Systems and Software Modeling

Jesper Andersson

Linnæus University

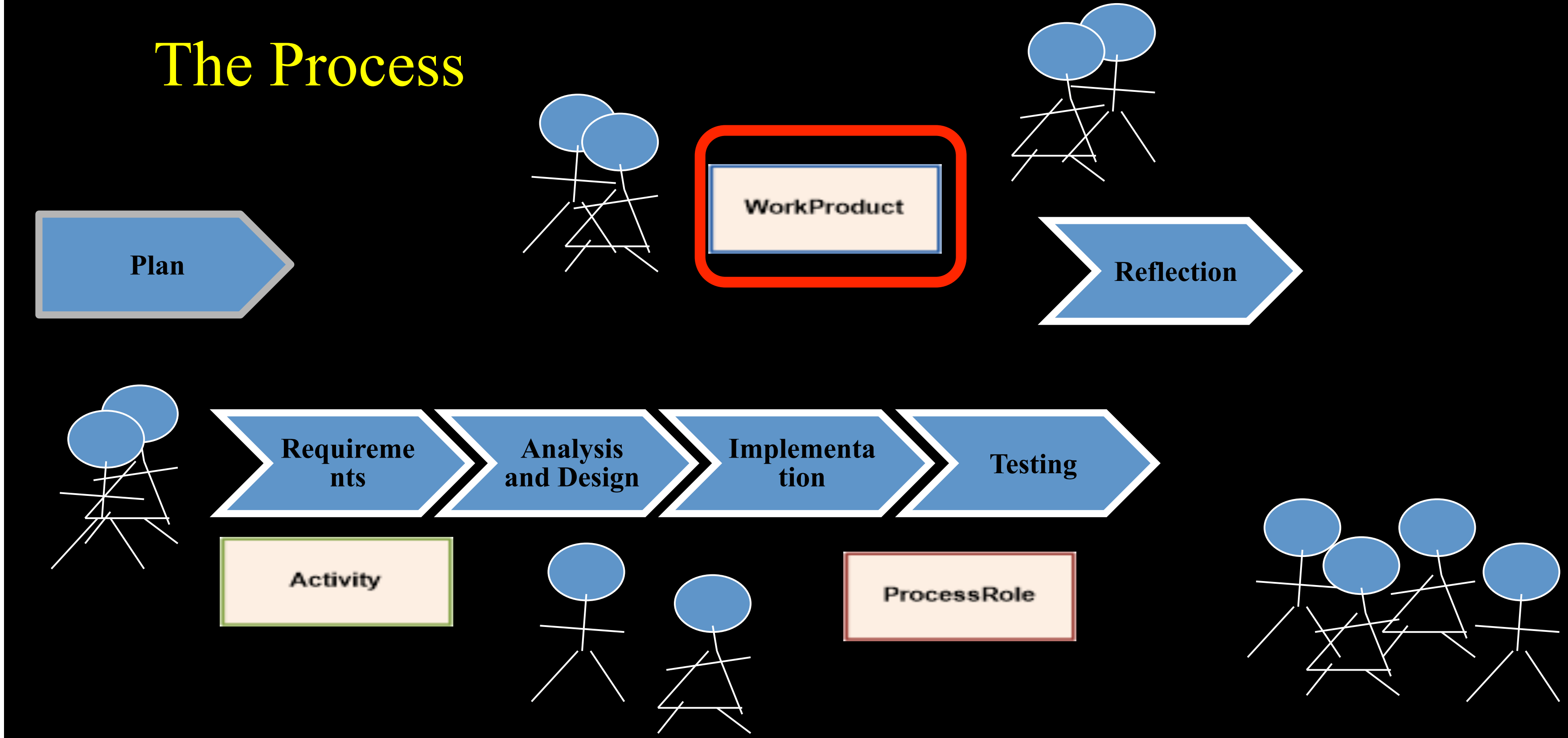
System and Software Design

- ✓ Characteristics
  - – Complex systems
  - – Invisible
  - – Group activity
  - – Always the “first time”
- ✓ Problems
  - – Align work
  - – Decompose for decentralization
  - – Coherence
  - – Conformance

Root cause?

Communication!

The Process



Plan

WorkProduct

Reflection

Requirements

Analysis and Design

Implementation

Testing

Activity

ProcessRole

Model

- ✓ A **model** is a theoretical construct that represents
  - **physical,**
  - **biological or**
  - **social processes,**

