# *C++ Software Engineering*

***for engineers of other disciplines***

Module 8

*"Software Engineering"*

*1st Lecture: SDLC*

ALTEN

***Autumn 2021***

***Gothenburg, Sweden***

***petter.lerenius@alten.se***

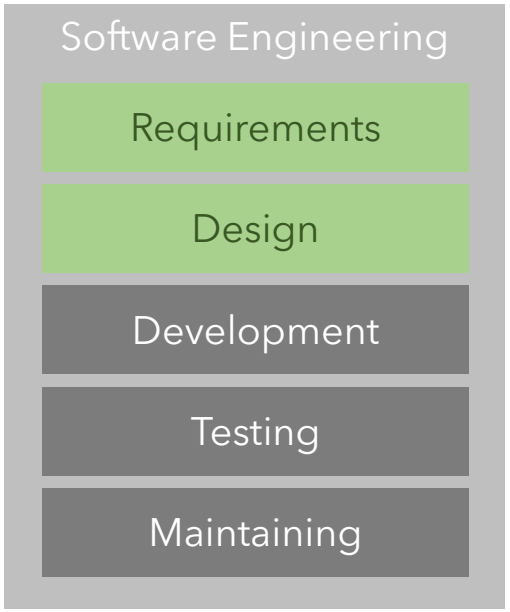
***rashid.zamani@alten.se***

# Software Engineering



ALTEN

- An engineering discipline focused on software development.
- Software = Executable + Documentation + Related Digital Media + Licensing.
- Software Engineering is an *ever-evolving* science, and lots of practical knowledge.
- Software engineering allows:
  - Tackling complex problem
  - Minimize costs
  - Increase quality and effectiveness



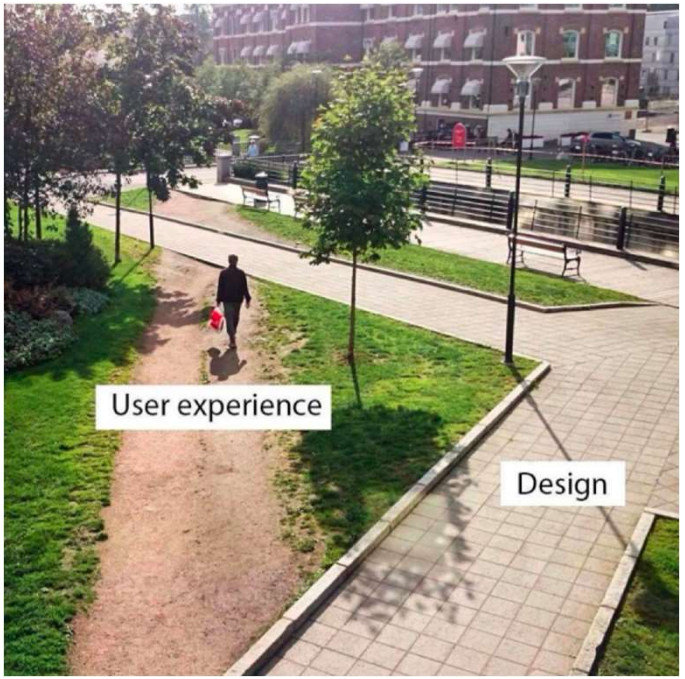

- Software Engineers are not the only people in software development, other roles, such as managers, business, and legal roles are vital.
- A software engineer role could only *focus* on one area of software engineering or more.

# Software Requirements



ALTEN

- Requirement the is needs of *stakeholders* which the *software* supposed to satisfy.
- Requirements are both functional (capability) & non-functional (condition).
- Gaining knowledge about the problem from different sources through *requirement elicitation.* Requirements are then *refined* and *specified*.
- Through out the project, requirements are traced, prioritized, document, and might even be modified through a process called requirement management.
- At the end of the project (acceptance test) requirements are *validated & Verified.*

