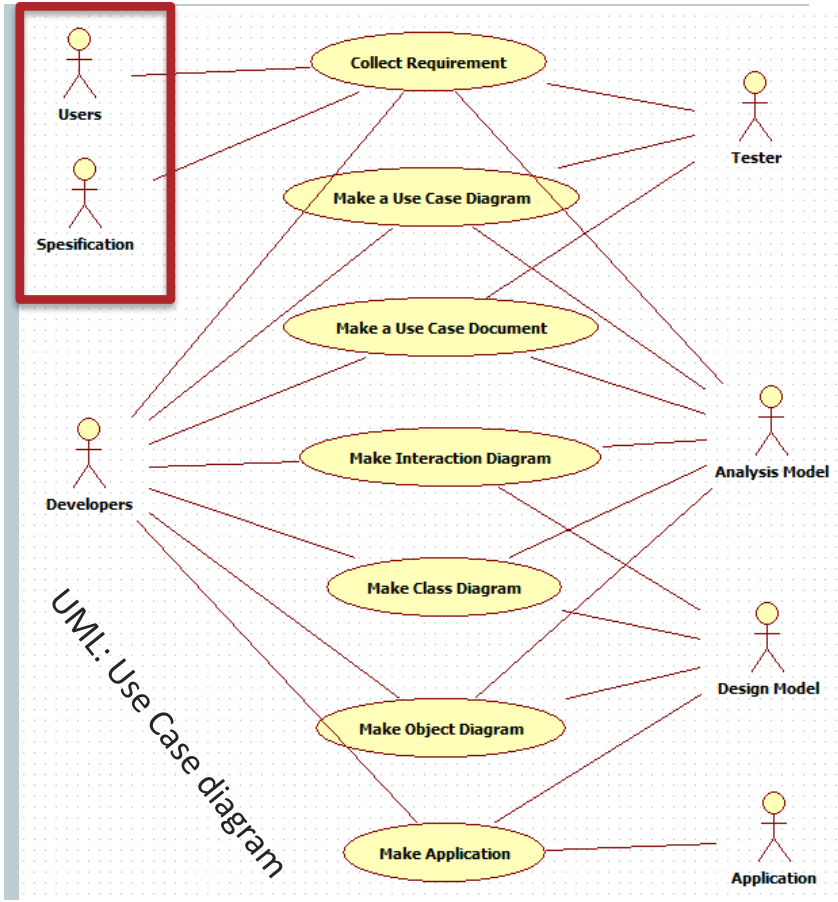
University of South-Eastern Norway

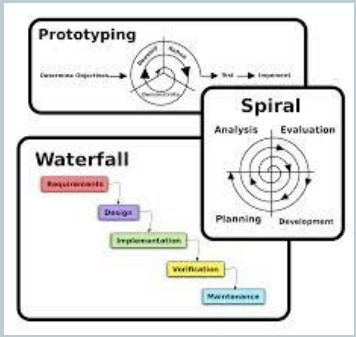
An introduction to Unified Modeling Language (UML) in Software Engineering

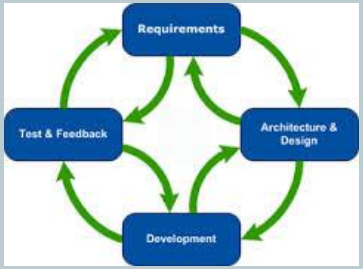
Nils-Olav Skeie

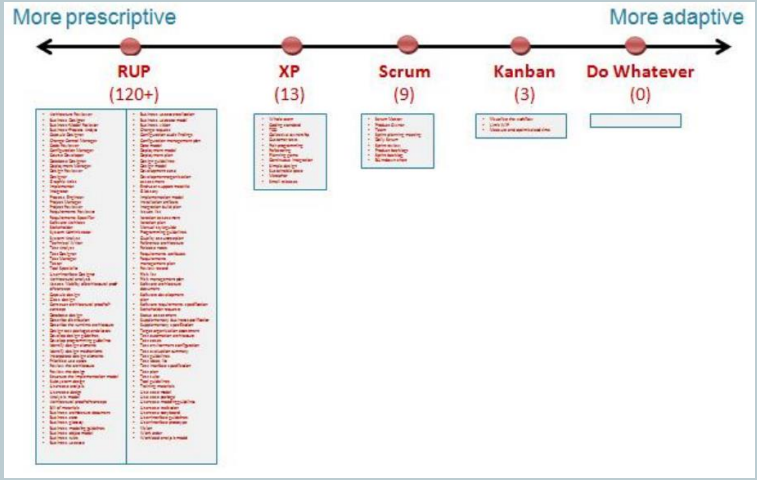


Overview

- Introduction,
  - Software development process
- Analysis,
  - Use Case,
- Design,
  - Interaction diagrams,
  - Class diagram,
- Testing,
- Conclusion.