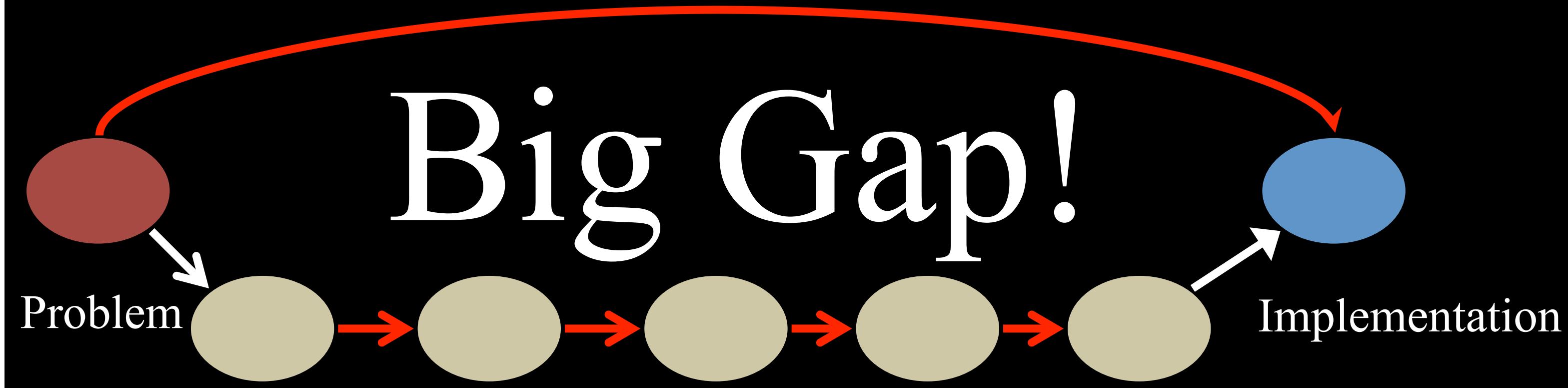
with a set of variables and a set of logical and quantitative relationships between them.

Models

- ✓ Models are **constructed** to **enable reasoning** within an idealized logical framework about these processes.
- ✓ **Idealized** means that the model may make explicit assumptions that are known to be false in some detail → Simplifications!.

The Stepwise Refinement Principle

...stepwise refinement can be viewed as a sequence of elaborations that result in the formation of a program in a target language from an initial function specification… **N. Wirth**



Big Gap!

Problem

Implementation

Properties of Good Models

✓ **Reduce** Complexity and **Remove** Uncertainty

✓ Complexity - We have to deal with more information than we may comprehend!

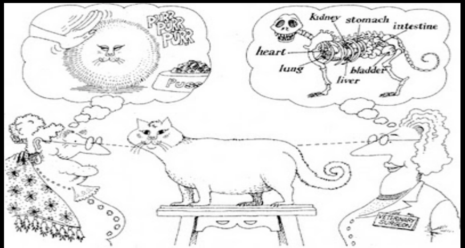
✓ Abstraction – Reduce information, FOCUS!

✓ Modularity – Divide models up

✓ Hierarchy – Structure models

✓ Information hiding – Encapsulate details

Abstractions



- ✓ Examples
  - – Object
  - – Class
  - – Interfaces
  - – Operation

"a ***simplified*** description, or specification, of a system that ***emphasizes some*** of the system's ***details*** or properties ***while suppressing others***.

A good abstraction is one that emphasizes details that are significant to the reader or user and suppresses details that are, at least for the moment, immaterial or diversionary." -- Shaw, M. 1984

Modularity

Decomposing a system in to its parts

- ✓ Logical or Physical modules

- ✓ Examples in Java
  - Classes (Logical)
  - Packages (Logical)
  - Files (Physical)

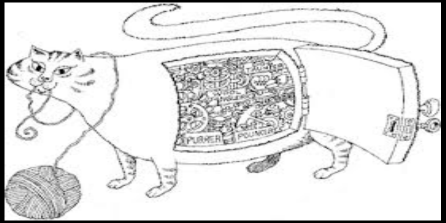
Hierarchy

✓ Compose subsystems into larger systems

TOY MICE
CLOCKWORK MOTORS
COMPONENT PARTS
ATOMS
ETC. ETC.

Hierarchy is the **ranking** or **ordering** of **abstractions**

Encapsulation – Information hiding



"the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; ***encapsulation serves to separate the contractual interface of an abstraction and its implementation.***"

- ✓ Examples
  - Class interface in Java
    - Attributes
    - Operations
  - Access modifiers (Java)

