

Software Engineering with UML

Tewfik Ziadi

tewfik.ziadi@edu.ece.fr

- Tewfik Ziadi
 - Doctor and Engineer in Computer Sciences.
 - Since 2005, Associate Professor at Univ. Paris 6
 - Teaching Software Engineering & UML at Paris 6
 - L3
 - Master 1
 - Master 2
 - My research interests concern also the different aspects of Software Engineering

What is Software Engineering?

- **Engineering**: The Application of Science to the Solution of Practical Problems.
- **Software Engineering**: The Application of Computer Sciences to Build Practical Software Systems



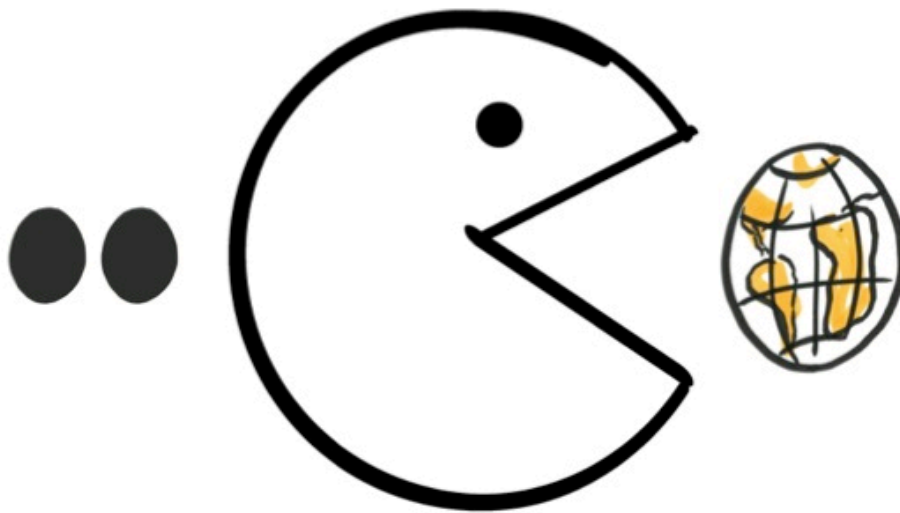
Specification

[illegible]

The software System

Why it's important?

Software is eating up the world*



* Marc Andreessen
in Wall Street Journal

5

Why it's important?

DJIA ▲ 21987.56 0.18%

S&P 500 ▲ 2476.55 0.20%



Nasdaq ▲ 6435.33 0.10%

U.S. 10 Yr ▼ -14/32 Yield 2.168%


Crude Oil ▲

THE WALL STREET JOURNAL.


HomeWorldU.S. PoliticsEconomyBusinessTechMarketsOpinionLife & ArtsReal Estate




WSJ








U.S. NEWS
Flooded Again, a Houston Neighborhood Faces ...



OPINION
The Houston Navy



BUSINESS
'Amazon' Sparks De Software-



ESSAY

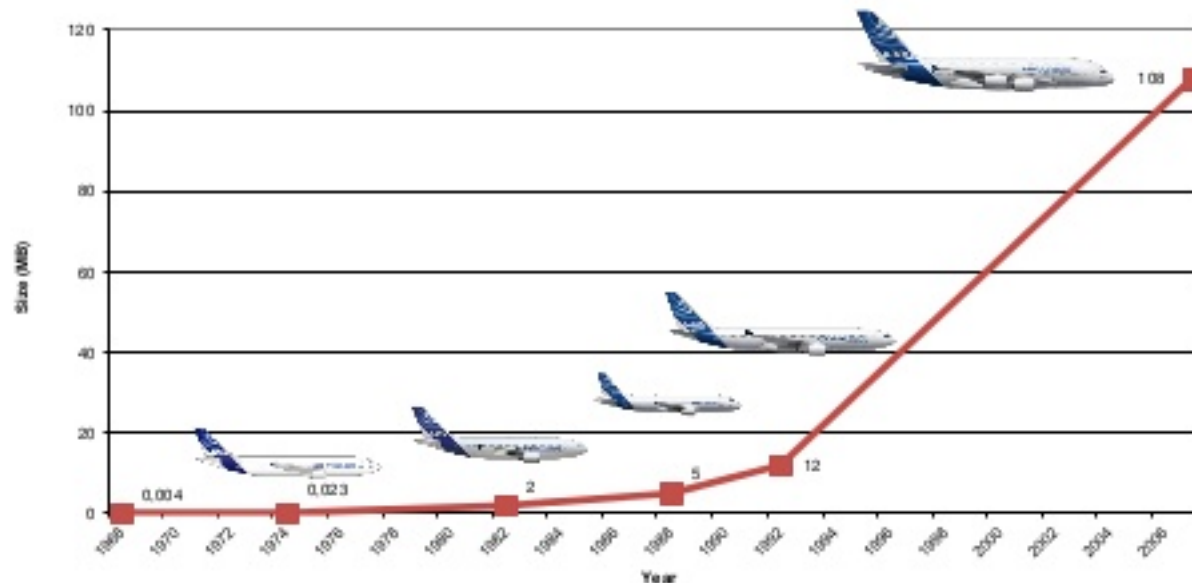
Why Software Is Eating The World

By Marc Andreessen
August 20, 2011

This week, Hewlett-Packard (where I am on the board) announced that it is exploring jettisoning its struggling PC business in favor of investing more heavily in software, where it sees better potential for growth. Meanwhile, Google plans to buy up the cellphone handset maker Motorola Mobility. Both moves surprised the tech world. But both moves are also in line with a trend I've observed, one that makes me optimistic about the future growth...

Why it's important?

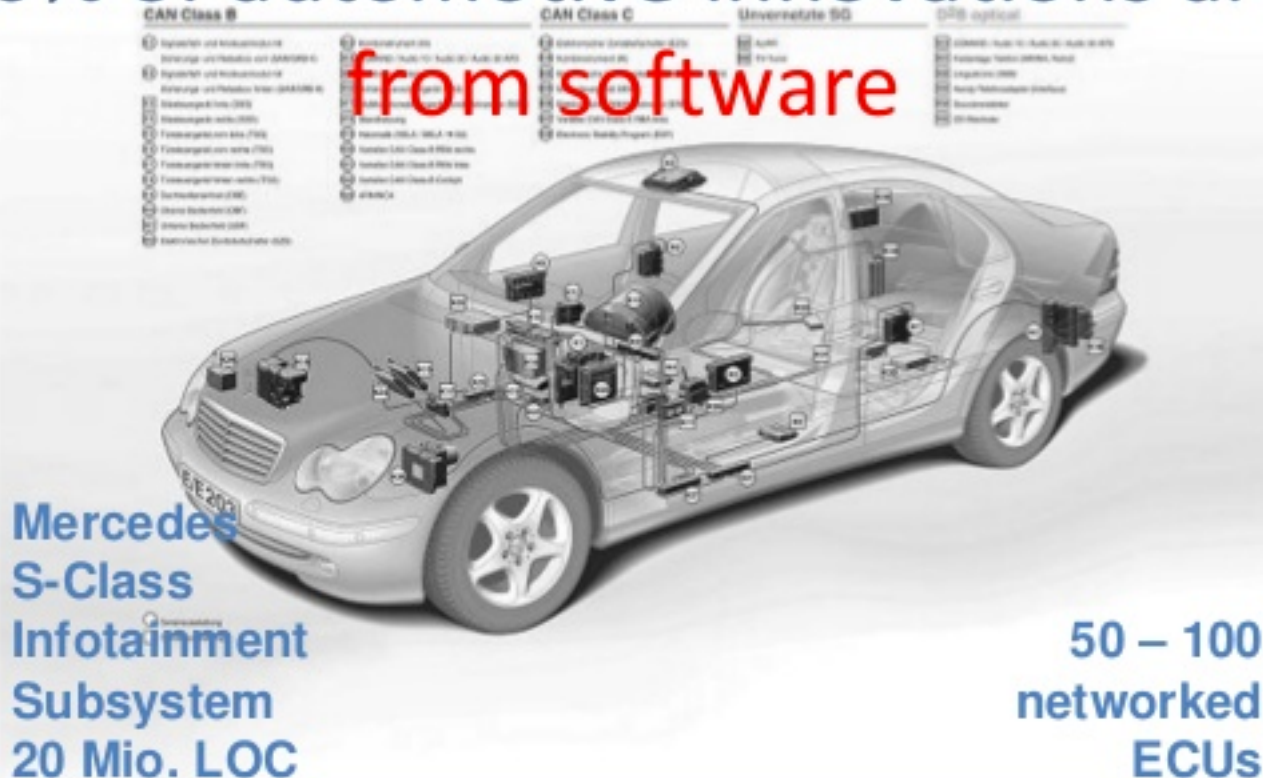
Software Embedded on Airbus Aircraft



Why it's important?

90% of automotive innovations are

from software



Why it's important?