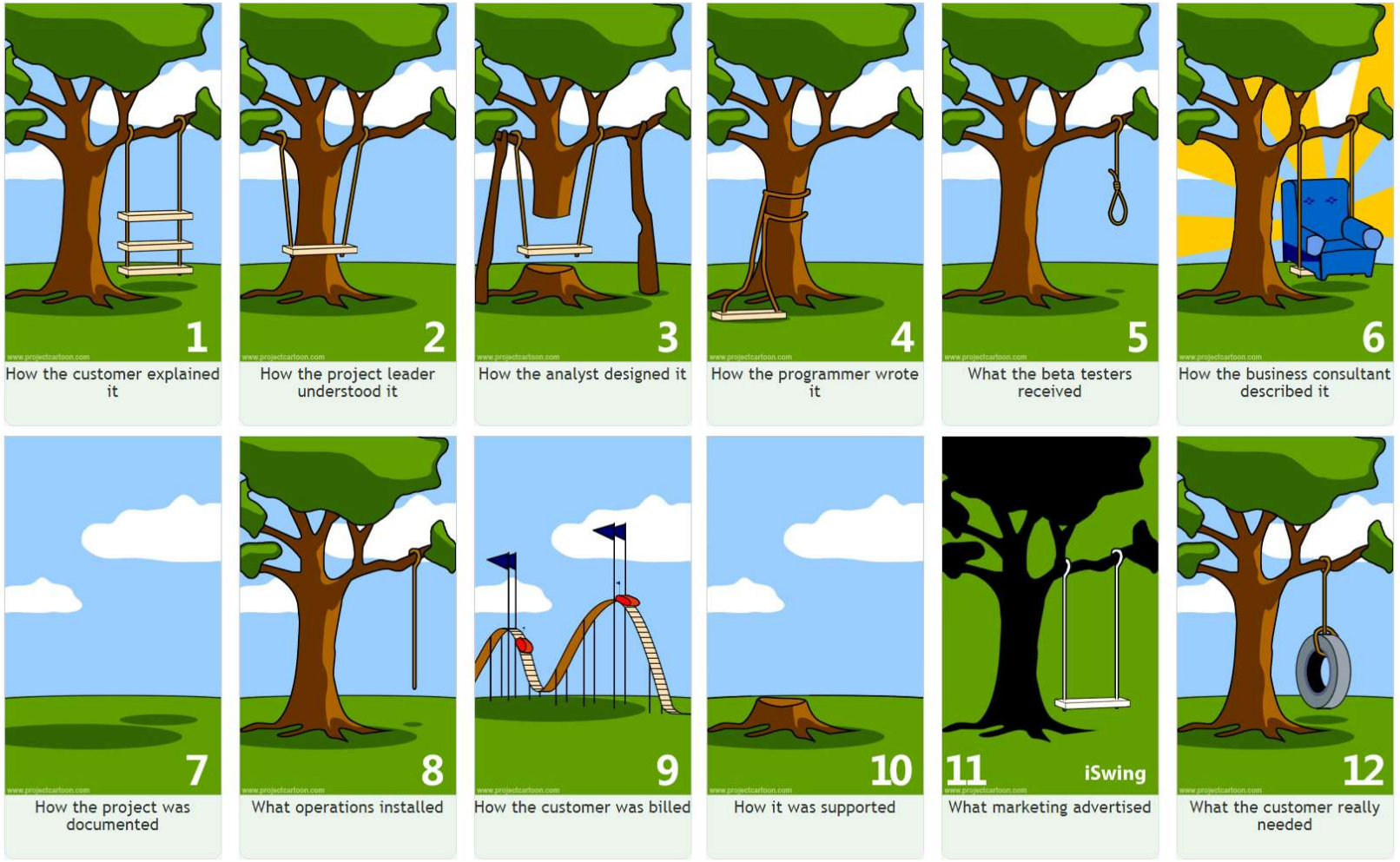



https://missunitwocents.tumblr.com/post/163749409715/desire-paths

- Requirements are the skeleton of the project and heavily affect architecture, design, and implementation of software. Massive requirement changes are costly, especially late in the development.

# Software Requirements






- Proper Documentation of the requirements ensure information integrity through out communications.