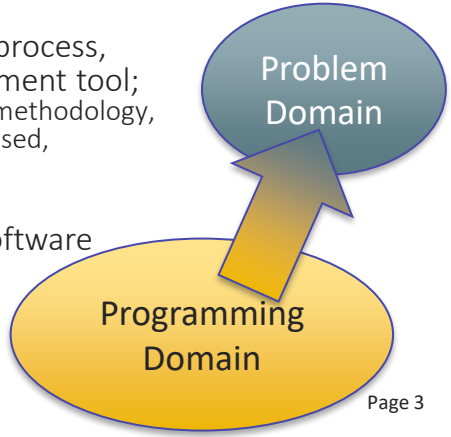


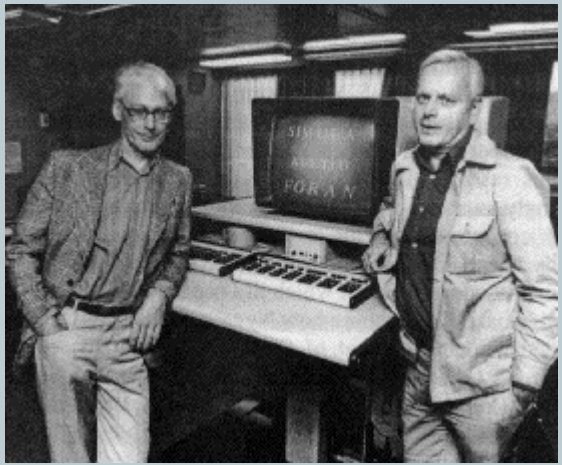




Software Development Process

- Iterations; Analysis, Design, Coding, Testing.
- Coding;
  - only a part of software development,
  - the easiest part of being a software engineer,
  - Every line of code is a potential point of failure,
- Developing software using UML;
  - UML is only a tool!
  - Used in a development process,
  - Select the UML development tool;
    - Tool depend on the methodology,
    - Software or paper based,
- Use case / User story
  - Main functions of the software
- Practice;
  - UML and tool,
  - ”small steps”





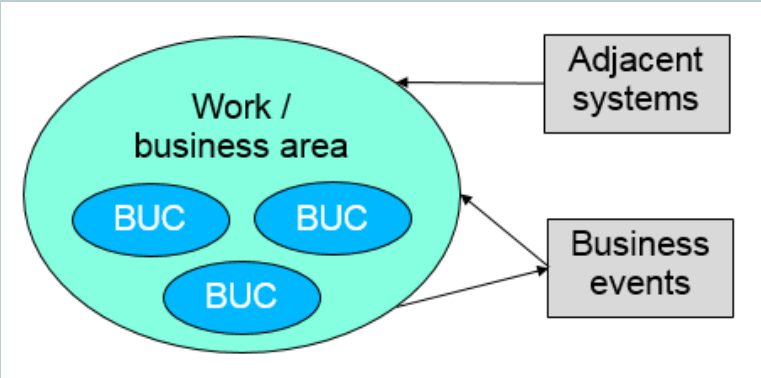
Simula67: Ole-Johan Dahl and Kristen Nygaard

Person
-name : String -birthDate : Date
+getName() : String +setName(name) : void +isBirthday() : boolean

Book
-title : String -authors : String[ ]
+getTitle() : String +getAuthors() : String[ ] +addAuthor(name)

Object-oriented development

- Class;
  - An abstract definition of some sort of function in the problem domain,
  - Consists of a name, data and methods.
- Object;
  - An instance of a class in computer memory, with valid data,
- OOAD;
  - Using Object-Oriented methods for analyzing and designing applications,
  - Assigning responsibilities to each object,
  - Let objects cooperate to solve a specific task or a set of tasks.

