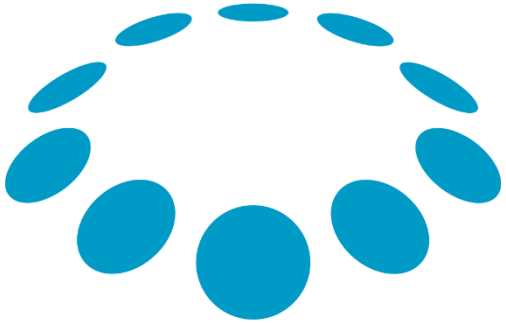
Advanced Software Development

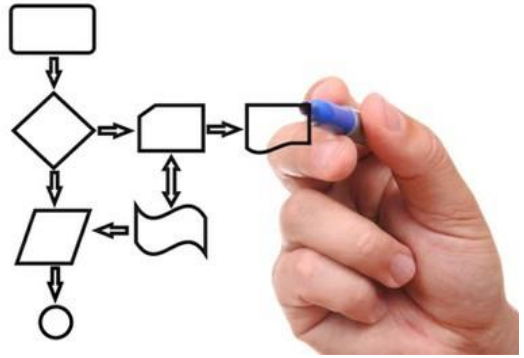
Dr. Eliahu Khalastchi

2017



המסלול האקדמי
המכללה למינהל

Course Goals



You will be able to apply

- Object Oriented Analysis
  - Use cases
  - Interaction diagrams
  - UML class diagrams
- Object Oriented Design
  - Design Patterns
  - Architectural Patterns
- Object Oriented Programming
  - Java
  - Additional C#, C++ examples
- In addition…
  - Data structures
  - Streaming data (Files, Sockets)
  - Concurrency patterns
  - Client-Server

Course Syllabus

Introduction

To software engineering

History

- In the 1940s the instructions were wired to the computers – not flexible
- In the 1950s programming languages started to appear
  - Fortran, Algol, Cobol
- The term "software engineering“
  - coined first by Anthony Oettinger and then used by Margaret Hamilton
  - the title for the world's first conference on software engineering (1968)

- Up until then, it was a secondary branch of Computer Science

- The “Software Crises”

History

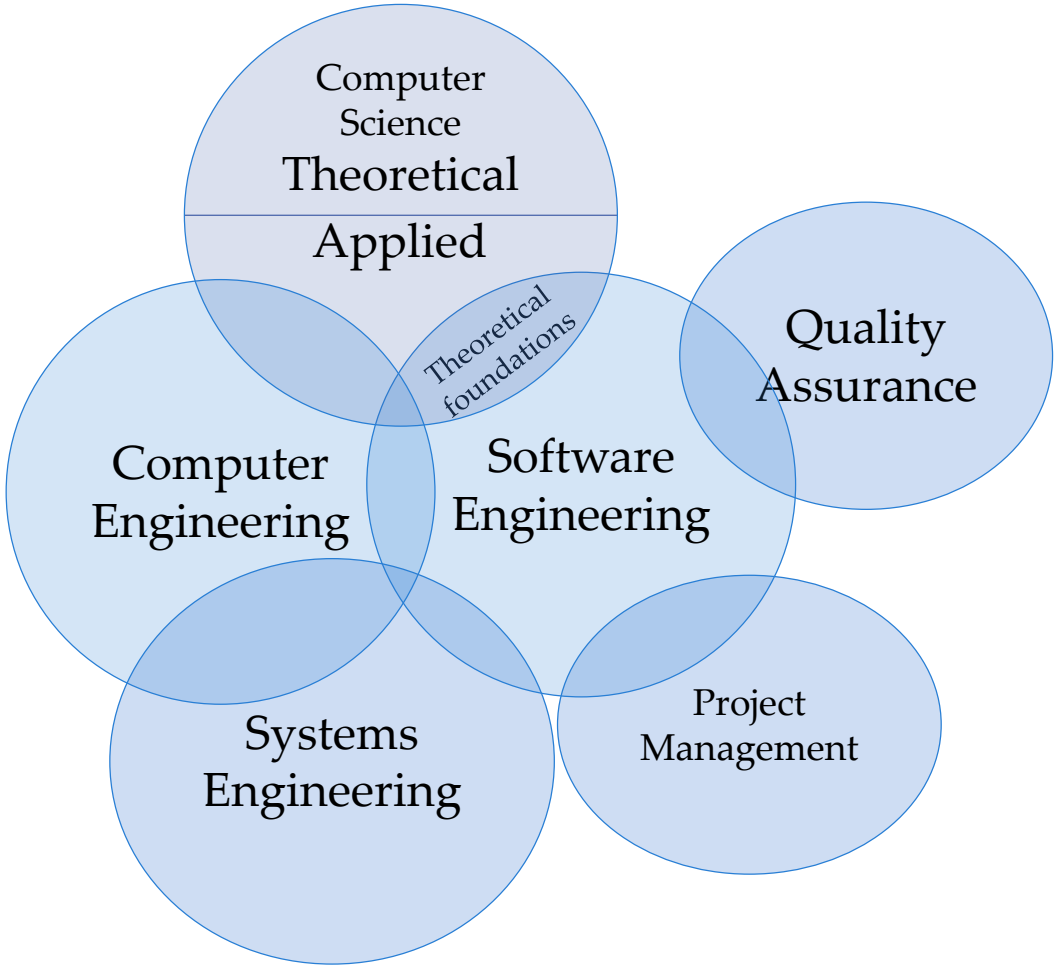
- The discipline of software engineering was created to
  - address poor quality of software
  - get projects exceeding time and budget under control
  - ensure that software is built
    - Systematically
    - Rigorously
    - Measurably
    - On time
    - On budget
    - Within specification

- Engineering already addresses all these issues
- Hence the same principles used in engineering can be applied to software

History

- 1996 – the first BSc academic program (US, Israel)
- 1999 – "Software Engineering Body of Knowledge" (SWEBOK) is completed
  - A document, an **international standard**
  - Specifying a guide to the generally accepted Software Engineering Body of Knowledge
  - Latest version in 2004
- Most of the professionals in the field have background in Computer Science
  - But are accepted to work as software engineers
- Software engineering is now an independent field

Related Fields



Software Engineering

As opposed to other engineering disciplines, software engineering

- **Does not ensure** consistent, reliable, or even useful products
- The methods are
  - Not uniformed
  - Not regulated
  - Mostly based on rules of thumb
  - Lack sold mathematical foundations

- Is it Engineering? Science? Art?

Software Engineering vs. Computer Science

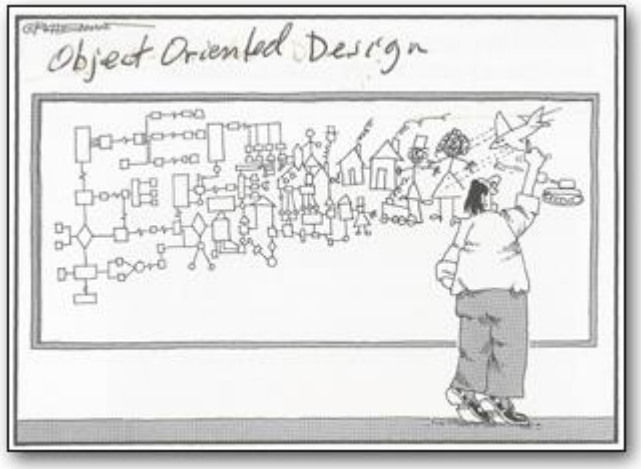
Topic	Software Engineering	Computer Science
Ideal	Software system development	Discovering truths about computability, algorithms, and other theoretical topics
Emphasis	Developing software that has a value to a user	General truths about the complexity, correctness, soundness and completeness of algorithms
Goal	Creating useful software	Creating algorithms
Additional knowledge	Engineering, project management, technology	Math

Software Engineering vs. Traditional Engineering

Topic	Software Engineering	Traditional Engineering
Fundamentals	Computer science, Discrete mathematics	Physics, chemistry, Calculus
Methods	Rules of thumb, not based on a solid mathematical foundation	Uniformed, regulated, based on proven mathematical models
Development cost	Almost ½ of the total cost. Tools are relatively cheap.	Relatively cheap.
Production cost	Trivial	Most of the effort goes to production
Product life span	Several years	Decades

Introduction

To **Object Oriented** Software Engineering



Object Oriented Programming

Analysis

Design

Implementation

Understand our problem & required functionality — **Analyzer**

Plan a solution that meets these requirements — **Architect**

Implement the design with an Object Oriented Programming Language — **Engineer**

Analysis

Understand our problem & required functionality

Object Oriented Design

Plan a solution that meets these requirements

↓

Diagrams, sketches, etc.
No code!