https://www.zentao.pm/share/trees-wing-project-management-cartoon-97.html

# Software Design

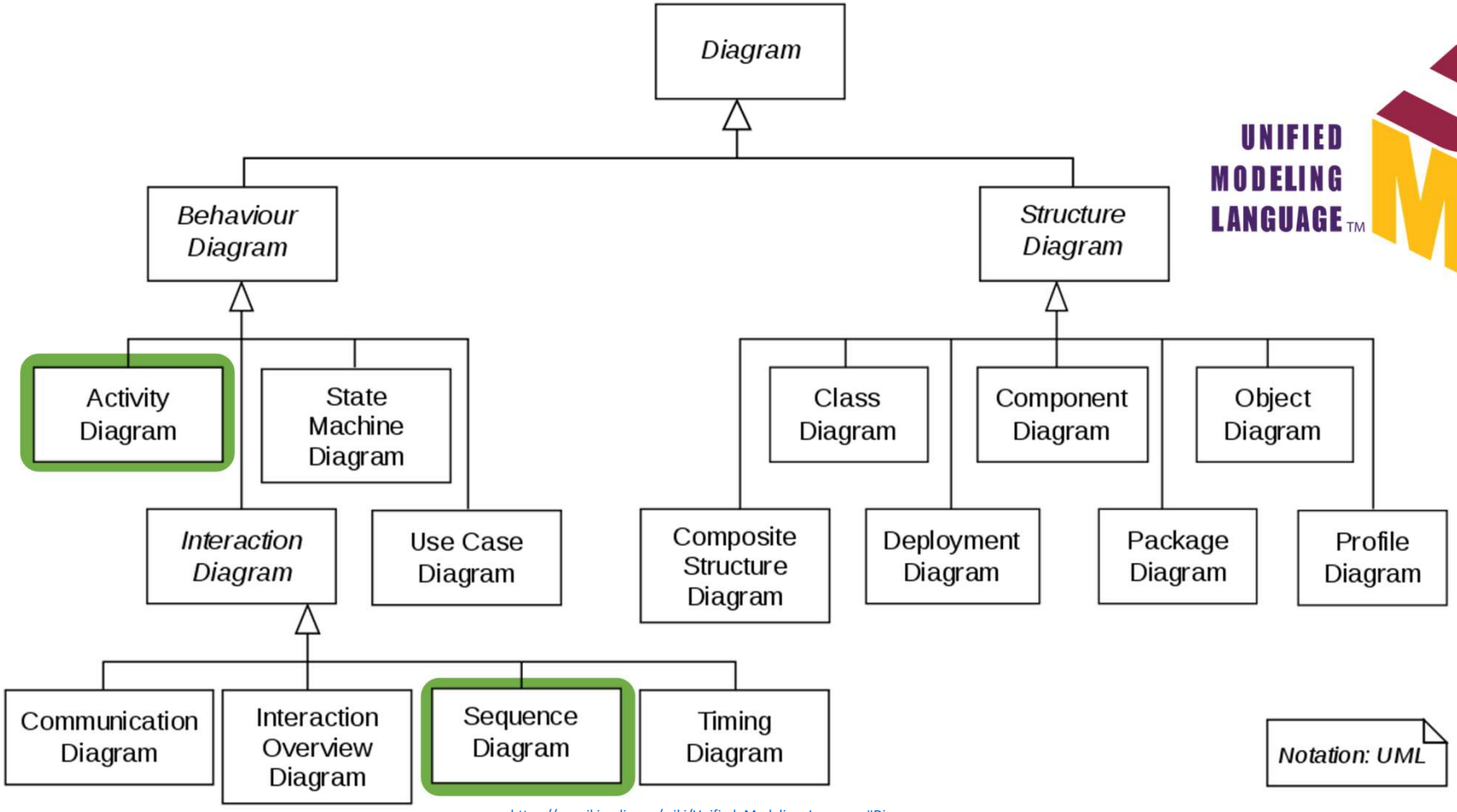



- Software Design is *transforming* requirements into:
  - ***Architectural Design***: highest abstract version – sub-system components are identified.
  - ***High-level Design***: identifies more details of each subsystem and their modules and their interactions.
  - ***Detailed Design***: implementation detailed and logical structure of module, and algorithm used.
- *Modeling Languages* are used to *express information, knowledge* or *system* structure.

*"Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraints."* https://en.wikipedia.org/wiki/Software_design