GE CEO Jeff
Immelt

“Every industrial company will
become a software company”

http://www.ge.com/ar2013/pdf/GE_AR13_Letter.pdf

Why it's important?

The consequences with software problems can be very important:

- Therac 25, '85-'87 : 2 deaths
- Syst. Bagages Aeroport Denver, '95 : 16 months, 3.2 Mds\$
- Ariane 5 vol 88/501, '96: destr., 850 M\$
- Mars Climate Orbiter & Mars Polar Lander, '99 : destr.

Why it's important?

The cost of software fails in 2016



USD \$1.1 trillion
in assets



363 companies
affected



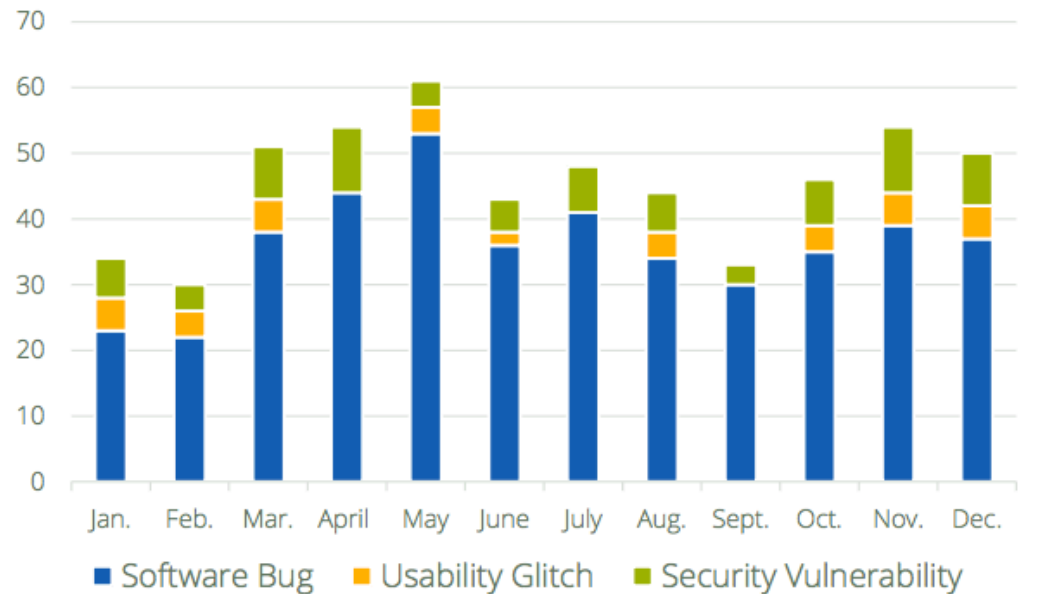
4.4 billion customers
affected



315 ½
lost years

source: tricentis.com 2016

RAYGUN



Type of fail by month, 2016

Issues for Software Engineering

- Now software systems are more and more **complex**

Example :

- Air traffic control system
- Reservation system e.g.; SNCF
- System Management of Schools
- Media System
- Beyond Systems of Systems
 - Airport Management

SE for complex systems

- Develop such systems needs **many programming languages** :
 - XML, Java, Javascript, .Net, C#, perl, python, C, C++, Ruby, SQL, PLSQL, ...
- Different **environments**
 - Local
 - Network
 - Real time
 - Critical
 - Nomad

SE for complex systems

- Various stakeholders
 - Project Manager (Advanced Project)
 - Developer (method, algorithm, dependence API)
 - Tester (test definition, test execution)
 - Deployer (put into production, testing scalability)
 - Quality (compliance charter, metrics)
 - CLIENT (met need)
- How to improve the development of this type of complex systems?
 - The quality
 - Productivity (time, cost)

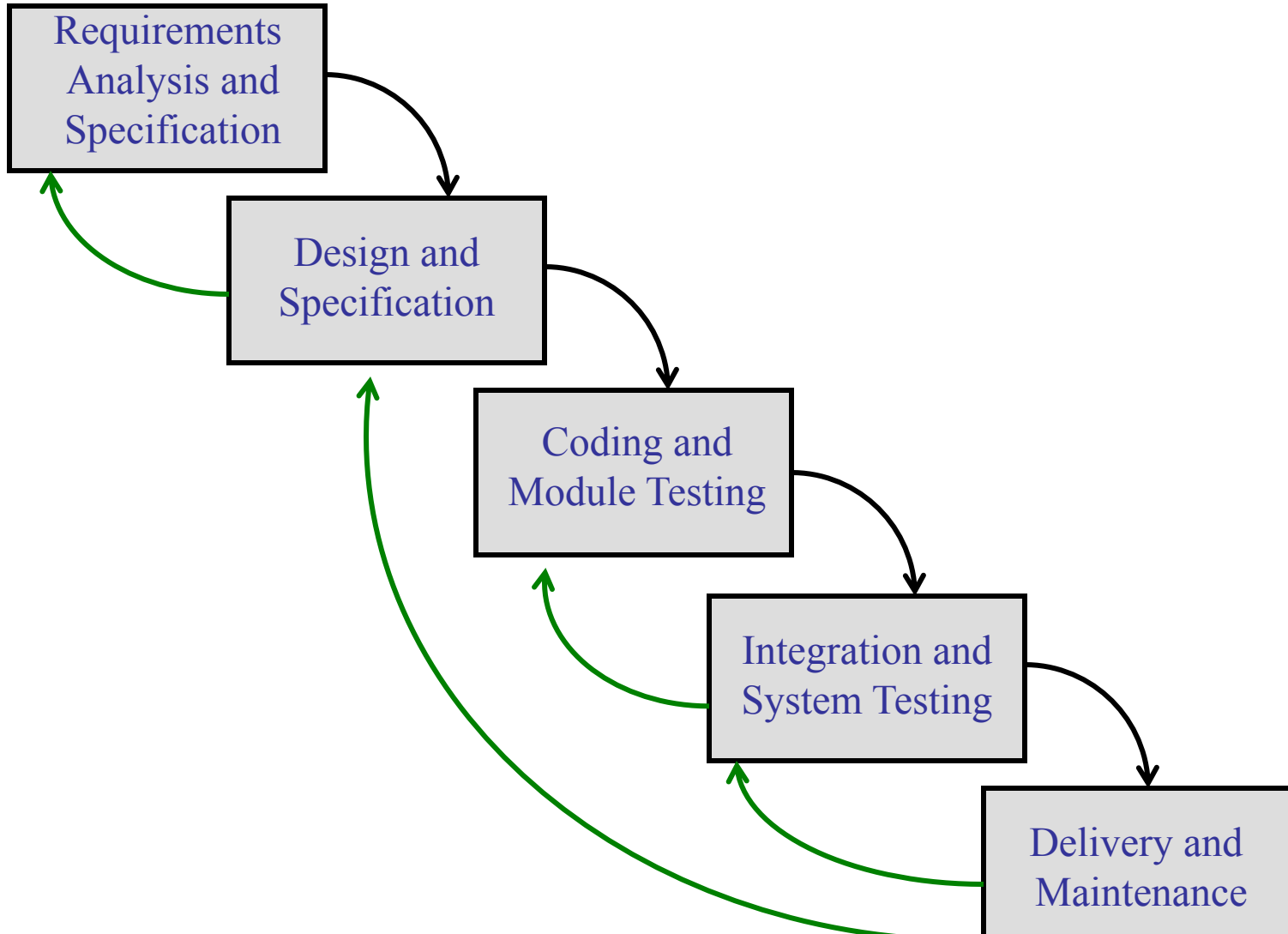
Requirements for Software Engineering

- Requirements for Software Engineering :
 - REQ1: A **process** (method). A set of activities that we can follow to develop the software.
 - REQ2: **Languages** and **tools** that can be used in the process.