Why use object orientation?

- Procedural programs are difficult to manage and plan
  - A long list of statements with data & logic
  - Instead, we divide the code into objects

Objects are

- Abstractions of beings derived from the problem or the solution
  - They represent the way we think about the problem
- Self-contained, self-managed mini-programs
  - Contain their own data & logic
  - Easier to maintain
- Reusable

- Objects communicate among themselves by messages, not calls
  - i.e., methods, not functions with globally shared data

- The functionality is described by services offered by objects

```
Begin["`Private`"];
GausssSolveArraySlice[Real[n_, n_] ain_, Real[n_, m_] bin_, Integer iterations_] → Real[n, m] :=
 Module[{Real[n] dumc, Real[n, n] a, Real[n, m] b, Integer[n] {ipiv, indxr, indxc},
   Integer {i, k, l, irow, icol}, Real {pivinv, amax, tmp}, Integer {beficol, afticol, count}},
  For[count = 1, count ≤ iterations, count = count + 1, {a = ain;
    b = bin;
    For[k = 1, k ≤ n, k = k + 1, ipiv[[k]] = 0];
    For[i = 1, i ≤ n, i = i + 1,
     (*find the matrix element with largest absolute value*)
     amax = 0.0;
     For[k = 1, k ≤ n, k = k + 1,
      If[ipiv[[k]] == 0,
       For[l = 1, l ≤ n, l = l + 1, If[ipiv[[l]] == 0, If[Abs[a[[k, l]]] > amax, amax = Abs[a[[k, l]]];
           irow = k;
           icol = l]
        ]
       ]
     ];
     ipiv[[icol]] = ipiv[[icol]] + 1;
     If[ipiv[[icol]] > 1, "*** Gauss2 input data error ***" >> "";
      Break];
     (*if irow ≠ icol, then interchange rows irow and icol in both a and b*)
     If[irow ≠ icol, For[k = 1, k ≤ n, k = k + 1, tmp = a[[irow, k]];
       a[[irow, k]] = a[[icol, k]];
       a[[icol, k]] = tmp];
      For[k = 1, k ≤ m, k = k + 1, tmp = b[[irow, k]];
       b[[irow, k]] = b[[icol, k]];
       b[[icol, k]] = tmp]];
     indxr[[i]] = irow;
     indxc[[i]] = icol;
     If[a[[icol, icol]] == 0, Print["*** Gauss2 input data error 2 ***"];
      Break];
     (*prepare to divide by the pivot and subsequent row transformations*)
     pivinv = 1.0 / a[[icol, icol]];
     a[[icol, icol]] = 1.0;
     a[[icol, _]] = a[[icol, _]] * pivinv;
     b[[icol, _]] = b[[icol, _]] * pivinv;
     dumc = a[[_, icol]];
     For[k = 1, k ≤ n, k = k + 1, a[[k, icol]] = 0];
     a[[icol, icol]] = pivinv;
     For[k = 1, k ≤ n, k = k + 1, If[k ≠ icol, a[[k, _]] = a[[k, _]] - dumc[[k]] * a[[icol, _]];
       b[[k, _]] = b[[k, _]] - dumc[[k]] * b[[icol, _]]]]
    ];
    For[l = n, l ≥ 1, l = l - 1, For[k = 1, k ≤ n, k = k + 1, tmp = a[[k, indxr[[l]]]];
      a[[k, indxr[[l]]]] = a[[k, indxc[[l]]]];
      a[[k, indxc[[l]]]] = tmp]])
  ];
  b];
End[];
EndPackage[];
```

Why use object orientation?

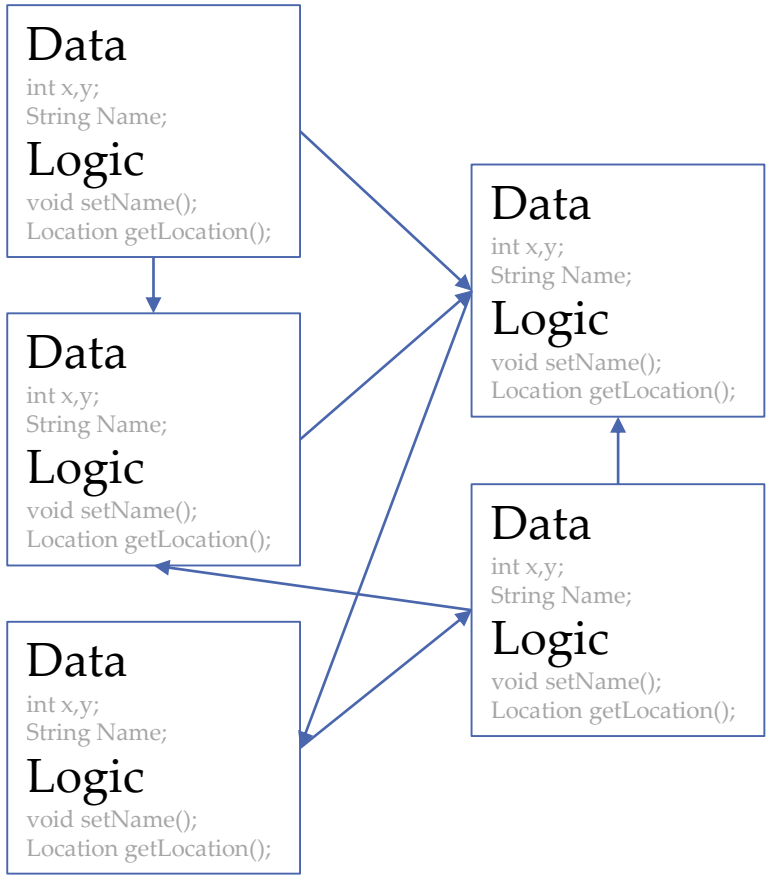
- Procedural programs are difficult to manage and plan
  - A long list of statements with data & logic
  - Instead, we divide the code into objects

Objects are

- Abstractions of beings derived from the problem or the solution
  - They represent the way we think about the problem
- Self-contained, self-managed mini-programs
  - Contain their own data & logic
  - Easier to maintain
- Reusable

- Objects communicate among themselves by messages, not calls
  - i.e., methods, not functions with globally shared data

- The functionality is described by services offered by objects



Objects

- Represent physical or virtual objects of the real world
  - E.g., a car, an email account
- Are separated from one another
  - Have their own existence, identity, that is independent of other objects
- May contain other objects
- Have characteristics, attributes
  - Describe the current state of the object
  - The state of one object is independent from another
    - E.g., closing a door does not close all the doors
- Have behaviors that are specific to the type of object

