Validation

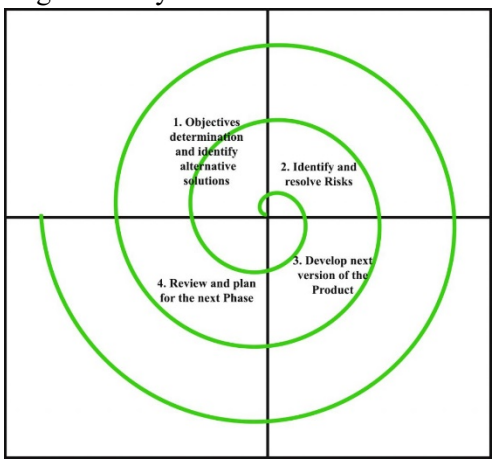
- I build the right product
- I did not build what the client was expecting

Verification

- I build the product right
- I build what the client was expecting but I built in the wrong manner (there's a bug)

Maintenance takes large portion of cost in the software development, often 50% of the total cost

Development workflows

- Waterfall Model
    - No backward
    - Boolean decision "go or not go"
    - Linear progression
    - Varies alternative views for the model
    - Feasibility design → requirement analysis → Design → Implementation → testing → Deployment → Maintenance
- V – model
    - Process steps move upwards after the code phase
    - Shows the relationship between the development and the corresponding verification and validation activity
- Spiral model customer involvement is high
    - Breaking a project into smaller segments
    - A cycle processes
    - Begin each cycle with the identification of the stakeholder's goals
    - 

risk reduction

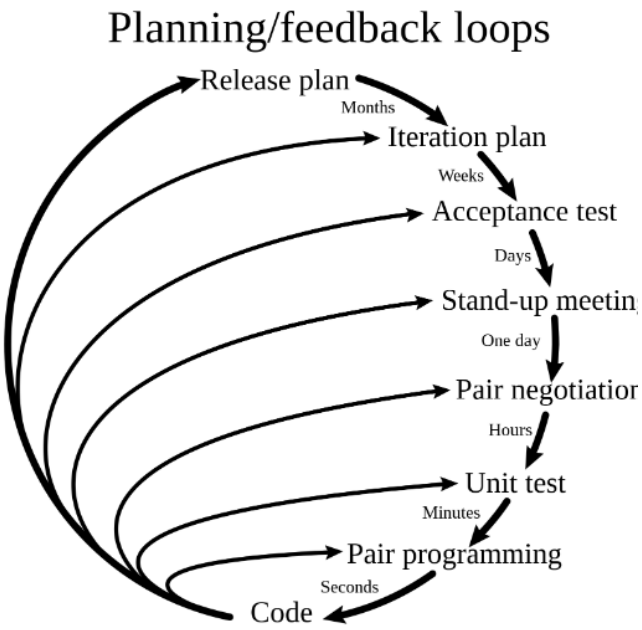
Prototyping

- Create incomplete version of the software program being developed for review

Agile development test driven development

- An iterative approach to project management and software development that helps teams deliver value to customers faster and with few problems
- Agile team delivers work in small, but consumable, increments.

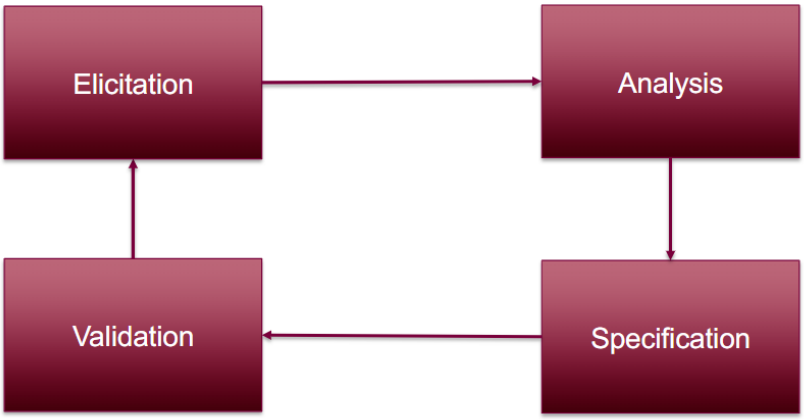
- Requirements, plans and results are evaluated continuously so teams have a natural mechanism for responding to change quickly
- Exists in different versions
    - SCRUM
        - Design for small teams
        - Product backlog
        - Sprint Planning
        - Sprint
        - Sprint review
        - Sprint retrospective → how can we improve the process
    - Extreme Programming (XP)
        - Four basic activities
            - Code, testing, listening, designing