REQ1: The software process

- A structured set of activities required to develop a software system.
- Many different software processes:
 - Waterfall
 - V-Cycle
 - ..ect

Waterfall Process Model



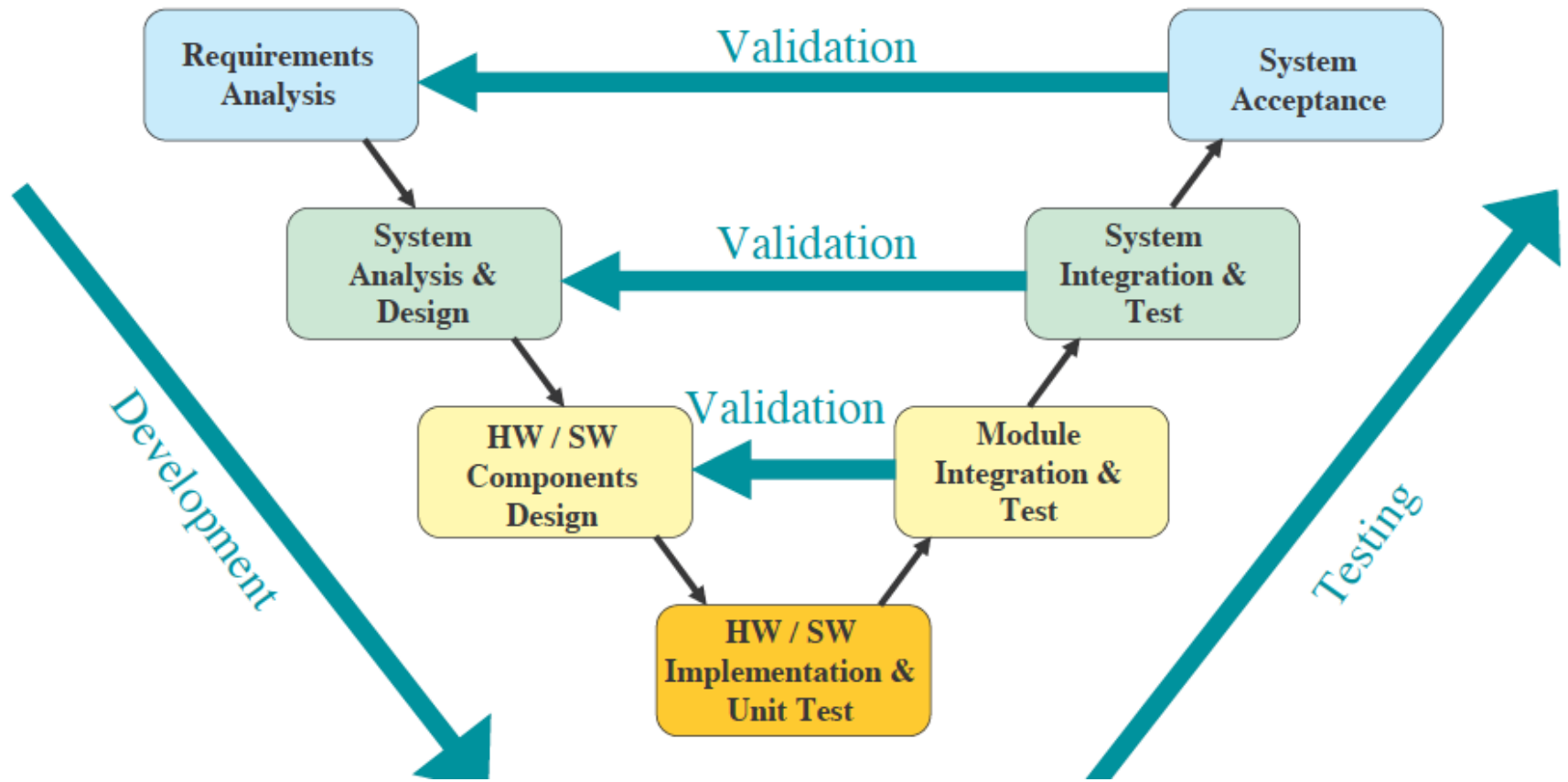
Software Lifecycle of Waterfall Model

- **Requirements Analysis and Specification**
 - What is the Problem to Solve?
 - What Does Customer Need/Want?
- **Design and Specification**
 - How is the Problem to be Solved?
- **Coding and Module Testing**
 - Writing Code to Meet Component/Module Design Specifications
- **Integration and System Testing**
 - Integration of Components/Modules into Subsystems
 - Integration of Subsystems into Final Program

Software Lifecycle of Waterfall Model

- **Delivery and Maintenance**
 - System Delivered to Customer/Market
 - Bug Fixes and Version Releases Over Time

V-Cycle Process Model



The software process

- Many different software processes but all involve:
 - **Analysis** - defining what the system should do; **WHAT**
 - **Design and implementation** - defining the organization of the system and implementing the system; **HOW**
 - **Validation** - checking that it does what the customer wants;
 - **Maintenance & Evolution** - changing the system in response to changing customer needs.

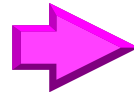
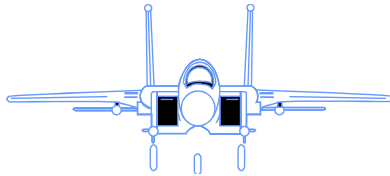
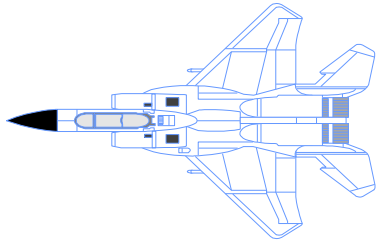
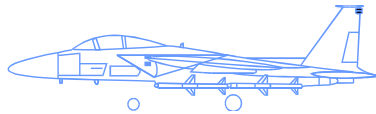
REQ2: Languages

- Limitation of programming languages
 - Only one level of abstraction
 - Only one level of diversity

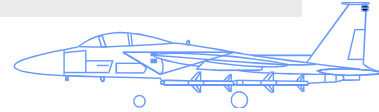
➔ Modeling Languages

What Is a Model?

- A model is a **abstraction of the reality**.
- Follows a formal set of rules.
- That facilitates interpretation by man and machine.



Why model?



- We build models of complex systems because you cannot comprehend such a system in its entirety.
- We build models to better understand the system you are developing.
- We build models to communicate (working inside a team).

Warning : abstraction = simplification?

Modeling simplifies the comprehension and the communication around the problem. However, it does not simplify the problem itself!

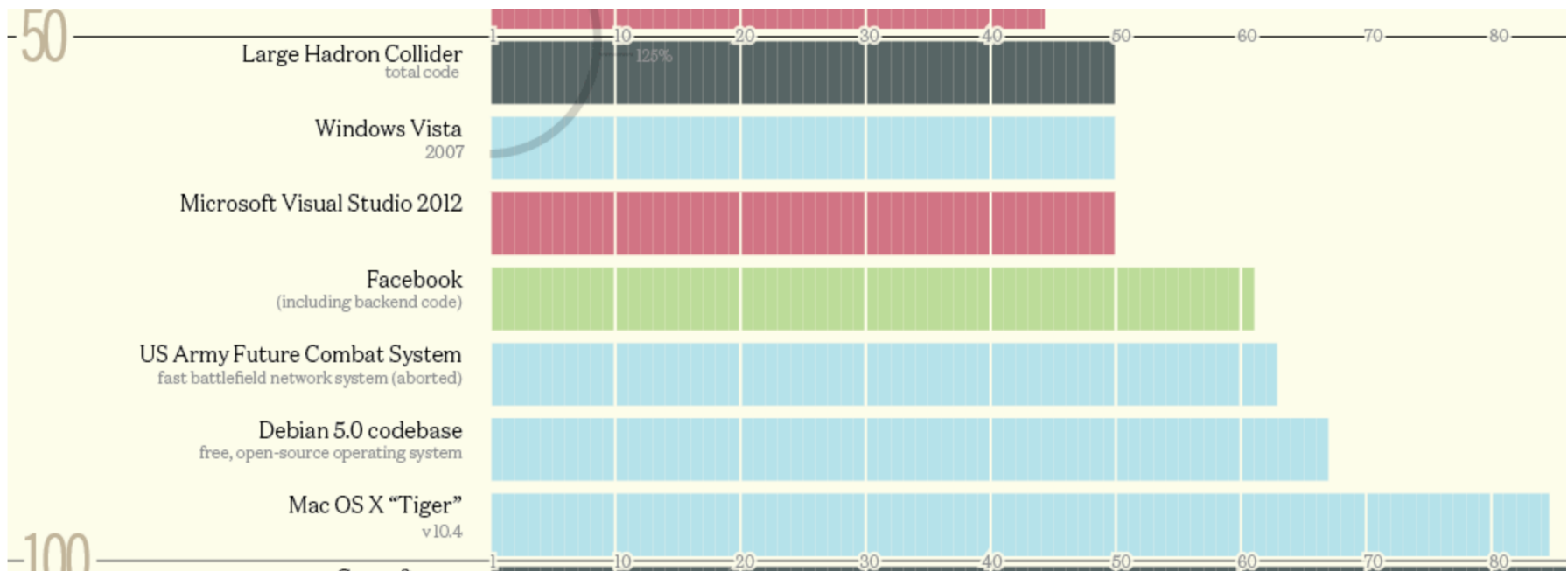
Example: Google Maps

- Different abstraction levels
- Different views



Why model?

To manage the complexity



Why model?

To perpetuate the expertise

- Some projects may take **many years**
 - Not always the same people working on the project
 - Need to capitalize a knowledge independent from code and technologies.
 - Capturing the business without worrying about technical details
- Examples of projects:
 - Air control system (Thalès): Project ~ 8 years, life 40 years
 - The construction of a new plan (Airbus): Project ~ 10 years, life 50 ans

Why model?