Identity
Attributes
Behaviors

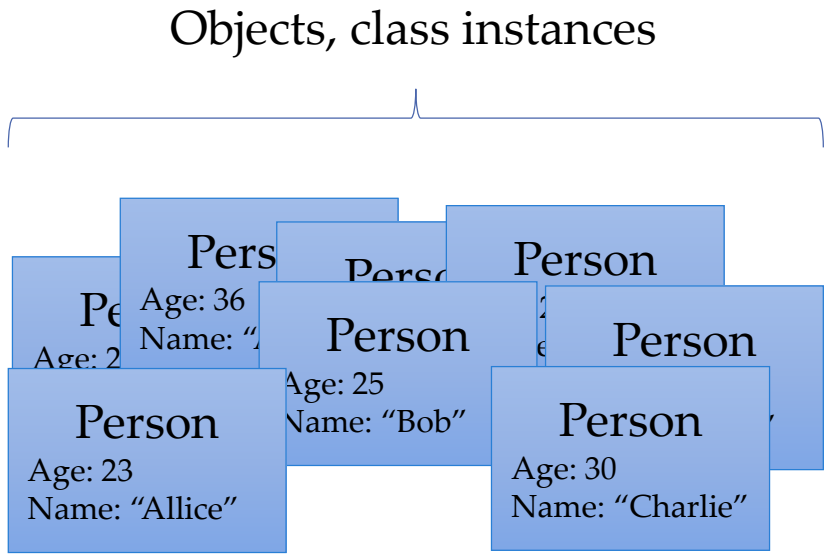
Objects

Person allice;	Person bob;	EmailAccount bobsEmail;	
Person	Person	EmailAccount	Identity
Age: 23 Name: “Allice”	Age: 25 Name: “Bob”	Address: “bob.sponge@gmail.com” Unread: 5,432	Attributes
walk(); talk();	walk(); talk();	send(); receive();	Behaviors

Classes

- The "blueprint" of objects
- Object oriented design is about classes

- A class defines:
  - A Name
    - Also known as a Type
  - Attributes
    - Properties, Fields, Data members
  - Behaviors
    - Operations, methods, member functions



A class

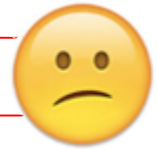
```
public class Person{
  private int age;
  private String Name;
  public Person(int age, String Name){
        this.age=age;
        this.name=name;
  }
}
```

How do we know which objects to create??

Object Oriented Analysis & Design

Software Development Methodologies

Is there one process that will take us step by step, from start to finish, to make our software?

No. There are dozens. 