Agile	Waterfall
Small projects	Large projects
Every Day software	Mission Critical
Fast Requirements Change	Stable Requirements
Small Teams	Large Teams
Skill Experts	Diverse Team
Stable Team (people remain)	Unstable Team (people leave the company)
Senior Developers	Diverse Experties

Requirement Engineering

- concerns a set of **activities** to define the **purpose** of a software system and to identify the **context** in which it will be used
- A need or constraint imposed by a stakeholder
- A capability or property that a system shall have
- Challenges
    - Broad scope
    - Multiple concerns
    - Multiple stakeholders with different background → cause conflicts
- Life cycle



    -
- Purpose of SRS (**Software requirements specification**)
    - Communicates an understating of the requirements
    - Contractual → may be legally binding
    - Baseline for project planning and estimation
    - Baseline for software evaluation
    - Baseline for change control
- Users
    - Costumers and users
    - Systems analysts, requirement analysts
    - Developers, programmers
    - Testers
    - Project managers
- SRS should **address**
    - Functionality
    - External interfaces
    - Performance
    - Attributes
- SRS qualities
    - Completeness, pertinence, consistency, unambiguity, verifiability, feasibility, comprehensibility, traceability, good structing, modifiability
- SRS errors
    - Incompleteness, inadequate requirements, contradictions, ambiguities, unfeasible goals, noise, overspecification, lack of clarity, poor structing

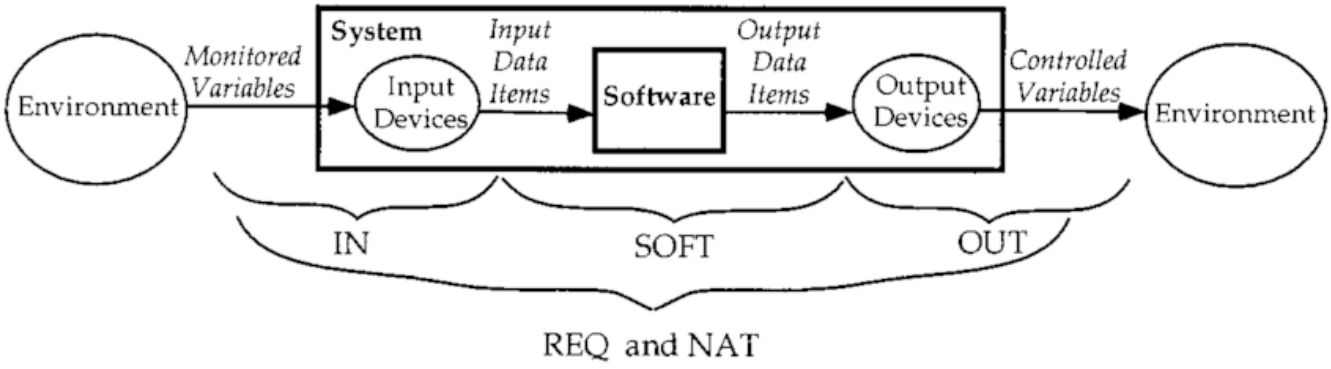
- Goals use “Should” → The user should []
- Requirement use “Shall” → The system shall []
- Statements of fact use “Will” → The system will []