Software - Models

✓ All software systems contain models of the real-world

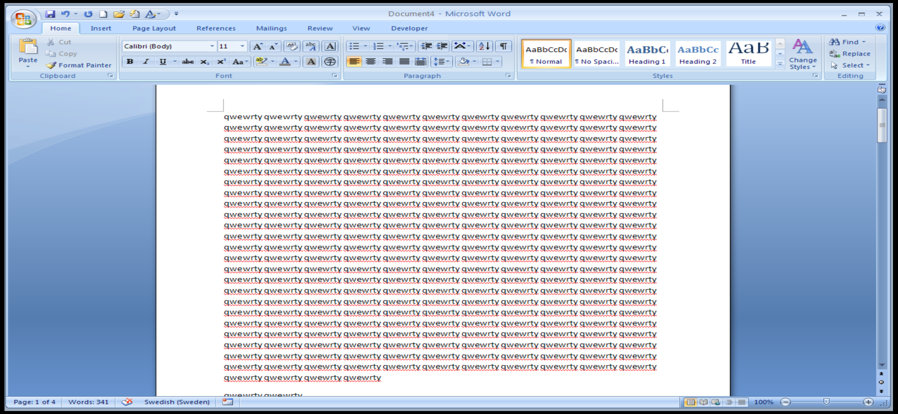
✓ Examples

- Games
- Control Software
- Fly-by-wire
- Information systems

Models? – Word Processor

typesetter

typewriter



qwewrty qwewrty

qwewrty qwewrty

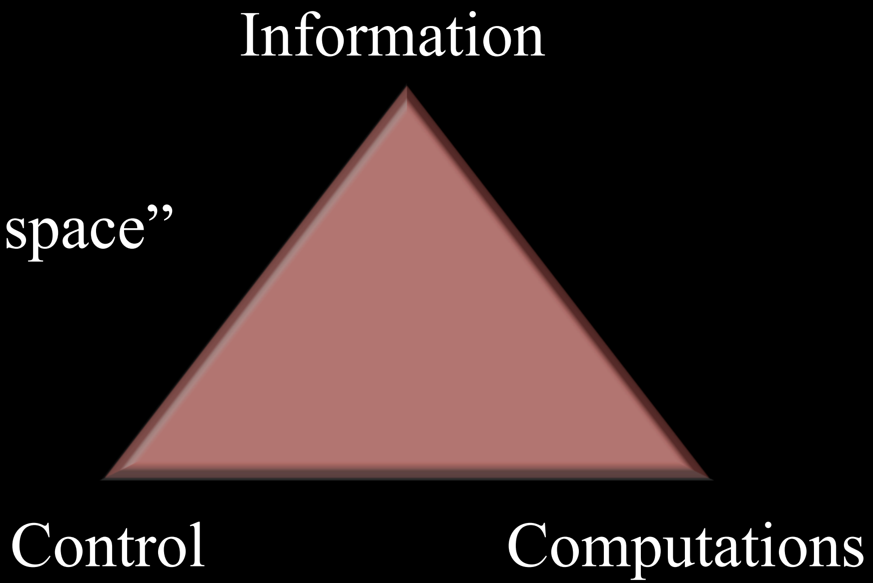
paper

dictionary

proofreader

Software Systems

“the system space”



Information

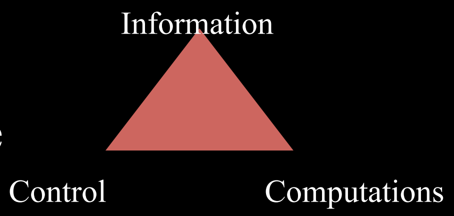
Control

Computations

Models – Views

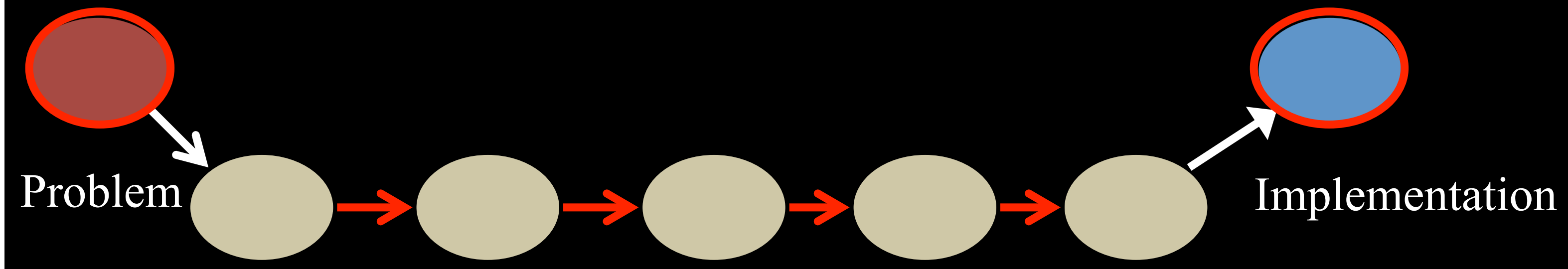
- ✓ The Computer System Space
- ✓ A software system must be described using several different models!
- ✓ Why do we need views?

- Different stakeholders
- Focus on specific details
- Does not require the big picture



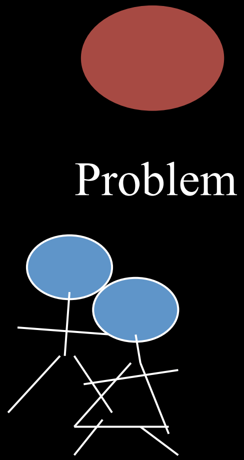
Development

- ✓ Every piece of behavior in system must be provided for, in a sensible way
- ✓ We must model the different aspects for the technology we choose for our project.