Increase productivity

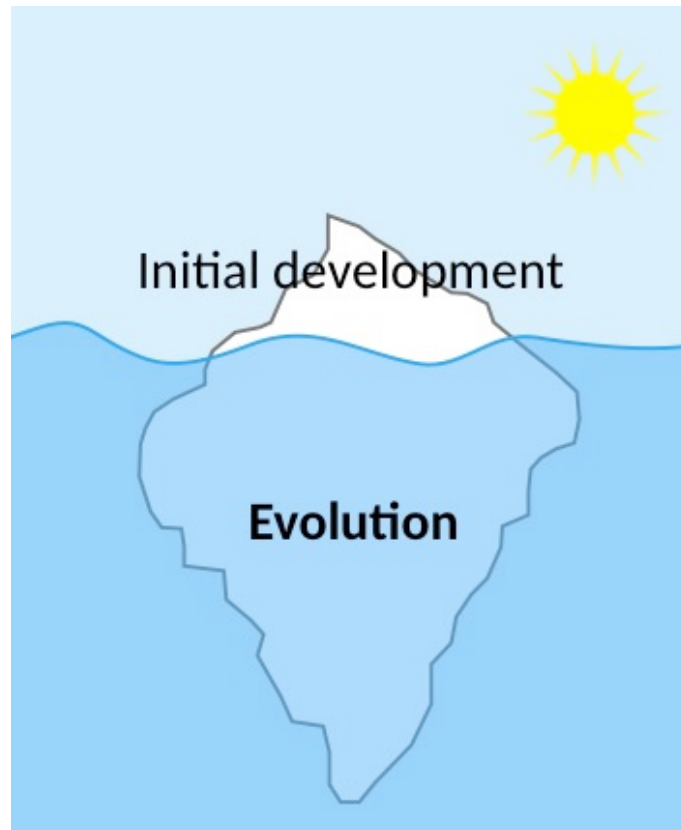
- Models can be directly connected to a variety of programming languages.
 - Code generation from models (**Model-Drive Engineering** vision)

Manage variability

- The notion of **Software Product Lines** to manage variants of the same software
- Exemple: Mobiles
 - billions of phones (
 - Many thousands of versions of the same software

Why model?

- To facilitate software evolution management



Software = Code ?

- Is it important to ask this question?
- Before yes but it's not the case today:
 - **Software = Documentation + Models + Code**
 - Many models, many views for the same software
 - Models to generate the documentation
 - Models to generate the source code (100% in some cases)

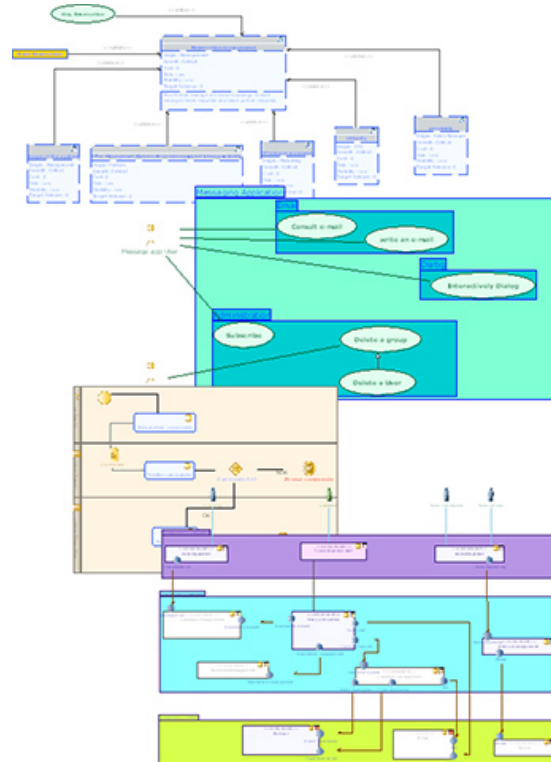
But what kind of modeling language that we can use?

- Many existing modeling languages
- But only one **LANGUAGE** for OO modeling
- **UML (Unified Modeling Language): Why we need to use it?**
 - It's **THE standard** (used in more than 80% of IT projects)
 - Standardised by OMG (Open Management Group)
 - Many existing tools and documents (books, tuto, forums, etc.)

Models as the core elements in SE



Specification

[illegible]

The source code

UML

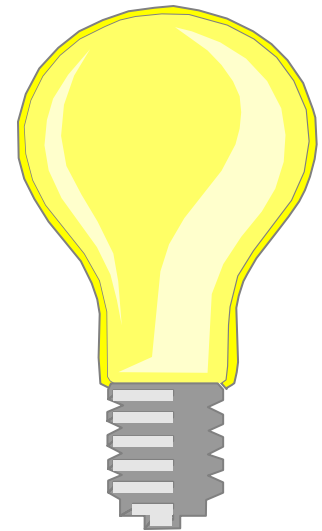
What Is the UML?

- The UML is a language for
 - Visualizing
 - Specifying
 - Constructing
 - Documenting
- the artifacts of a software-intensive system.



The UML Is a Language for Visualizing

- Communicating conceptual models to others is prone to error unless everyone involved speaks the same language.
- There are things about a software system you can't understand unless you build models.
- An explicit model facilitates communication.



The UML Is a Language for Constructing

- UML models can be directly connected to a variety of programming languages.
 - Maps to Java, C++, Visual Basic, and so on
 - Tables in a RDBMS or persistent store in an OODBMS
 - Permits forward engineering
 - Permits reverse engineering

The UML Is a Language for Documenting

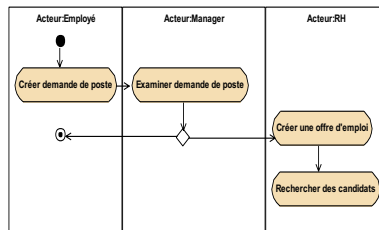


Diagramme d'activit  

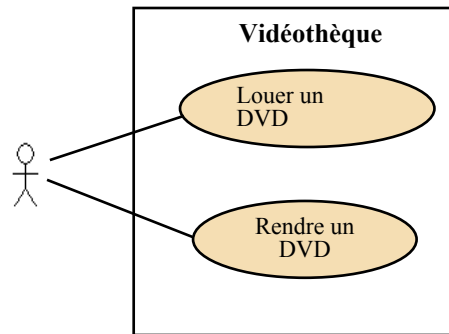


Diagramme de Cas d'Utilisation

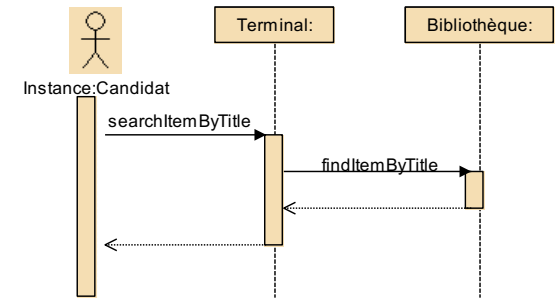
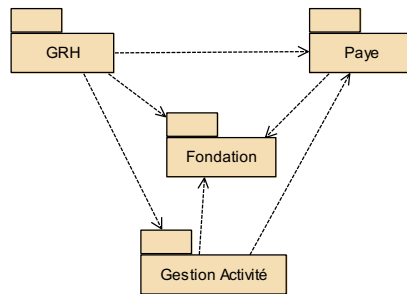