Incremental development

Waterfall

Prototyping

Iterative Development

Spiral development

Cleanroom

Crystal Clear

Scrum

Agile

Feature-Driven Development

Rapid

SSADM

Rapid application development

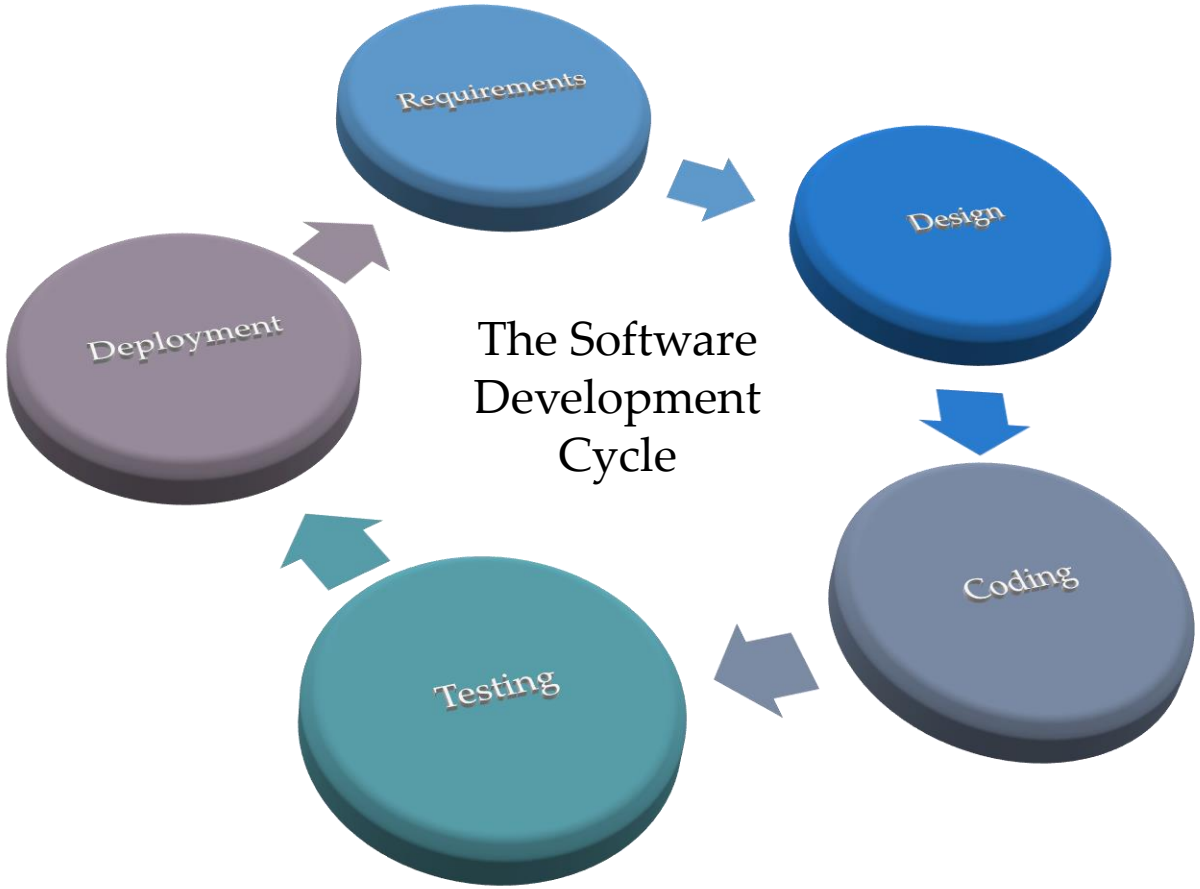
Extreme programming

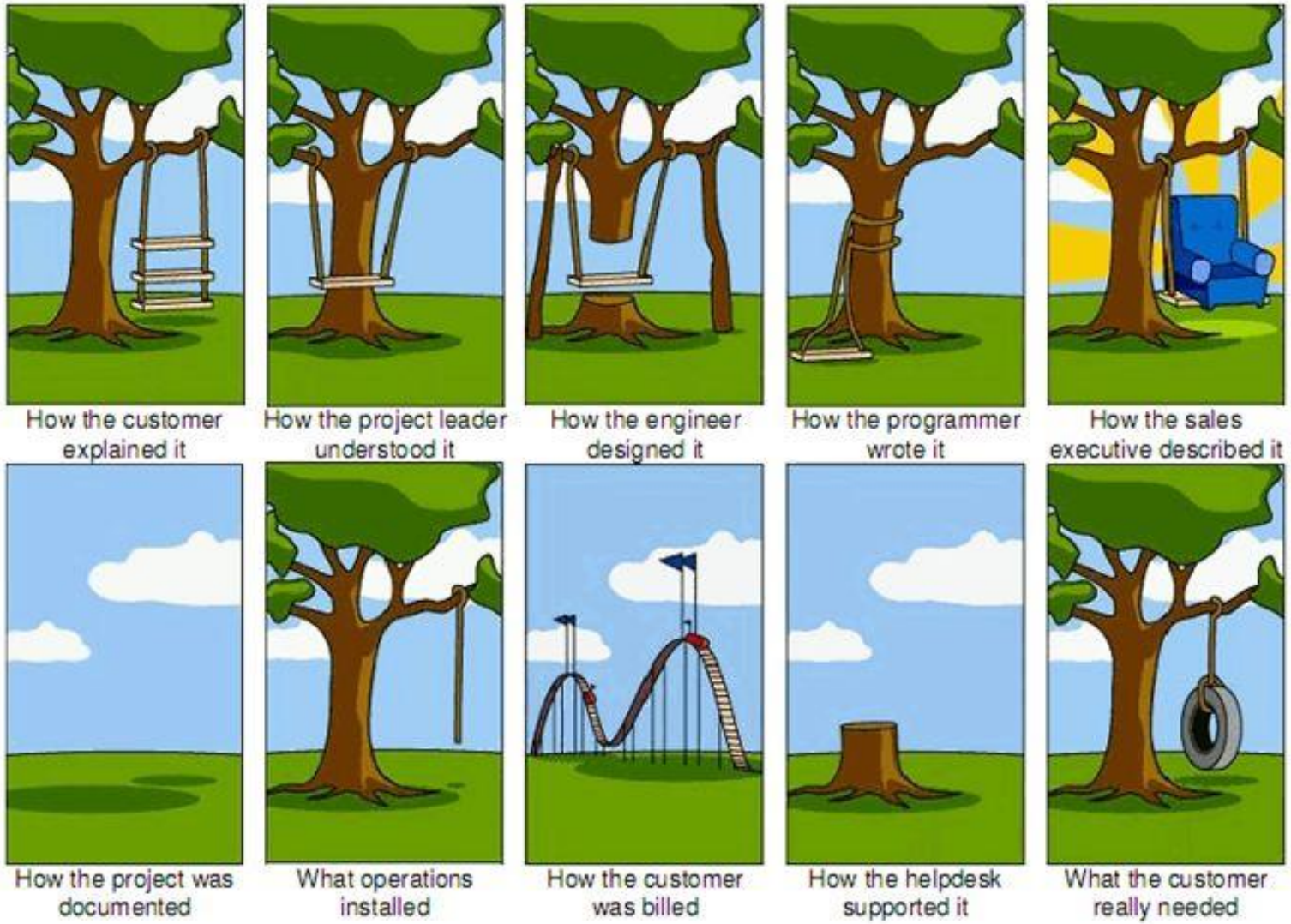
Behavior-Driven Development

Adaptive Software development

Core Activities

- Requirements
- Design
- Construction
- Testing
- Debugging
- Deployment
- Maintenance

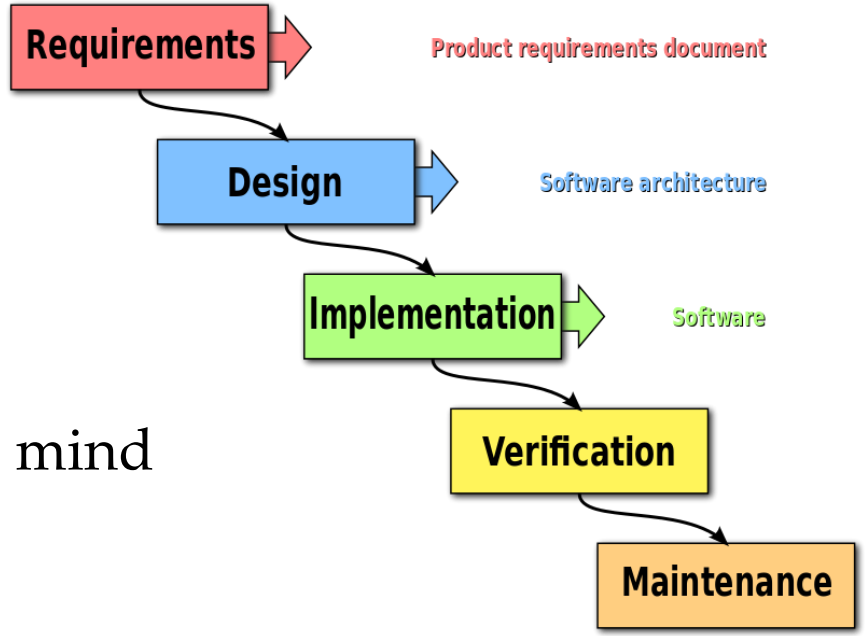



How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

The waterfall approach

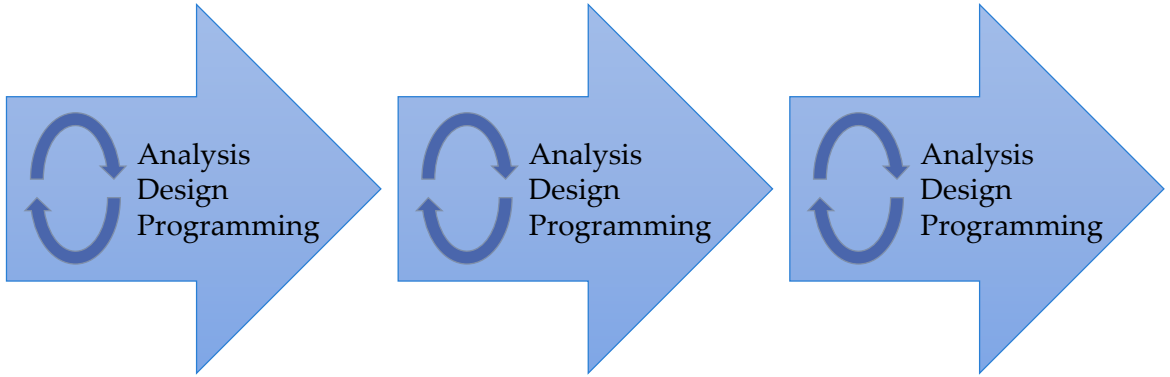
- A linear approach, one life cycle
- Upon completion of a step, you move to the next
- Early and extensive formalization of requirements
  - Almost ½ of the development time is dedicated to **documentation**, **not code**.

- It does not work!
- During implementation
  - Encounter problems you didn't think of
  - New requirements
  - The customers (and you) are going to change their mind



A software development methodology

- Should be responsive
  - Should allow the addition of new **features**
  - Should allow to **bugs** to be **fixed**
- Should allow **continual** programming,
  - Supported by continual analysis and design