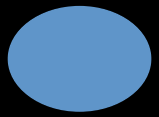


Model of the Problem

- ✓ Capture a teams understanding of the problem
- ✓ Two categories
  - Directed towards end-users
  - Directed towards developers
- ✓ Properties:
  - Understandability, expressiveness
  - Precision, Verifiability



Model of Implementations

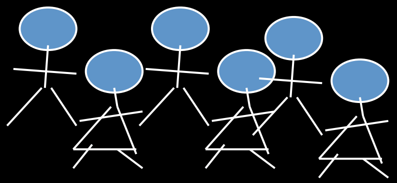


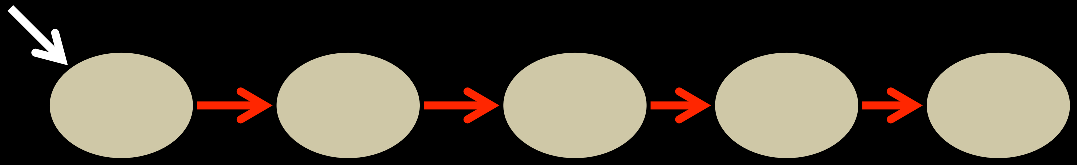
Implementation

- ✓ Programming Languages
- ✓ Other specification languages
  - – Configuration languages
  - – DBMS languages
  - – Build and Deployment scripts
- ✓ Properties:
  - – Precise
  - – Transformation

Models while in Transition

- ✓ Some oriented towards understanding a problem
- ✓ Some oriented towards a solution
- ✓ Purpose and Target group
  - – Conceptual models
  - – Physical models
  - – Static
  - – Dynamic





Linnæus University
Sweden

Model – Views

- ✓ All models depict elements and their relationships
- ✓ Purpose
  - Static – Does not depict any change!
  - Dynamic – Illustrates change!
  - Conceptual – A model which main use is reasoning and decisions.
  - Physical – A model which models physical, real, entities

Conceptual vs. Physical

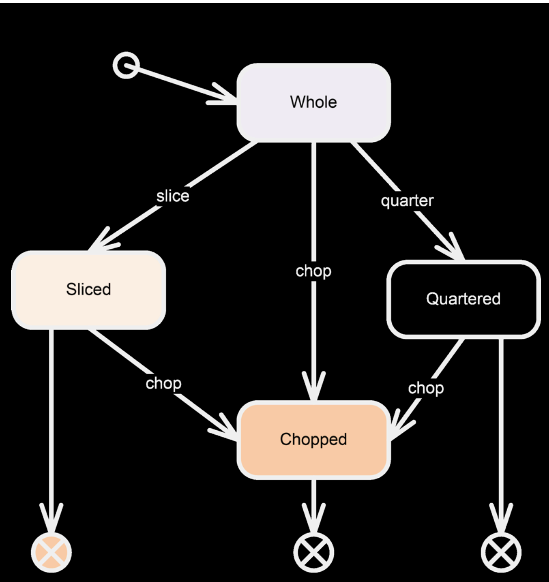
Tomato

Knife

Cut Board

```
theCutBoard.put(theTomato);
theKnife.chop(theCutBoard.getItems());
```

Static vs. Dynamic

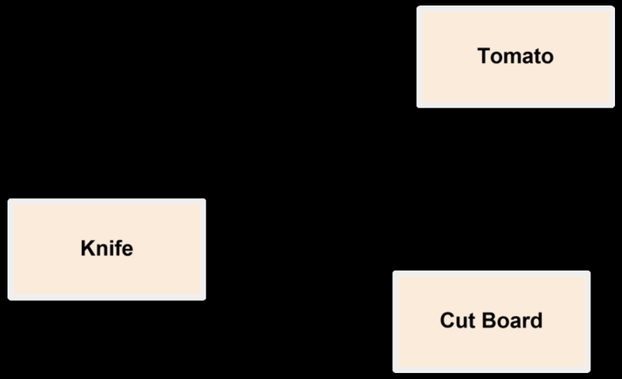


```
class Knife {

   private float length;
   private Manufaturer make;
   …

   public void chop() { Collection<IChoppable> objects }
   public void stab() { IStabbable object …}
   public void slice() { ISliceable object …}
   …
}
```

Linnæus University
Sweden

Conceptual – Static

- ✓ What do we have!
- ✓ Which objects are used to
  - Describe a problem
  - Describe a solution?


Tomato

Knife

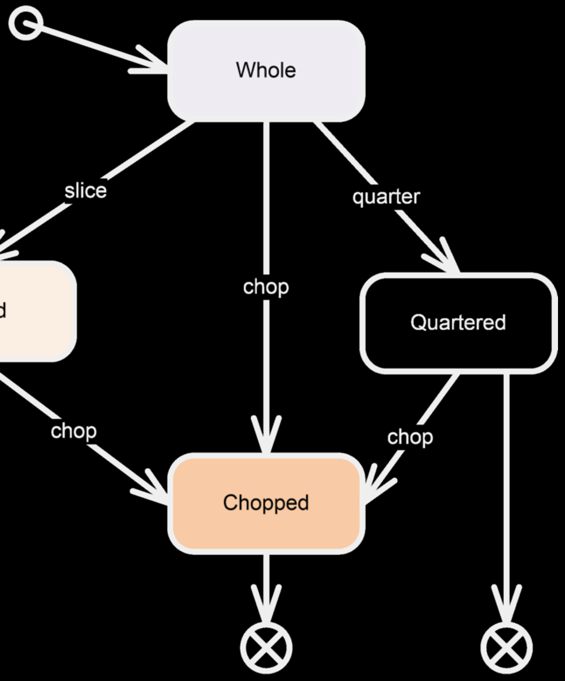
Cut Board

- ✓ Does not change at runtime!