# UML



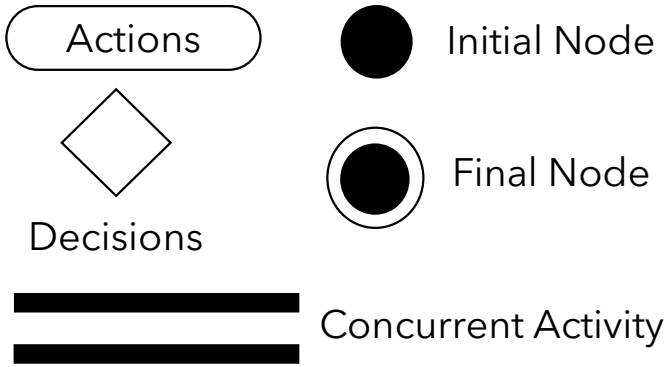


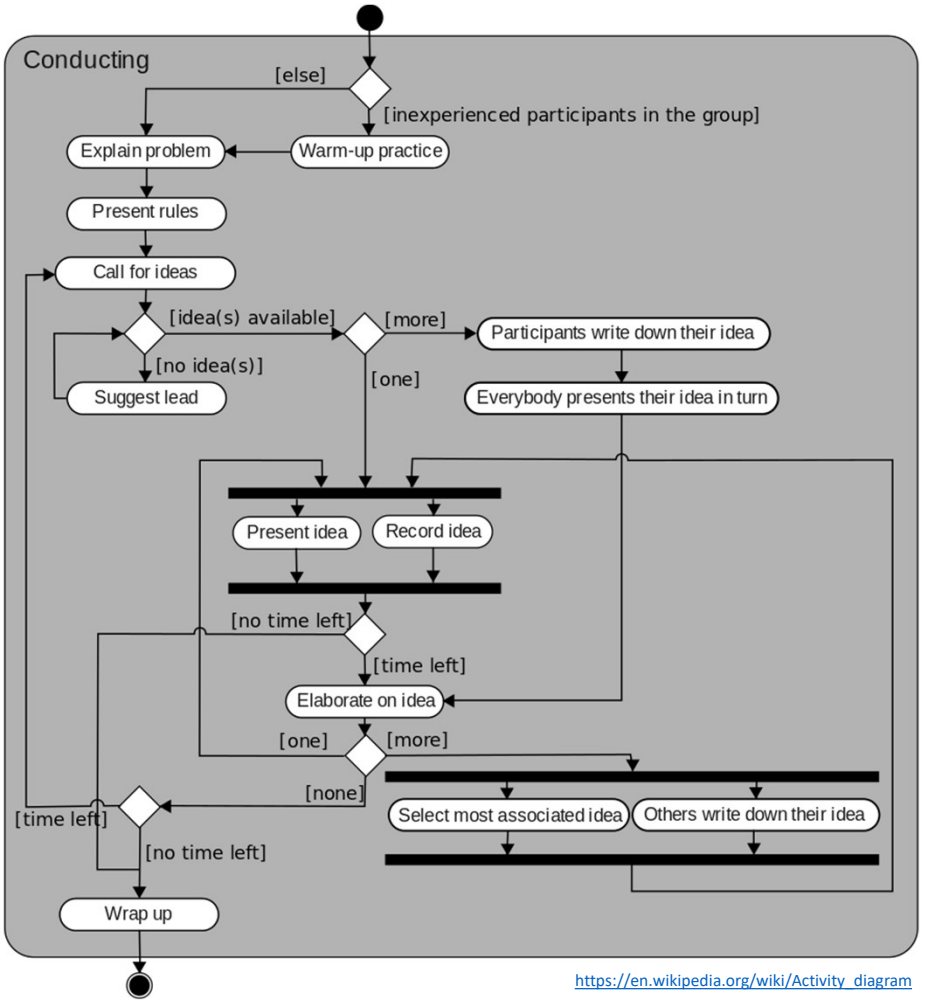

https://en.wikipedia.org/wiki/Unified_Modeling_Language#Diagrams

# UML – Activity Diagram



- Represents workflow with support for concurrency and iteration.
- Combination of DFD & flowchart.



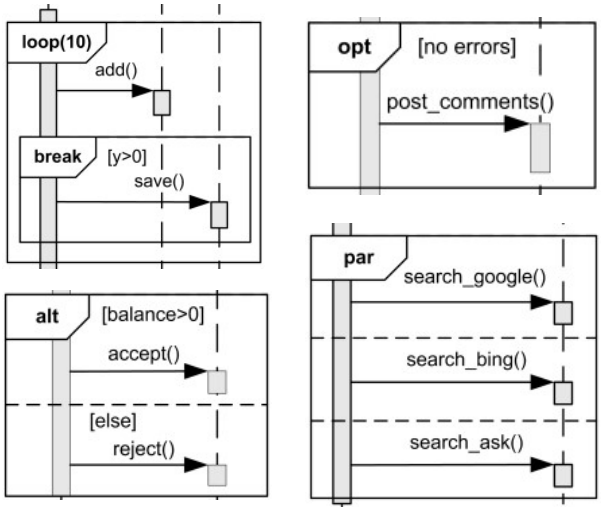




https://en.wikipedia.org/wiki/Activity_diagram

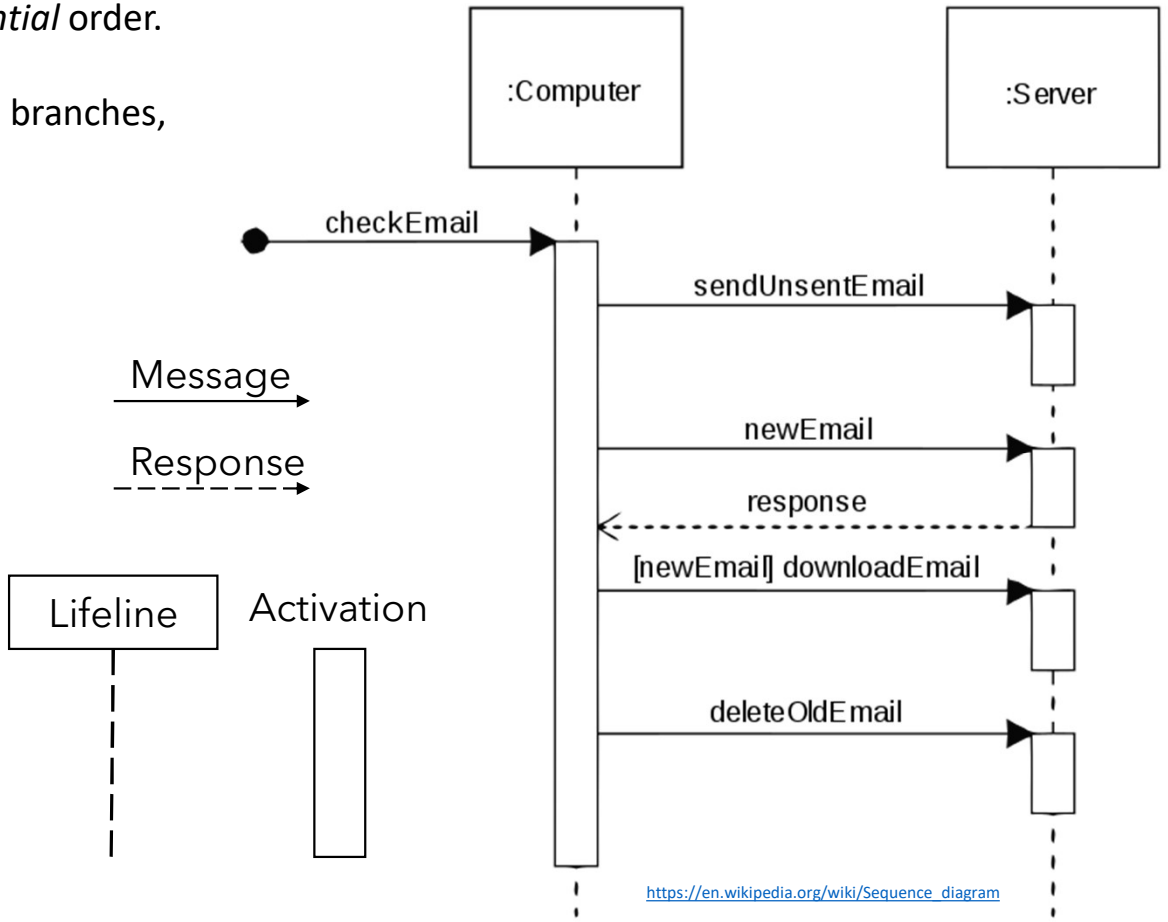
# UML – Sequence Diagram



- Demonstrates *object* interactions in *sequential* order.
- Since UML 2.0, it possible to show loops, branches, concurrency in sequence diagram.




