Software Engineering and ModelLing

Jörg Kienzle

Overview

- What is Software Engineering?
- Software Development Processes
- Software Development Phases
- Model-Driven Engineering
- Object-Orientation
- UML
- Overview of Process we will use
  - Background on Fondue, Fusion, etc...
  - Overview of Models that we will build

What is Software Engineering

- Software:
  - Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system

According to the IEEE

- Software Engineering:
  - The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software
  - In other words: the application of engineering to software

Increasing Complexity

- Scope, complexity and pervasiveness of computer-based and controlled systems continue to increase
- Software assumes more and more responsibility

Challenges for SE

- 1960's: Cope with software correctness
  - Milestone: Floyd 'assigning meaning to programs'
- 1970's: Cope with project size
  - Milestone: Parnas, Yourdon: modularity & structure
- 1980's: Cope with variability in requirements
  - Milestone: Jackson, Meyer: modeling, object orientation
- 1990's: Cope with distributed systems and mass deployment:
  - Milestone: Szyperski: product-lines & components
- 2000's: pervasive software integration, system of systems, accelerating technological changes
  - Milestone: ?

Long-Term Availability

- AIRBUS A300 Life Cycle
  - Program began in 1972, production stopped in 2007
  - 2007-1972 = 35 years...
  - Support will last until 2050
    - 2050-1972 = 78 years !!

Dependability

- Consequences of systems failing
  - Annoying to catastrophic
  - Opportunities lost, businesses failed, security breaches, systems destroyed, lives lost

On June 4, 1996 an Ariane V rocket launched by the European Space Agency exploded just forty seconds after lift-off

Software Development Process

- A well-defined and well-managed software engineering process
  - Provides guidance as to the order of a team's activities,
  - Specifies what artifacts should be developed,
  - Directs the tasks of individual developers and the team as a whole, and
  - Offers criteria for monitoring and measuring a project's products and activities.

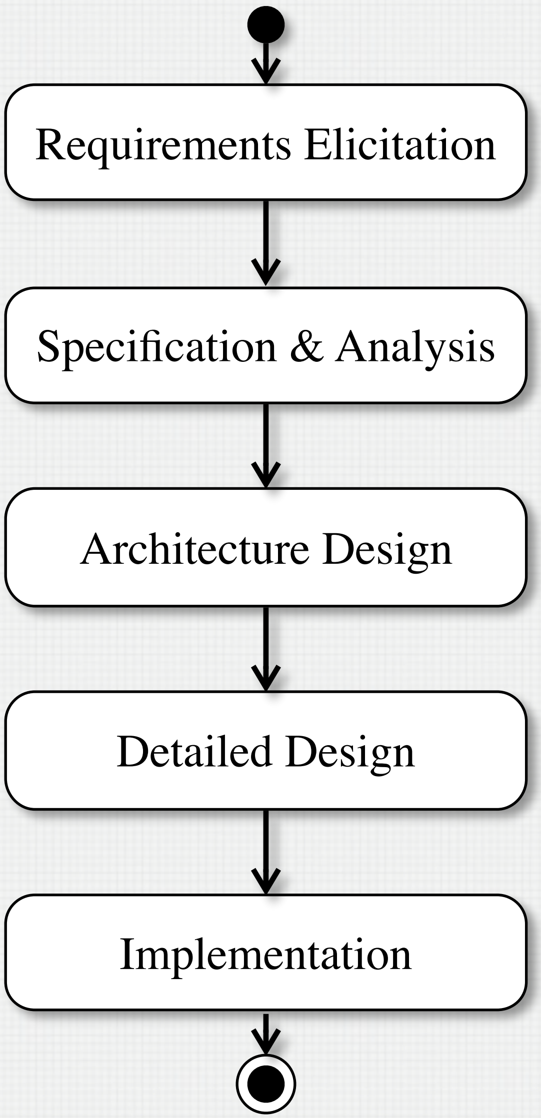
Software Process Activities

- Primary activities
  - Development
  - Operation
  - Maintenance
- Supporting activities
  - Documentation
  - Configuration management
  - Quality assurance
  - Verification and validation
  - Training
- Process-related activities
  - Management
  - Infrastructure
  - Tailoring
  - Process assessment

Systematic guidance on how to do this is called a development method

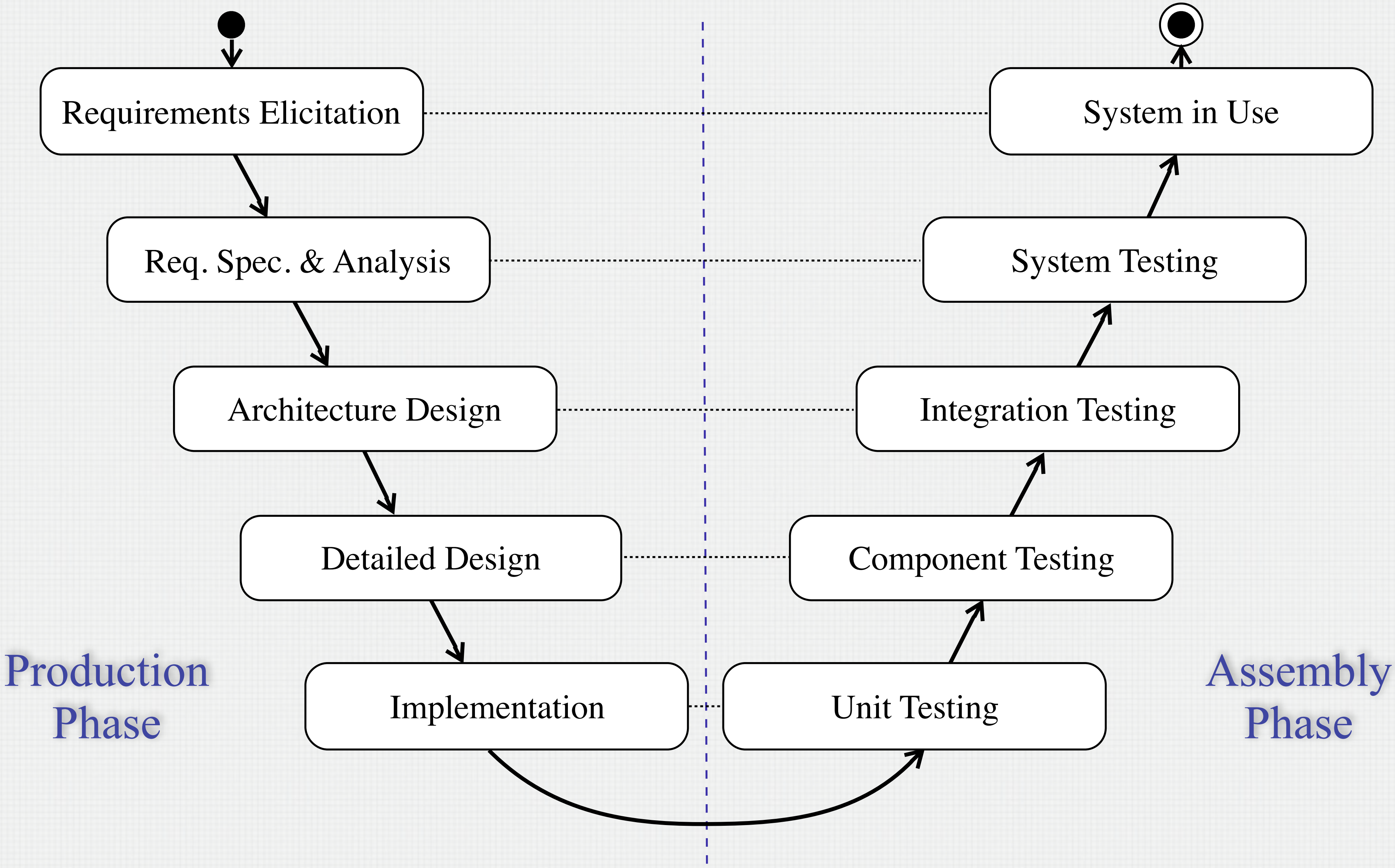
We will focus on this!