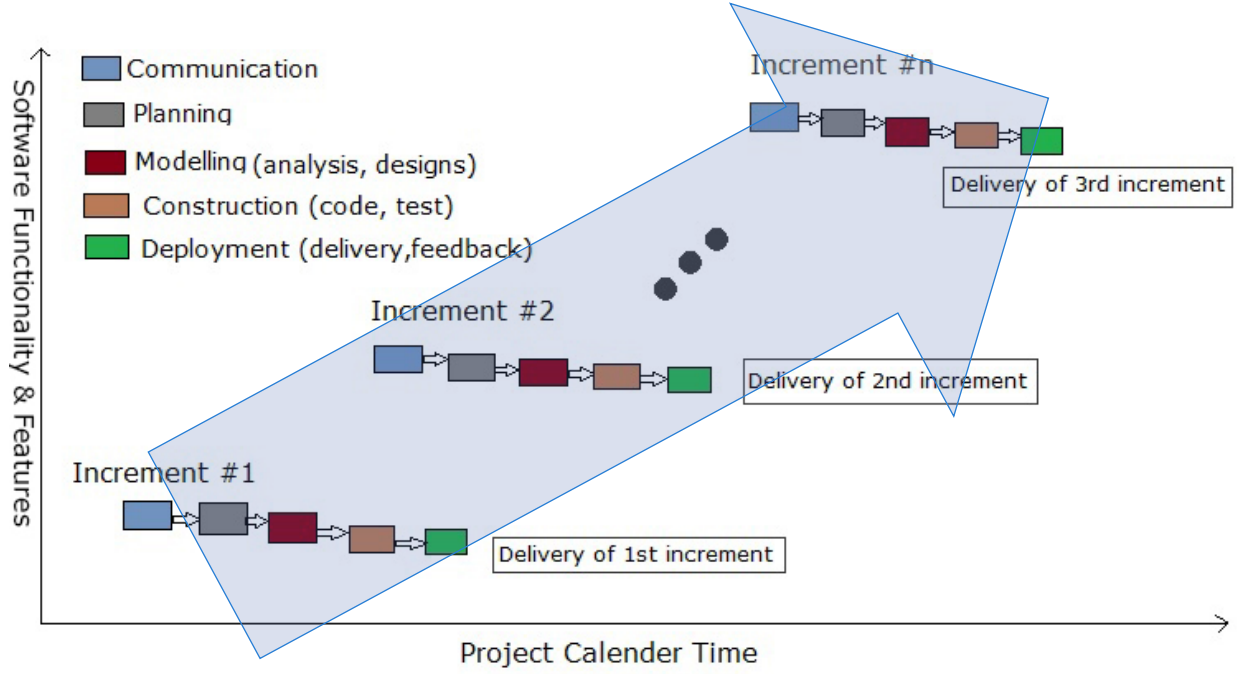


- It should be ***iterative***
  - Several incremental cycles
  - Each includes analysis, design and programming

Iterative Incremental Model

- The product is designed, implemented and tested incrementally
  - a little more is added each time

- Advantages
  - Few changes = easier to find bugs
  - Customer response
  - Initial product delivery is faster
    - (and costs lower)



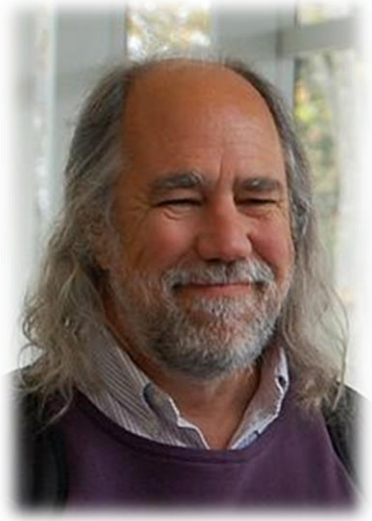
- Disadvantages
  - Resulting cost $>^{?}$ expected cost
  - Architectural problems as additional functionality is added

Object Oriented Analysis

Object Oriented Analysis

- **Analysis**: Emphasizes the an investigation of the **problem & requirements**
  - Rather than a solution (Design)

- **Object Oriented Analysis**:
  Describe the objects or concepts in the problem domain



- Grady Booch:

"Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain"

Object Oriented Analysis – the steps

- There are different methodologies, but the ideas are similar:

1. Gather **requirements**
2. **Describe** the application
3. **Identify** the main objects
4. Describe the **interaction** between these objects
5. Create a **class diagram**

These steps are not done once.

In an iterative approach you continually revisit and refine these steps over the time of your software development

Object Oriented Analysis – the steps

- There are different methodologies, but the ideas are similar:

1. Gather **requirements**
2. **Describe** the application
3. **Identify** the main objects
4. Describe the **interaction** between these objects
5. Create a **class diagram**

A conceptual model

Object Oriented Analysis – the steps

- There are different methodologies, but the ideas are similar:

1. Gather **requirements**
2. **Describe** the application
3. **Identify** the main objects
4. Describe the **interaction** between these objects
5. Create a **class diagram**

Gather Requirements

- Functional requirements
  - What does the application suppose to do?
  - Features
- Non-Functional requirements
  - Documentation / Help
  - Legal
  - Performance
  - Support
  - Security

Examples:
Functional requirements:
the app MUST generate a maze
the app MUST allow the user to solve a maze
the app MUST solve a maze automatically

Non-Functional requirements:
the app MUST return a solution within 3 seconds

FURPS

- Functional Requirements
  - Features
- Usability Requirements
  - Documentation, tutorials
- Reliability Requirements
  - Recovery, acceptable failure rates
- Performance Requirements
  - Response times, memory usage
- Supportability Requirements

FURPS+

- Design Requirements
  - Must be an Android app
- Implementation Requirements
  - Programming language
- Interface Requirements
  - With other systems
- Physical Requirements
  - Must run on a device with a camera

Object Oriented Analysis – the steps

- There are different methodologies, but the ideas are similar:

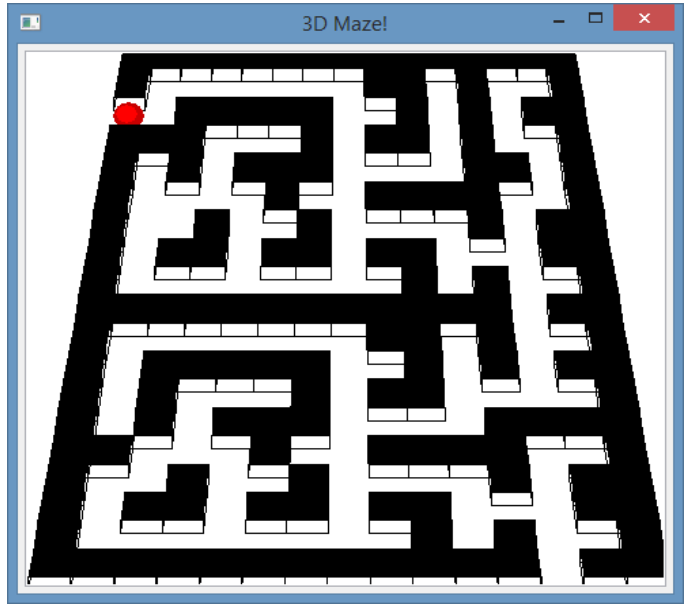
1. Gather **requirements**
2. **Describe** the application
3. **Identify** the main objects
4. Describe the **interaction** between these objects
5. Create a **class diagram**

Describe the application

- From the user point of view
- How does the user accomplish a certain goal with our application
- Write it in a free, every day, non-technical language
  - Use case
  - User stories / scenarios

- Example

1. The user asks to create a maze
2. The maze is displayed in the UI
3. The user moves a character to solve the maze
4. The user asks for a solution
5. The solution is displayed in the UI



Use Cases

Describe the application

Use Case

- We have written the requirements (“the app must have this feature”)
- Now we focus on the different users
- Use cases describe how the application should work
  - From the users perspectives

- A Use Case has a
  - Title – ***What*** is the goal?
  - Actor – ***Who*** desires it?
  - Scenario – ***How*** is it accomplished?

This is not an isolated feature.
It is an interaction
in full context

Other typical details of a Use Case

- Preconditions
  - Conditions that must be satisfied before the action can be taken
- Triggers
  - The events that lead to the execution of the action
- Main Success Scenario (sunny day scenarios)
- Alternative Scenarios
  - Also details exceptions and unsuccessful scenarios

Example: Withdraw Money from ATM

- Preconditions
  - The system has to be online
  - The money is above a minimal threshold

- Triggers
  - User's request
- Main Success Scenario
  - User logs in, System displays a menu, User selects "money withdrawal", System registers the transaction, System counts the money, etc.
- Alternative Scenarios
  - There isn't enough money in the ATM
  - System's informs the user, System aborts the action, System displays a menu, etc.