https://www.uml-diagrams.org/sequence-diagrams-combined-fragment.html

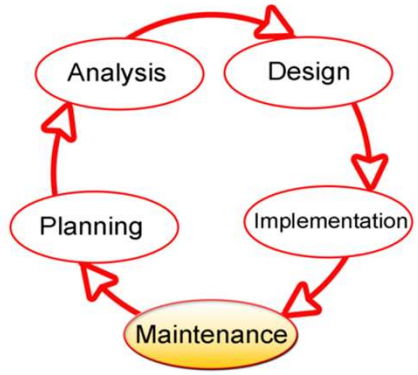



https://en.wikipedia.org/wiki/Sequence_diagram

# SDLC



- System Development Life Cycle is the complete process of an *information system.*
- There are different SDLC models & methodologies:
  - ***Sequential***: focuses on correct and complete planning as the guideline for the project (BDUF).
  - ***Iterative***: implement and enhance modules (limited scope) over many iterations.
  - ***Agile***: concentrates on *micro-processes* which welcome modifications through out the whole development cycle




https://en.wikipedia.org/wiki/Systems_development_life_cycle#Design

- Maintenance is a step in which the cycle could end.

# SDLC – IEEE Documentation



ALTEN

- IEEE provides standard for documenting all the processes within SDLC.
- Each document is well-defined in a separate standard clause.
- Software Requirement Specification (SRS) details *all* the project's requirements as well as providing an overall description of the product defining:
  - Scope of the work
  - All interfaces
  - Operational, design, and environmental constraints, assumptions, and dependencies
  - Product functions
  - User Characteristics

**IEEE software life cycle**

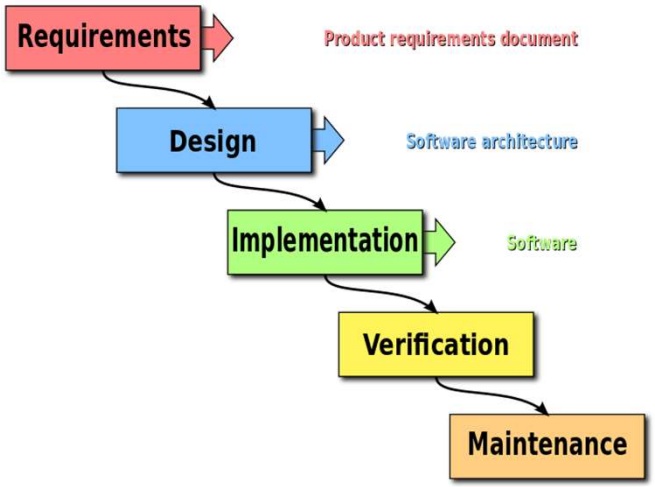
SQA – Software quality assurance IEEE 730
SCM – Software configuration management IEEE 828
STD – Software test documentation IEEE 829
SRS – **Software requirements specification IEEE 830**
V&V – Software verification and validation IEEE 1012
SDD – Software design description IEEE 1016
SPM – Software project management IEEE 1058
SUD – Software user documentation IEEE 1063

- IEEE documentation style is heavily linked to BDUF SDLCs and in most projects are considered as *over documentation*.

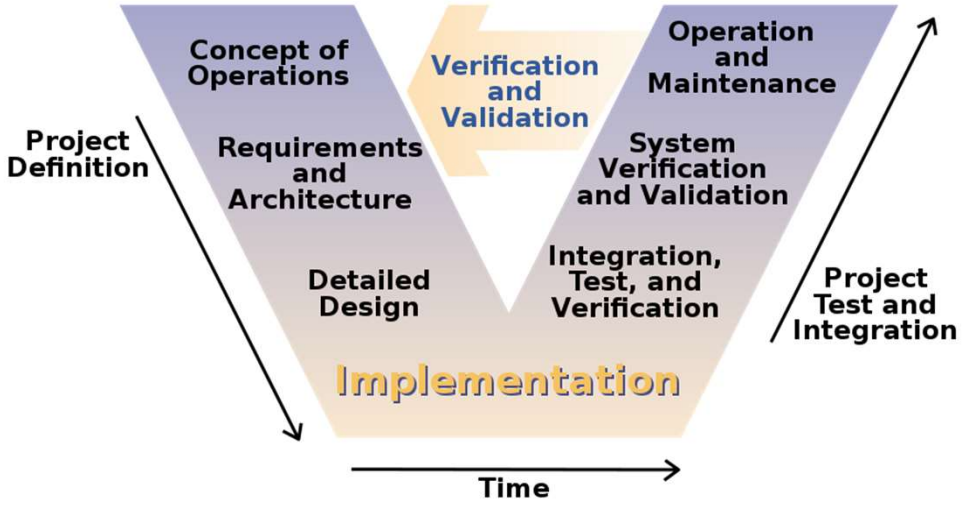
# BDUFs



- Big-Design-Up-Front SDLCs, put a lot of focus in the beginning to capture all the requirements, hence the name.
- Capturing all the requirement up-front is not possible in all domain. For complex projects, late release times could be a hurdle.