Classic Waterfall Model



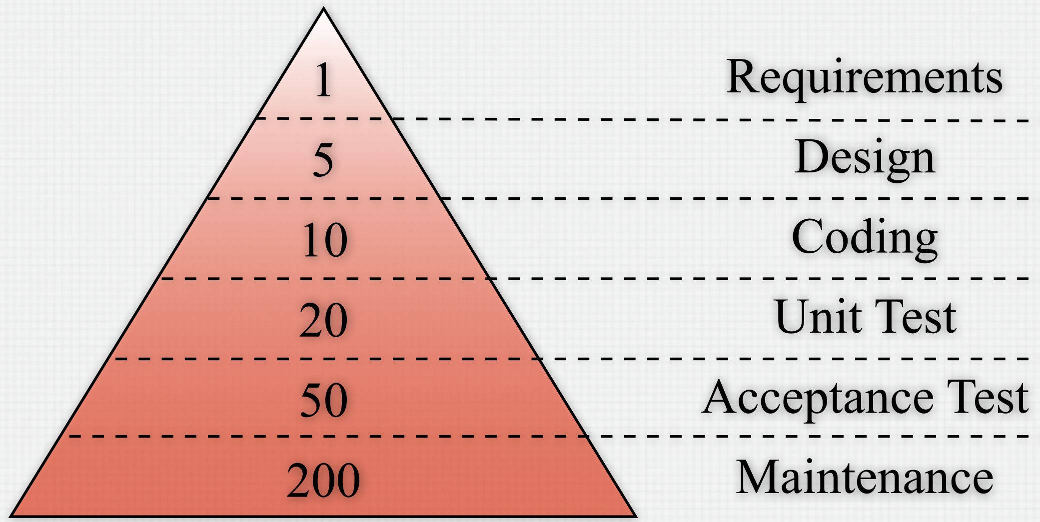
Why?	Discover functionality / user expectations Non-functional requirements / qualities
What?	Produce a complete / unambiguous description of the problem domain and the requirements
How?	Elaborate a system architecture that fulfills the requirements
How exactly?	Allocate responsibilities to modules Design how functionality is achieved at run-time
	Elaborate low-level algorithms Implement your modules

Software Development V Model



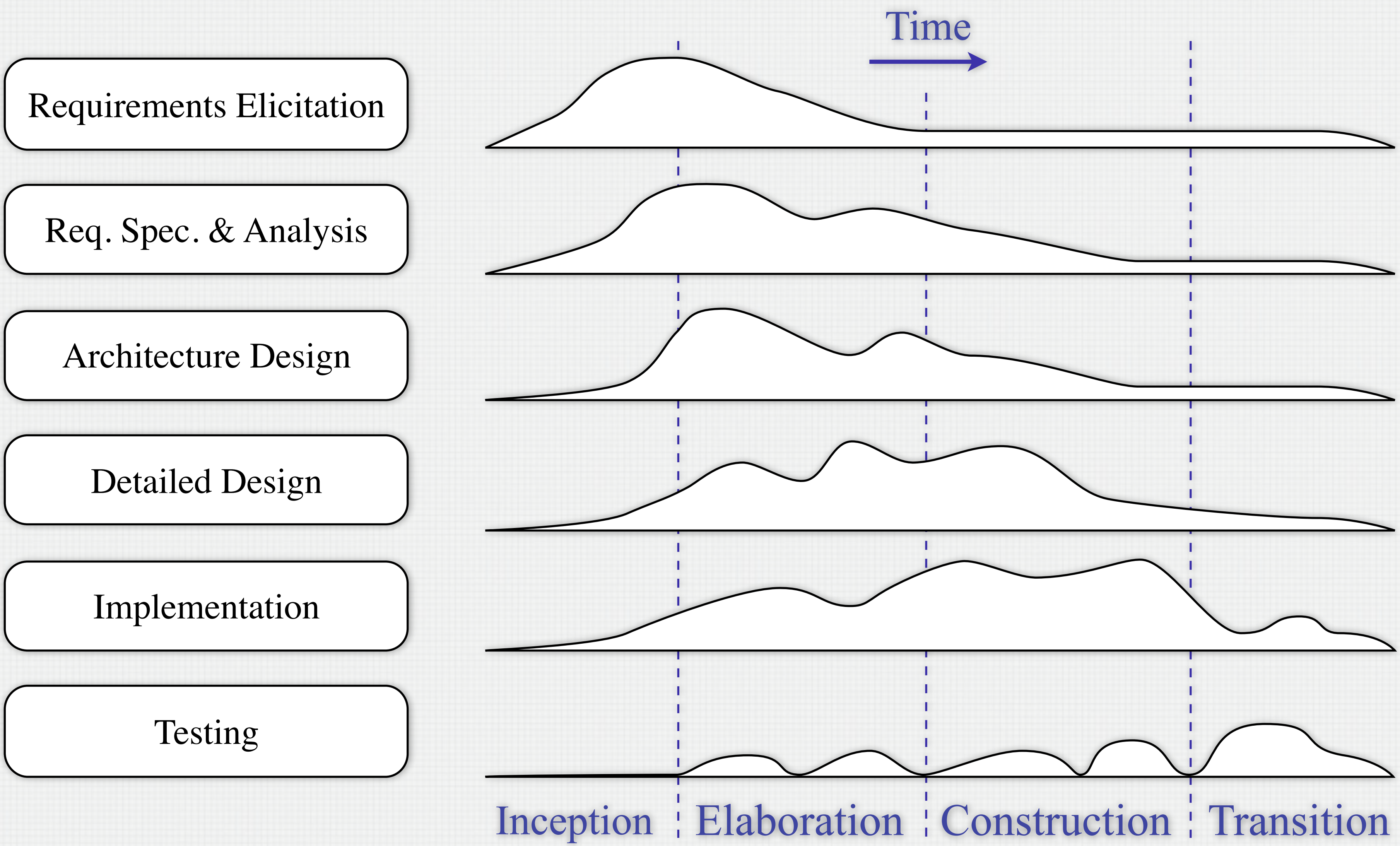
Importance of "Good" Requirements

- Faults / omissions made at the requirements stage are expensive to fix later
  - Stated requirements might be implemented, but the system is not one that the customer wants
- Need to determine and establish the precise expectations of the customer!