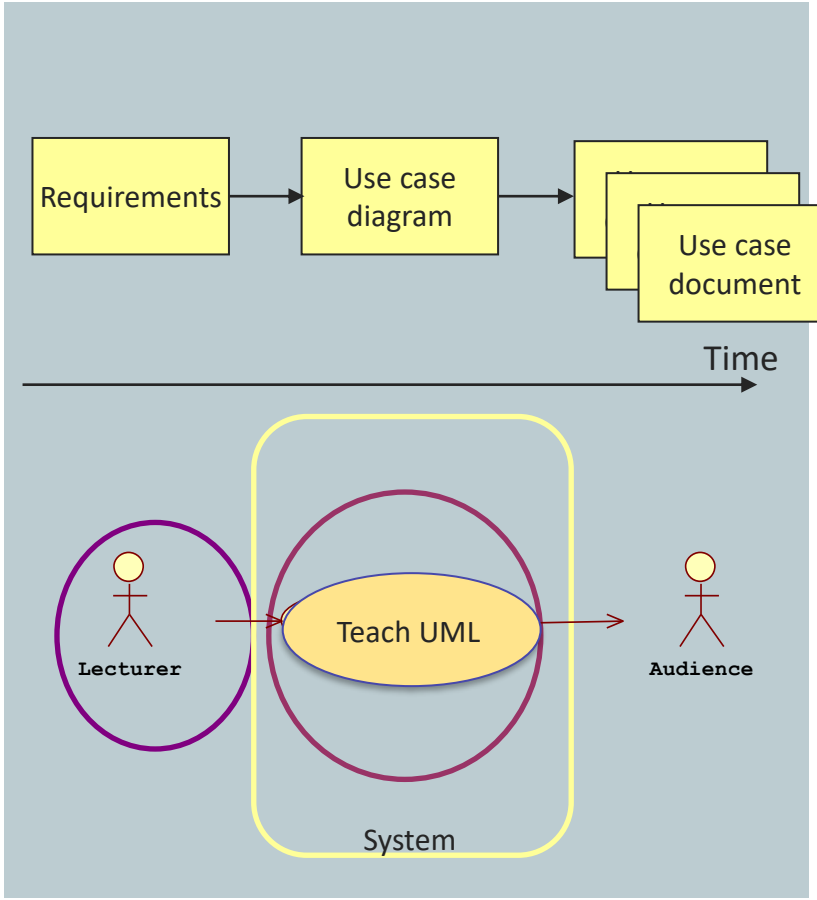


Business Area / Specification

- Work / Business area
  - Business activities of the product owner,
  - Activities that the owner wants to improve,
  - Business use cases (BUC),
- Specification - USER
  - An oral and/or written description of a challenge,
- Requirements - DEVELOPER
  - Make a document describing the requirements for the system,
  - Testable.

Exercise: Business Area / BUC

- Software for a Washing Machine
  - Specification?
  - Business Use Cases?
  - Requirements?