https://en.wikipedia.org/wiki/Waterfall_model




https://en.wikipedia.org/wiki/V-Model_(software_development)

# Iterative vs Incremental

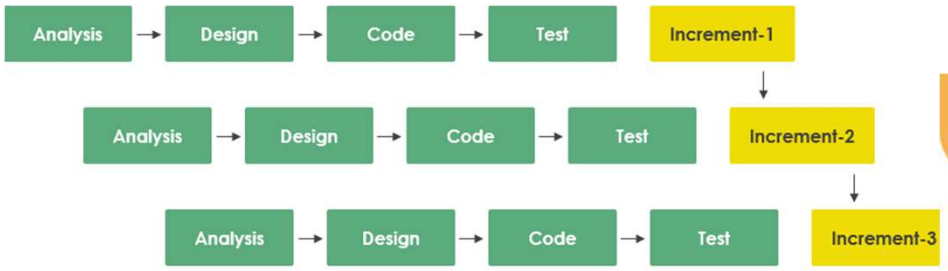


- Both iterative and incremental model provide the development team with feedback from the previous increment or iteration.
- Iterative development is suitable for scenarios where the final solution is hard to be defined.
- It is common to use a *hybrid* of the two.

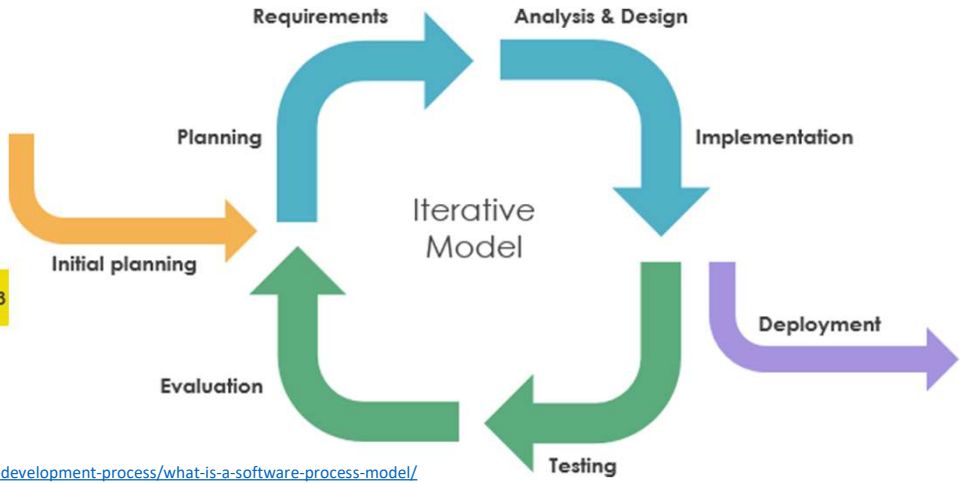

Incremental Development


Iterative Development

https://alisterbscott.com/2015/02/02/iterative-vs-incremental-software-development/




Incremental Model




https://www.visual-paradigm.com/guide/software-development-process/what-is-a-software-process-model/

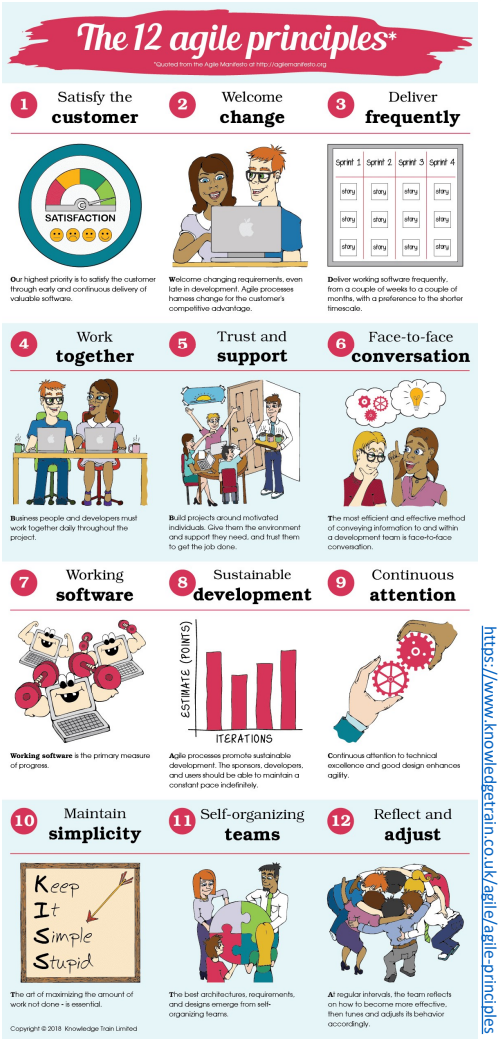
# Agile



- Agile is *iterative, incremental,* and *evolutionary* development.
- Through out many *timeboxes* a.k.a. ***sprint***s, *cross-functional* members, plan, analyze, design, code, test, and finally *releases* a minimal ***demoable*** *deliverable*.
- Frequent *releases* minimizes overall risks and can welcome changes.
- ***Lean*** software development, supported by *Agile community* is built on seven principles:
  - Eliminate waste
  - Amplify learning
  - Decide as late as possible
  - Deliver as fast as possible
  - Empower the team
  - Build integrity in
  - Optimize the whole