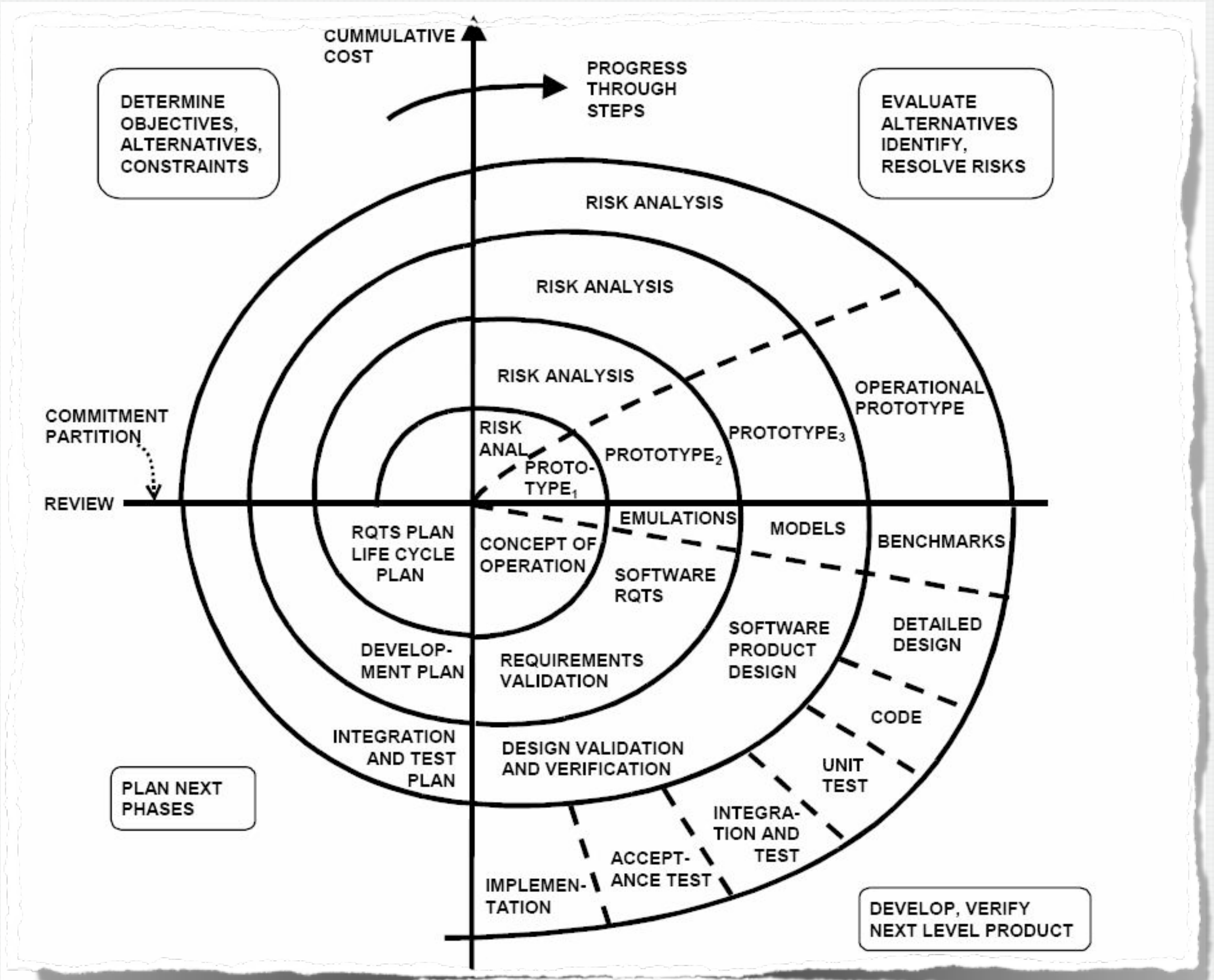


Relative Cost to Repair a Defect
at Different Lifecycle Phases [Davis 93]

Iterative Software Development



Spiral Methodology



Why Modelling?

"Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose."

Jeff Rothenberg
The Nature of Modeling
John Wiley & Sons, August 1989

Why Modelling?

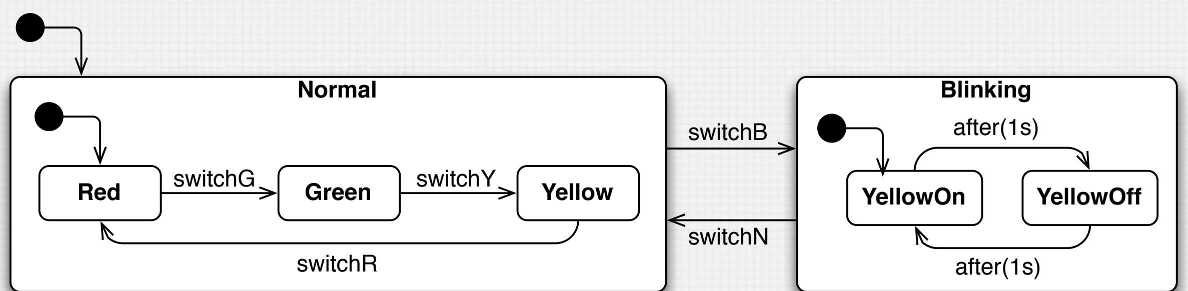
"A model represents reality for a given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality."

Jeff Rothenberg
The Nature of Modeling
John Wiley & Sons, August 1989

Modelling and Software Development

- A Model is a simplified representation of an aspect of the world for a specific purpose
- We use models to better understand a system
  - For a observer A, M is a model of an object O, if M helps A to answer questions about O. (Minsky)
- A model helps to understand, communicate and build
- Modelling and engineering: model something not yet existing!

M1 - Modelling Space

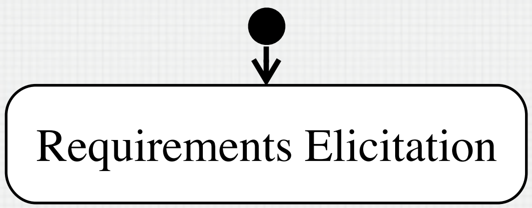


M0 - The World

Model-Driven Engineering

- A unified conceptual framework in which software development is seen as a process of model production, refinement and integration
- Models are at the centre of the development activities
  - Models are built representing different views of a software system using different formalisms, i.e., modelling languages, at different levels of abstraction, for different purpose
  - Models are connected by model transformations
    - High-level specification models are refined / combined / transformed to include more solution details until the produced models can be executed
- Tools are of major importance to effectively create, manipulate and transform models

Example Models

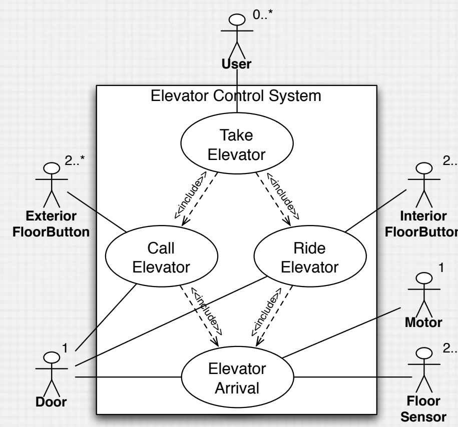


Why?

Generated Artifacts

Text Documents

CRC Cards



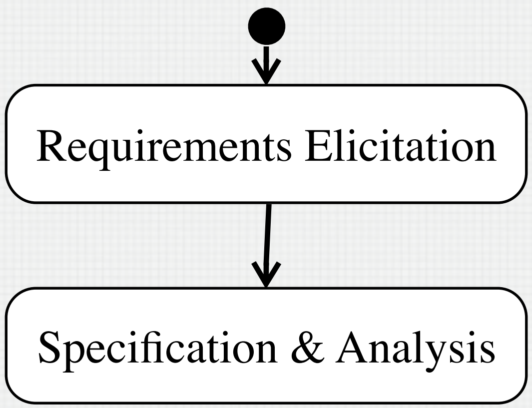
Use Case Diagram

Use Case: TakeElevator
Scope: Elevator Control System
Primary Actor: User
Intention: The intention of the *User* is to take the elevator to go to a destination floor.
Level: User Goal
Main Success Scenario:
1. *User* **Call[s]Elevator**
2. *User* **Ride[s]Elevator**

Extensions:
1a. Cabin is already at *User's* floor…
1b. *User* is already inside…

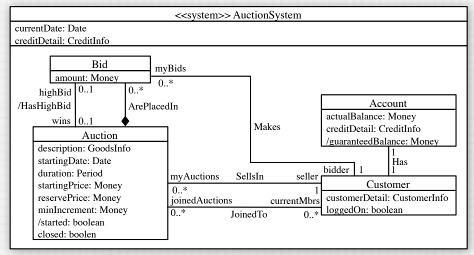
Use Cases

Example Models

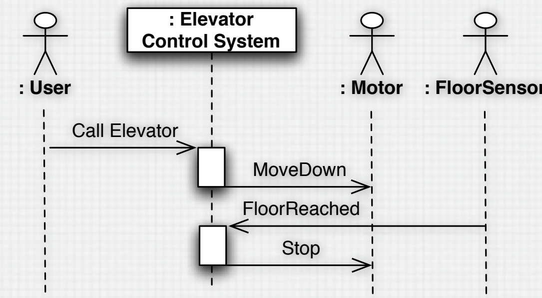


What?