Conceptual – Dynamic

- ✓ What is happening in a system?
- ✓ What happens in
  - – A problem
  - – A solution

- ✓ Describes change



Linnæus University
Sweden

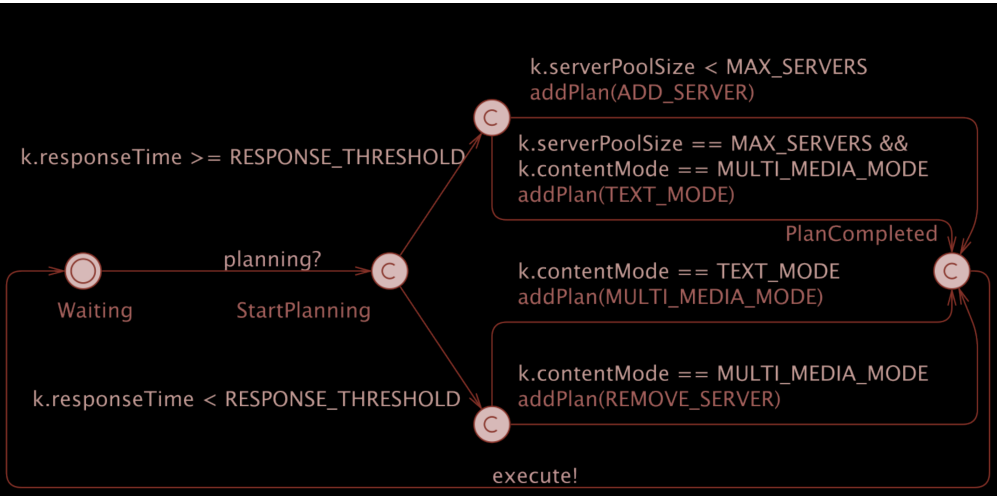
How are models expressed?

- ✓ Natural language
- ✓ Specific modeling languages
  - Formal
  - Semi-formal
  - Graphical

Formal Models

- ✓ Precise models
- ✓ Syntax and Semantics
- ✓ Mathematical foundation

- ✓ Used for
  - – Exact transformation
  - – Verification


k.serverPoolSize < MAX_SERVERS
addPlan(ADD_SERVER)
k.responseTime >= RESPONSE_THRESHOLD
k.serverPoolSize == MAX_SERVERS &&
k.contentMode == MULTI_MEDIA_MODE
addPlan(TEXT_MODE)
PlanCompleted
planning?
Waiting
StartPlanning
k.contentMode == TEXT_MODE
addPlan(MULTI_MEDIA_MODE)
k.contentMode == MULTI_MEDIA_MODE
addPlan(REMOVE_SERVER)
k.responseTime < RESPONSE_THRESHOLD
execute!
C
C
C
C

UML

- ✓ UML is a modeling language to express and design documents, software
  - Particularly useful for OO design
  - Not a process!
  - Independent of implementation language
- ✓ Combines techniques from various domains (views!!)
  - Data modeling (ER- Diagrams)
  - Business modeling (Work flows)
  - Object modeling
  - Component modeling

UML

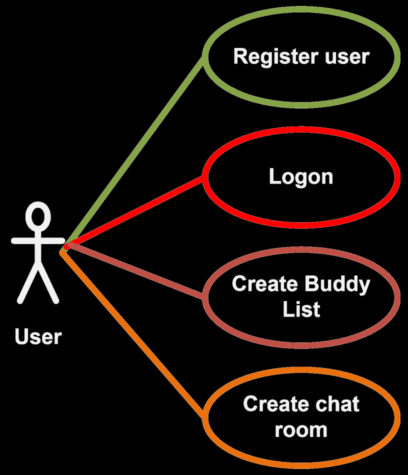
- ✓ Standardized graphical notation for
  - – Specifying, visualizing, constructing, and documenting software systems
- ✓ Language can be used from general **initial** design to very specific **detailed** design
- ✓ Increase understanding/communication of product to customers and developers
- ✓ Support for UML in many software IDEs

UML - Models

- ✓ **Functional** – Depicts the functionality of the system from the user's Point of View. Includes Use Case Diagrams
- ✓ **Object** – Captures the structure and substructure of the system using objects, attributes, operations, and associations. Includes Class Diagrams.
- ✓ **Dynamic** – Demonstrates the system’s internal behavior. Includes Sequence Diagrams, Activity Diagrams and Statechart Diagrams.