Requirement specification

- Specification is a precise description of behaviors
- Syntax
    - Rules that define the combinations of symbols that are considered to be correctly structured
- Semantics
    - Assigns computational meaning to valid strings in a programming language syntax
- Flowchart
- Pseudocode specification
- Tabular expression
    - Includes inputs, outputs, and conditions of different parameters
    - In short, **conditions and results**
- Requirement table
    - Index
    - Summary
    - Duration
    - Post condition
    - Action

4 variable model

- Describes the essential relationship between requirements and design
- Illustrates why the software design has different inputs and outputs from the requirements



- Monitored variables
- Input variables
- Output variables
- Controlled variables
- IN represents the function performed by the input hardware
- Completeness → Simulink completeness **FALSE**
- Simulink completeness → Completeness **TRUE**

The way to check if the requirement is completed

- Input condition 1 OR Input condition 2 OR Input condition 3 OR Input condition n == TURE

Input	Output
x>0	
x<=0	

x > 0 == A

x <= 0 == not(A)

- **Inconsistency issues**: occur when the block can perform different behaviors for a single combination of input values
- **Academic unambiguity**: Condition i AND Condition j == FALSE

Input	Output
x>0	
-2<=x<=2	
x<-2	

For x = 1

A = True, B = True, C = False

A Λ B == Ture

A Λ C == False

B Λ C == False

Inconsistency (depends on outputs) and ambiguity

Input	output	When x is 45
x < 10	0	No Inconsistency
[10,50]	1	Ambiguous
x > 40	1	

Importance of timing

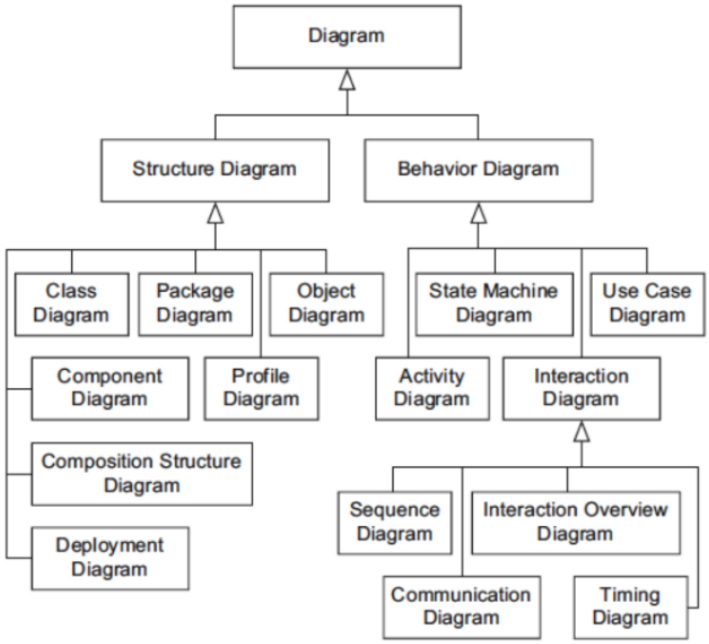
- Meeting timing requirement is just as important as getting the logic of functionality correct in **Hard Real-Time system**
- In **Soft Real-time system**, timing requirement is important, but it is usually treated as a probabilistic requirement
- Response Allowance (RA)
    - Specifies an allowable processing delay
    - Each controlled variable must have an RA specified for it

Software design

- A process that enables the creation of a specification of a software artifact
- Requirement specifications are formal descriptions that domain experts can read and understand
- Software design document (SDD) provides a specification for the software design
- Often software architecture is produced prior to a software design
- Software architecture
    - Shows gross structure and organization of the system to be defined
- Principles
    - Divide and conquer
    - Increase cohesion
    - Reduce coupling
    - Increase abstraction
    - Increase reusability