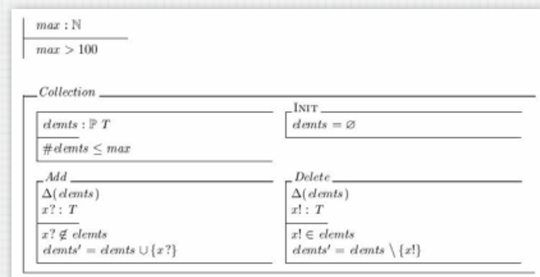
Generated Artifacts



Concept Model

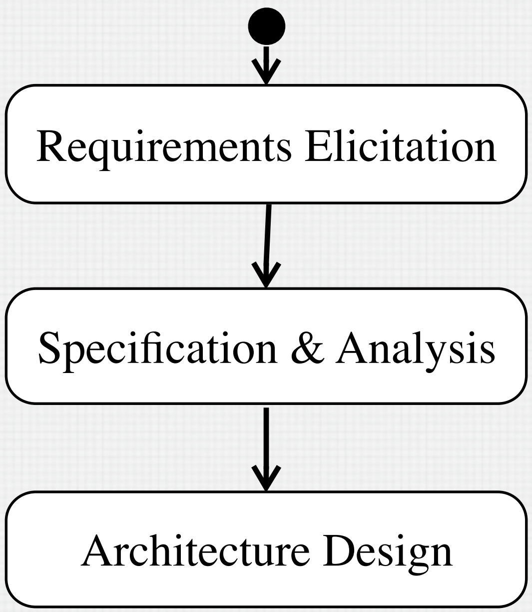


Environment Model



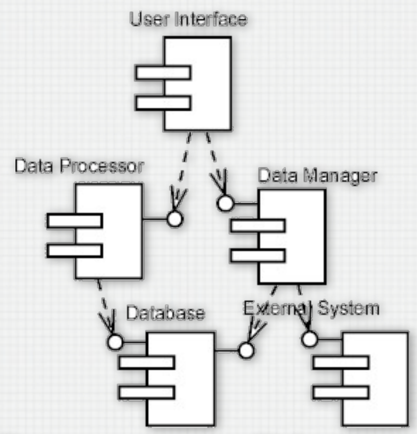
Z or B or OCL Specification

Example Models

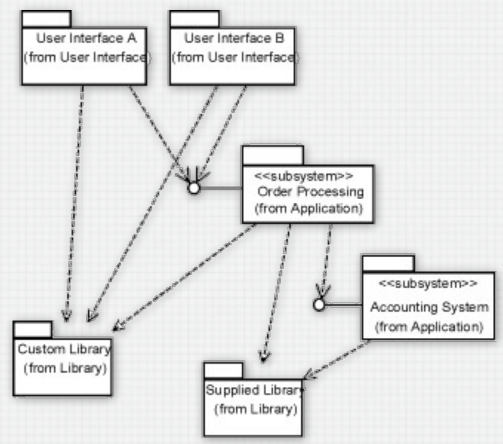


How?

Generated Artifacts



Component Diagram

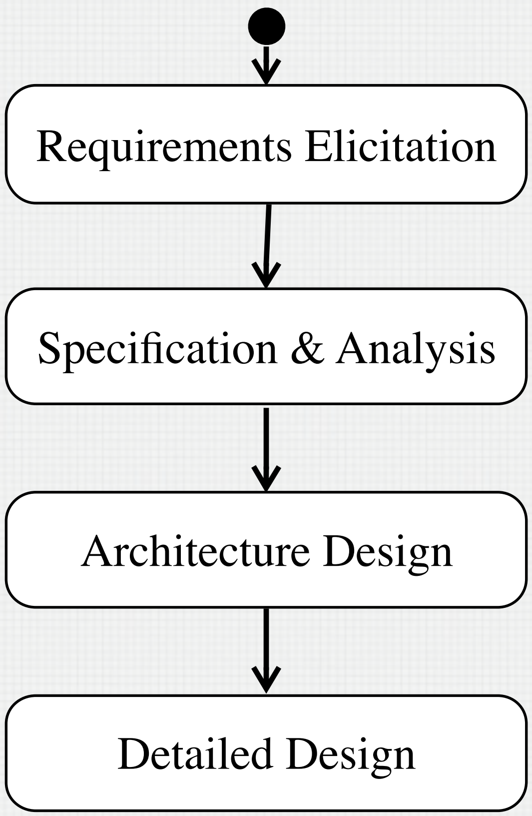


Package Diagram

```
archInstance {

  componentInstance {
    description = "Component 1"
    interfaceInstance{...}
  }

  connectorInstance{
    description = "Component 1"
    interfaceInstance{...}
  }

  linkInstance{
    description = "Comp1 to
     Conn1 Link"
    point {
      (link) anchorOnInterface =
     "#comp1.IFACE_BOTTOM"
    }
  }
}
```

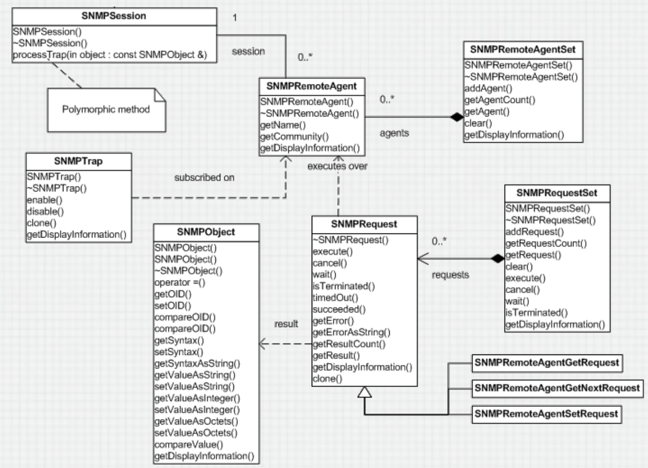
ADL Code

Example Models

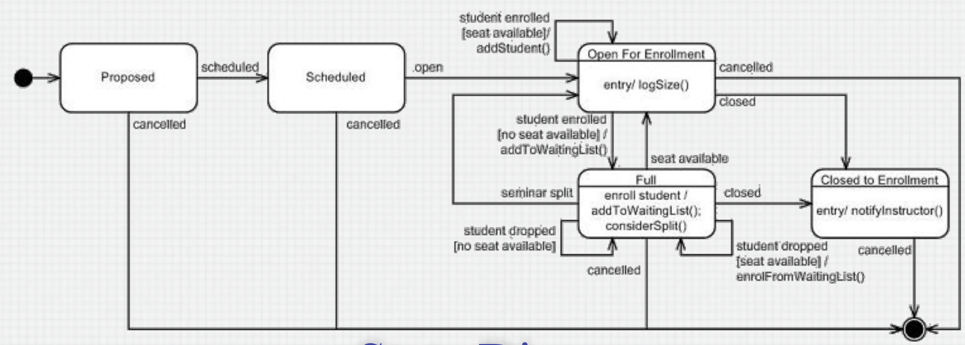


How exactly?