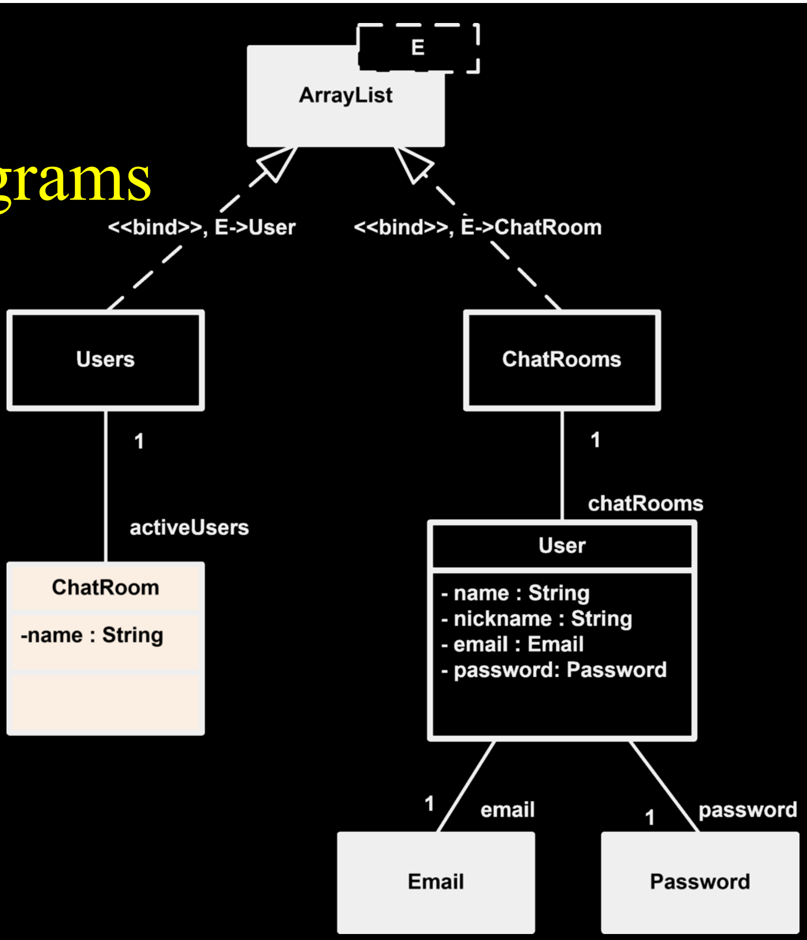
Functional Model – Diagrams

- ✓ Use Case Diagram - Shows use cases, actors, and their interrelationships.



Object Model – Diagrams

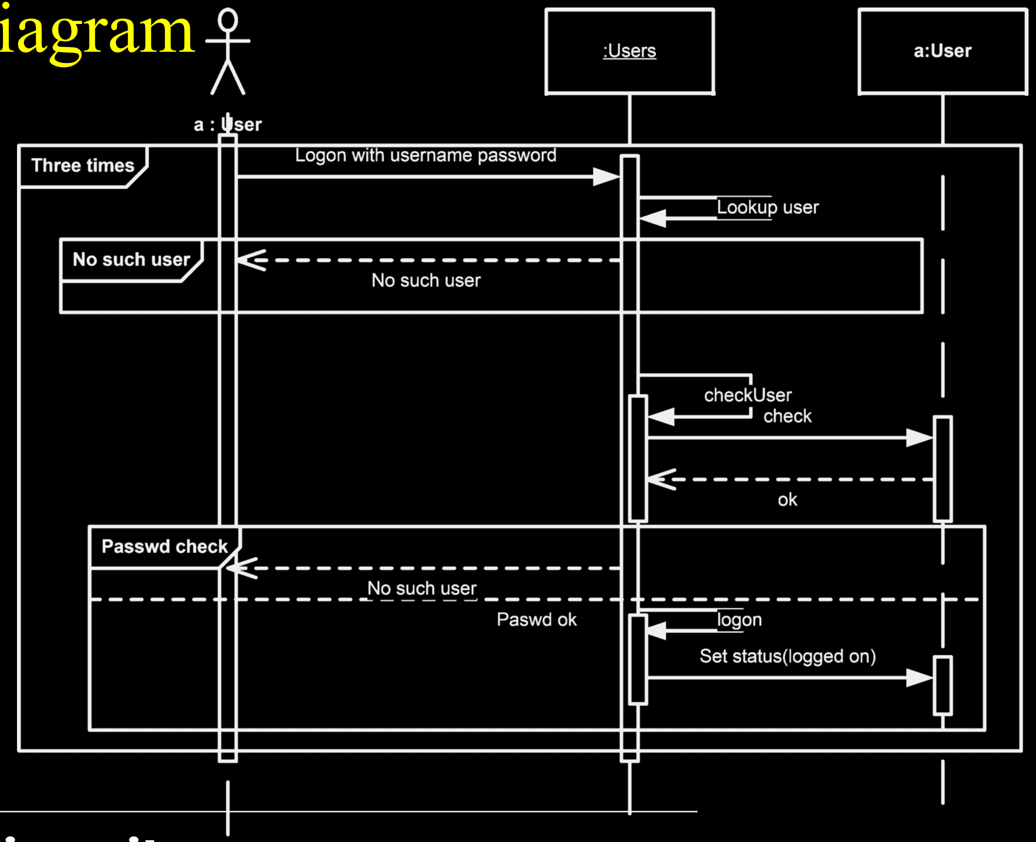
- ✓ Class Diagram - Shows a collection of static model elements such as classes, interfaces, and types, their contents, and their relationships.