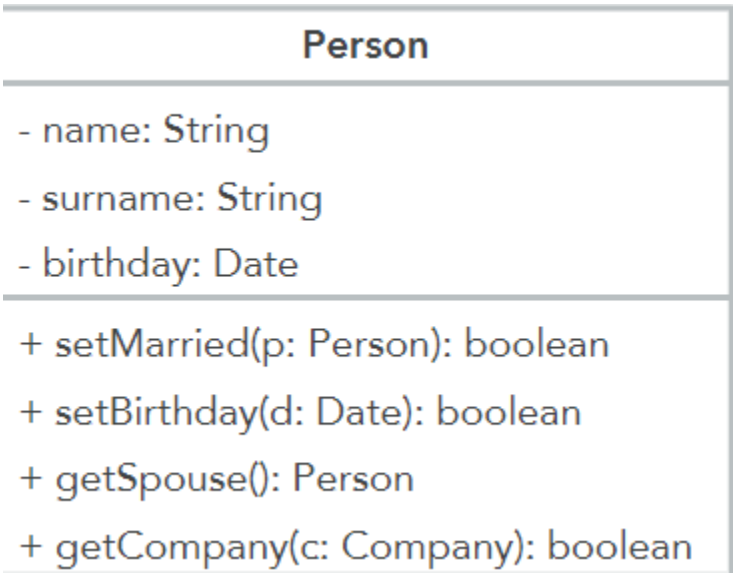
Unified Modeling Language

- Describe the structure, functionalities, and performances of the software



Class diagram

- Made by three parts
- Name, Attributes (represents its state), and methods (describes its behavior)



+ means public

- means private

means protected

Attribute

name: type

Method

name (parameters): return type

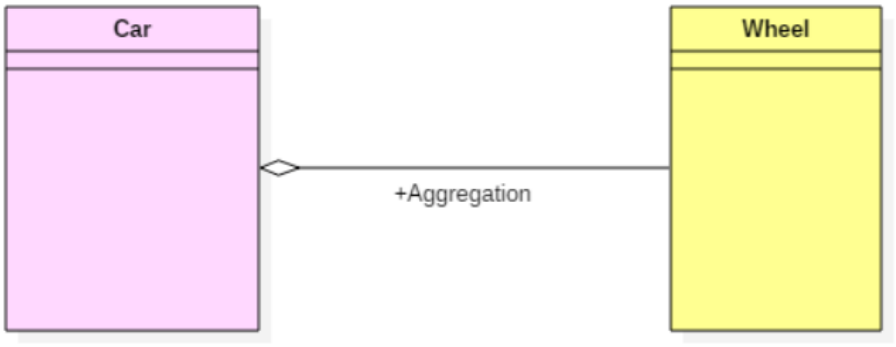
Class diagrams relationship

https://www.guru99.com/association-aggregation-composition-difference.html#uml-association

- Always have arrows
  - On the left and right side of arrow, some numbers may label.
  - 0…* means 0 to many
  - 0…1 means 0 to 1
- Association
  - Basic connection
  - If two classes in a model need to communicate with each other, there must be a link between them
  - The symbol looks like 

A straight line
  - **Direct association**
    - The flow is directed
    - The association from one class to another class flows in a single direction only
    - The flow from server to client only

- Aggregations
  - Particular type of association
  - Indicate that an object is a part of another object
  - Child can exist independently without parent
  - Delete the class, the child class also exist

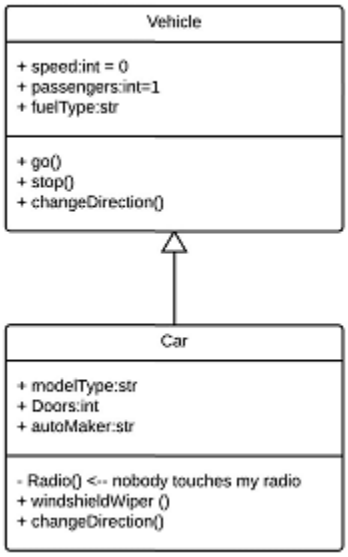


  - A wheel is part of the car, wheel can exist independently without car
  - **Wheel is part of the car**

- Composition
    - The components must exist together
    - The child cannot exist independent of parent
    - Example
        - House (parent) and Room (child)
        - Rooms don't exist separate to a house
        - **Room is part of the house**