Diagramme de s  quences



Digramme de packages

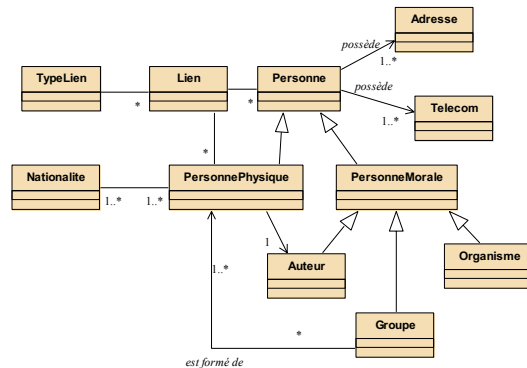


Diagramme classes

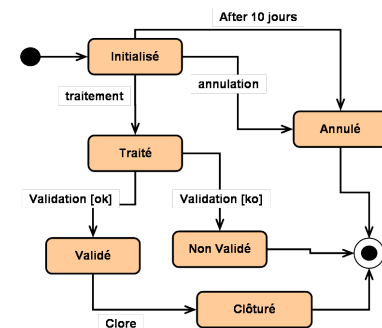
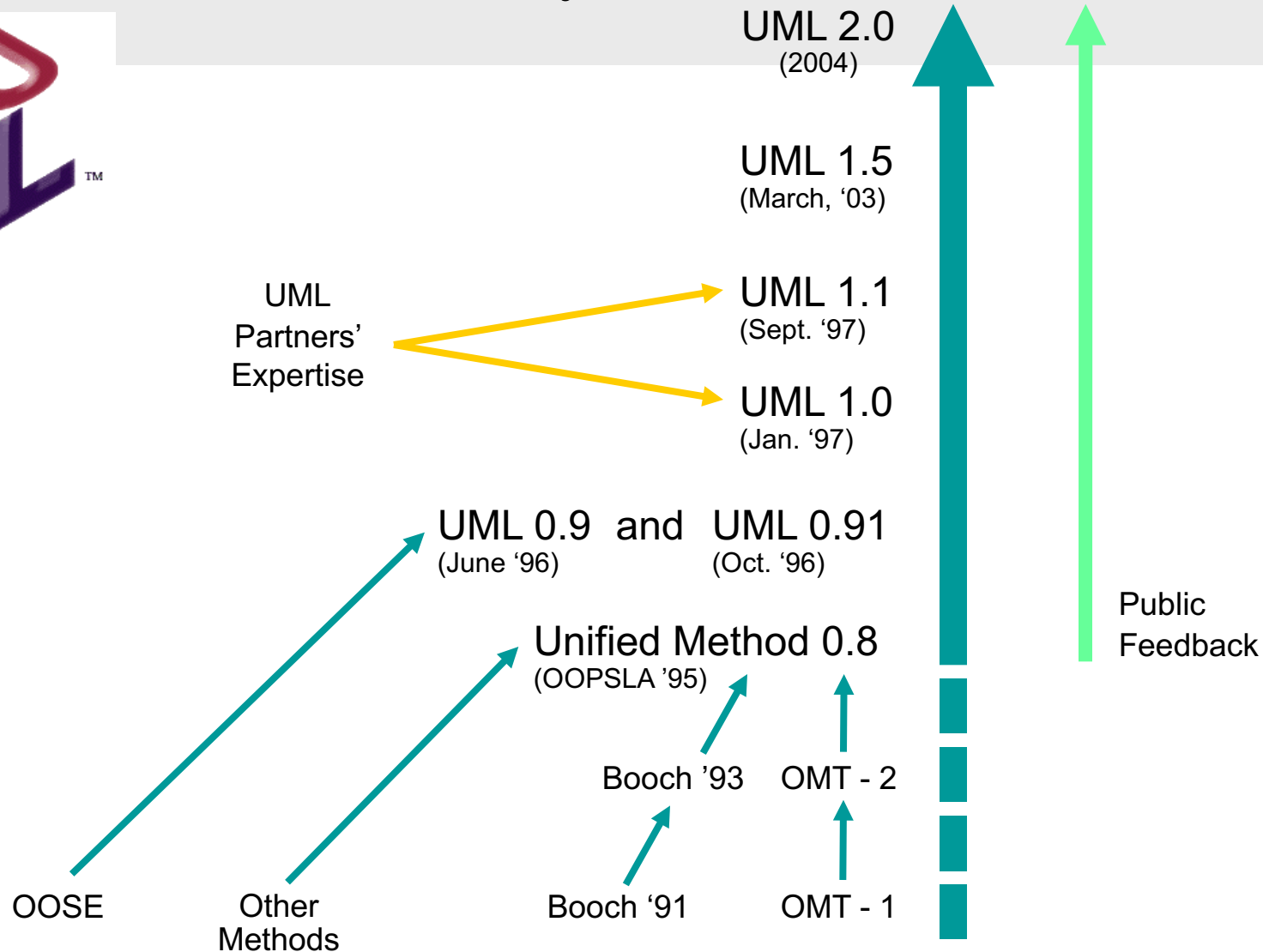
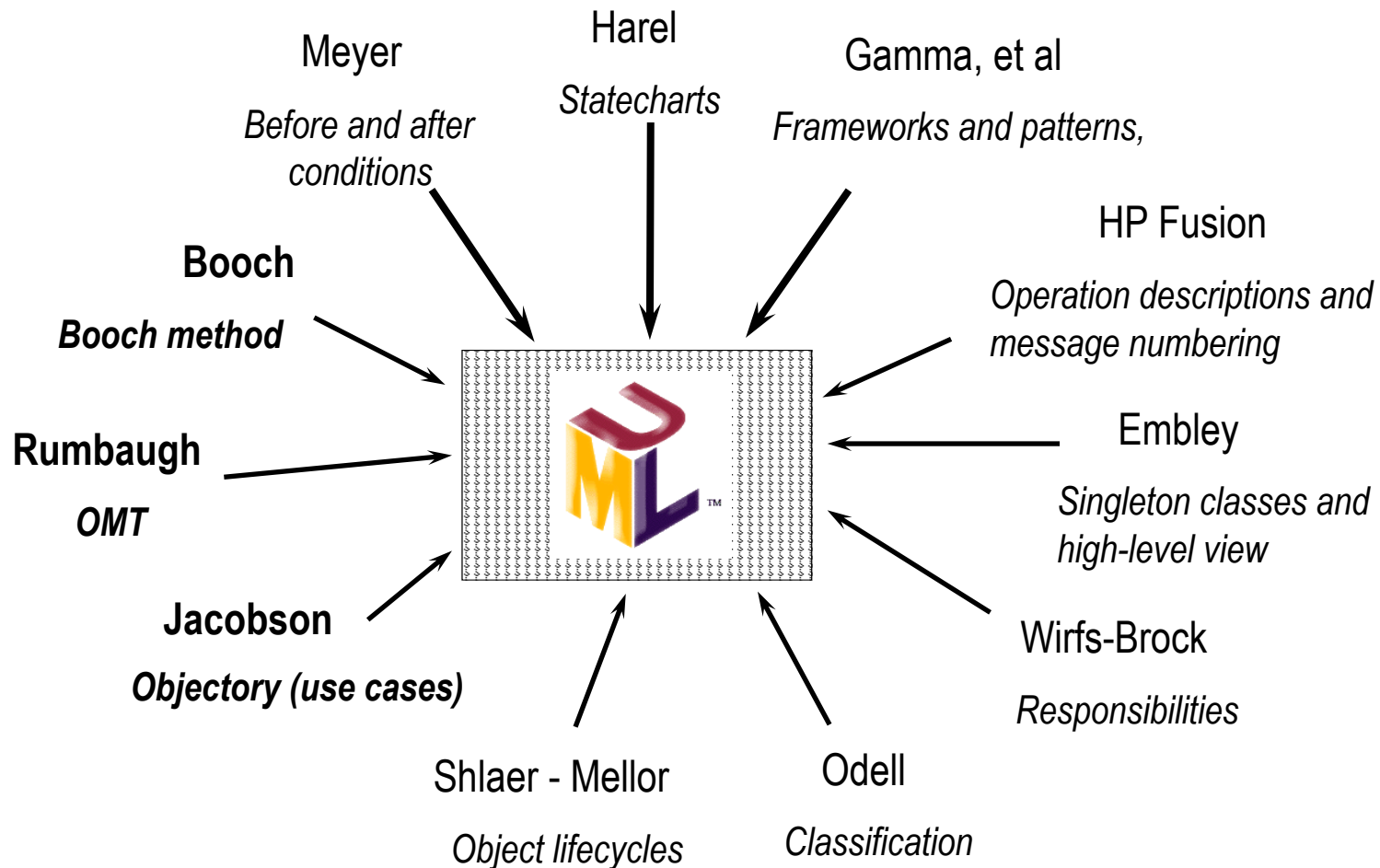


Diagramme   tats/transitions

History of the UML



Inputs of UML



The diagrams of the UML 2.0

- **Activity diagram**
High-level business processes, including data flow, or to model the logic of complex logic within a system.
- **Class diagram**
Shows a collection of static model elements such as classes and types, their contents, and their relationships.
- **Use case diagrams + descriptions**
Defines the functional requirements of a system. Shows use cases, actors, and their interrelationships.
- **Component diagram**
Depicts the components that compose an application, system, or enterprise. The components, their interrelationships, interactions, and their public interfaces are depicted.
- **Deployment diagram**
Shows the execution architecture of systems. This includes nodes, either hardware or software execution environments, as well as the middleware connecting them.
- **State Machine diagram**
Describes the states an object or interaction may be in, as well as the transitions between states. Formerly referred to as a state diagram, state chart diagram, or a state-transition diagram.

The diagrams of the UML 2.0

- **Communication diagram**
Shows instances of classes, their interrelationships, and the message flow between them. Communication diagrams typically focus on the structural organization of objects that send and receive messages. \ Formerly called a Collaboration Diagram.
- **Composite structure diagram**
Depicts the internal structure of a classifier (such as a class, component, or use case), including the interaction points of the classifier to other parts of the system.
- **Interaction overview diagram**
A variant of an activity diagram which overviews the control flow within a system or business process. Each node/activity within the diagram can represent another interaction diagram.
- **Object diagram**
Depicts objects and their relationships at a point in time, typically a special case of either a class diagram or a communication diagram. \
- **Package diagram**
Shows how model elements are organized into packages as well as the dependencies between packages
- **Timing diagram**
Depicts the change in state or condition of a classifier instance or role over time. Typically used to show the change in state of an object over time in response to external events.