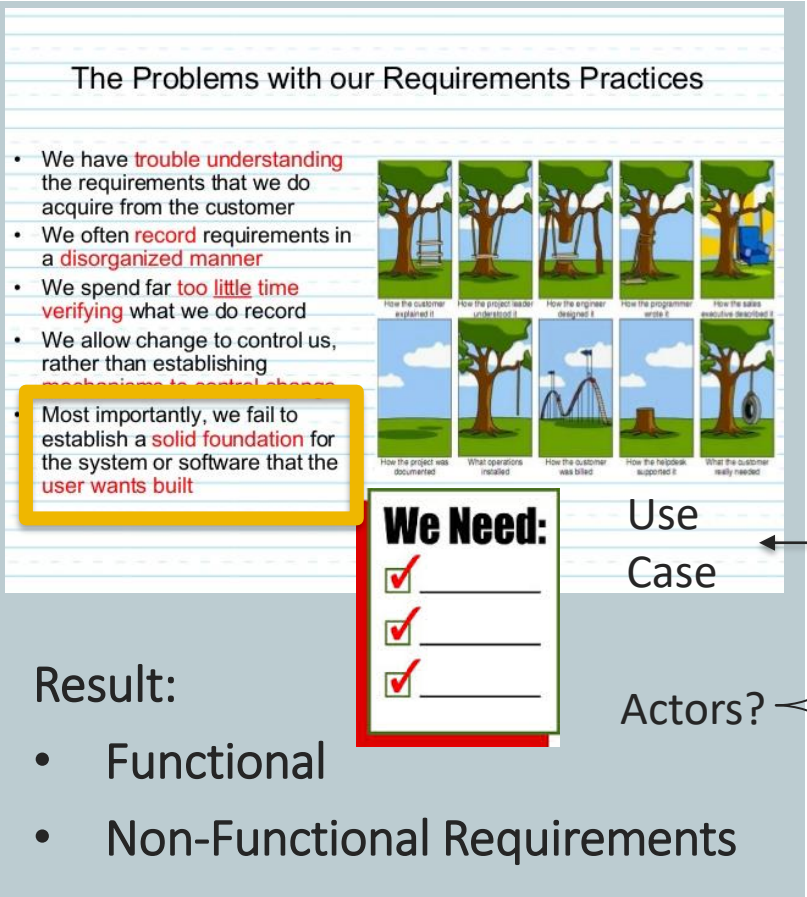


Analysis

- Collect requirements,
- Make use case diagram,
- Make use case document,

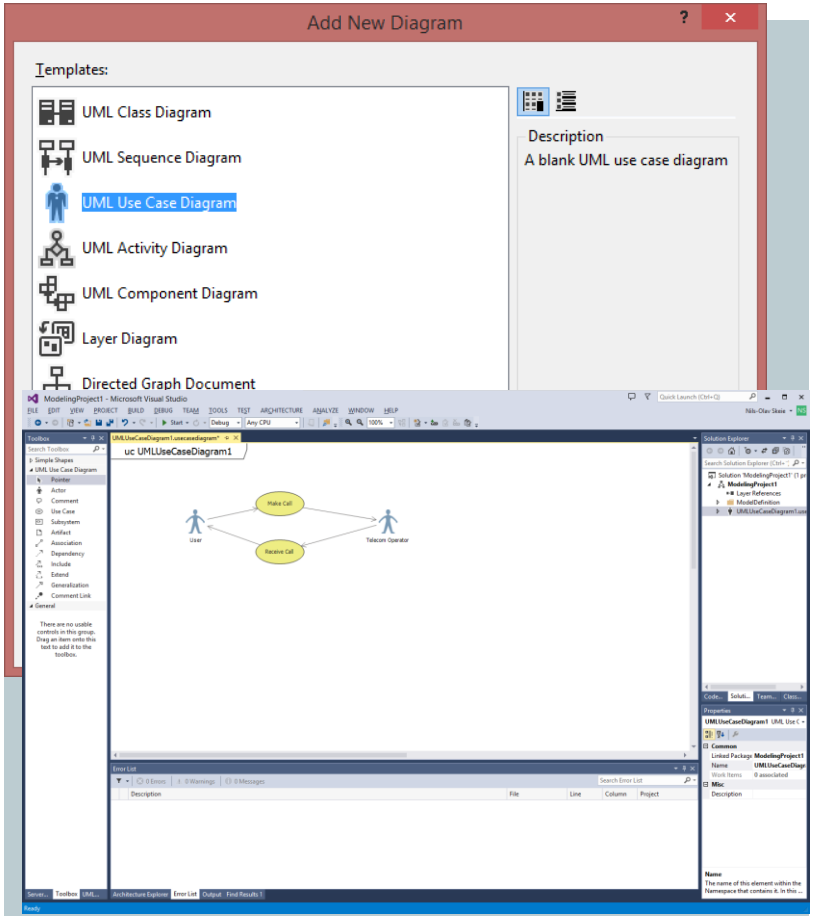
- Use case diagram,
  - Use case; main functions of the software,
  - Actors; I/O of the software,

- Use case document,
  - Text description of each use case.



Requirements

- Use the FURPS+ letters, **make a text document**;
  - F is Functionality;
    - Main functions of the software, the use cases, starts with a verb,
  - U is Usability;
    - How to interact with the software, human factors, help, documentation,
  - R is Reliability;
    - Predictability, Accuracy, Mean time to failure,
  - P is Performance;
    - Speed, Resource consumption, Throughput, Response time,
  - S is Supportability;
    - Testability, Adaptability, Maintainability, Configurability,
  - + (extra)
    - Implementation, licenses, administration, interface to external systems,



Use case diagram

- How software will fulfill the requirements of the external actors,
- Consists of a set of actors and use cases,
- Actor
  - ”something” requires a function or service of the software,
  - Often a person, hardware device, software function (OS) or another computer system,
- Use case
  - Main functions of the applications,
  - The functions required to/from the actors,
  - Use a verb in the use case name,
  - Functional section from the requirements,
  - More details for each use case in a use case document.

Exercise: Use case diagram

- Requirements?
- Main functions of software?
- Any actors?
- Make a use case diagram.

Use case document

Fully dressed use case document

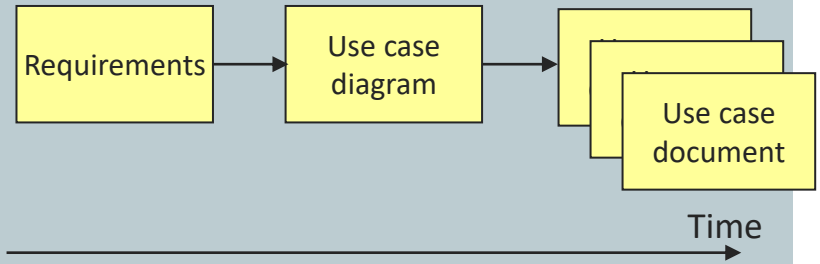
	Use case section	Comment
1	Use case name	Start with a verb
2	Scope	The system under design
3	Level	"user goal" or "sub function"
4	Primary actor	The main user of the function
5	Stakeholders and Interests	Who cares, and what do they want
6	Preconditions	What must be fulfill before starting
7	Success Guarantee	What must be fulfilled at a successful completion
8	**Main success scenario**	A typical set of events
9	**Extensions**	Alternative scenarios of success or failure
10	Special requirements	Related non-functional requirements
11	Technology list	Different I/O methods and data formats
12	Frequency of occurrence	Investigation, testing and timing of implementation
13	Miscellaneous	Such as open issues