Generated Artifacts

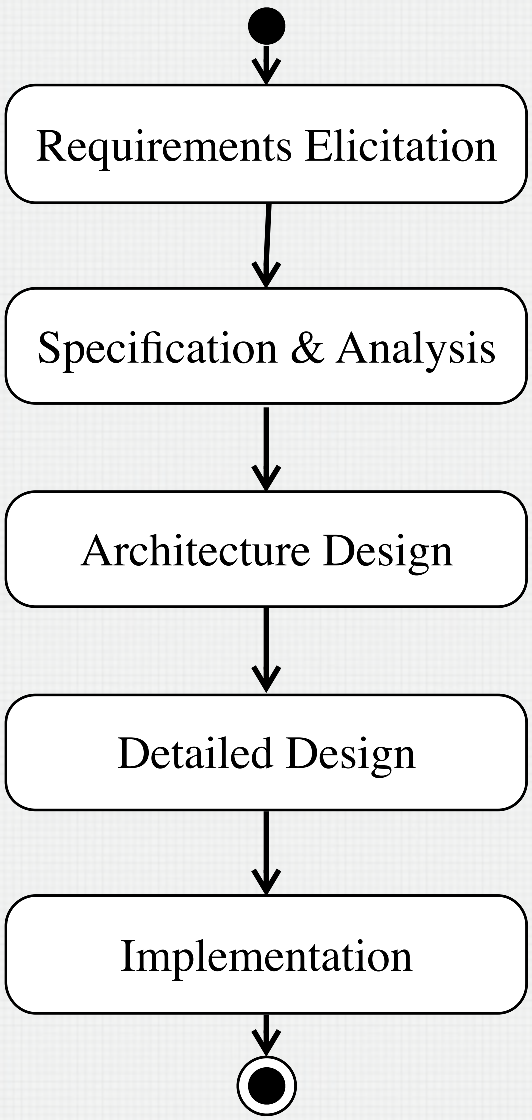


Class Diagram

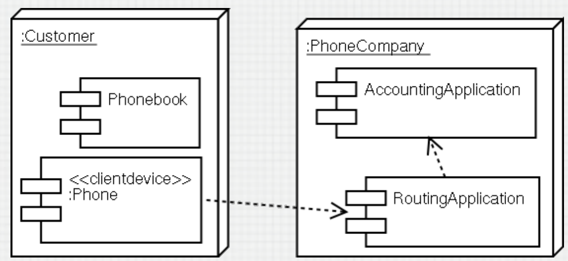


State Diagram

Example Models



Generated Artifacts



Deployment Diagram

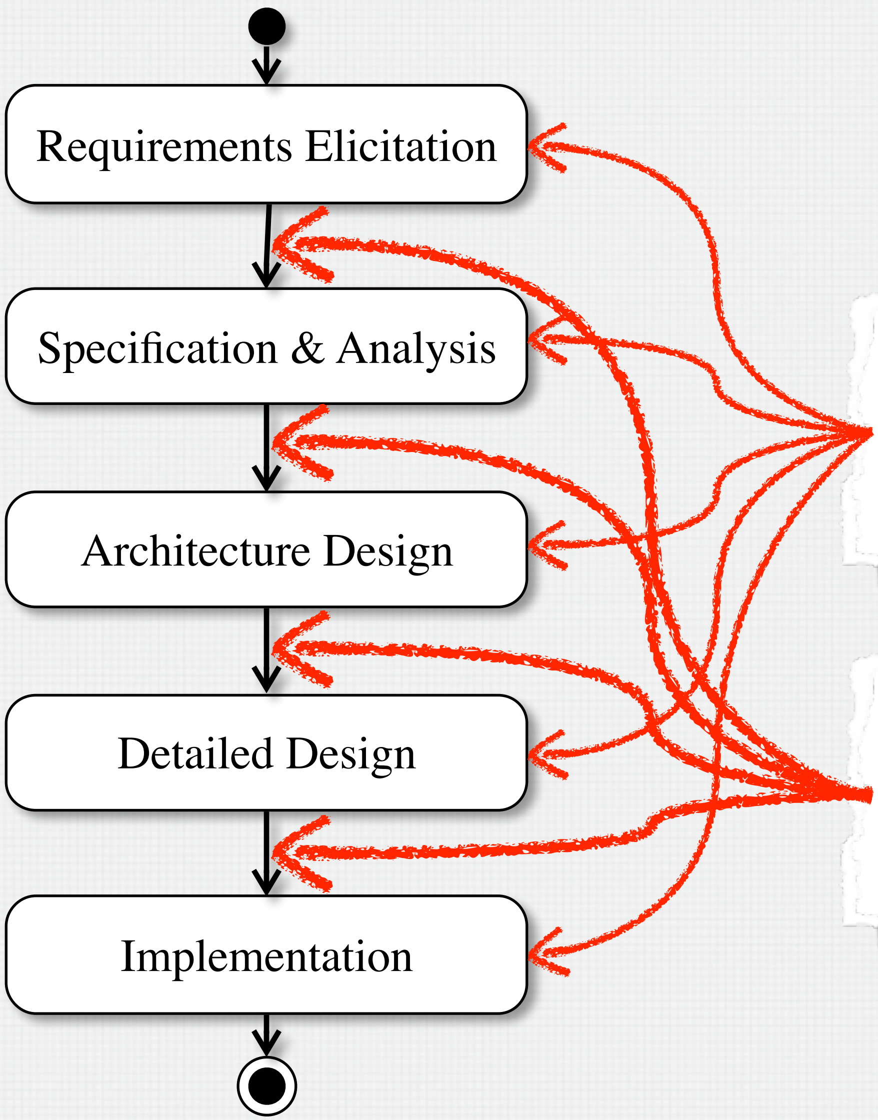
```java
public class Asteroid
  extends Model {

  //position
  float xPos;
  float yPos;

  //dynamics
  float speed;

  public Asteroid() {
    xPos = ConstantWORLD_MAX_X;
    yPos = 0;
  }

  public void moveAsteroid() {
    xPos = xPos - speed;
  }

  public boolean outOfBounds() {
    return xPos < 0;
  }
}
```

Java Code

```xml
<?xml version="1.0"?>
<config version="1.0" serial="137"
timestamp="1145938502.12">
  <Lib>
    <Account>
      <LastUsed>1145938465</LastUsed>
      <LocalData>4215</LocalData>
      <Migration>63</Migration>
    </Account>
    <Call>
      <IncomingPolicy>everyone</IncomingPolicy>
      <MicVolume>96</MicVolume>
      <SkypeInPolicy>everyone</SkypeInPolicy>
    </Call>
  <UI>
    <Profile>
      <LastOnlineStatus>2</LastOnlineStatus>
    </Profile>
  </UI>
</config>
```

Configuration File

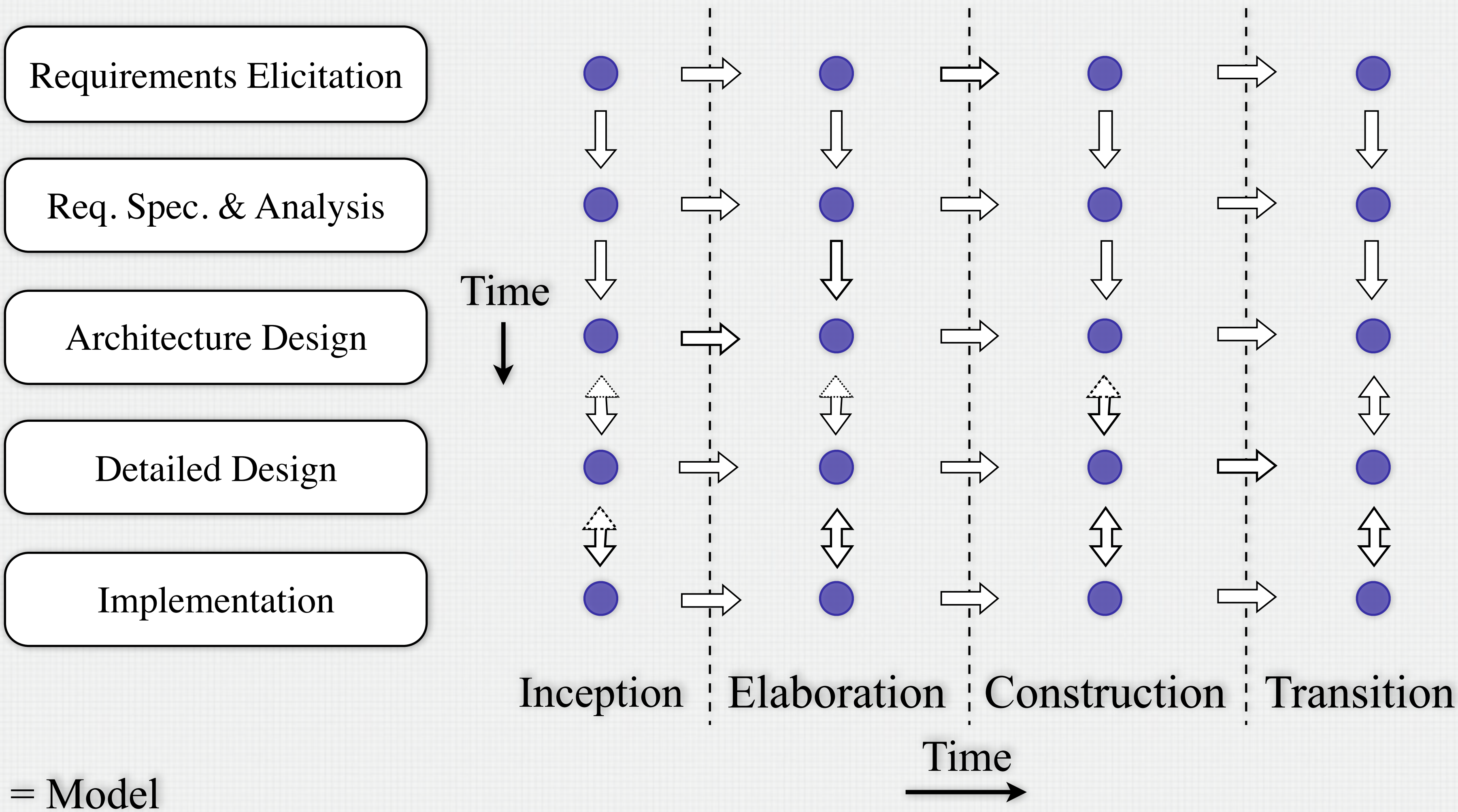
Model-Driven Engineering



Models are created at the right level of abstraction using the most appropriate modeling formalism