Organisation of the course (1/4)

- Week 1 : Introduction to SE with UML
 - Part 1 : Course
 - Part 2 : Exercises/ Lab work
- Week 2 : Requirement analysis: Use case diagrams
 - Part 1 : Course
 - Part 2 : Exercises/ Lab work
- Week 3 : Structural modeling class diagrams + design pattern
 - Part 1 : Course
 - Part 2 : Exercises/ Lab work
- Week 4: Structural modeling : Lab work and work on project.
 - Part 1 : Exercises/ Lab work
 - Part 2 : Work on the project.

Organisation of the course (2/4)

- Week 5 : Sequence diagrams + state machines
 - Part 1 : Course
 - Part 2 : Exercises/ Lab work
- Week 6 : Sequence diagrams + state machines: Lab work
 - Part 1 : Exercises/ Lab work
 - Part 2 : Work on the project
- Week 7 : Code generation & reverse engineering
 - Part 1: Course
 - Part 2: Exercises/ Lab work
- Week 8 : Software Testing
 - Part 1 : Course
 - Part 2 : Exercises/ Lab works

Organisation of the course (3/4)

- Week 9 : Software Product Lines
 - Part 1 : Course
 - Part 2 : Exercises/ Lab work

Organisation of the course (4/4)

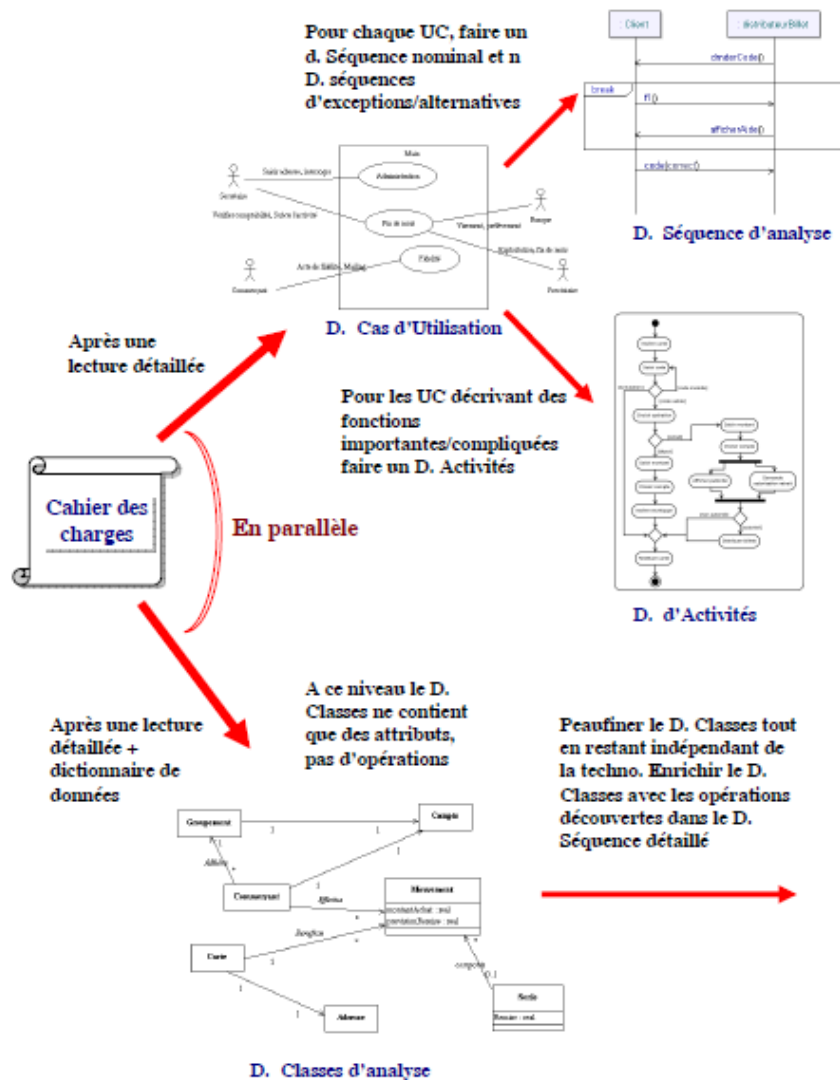
Evaluation

- Intermediate Written Test (Week October 16) ==> (20%)
- Long Mini Project (30%)
- Final Exam (50%)

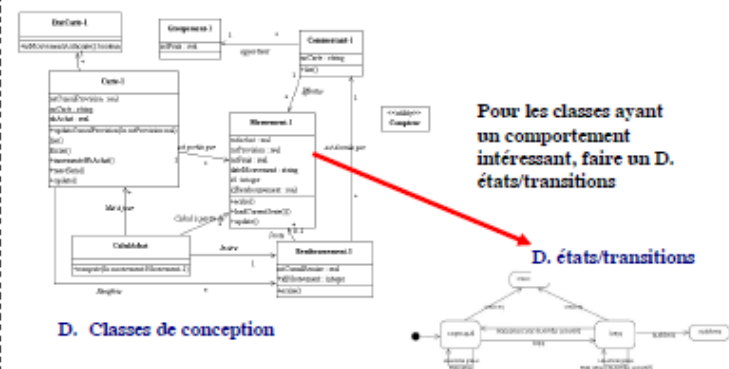
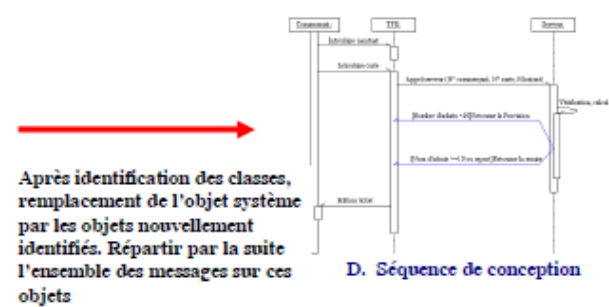
Long Mini Project (using the Modelio UML tool)

- Applying the concepts to develop a system InterimECE (the specification document is available on the campus web page).
- Groups of 2 students (maximum 3).
- One final report (deadline before Week 8).

Process & diagrammes



Besoins & Analyse



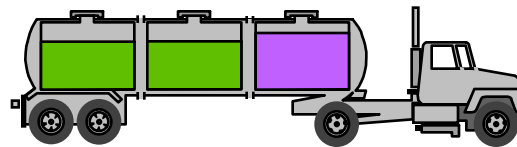
Conception

OBJECT-ORIENTED MODELING PRINCIPALS

What Is an Object?

- Informally, an object represents an entity, either physical, conceptual, or software.

- Physical entity



Truck

- Conceptual entity



Chemical Process

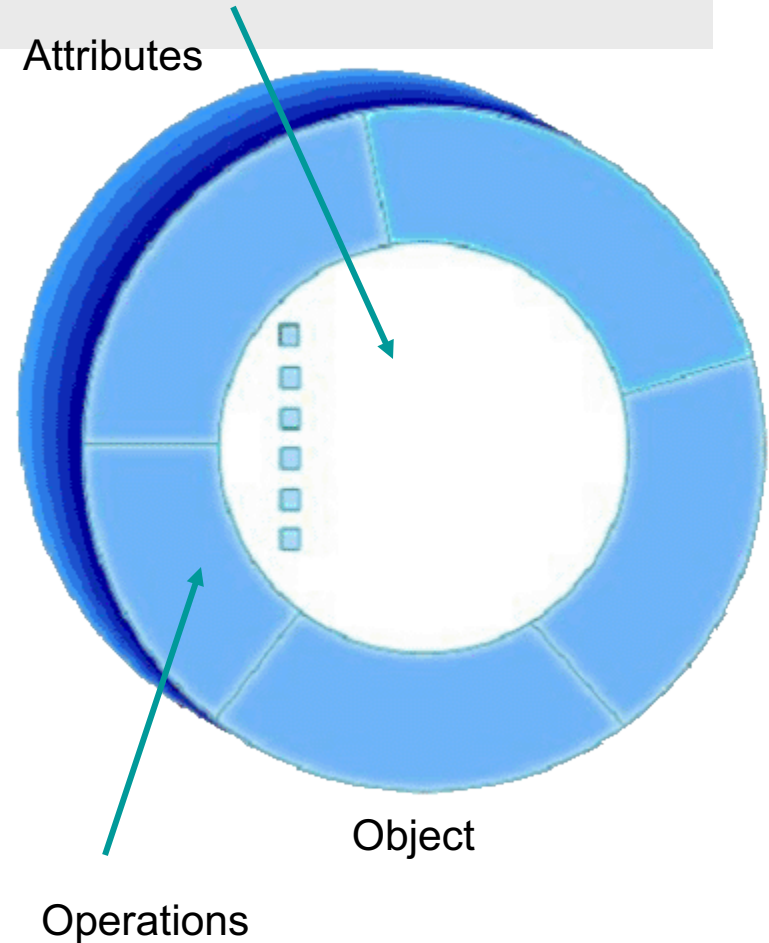
- Software entity



Linked List

A More Formal Definition

- An object is an entity with a well-defined boundary and identity that encapsulates state and behavior.
 - State is represented by attributes and relationships.
 - Behavior is represented by operations, methods, and state machines.