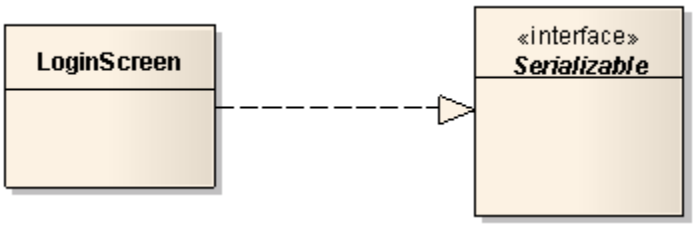
- Reflective association
    - Create associations between a class and itself
- Inheritance (generalization)

Car will inherit all the attributes and methods of the parent class vehicle

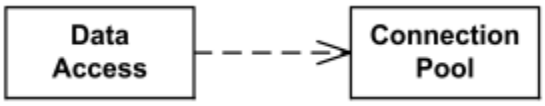


https://www.uml-diagrams.org/class-reference.html

- Interfaces



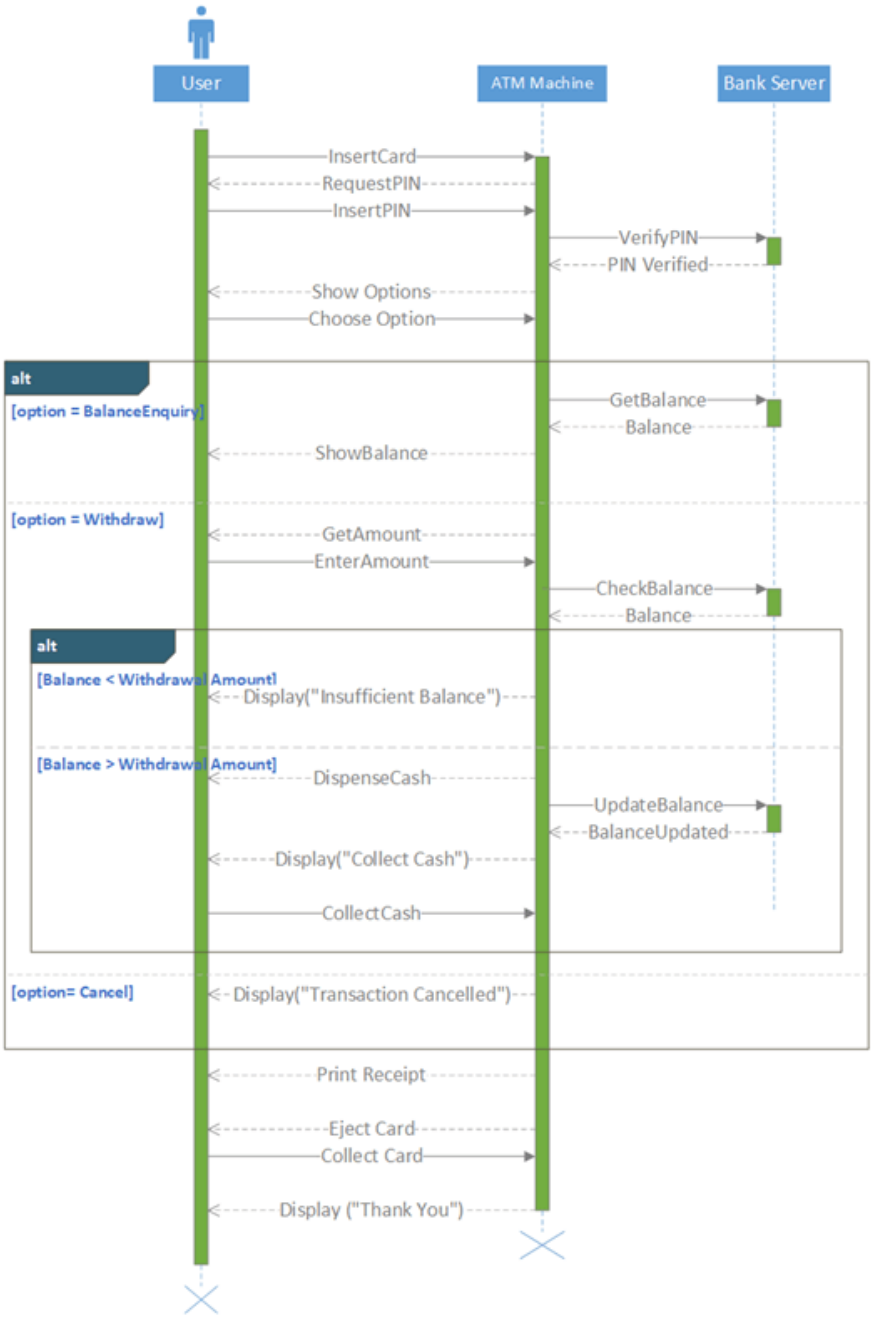
    - LoginScreen class implements the serializable interface
- Dependency