*"[Agile has] come to value:* http://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"*




https://www.knowledgetrain.co.uk/agile/agile-principles

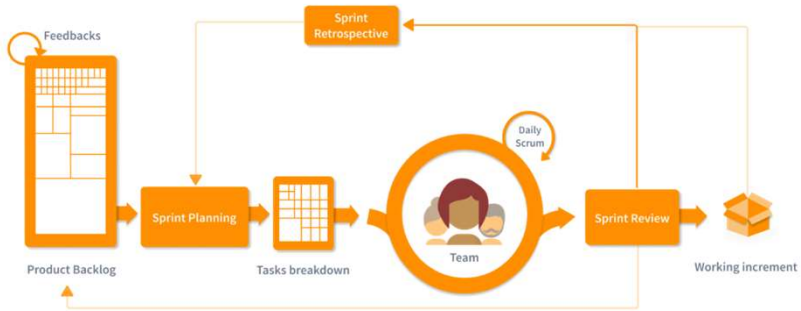
# Agile -- Terminology

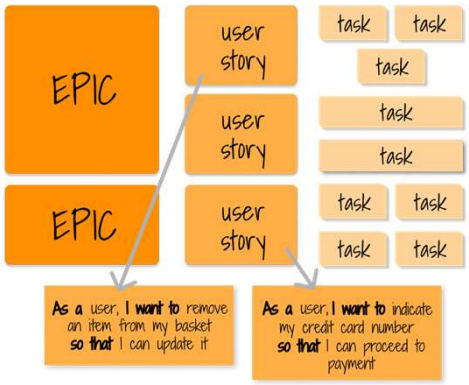


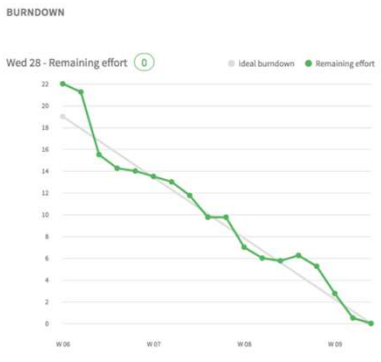

| keyword | Description |
|---|---|
| Epic | Big, *coarse* chunk of work |
| Spike | Timed effort to breakdown *Epics* |
| User Story | High-level definition of a requirement |
| Task | Smallest, most *refined* unit of work which is *"estimatebale"* |
| DoD | Definition of Done for a *Task* |
| Backlog | Prioritize repository for items |
| Sprint Backlog | *Spikes*, *Stories* and *Tasks* planned for a sprint |
| Product Backlog | Backlog for all the *items* in a project |
| Release Plan | *Group* of *Stories* planned for realese at the end of *the Sprint* |

# Scrum








https://www.tuleap.org/agile-scrum-in-10-minutes/

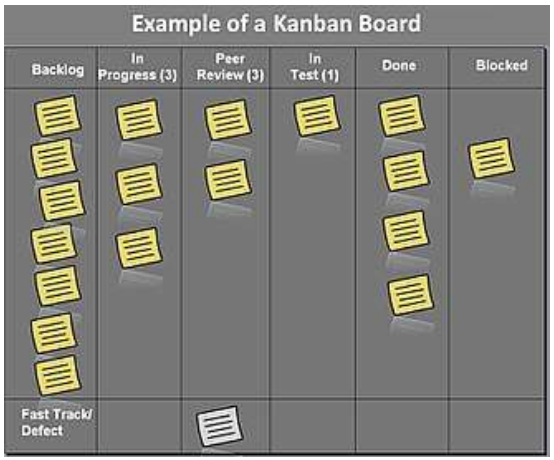
# Kanban vs Scrum



ALTEN

- *Both methods are agile*
- *Both use metrics to determine the optimum amount of work in progress*
- *Both encourage continuous improvement through transparency and collaboration*
- *Both focus on delivering frequent releases*
- *Both are centered around self-organizing teams*
- *Both require splitting the work into smaller tasks*

https://reqtest.com/agile-blog/scrum-vs-kanban/



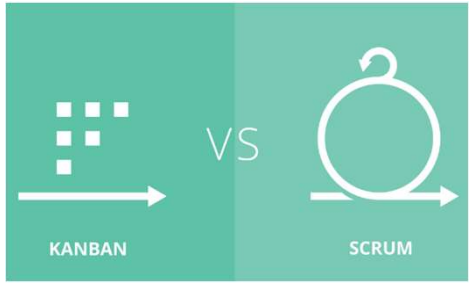
# Kanban



ALTEN

| Scrum | Kanban |
|---|---|
| Time-boxed iterations are an essential part | Time-boxed iterations are optional |
| Team commits to a specific amount of work for each iteration | No such commitment is required |
| Velocity is the default metric for planning and managing the project | Lead time is default metric for planning and managing the project |
| Cross-functional teams prescribed | Specialist teams allowed |
| Items must be broken down so they can be completed within a single sprint | No particular item size is required |
| Burn-down chart used | Only the board is used |
| Implicit WIP limits in a sprint | Explicit WIP limits per workflow |
| Estimation necessary | Estimation optional |
| Cannot add items to ongoing iteration | Can add new items whenever capacity is available |
| The sprint backlog is owned by the team | The Kanban board may be shared by multiple teams or individuals |
| Has definite roles | Doesn't have any roles |
| Uses a prioritized product backlog | Prioritization of tasks is optional |