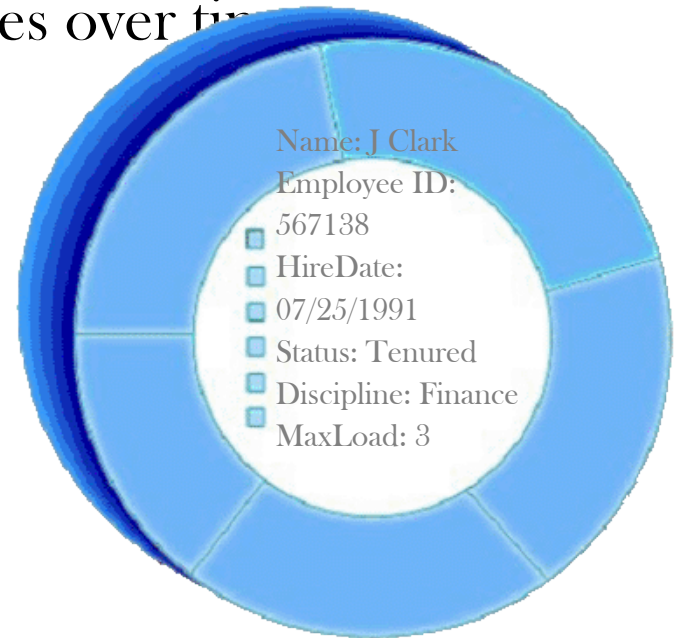


An Object Has State

- State is a condition or situation during the life of an object, which satisfies some condition, performs some activity, or waits for some event.
- The state of an object normally changes over time.



Name: J Clark
Employee ID: 567138
Date Hired: July 25, 1991
Status: Tenured
Discipline: Finance
Maximum Course Load: 3 classes



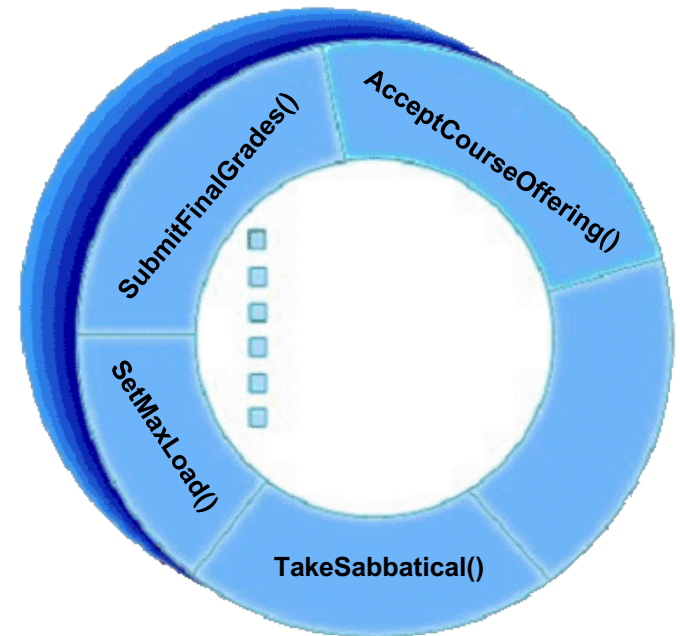
Professor Clark

An Object Has Behavior

- Behavior determines how an object acts and reacts.
- The visible behavior of an object is modeled by a set of messages it can respond to (operations that the object can perform).



Professor Clark's behavior
Submit Final Grades
Accept Course Offering
Take Sabbatical
Set Max Load



Professor Clark

An Object Has Identity

- Each object has a unique identity, even if the state is identical to that of another object.



Professor “J Clark”
teaches Biology



Professor “J Clark”
teaches Biology



Dupont et Dupondt

Basic Principles of Object Orientation



Object Orientation

Abstraction

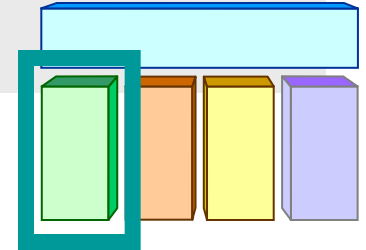
Encapsulation

Modularity

Hierarchy

What Is Abstraction?

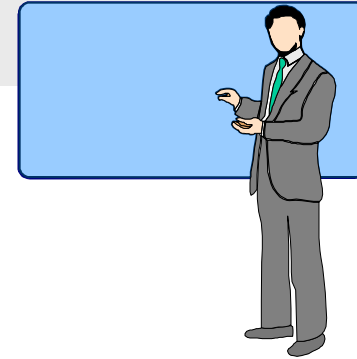
- A representation of an entity that distinguishes it from all other kinds of entities.



Example: Abstraction



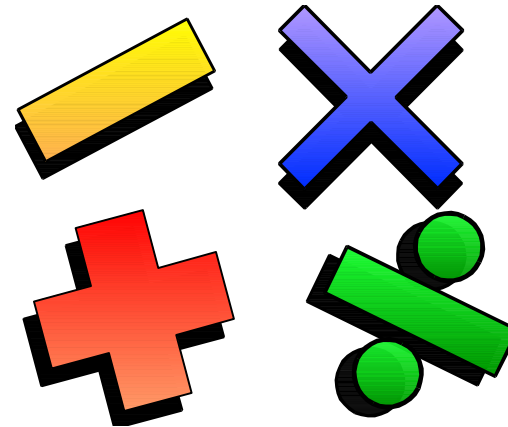
Student



Professor



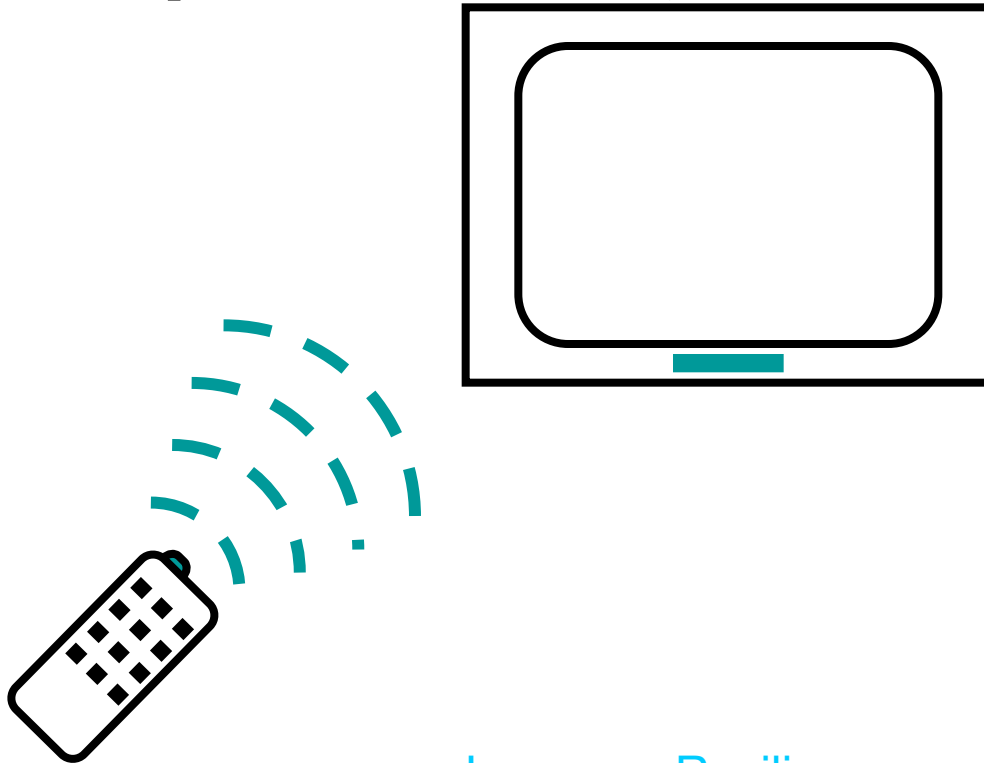
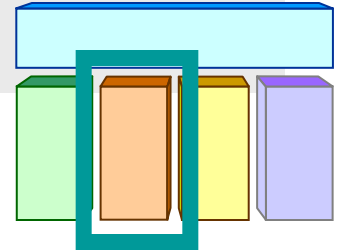
Course Offering (9:00 a.m.,
Monday-Wednesday-Friday)



Course (e.g. Algebra)

What Is Encapsulation?

- Hides implementation from clients.
 - Clients depend on interface only.