Use case document (iterations)

- Select the most important use case,
- Use case documents (FDUCD):
  - Describe in a text document,
  - Brief, Casual, or **Fully** dressed.
  - Different templates available,
    - Preconditions,
    - Success Guarantee,
    - Main Success Scenario (basic flow),
      - Actor events.
    - Extensions (conditional/branches),
      - Actor events.
  - Important sections; (6, 7) 8, 9 and 12.

Exercise: Use case document

- Select the most important use case,
- Describe the use case in more detail!
- Main success scenario / Extensions

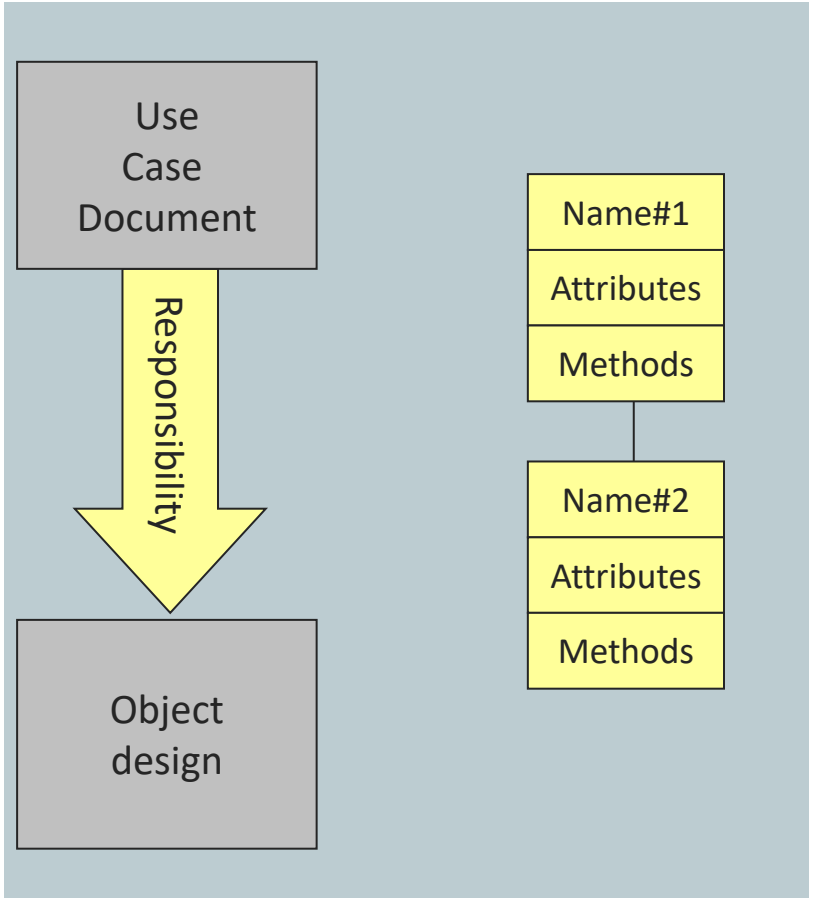


Documentations – part of:

- (SRS: Software Requirements & Specifications)
- **SRD: Software Requirements and Design**