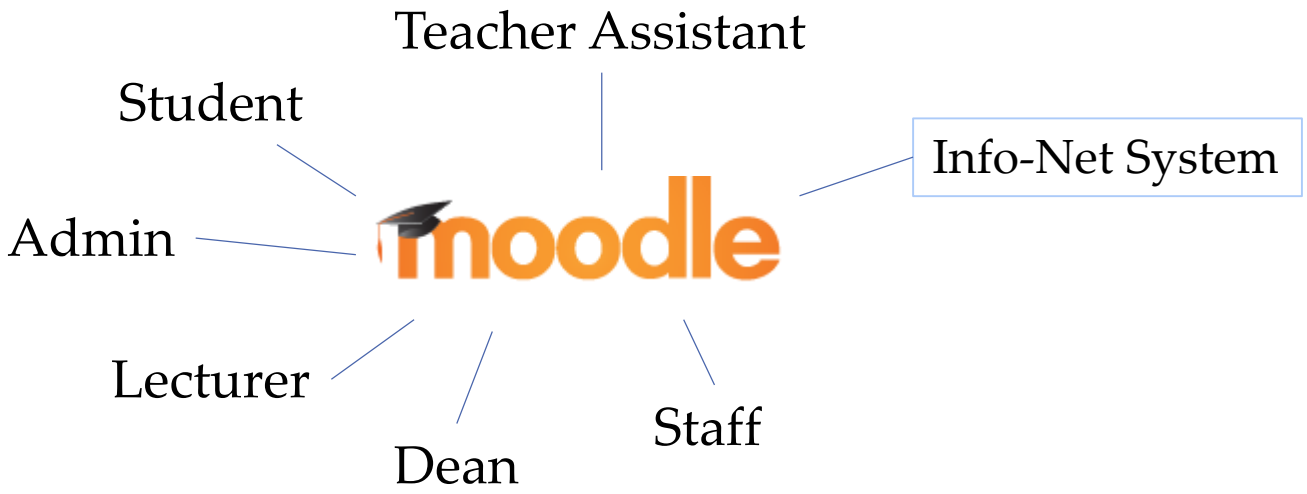
Use Case: Title

- An **active-verb** (short) goal phrase
- that names the goal of the actor
- Examples
  - **Register** a new Student 
  - **Enroll** a course
  - **Submit** a paper
  - **Grade** a paper
  - **View** class participants
  - **Upload** course material
  - **View** course material

Student registration 

Use Case: Actor

- An actor is not a generic user…
- "Actor" refers to any external system that acts on our system
  - Different types of users
    - Each with different goals
  - Other machines…

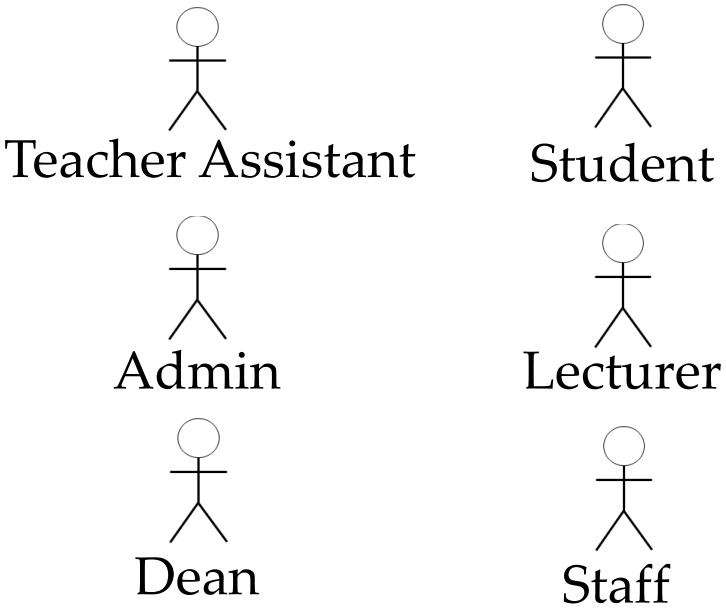


Identifying the Actors

- Does your system need to **interact** with **other systems** / organizations?
  - Backup systems, web services, external data sources, etc.
- Do you need to **distinguish** between **roles** or security groups?
  - Guests, Registered, Administrator, Owner, etc.
- Are there any different **job titles** or departments?
  - Staff, Dean, Lecturer, Teacher Assistant, Board member, etc.

Identifying the Actors

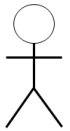
- Actors are identified according to goals



Courses Syllabus System

Identifying the Actors

- Actors are identified according to goals

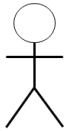


Primary Actor

Viewer

Courses Syllabus System

Secondary Actor



Editor

Use Case: Scenario

- Describes the details of accomplishing this one goal
  - As a paragraph
  - As a list of steps
- The typical user should understand it

- Example:
  - **Title**: Submit a paper
  - **Actor**: Student
  - **Scenario**:
    - Student logs in
    - Student views the course page and enters a submission box
    - Student uploads a paper
    - System validates the size of the paper
    - System stores the paper in the submission box
    - System responds with confirmation
    - …

This is not a pseudo code
It is the normal expected flow,
described from the perspective of a user

Identifying Scenarios

- We look for
  - User focused goals,
  - each with several steps,
  - that can be achieved in one encounter

- Examples
  - Submit a paper
  - Upload course material
  - View syllabus

- Log in
- Build a course

Be concise

- Don’t go into too much details
- The System connects the server through HTTPS, and uses XML to submit the provided search problem to be solved, then waits for a response, in the form of a Solution object 
- The system requests a solution to the search problem 

Focus on Intention, not UI

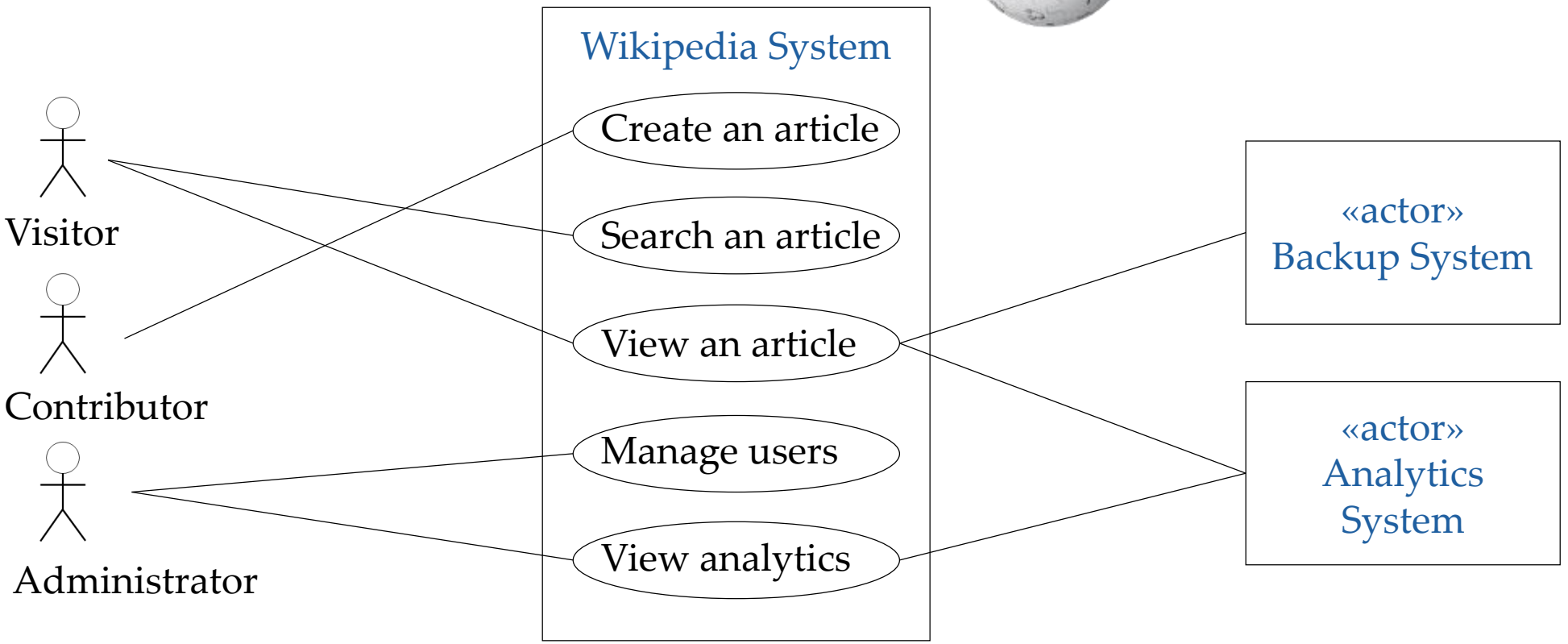
- Example:
  - **Title**: Submit a paper
  - **Actor**: Student
  - **Scenario**:
    - Student logs in
    - Student views the course page and enters a submission box
    - Student uploads a paper
    - System validates the size of the paper
    - System stores the paper in the submission box
    - System responds with confirmation
    - …