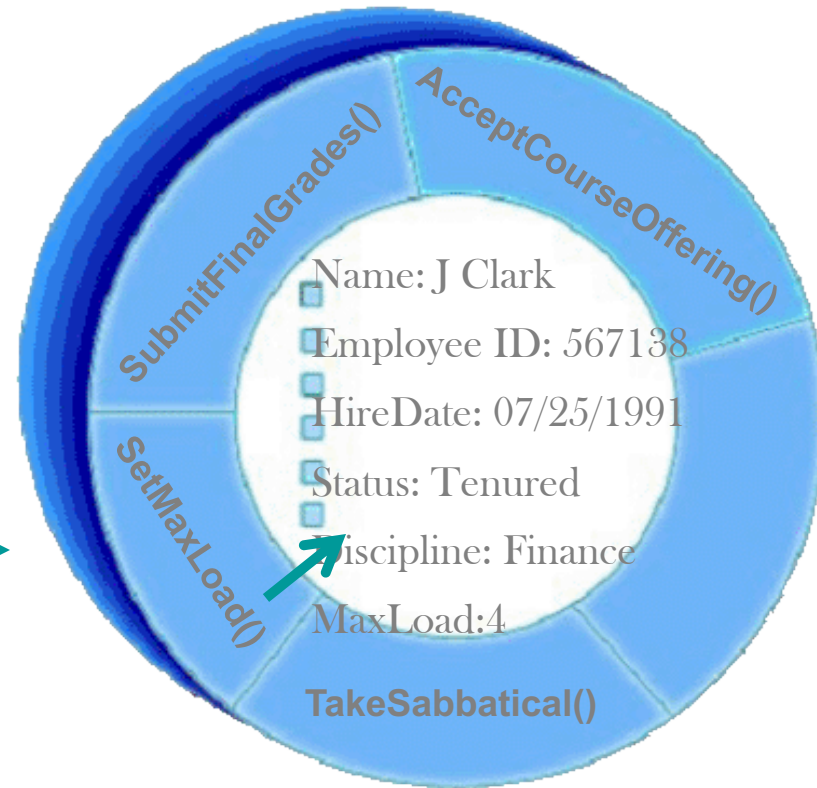
Improves Resiliency

Encapsulation Illustrated

Professor Clark

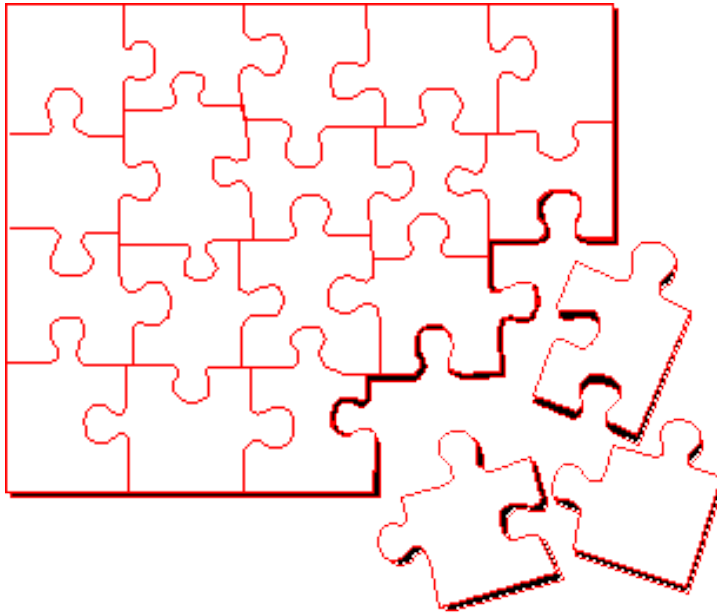
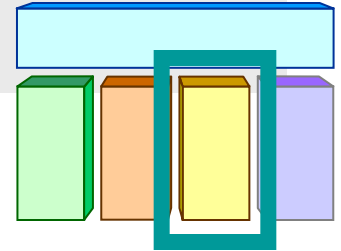
- Professor Clark needs to be able to teach four classes in the next semester.

SetMaxLoad(4)



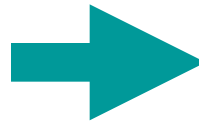
What Is Modularity?

- Breaks up something complex into manageable pieces.
- Helps people understand complex systems.

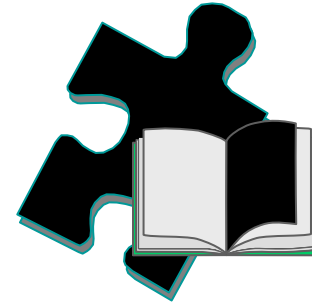


Example: Modularity

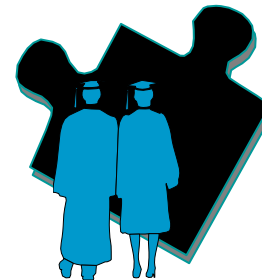
- For example, break complex systems into smaller modules.



**Billing
System**

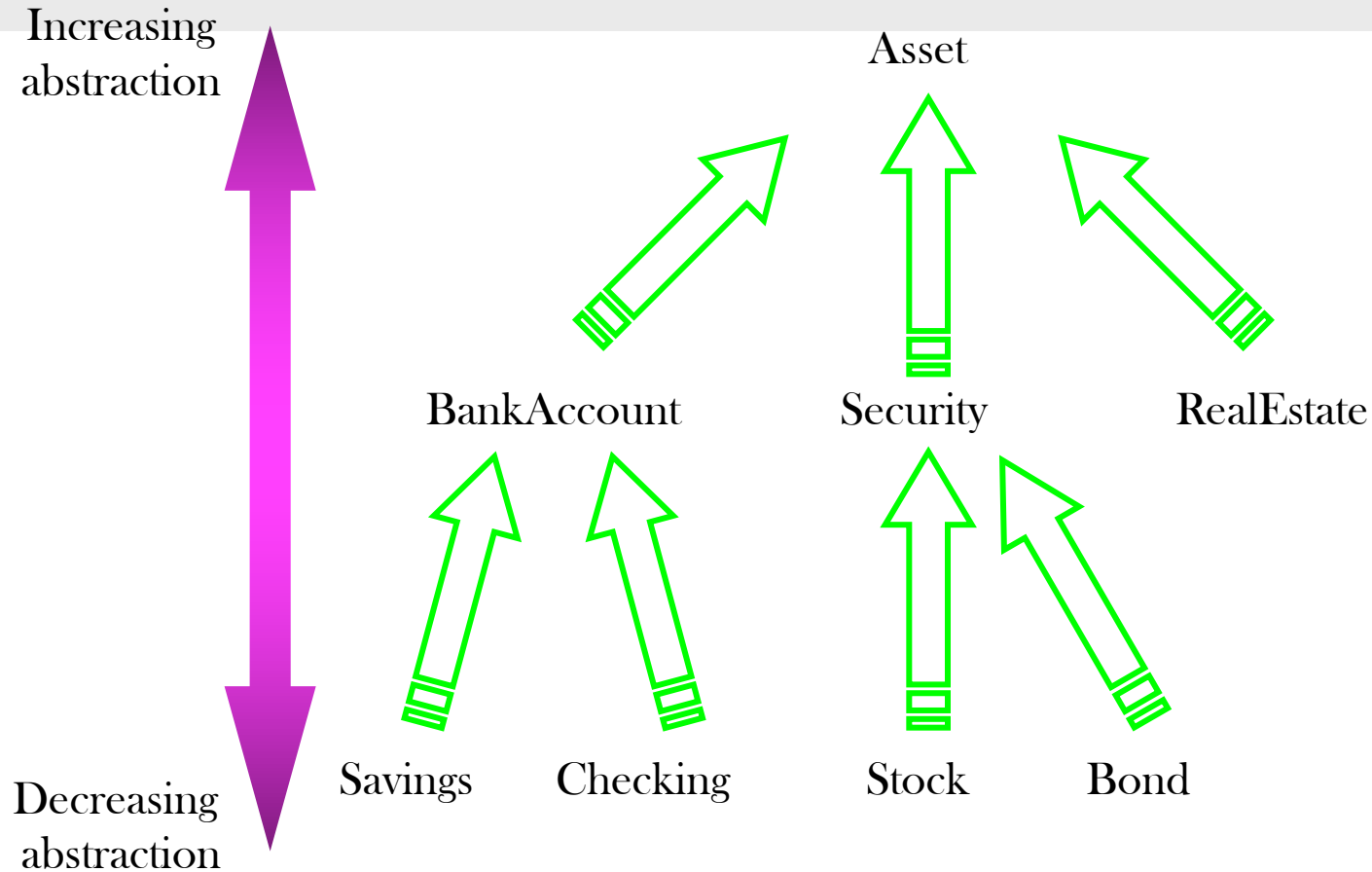


**Course
Catalog
System**



**Student
Management
System**

What Is Hierarchy?



Elements at the same level of the hierarchy should be at the same level of abstraction.

What Is Polymorphism?

- ♦ The ability to hide many different implementations behind a single interface.

