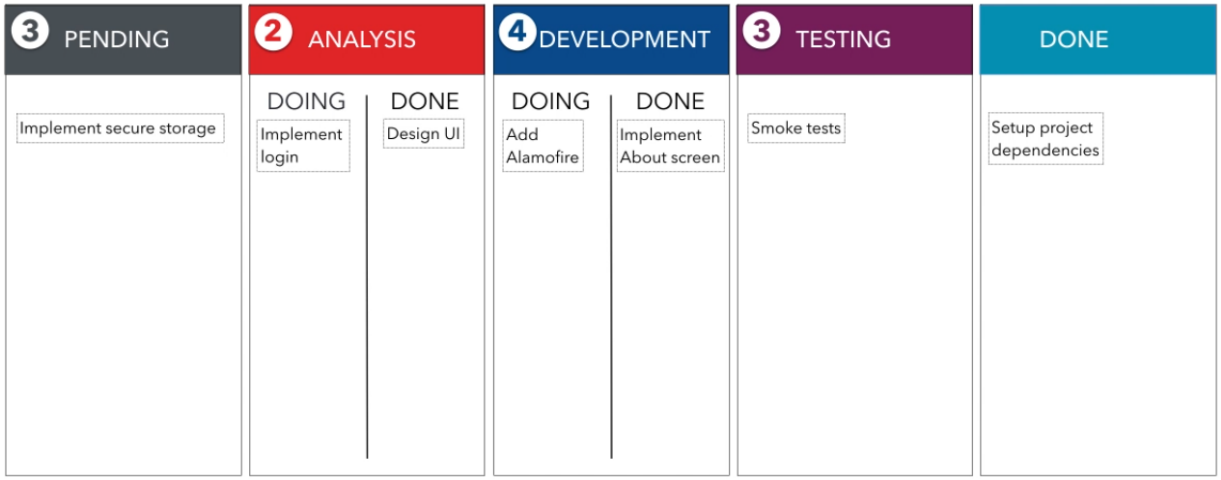
★ Kanban System (objective: Work should not pile up)

⇒ Kanban uses a big board to visualize pending work and the current capacity assigned to the given development phase.

⇒ Each column represent a phase in the development cycle.

⇒ The cards represents work item as they flow through the development cycle.

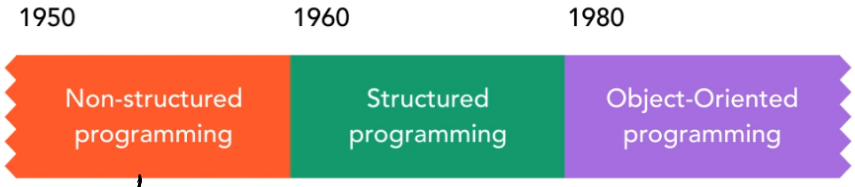
⇒ On the top of each column, there is a number representing the limits on the number of cards allowed for a given phase.



{ Example of a typical Kanban board

⇒ This simple mechanism prevents work to pile up and shows bottlenecks dynamically.

History of Programming



⇒ Code consists of sequential order of instructions.

⇒ Each state will go in a new line

⇒ Example: Basic

⇒ Problem
difficult to understand and maintain.

⇒ Breaks down code into logical steps.

⇒ They rely on Subroutines which contains a set of instructions to be carried out.

⇒ Example: C

Variable

Data Structure

⇒ Structured programming could not address all the increased complexity.

⇒ Object-Orientation was the next big step.

⇒ The main idea was to split the program into self contained objects.

⇒ Each object represent the part of the system that gets mapped to a distinct entity.

⇒ The object which forms the system interacts with each other.

⇒ An object functions as a seperate program by itself. It operates on its own data and has a specific role.

The Unified Modeling Language

⇒ Graphical notation to communicate the design of software systems.

↳ We can use these diagram to describe the objects that forms a system and there interactions.

⇒ UML has many diagram types:

(A) Functional diagram

(1) Use-Case diagram

↳ Describes the functional model of the system from users point of view

(B) Structural diagram

(1) Class diagram

⇒ Can be used to describe a system in terms of objects, attributes, operations, and relations.

(C) Dynamic behaviour diagram

(1) Bahavioral diagram

⇒ Describes the functionality of a system focusing on what happens and the interactions between objects

⇒ UML is independent of any particular programming language, but it should be object orianted.

Class Diagram

⇒ Provides an overview of the classes that forms a software system and describes a static relationship between them.

⇒ In UML we represent a class by drawing a rectangle divided into three compartments

↳ It is quite common to leave out the attributes or the operations in early stage of development.

↳ First we need to figure out the required classes, we can add the details later as we realize whats needed to fulfil the required functionality.

Visibility

↳ Controls who can access the attributes and the methods of our class.

↳ UML uses the following symbols to describe the visibility levels

+ public

- private

protected

- You should hide everything that is not required for your systems proper functionalty.
  - Exposing too much leads to unexpected problems.
- We should provide public setters and getters instead of of allowing everybody to access our class data.

⇒ Describing parent child relation between classes.