This is no mention of:
mouse, click, button, select, link, Java script, object, etc.

Use Case Diagram

- A diagram of several use cases and multiple actors at the same time
- We can get an overview of their interaction within a certain context
- It is not a replacement for a written use case
- It is not a sequence diagram

The Diagram


Wikipedia System
Create an article
Search an article
View an article
Manage users
View analytics
Visitor
Contributor
Administrator
«actor»
Backup System
«actor»
Analytics
System

Object Oriented Analysis – the steps

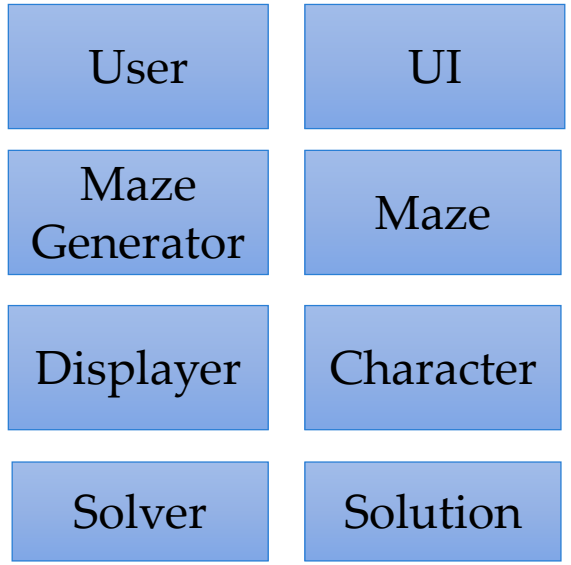
- There are different methodologies, but the ideas are similar:

1. Gather **requirements**
2. **Describe** the application
3. **Identify** the main objects
4. Describe the **interaction** between these objects
5. Create a **class diagram**

Identify the main objects

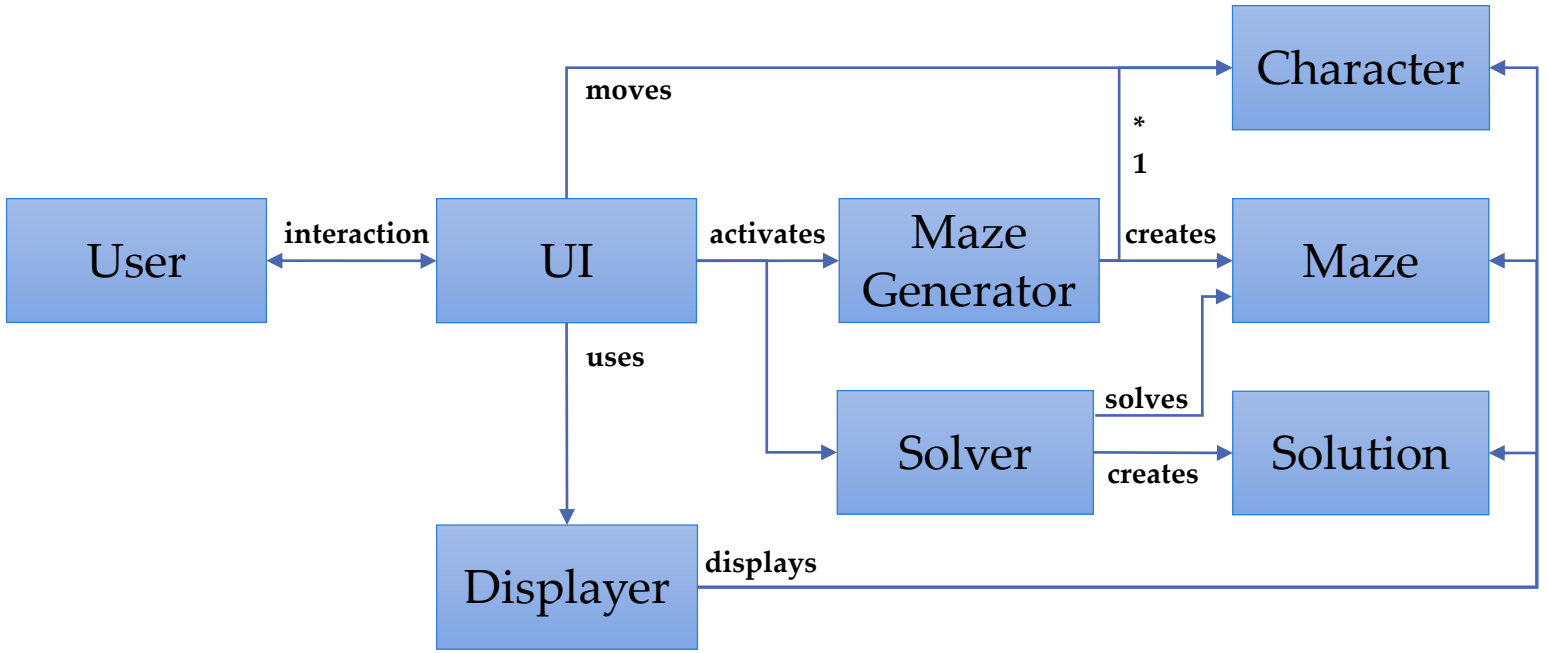
- The focus is shifted from the user to objects (object oriented)
- Not necessarily software-objects, but general entities of the application
- We create a **conceptual model**, not a full class diagram

1. A user asks to create a maze
2. The maze is displayed in the UI
3. A user moves a character to solve the maze
4. A user asks for a solution
5. The solution is displayed in the UI



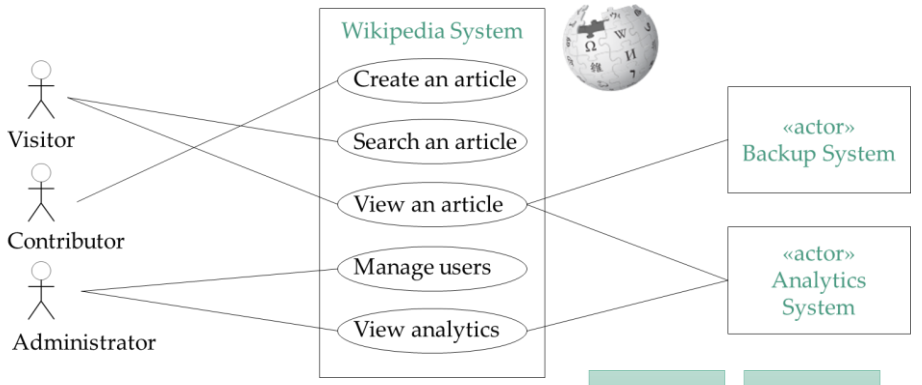
Describe the interaction of objects

- Simply put a line between boxes that have a relationship
- Describe the relationship