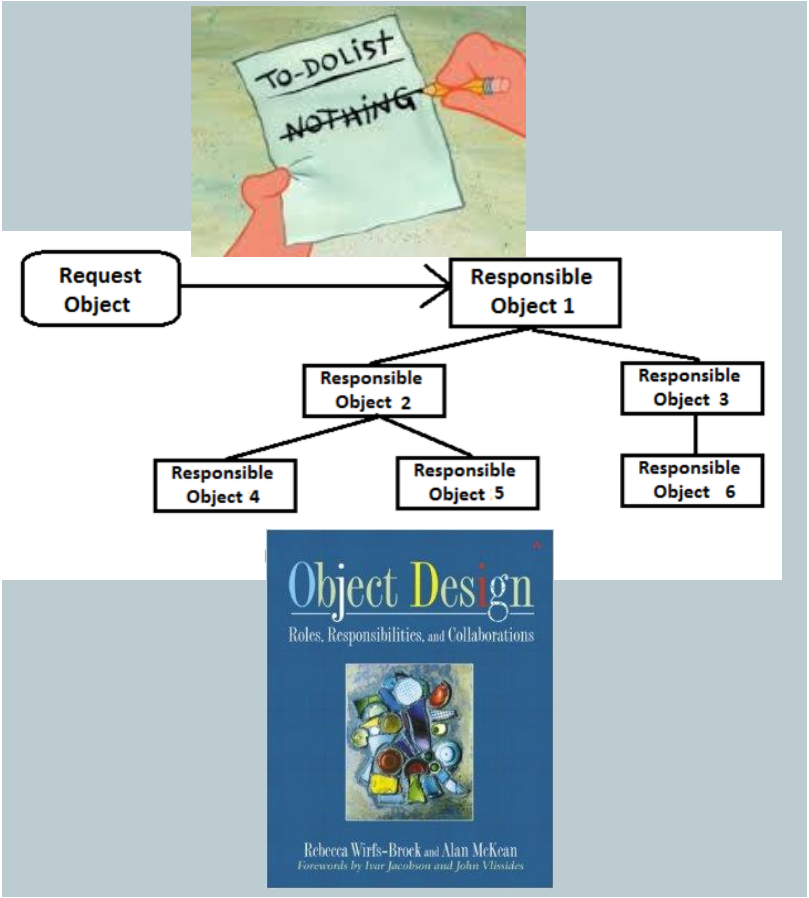
Analysis: Documents

- Specification and Requirements,
  - FURPS+,
  - Text,
- Use case diagram,
  - UML,
  - Main functions and actors,
- Use case document,
  - Text,
  - Description of each use case,
  - One document for each use case.



Analysis to Design

- Design:
  - How the software is going to do the work,
  - Structure,
    - Classes and objects,
  - Give responsibility to objects,
- Use design patterns.



Giving responsibility to objects

Two types of responsibilities (knowing/Doing):

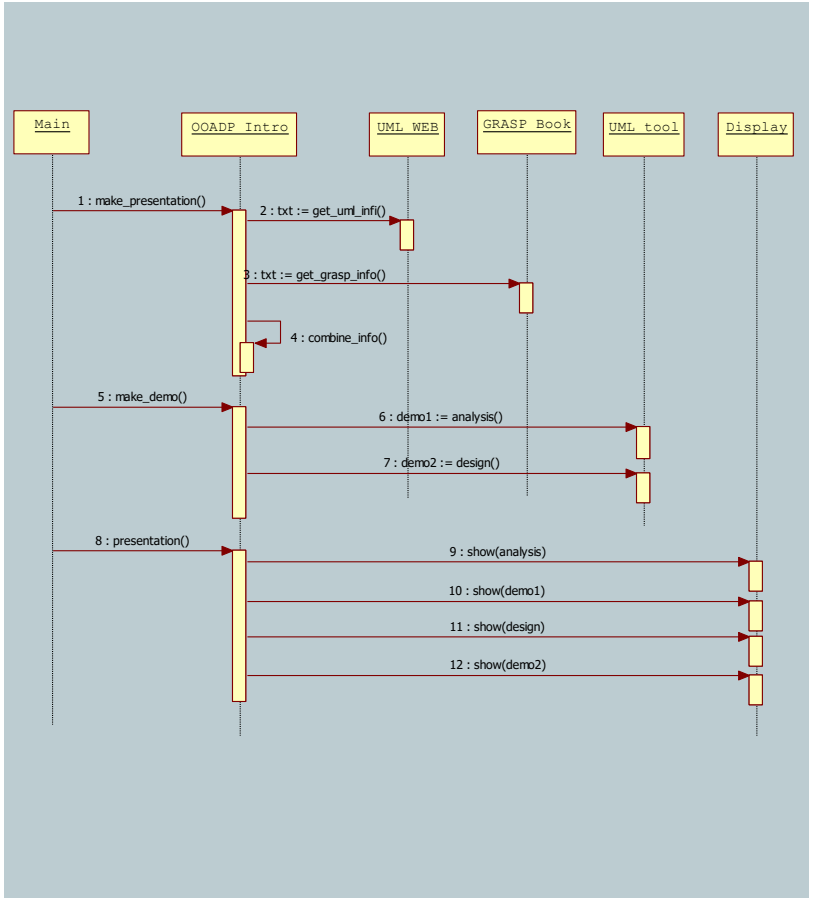
- knowing
  1. private encapsulated data,
  2. related objects,
  3. “things” it can derive or calculate
- doing
  1. doing something itself (creating an object or doing a calculation),
  2. initiating action in other objects,
  3. controlling and coordinating activities in other objects.

Design tasks:

1. Select the most important use case,
2. Select a controller class for this use case,
3. Make interaction diagrams for the use case,
4. Make a class and object diagram.