Data access depends on connection tool

Sequence diagrams

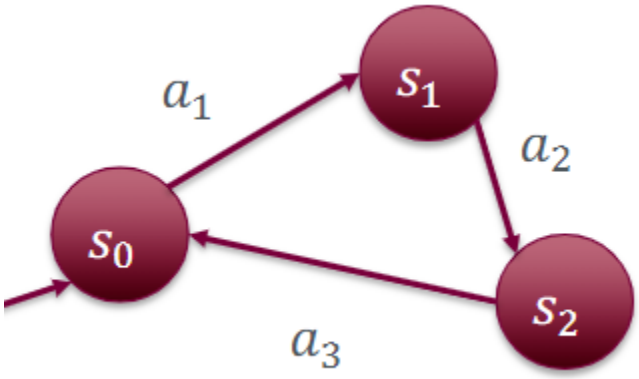
- Describe the behavior of a system and how the component/classes interact to solve a problem
- They model a system by considering
    - Entities
    - Messages exchange between entities
- https://support.microsoft.com/en-us/office/create-a-uml-sequence-diagram-c61c371b-b150-4958-b128-902000133b26#OfficeVersion=Newer_desktop_versions



- Synchronous message
    - Message without response to last entity or first entity
- Asynchronous message
    - The message has response to previous entity

Finite State Machines (FSM)

- Specify control flow aspects



Non-deterministic

- Doesn’t need input to decide next step

Git

- Git update
  - Update all the branches set to track remote ones
- Git checkout
  - Switches branches or restore working tree files
- Git clone
  - Make a clone or copy of repo in a new directory

Key differences between each software development model

- Waterfall
  - Linear
  - Don’t need to interact with customers
  - Do all the testing at the last step
  - Nonflexible
  - Requires early stage planning such as requirements
  - Least maintenance
  - Can not be reused
  - Works for unstable and diverse team
- V model
  - Basically, same as Waterfall model
  - Except the testing will be done for each step has done
    - Testing will start early stage
  - Expensive than waterfall
  - Defining requirements are needed
- Spiral model
  - More complex than other model
  - Flexibility
  - Low amount of risk
    - Resolve risk in every phase
  - Defining requirements are needed
  - Customer involves in this model
  - Evolutionary method
- Agile
  - Clients are highly involved
  - Works better for smaller teams
  - High flexibility
  - Don’t need to define requirement at the beginning of the project
  - Works incremental
  - Test driven development
- Scrum
  - A member of agile
  - Focus on the management and development of the project
  - Design for small teams
  - Break things into sprints (iterations) that can be completed in few weeks
- Extreme programming
  - Iterative steps

- XP team
  - Smaller team without a strict structure
  - Self-managing
- ÍConstant involvement of customer

Behavioral or dynamic view

- Emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects
- Sequence diagrams
- State machine diagrams

Structural view

- Emphasizes the static structure of the system using objects, attributes, operation, and relationships
- Class diagram

SOFTWARE DESIGN PATTERN

STRATEGY PATTERN

- Behavioral pattern
- Choose the way to behave with different algorithms or methods