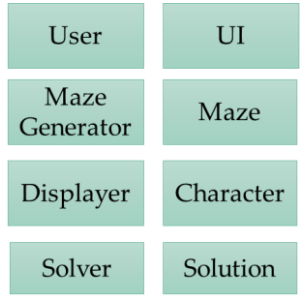
Object Oriented Analysis – the steps

1. Gather **requirements**

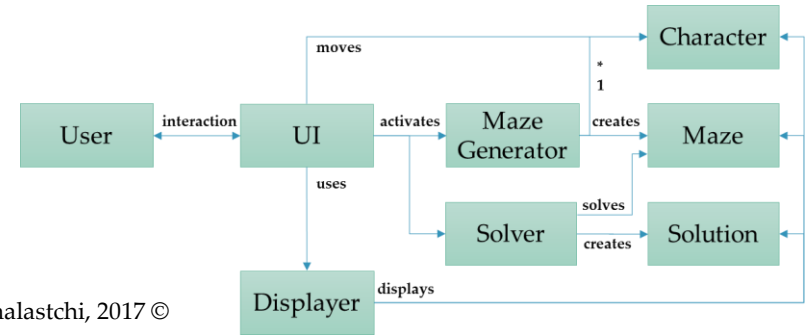


2. **Describe** the application

3. **Identify** the main objects



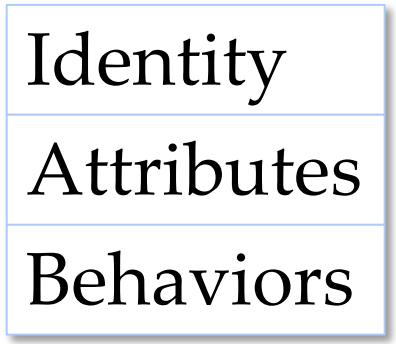
4. Describe the **interaction** between these objects



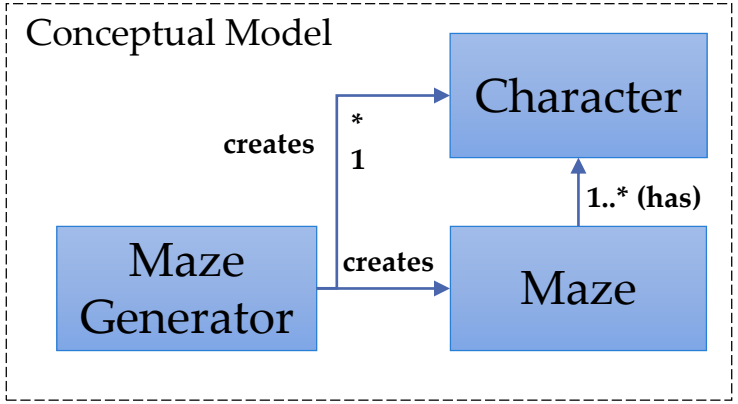
5. Create a **class diagram**

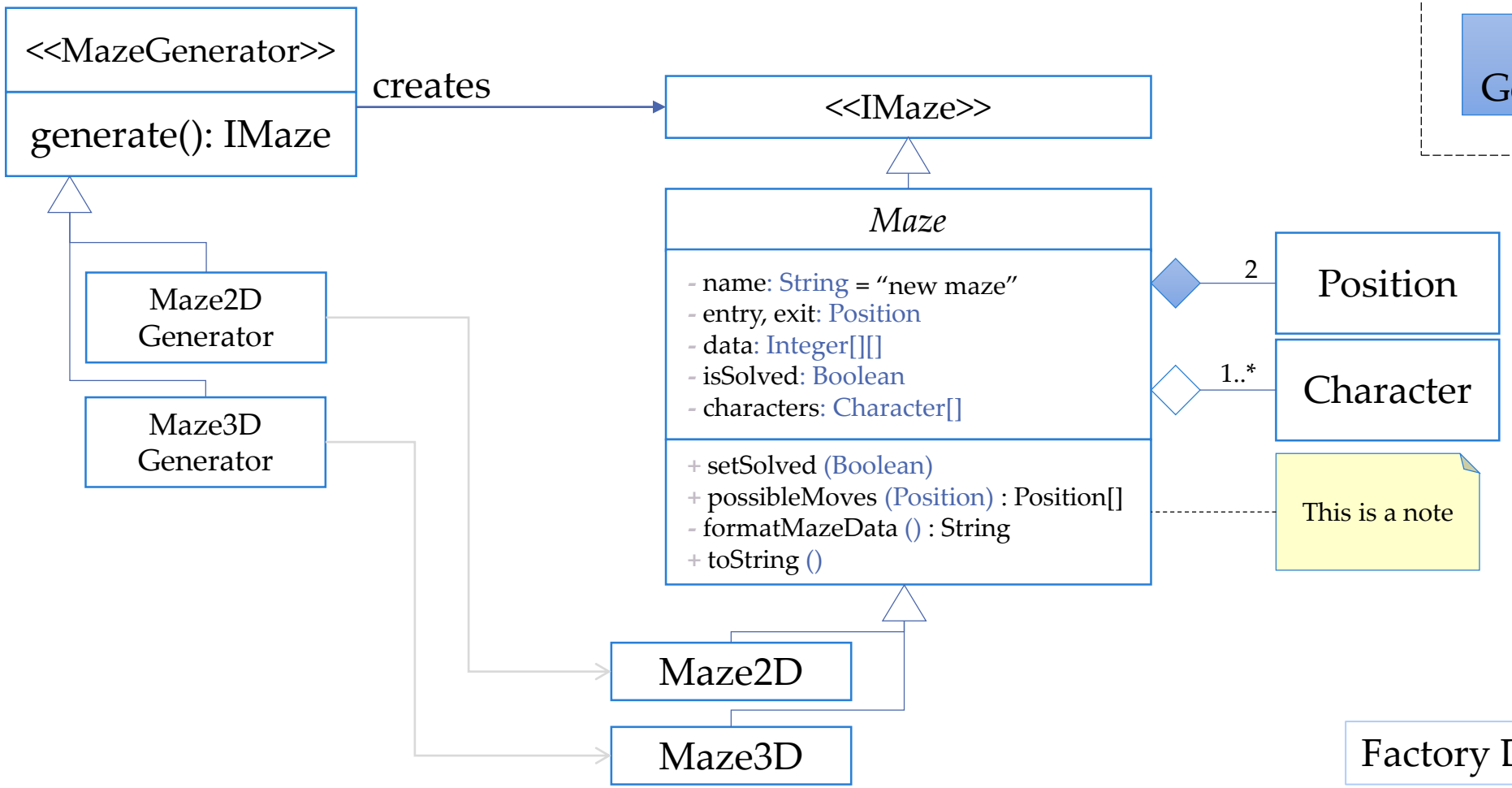
Creating a class diagram

- After the conceptual model is created, we can identify responsibilities
- We can **assign responsibilities** to classes
  - **One, and only one**, responsibility to a class
  - Each object is responsible of **its own attributes**
- These responsibilities define the class behaviors / methods



Class Diagram





Factory Design Pattern!