Model transformations map models at one level to models at another level of abstraction

Modern MDE



● = Model

⇨ = Refinement / Model Transformation

⬄ = Tool Support / Automated

On Modelling

- The choice of models and diagrams has a profound influence upon how a problem is attacked and how a corresponding solution is shaped
- Abstraction is a key to learning and communicating
- Every complex system is best approached through a small set of nearly independent views of a model; no single view is sufficient
- Every aspect of a system may be expressed at different levels of abstraction / fidelity

Object-Orientation

- Object-orientation is based on old principles
  - Abstraction
  - Information hiding and encapsulation
  - Modularity
  - Classification
- Object-orientation is based on a few concepts
  - Object
    - Groups together state and behaviour
  - Class
  - Inheritance
  - Polymorphism

Object-Orientation and SE

- Object-Orientation stems from object-oriented programming, but can be applied within the whole software development life cycle
  - Requirements Elicitation and Specification
  - Design
  - Implementation
  - Testing

Experience has shown that OO alone is not enough!

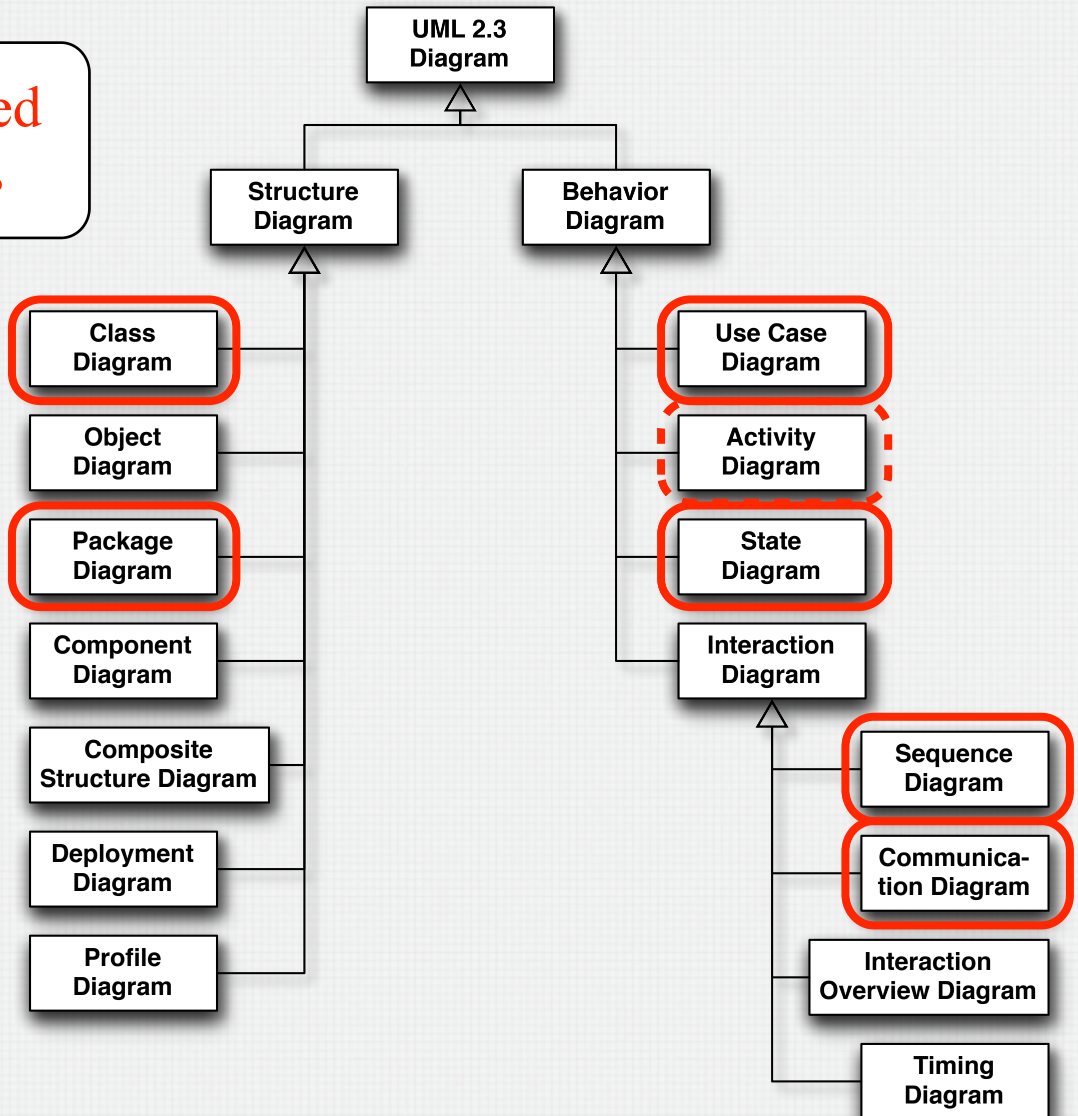
- Object-Orientation is a way of thinking about problems using models organized by real-world entities
- Object-Orientation is an engineering method used to create a representation of the problem domain and map it into a software solution

Scope of UML

- The Unified Modeling Language (UML) is a language for specifying, constructing, visualizing, and documenting the artifacts of a software-intensive system
- UML fuses the concepts of the Booch, OMT, and OOSE methods
- UML is a single, common, and widely usable modelling language
- UML incorporates the object-oriented community's consensus on core modelling concepts
- UML allows deviations to be expressed in terms of its extension mechanisms
- Current version: UML 2.4.1