https://reqtest.com/agile-blog/scrum-vs-kanban/



# SAFe



- Scaled Agile Framework (SAFe) is targeting large enterprises.
- Combines *Agile* and *Lean* techniques to provide

#1 Take an economic view

#2 Apply systems thinking

#3 Assume variability; preserve options

#4 Build incrementally with fast, integrated learning cycles

#5 Base milestones on objective evaluation of working systems

#6 Visualize and limit WIP, reduce batch sizes, and manage queue lengths

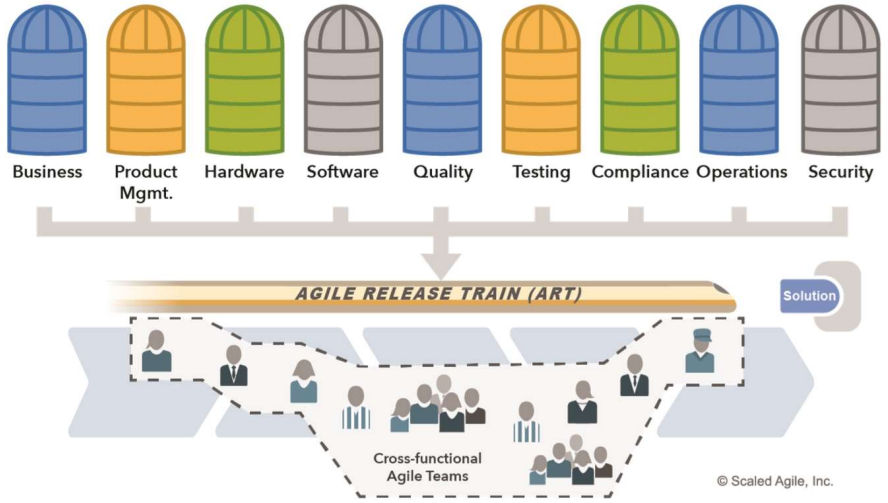
#7 Apply cadence, synchronize with cross-domain planning

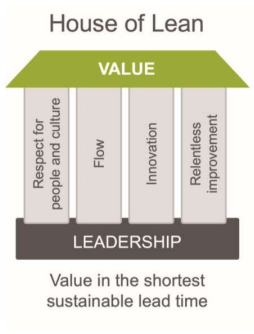
#8 Unlock the intrinsic motivation of knowledge workers

#9 Decentralize decision-making

#10 Organize around value







- SAFe could be deployed in different configurations, from basic a.k.a. *essential,* to *large solution*, *portfolio*, and *full*.

https://www.scaledagileframework.com/safe-5-0-white-paper/

# SAFe



Portfolio Backlog The Portfolio Backlog is the highest-level backlog in SAFe. It provides a holding area for upcoming business and enabler Epics intended to create and evolve a comprehensive set of Solutions.

Epics is a container for a significant Solution development initiative that captures the more substantial investments that occur within a portfolio. Due to their considerable scope and impact, epics require the definition of a Minimum Viable Product (MVP) and approval by Lean Portfolio Management (LPM) before implementation.

Features A Feature is a service that fulfills a stakeholder need. Each feature includes a benefit hypothesis and acceptance criteria, and is sized or split as necessary to be delivered by a single Agile Release Train (ART) in a Program Increment (PI).

Agile Release Train (ART) is a long-lived team of Agile teams, which, along with other stakeholders, incrementally develops, delivers, and where applicable operates, one or more solutions in a value stream.

Architectural Runway The Architectural Runway consists of the existing code, components, and technical infrastructure needed to implement near-term features without excessive redesign and delay.

Enabler supports the activities needed to extend the Architectural Runway to provide future business functionality. These include exploration, architecture, infrastructure, and compliance. Enablers are captured in the various backlogs and occur throughout the Framework.

Key Performance Indicators (KPIs) are the quantifiable measures used to evaluate how a value stream is performing against its forecasted business outcomes.

https://www.scaledagileframework.com/glossary/

# SAFe -- Essential



ALTEN

https://www.scaledagileframework.com/wp-content/uploads/delightful-downloads/ 2019/09/Scaled-Agile-Framework_Essential_A4-1.pdf

# SAFe -- Portfolio



ALTEN

https://www.scaledagileframework.com/wp-content/uploads/delightful-downloads/2019/09/Scaled-Agile-Framework_Portfolio_A4.pdf

# SAFe -- Full



https://www.scaledagileframework.com/wp-content/uploads/delightful-downloads/2019/09/Scaled-Agile-Framework_Portfolio_A4.pdf

# SAFe



 Explore Training at: https://www.scaledagile.com/training/calendar/