Dynamic Model - Diagrams

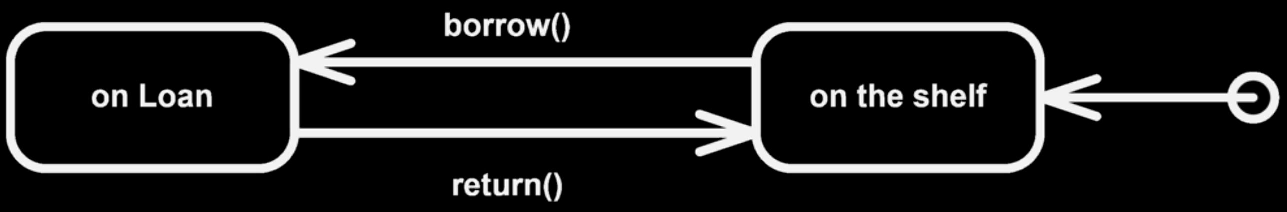
- ✓ **Sequence** - Models the sequential logic, in effect the time ordering of messages between classifiers. See UML Sequence diagram guidelines.
- ✓ **State chart** - Describes the states an object or interaction may be in, as well as the transitions between states. Formerly referred to as a state diagram, state chart diagram, or a state-transition diagram. See UML State chart diagram guidelines.
- ✓ **Communication** – 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. See UML Collaboration diagram guidelines.
- ✓ **Activity** - Depicts high-level business processes, including data flow, or to model the logic of complex logic within a system. See UML Activity diagram guidelines.