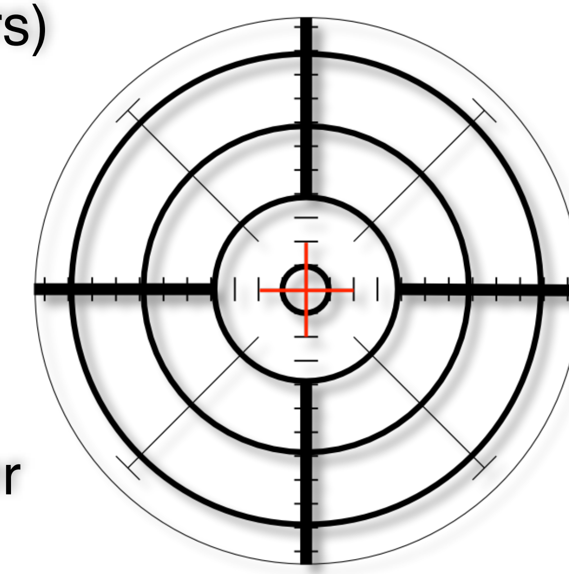
UML Diagrams

Notations Used in this Class



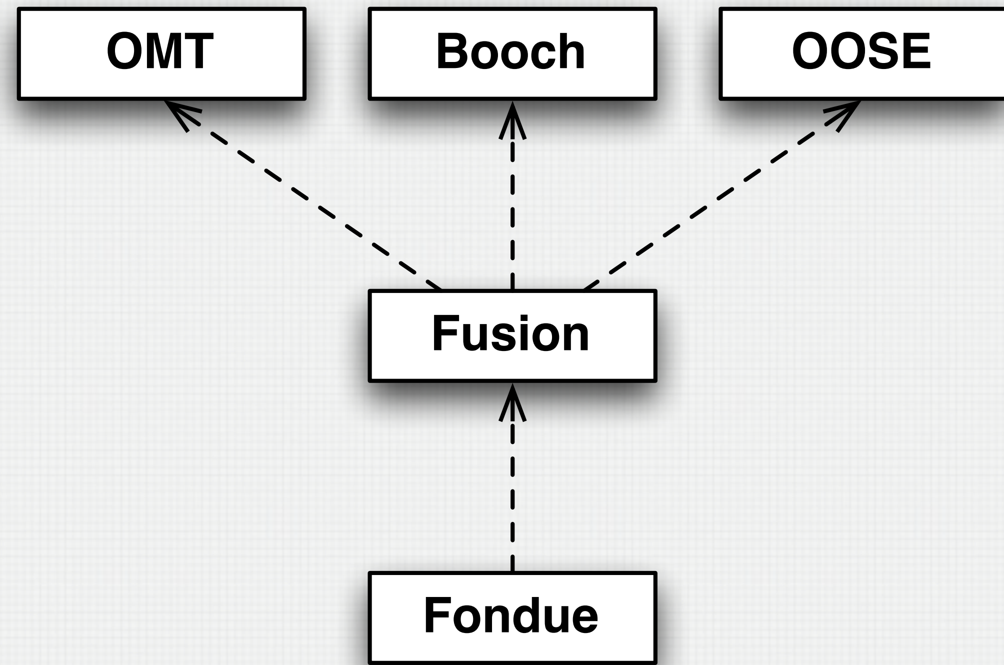
Outside the Scope of UML

- Programming Languages
  - UML is not intended to be a visual programming language
- Tools
  - UML does not define a tool interface, storage, or run-time model (which could be used by CASE tool developers)
- Process



  - UML is intentionally process independent
  - Processes by their very nature must be tailored to the organization, culture, and problem domain at hand
  - What works in one context would be a disaster in another
  - The selection of a particular process will vary greatly, depending on such things like problem domain, implementation technology, and skills of the team

Methods We're Using

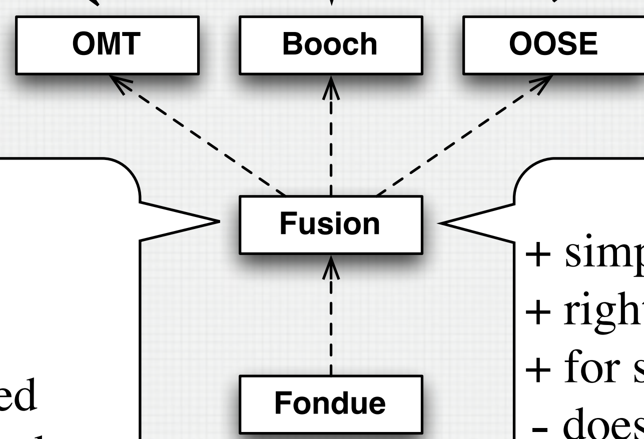


History of Fondue

Object Modeling Technique,
James Rumbaugh, 1991

Booch Method
Grady Booch, 1993
(Rational)

Object-Oriented Software
Engineering, Ivar Jacobsen, 1992



Derek Coleman, 1994
(Hewlett Packard)

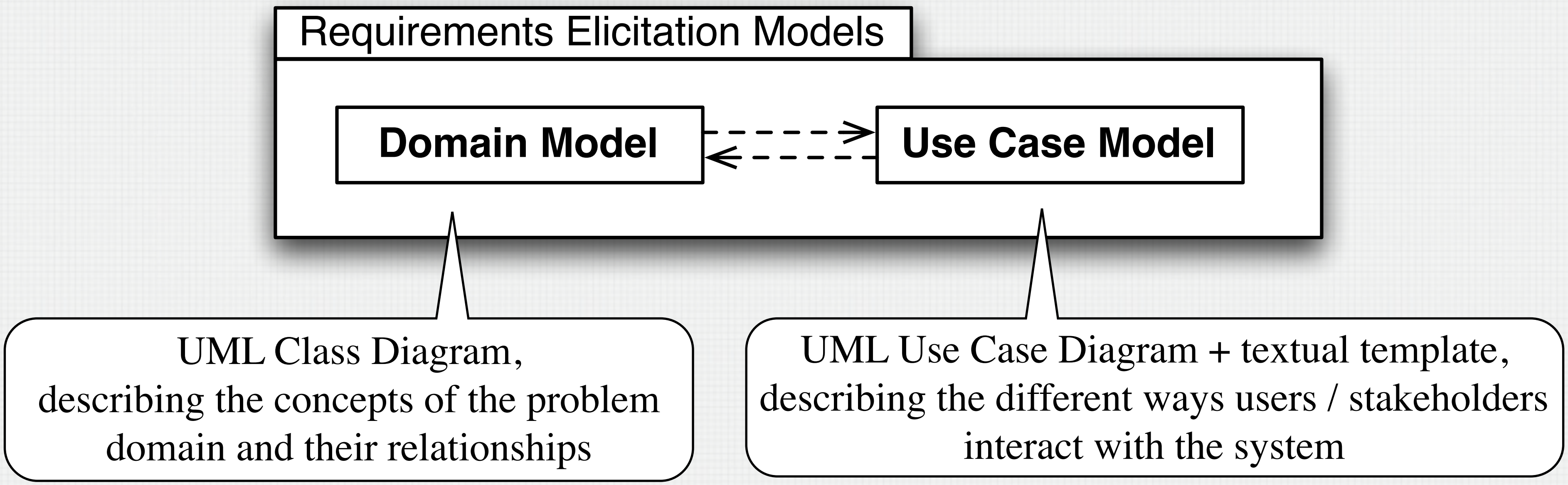
- 2nd generation object-oriented software development method based on OMT, Booch, OOSE, CRC Cards and formal methods
- Covers analysis, design and implementation of sequential reactive systems

+ simple to learn (300 page book)
+ right level of formality
+ for small/medium size systems
- does not use UML
- no requirements elicitation
- distributed, large systems
- concurrency
- missing user interface design
- data base design

What is Fondue?

- Fondue is a development method that extends the process and the models of the original Fusion method
- Fondue uses the UML notation
- Use cases are used during requirements elicitation
- Operations are specified formally in operation schemas by pre- and postconditions using the OCL
- Fondue follows the philosophy of model-driven engineering
  - For example: the domain model is refined into a concept model, then into a design class model and finally an implementation class model