Design Model

- Describe the scenario of a use case with collaborating objects,
- The system operation starts with a controller object,
- Use UML diagrams
  - interaction diagrams,
  - class diagrams,



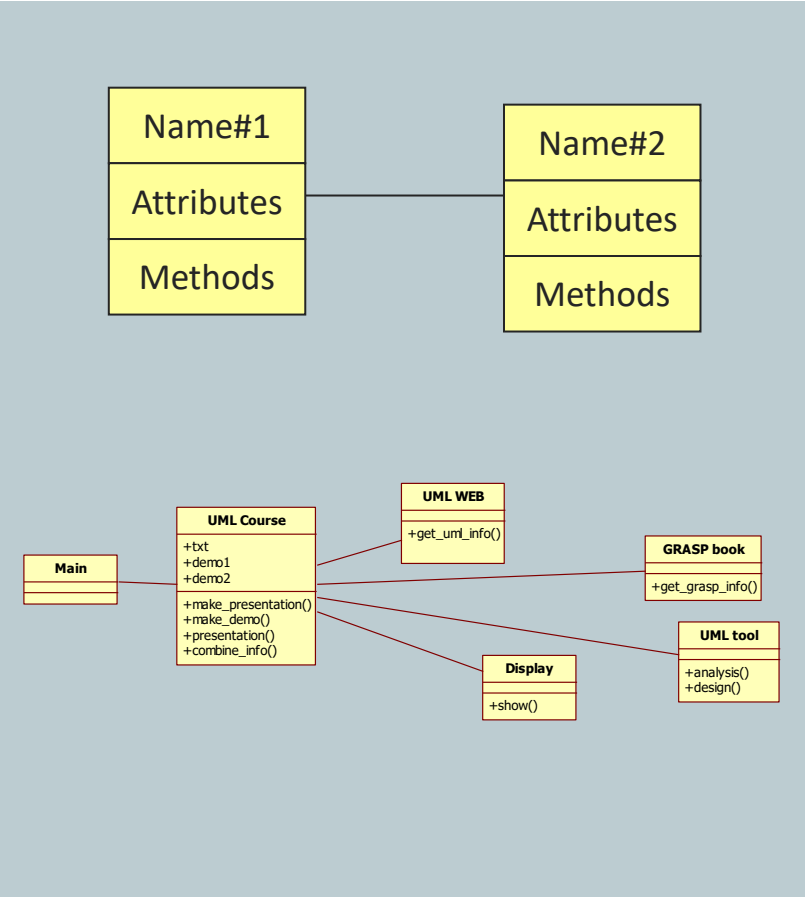
Interaction diagrams - Sequence diagram

- Shows the sequence of the messages in time,
- Simple notation,
- Takes a lot of horizontal space,
- Notation (”UML” Coding):
  - No answer: message()
  - With answer: ans:=message()
  - Condition: [x<10]:ans:=message()
  - Loop: *[i=1..n]: ans:=message()

- Use several objects for cooperation,
- Do not include the details, focus on the overview

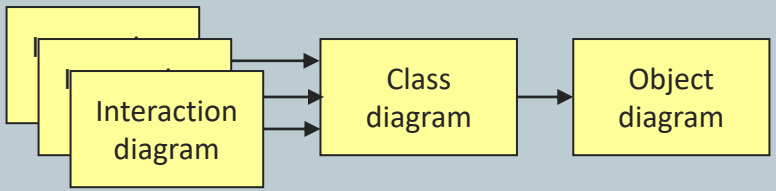
Exercise: Sequence diagram

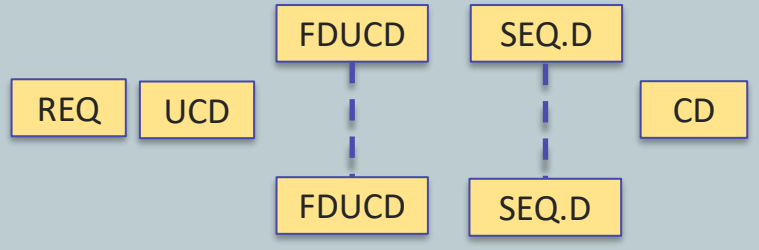
- Washing cloth at 40 deg. C



Class and Object diagrams

- Class and Object diagrams from the interaction diagrams,
  - One interaction diagram for each use case,
  - Common class and object diagrams.
- Get all the information from the interaction diagrams,
  - Class diagram and object diagram with names, attributes, and methods





Documentations – part of:

- (SDD: Software Design Document)
- **SRD: Software Requirements and Design**

Design: Documents

- Interaction diagram,
  - Sequence diagram, focus on time,
  - One diagram for each use case document,
  - Dynamic model of your software,
- Class diagram,
  - Static model of your software,
  - One common diagram,
- Object diagram,
  - Static model of your application,
  - One common diagram.