Sequence Diagram



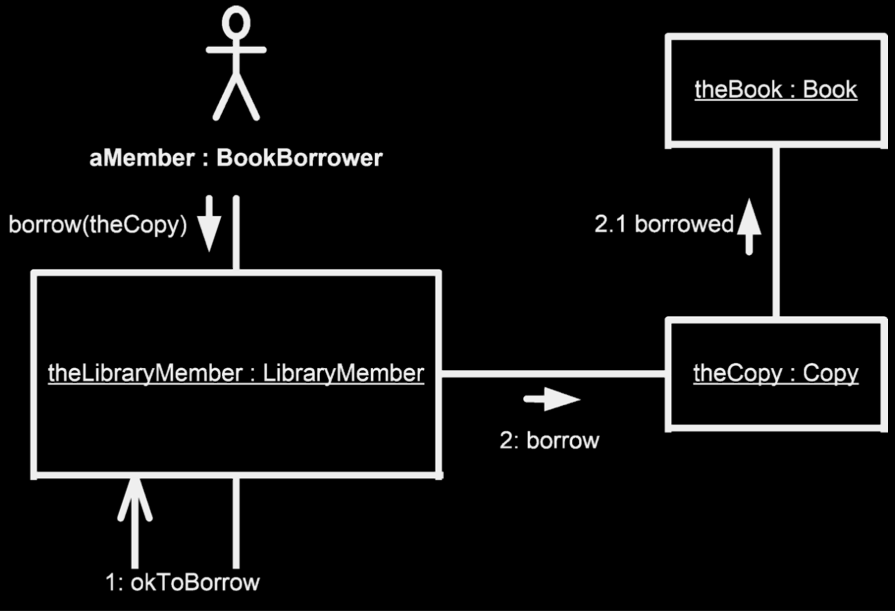
State Chart Diagram

- ✓ Depicts the states an object may be in and the transitions



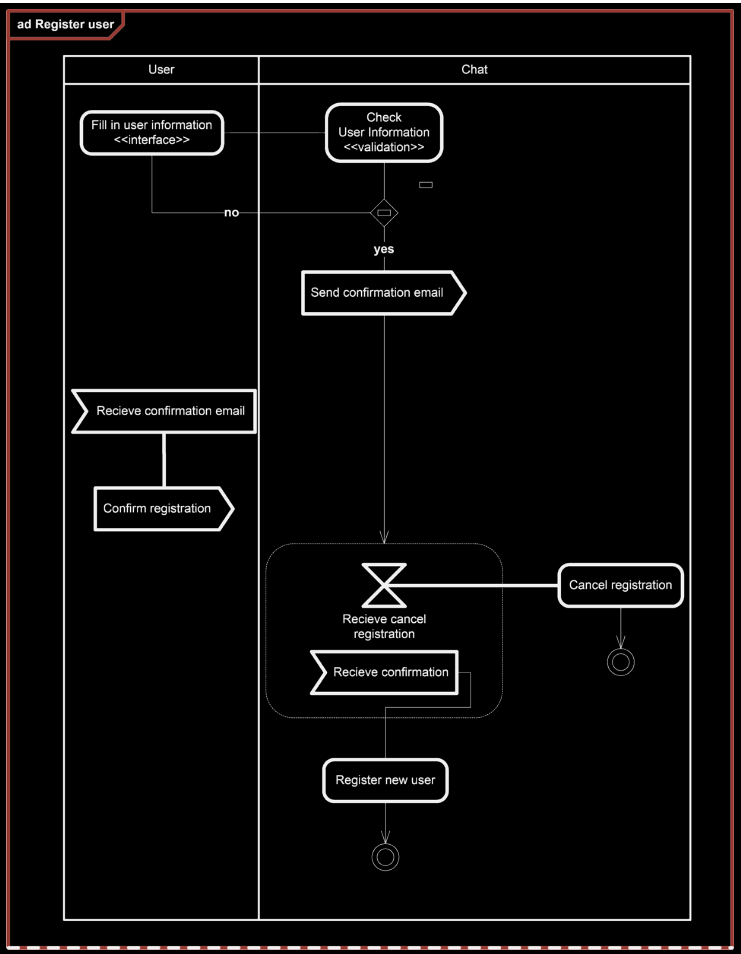
Communication Diagram

- ✓ Shows message flow between objects



Activity Diagram

- ✓ Models a workflow
  - Participants
  - Activities

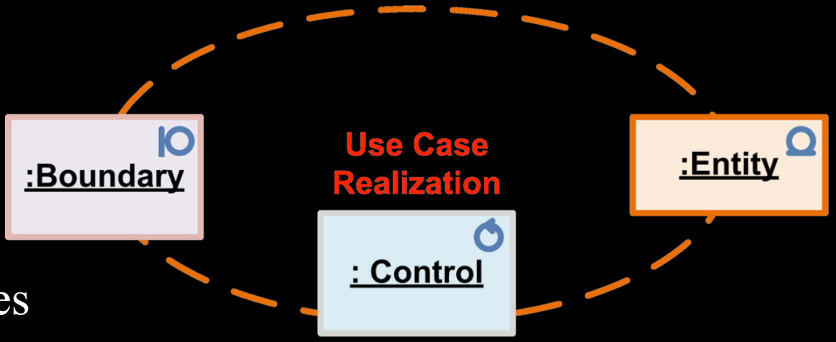


Linnæus University
Sweden

Additional UML Diagrams

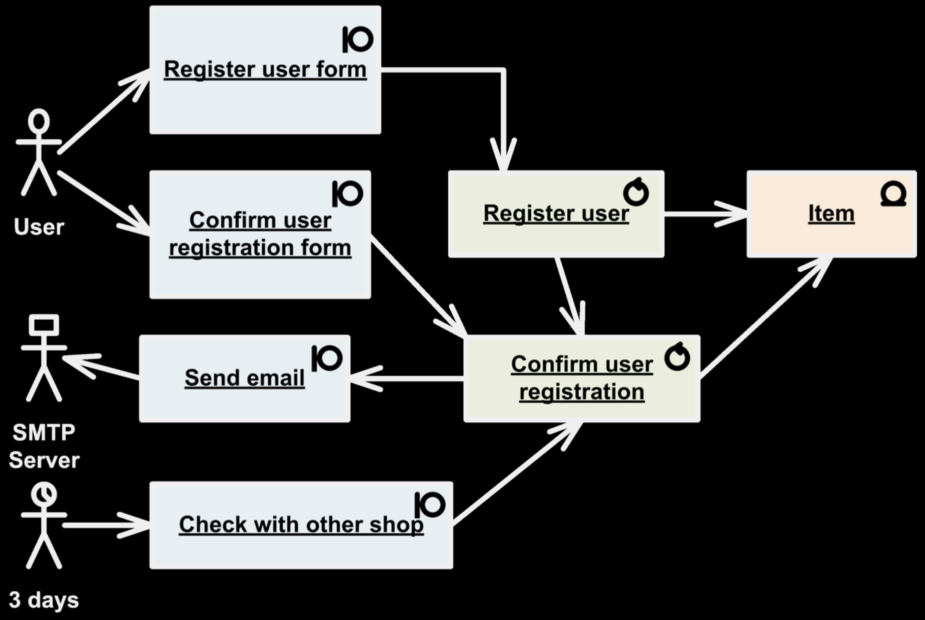
- ✓ Component – the components that compose an application, system, or enterprise, their interrelationships, interactions, and their interfaces are depicted.
- ✓ Composite structure - the internal structure of a classifier, including the interaction points to other parts of the system.
- ✓ Deployment – the execution architecture of systems.
- ✓ Interaction overview - A variant of an activity diagram which overviews the control flow within a system or business process.
- ✓ Object – objects and their relationships at a point in time, typically a special case of either a class diagram or a communication diagram.
- ✓ Package – Shows how model elements are organized into packages as well as the dependencies between packages.
- ✓ Timing – 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.

Robustness Diagrams



- ✓ Outlines a solution
- ✓ Combines elements of three types
  - Boundary
  - Control
  - Entity
- ✓ A robustness diagram is basically a simplified UML communication/collaboration diagram using ‘stereotyped objects’

Example:


Register user form
User
Confirm user registration form
Register user
Item
SMTP Server
Send email
Confirm user registration
3 days
Check with other shop

Today’s takeaways

- ✓ We create models for
  - Different purposes
  - Different target stakeholder
  - Different degrees of formality
- ✓ The models constitute our design language

Next Lecture

- ✓ Focus on Requirements models
- ✓ Requirements elicitation

- ✓ Use Case models
- ✓ Robustness models