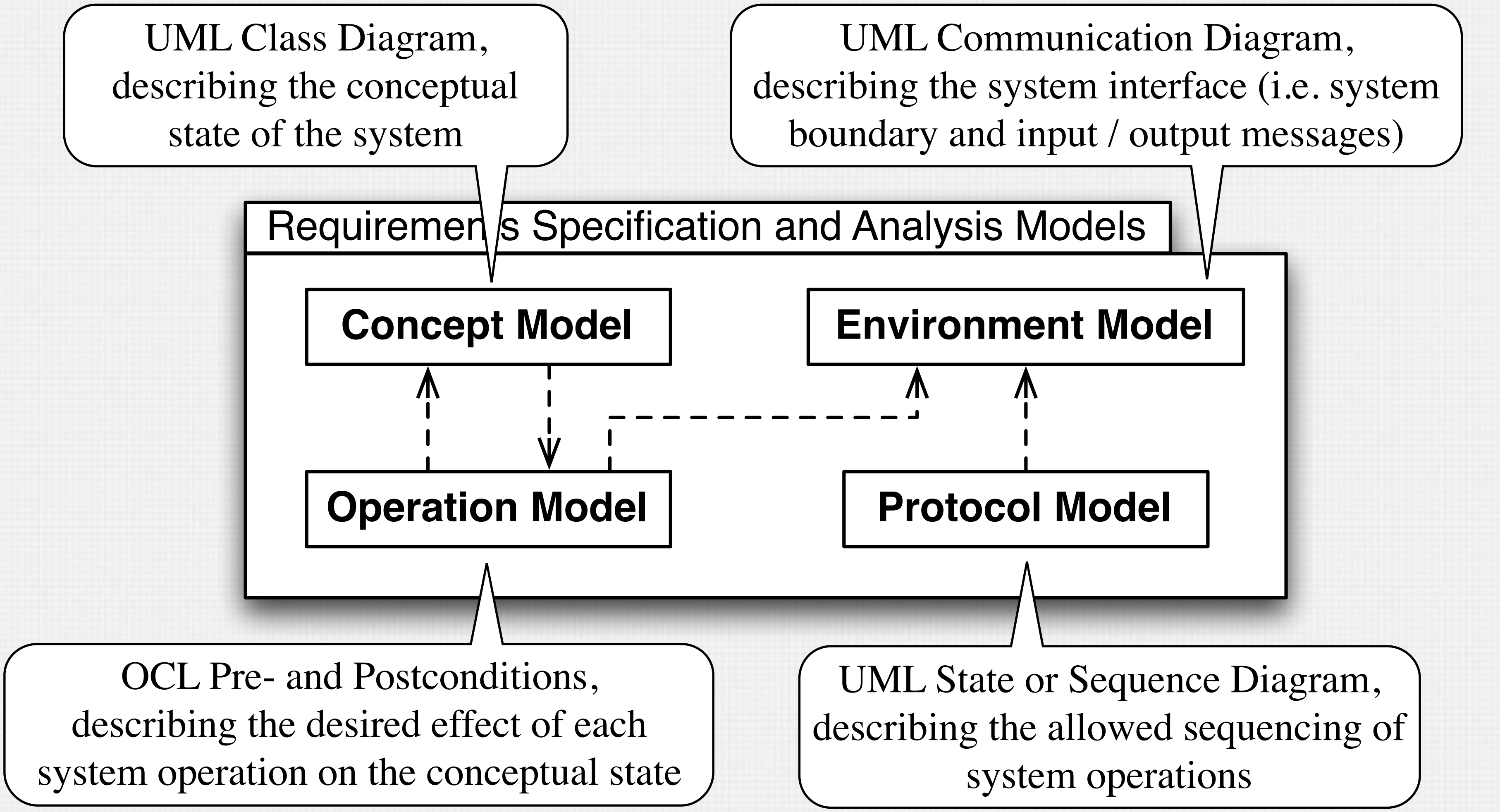
Fondue Models: Requirements Elicitation



Requirements Specification / Analysis

- The analyst defines the intended behaviour of the system
- Models are produced, which describe
  - The concepts that exist in the system.
  - The relationships between concepts.
  - The boundaries of the system.
  - The operations that can be performed on the system.
  - The allowable sequences of those operations.
- Fondue does not attach the operations to particular classes during analysis.

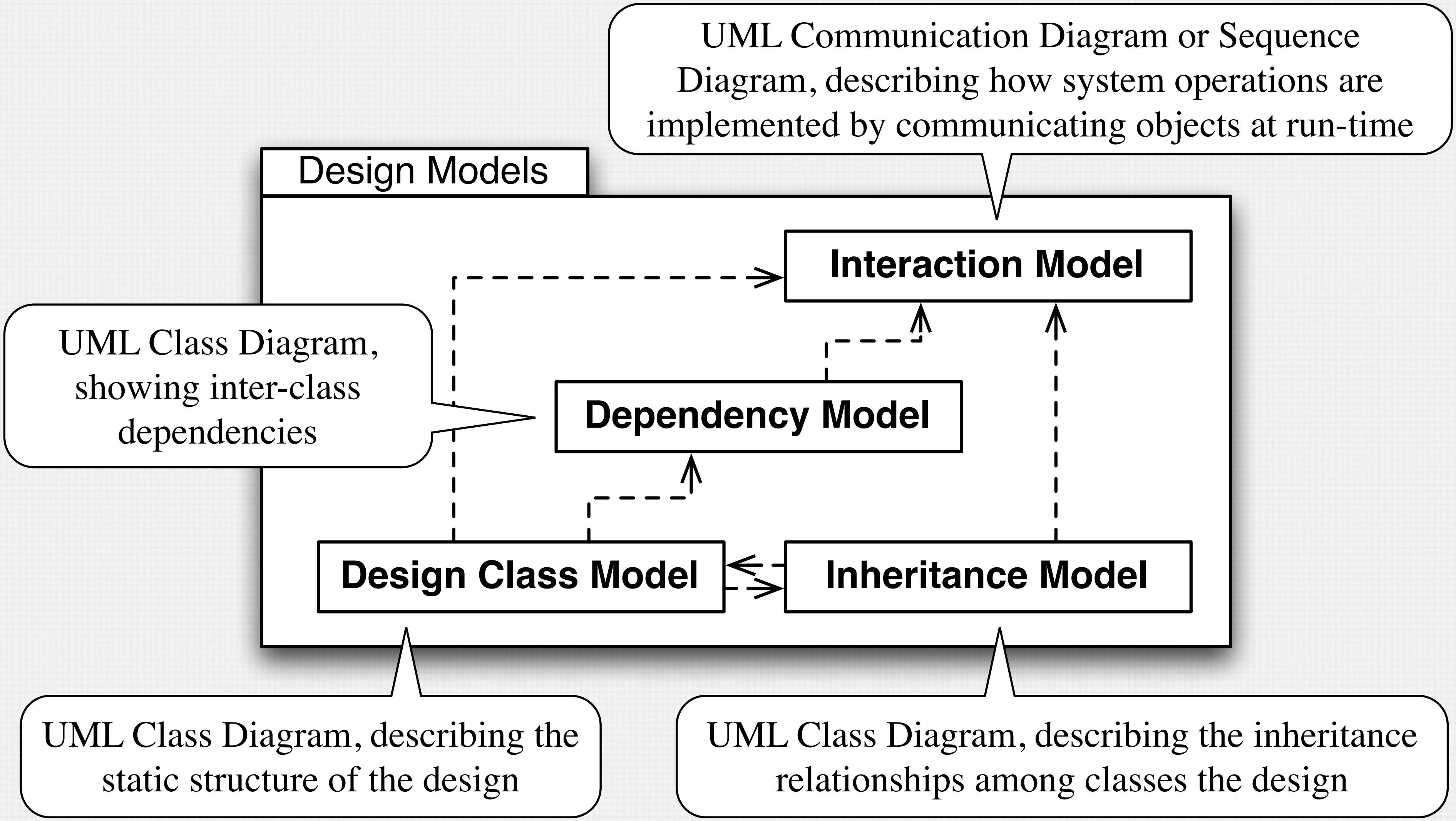
Fondue Models: Requirements Spec.



Design Activities

- The designer chooses how the system operations are to be implemented by interacting objects at run-time.
- Different ways of breaking up a system operation into interactions can/should be tried.
- The operations are attached to classes.
- The design phase delivers models that show:
  - How system operations are implemented by interacting objects.
  - How classes refer to one another (in order to achieve interaction).
  - How classes are related by inheritance.
  - The attributes and methods of classes.

Design: How?



UML Communication Diagram or Sequence Diagram, describing how system operations are implemented by communicating objects at run-time
Design Models
Interaction Model
UML Class Diagram, showing inter-class dependencies
Dependency Model
Design Class Model
Inheritance Model
UML Class Diagram, describing the static structure of the design
UML Class Diagram, describing the inheritance relationships among classes the design

Stepwise Refinement

- Initially, some classes end up with lots of responsibilities
  - Designers may need to investigate the substructures of some classes and their operations
- Hierarchical decomposition is used
  - The class is regarded as a subsystem
  - The analysis and design phases are applied to the subsystem
- Commonalities between classes are discovered
  - Inheritance is used to refactor common structure and behaviour

Implementation: Build It

- The design is mapped to a particular programming language.
- Fondue provides guidance on how this is done:
  - Inheritance, attributes, and methods are implemented in classes (if the construct is provided by the programming language).
  - Object interactions are implemented as calls to methods belonging to classes.
  - The permitted sequences of operations are recognized by a finite state machine.
- Result: Implementation Class Model (Class Diagram / Text / Code)

Questions