Testing

- More than 50% of errors may arise before coding,
- Testing in each iteration,
- Deploy an early version for the customer,
- For testing:
  - Requirements,
  - Use Case Diagram,
  - Use Case Documents (Fully Dressed Use Case Document),
- Most of the test documents produced **before** coding.

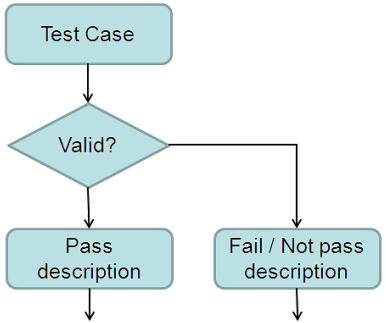
Sample Template for a Unit Test Plan

Sr.	Requirements	Typical Components	Detailed Description
1)	Introduction	a) Test Strategy and Approach	
		b) Test Scope	
		c) Test Assumptions	
2)	Walkthrough (Static Testing)	a) Defects Discovered and Corrected	
		b) Improvement Ideas	
		c) Structured Programming Compliance	
		d) Language Standards	
		e) Development Documentation Standards	
3)	Test Cases (Dynamic Testing)	a) Input Test Data	
		b) Initial Conditions	
		c) Expected Results	
		d) Test Log Status	
4)	Environment Requirements	a) Test Strategy and Approach	
		b) Platform	
		c) Libraries	
		d) Tools	
		e) Test Procedures	
		f) Status Reporting	

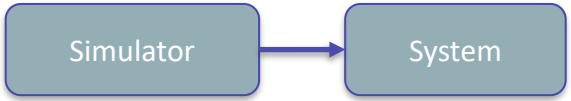
http://www.softwaretestinggenius.com

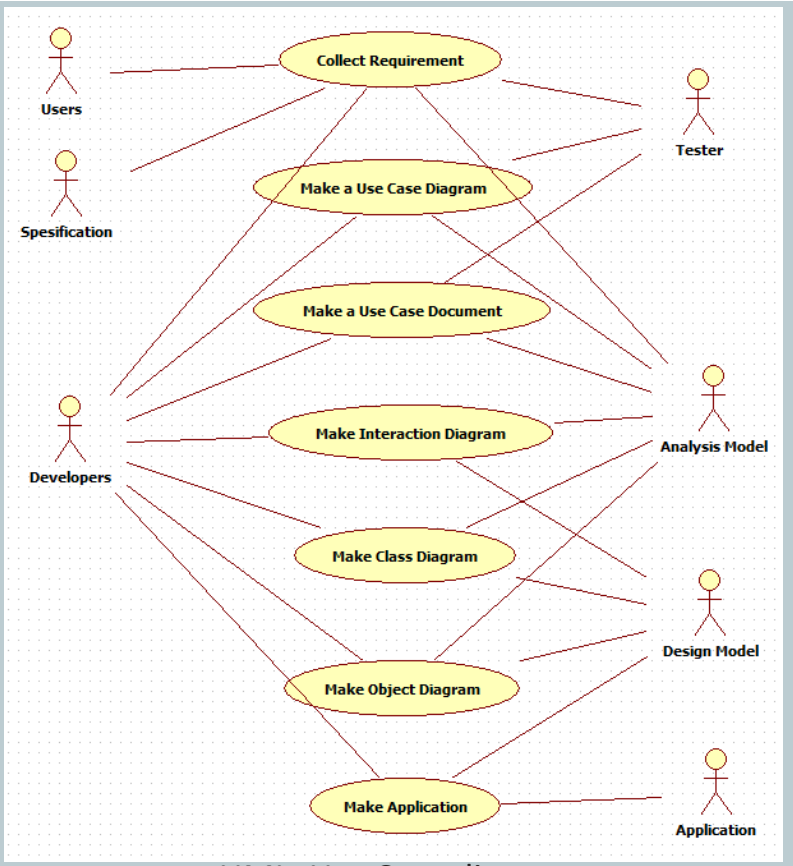
Testing – Test Plan

- Based on requirements,
- Test plan document;
  - System description,
  - Test cases,
  - Responsibility.



- Integrated test options;
  - Simulator,
  - Methods.





UML: Use Case diagram

Conclusions

- Use a development process
- Use time in the analysis phase,
  - Collect requirements,
  - Make use case diagram and use case documents,
- Design phase,
  - Assign responsibility to objects,
  - Sequence diagram; dynamic information,
  - Class diagram; static abstract information,
  - Object diagram; static runtime information,
  - Testing is based on requirements.