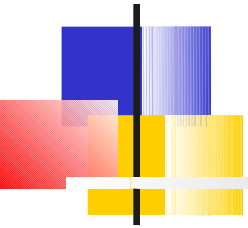
Information Systems Concepts

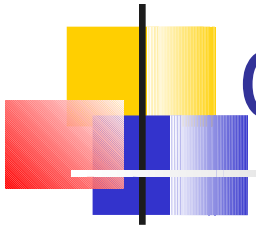
Modelling Concepts



Roman Kontchakov

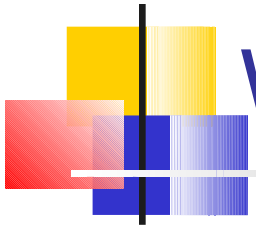
Birkbeck, University of London

Based on Chapter 5 and 7 of Bennett, McRobb and Farmer:
Object Oriented Systems Analysis and Design Using UML, (4th Edition), McGraw Hill, 2010



Outline

- Models and Diagrams
  - Section 5.2 (pp. 114 – 122)
- What Must a Requirements Model Do?
  - Section 7.2 (pp. 181 – 184)

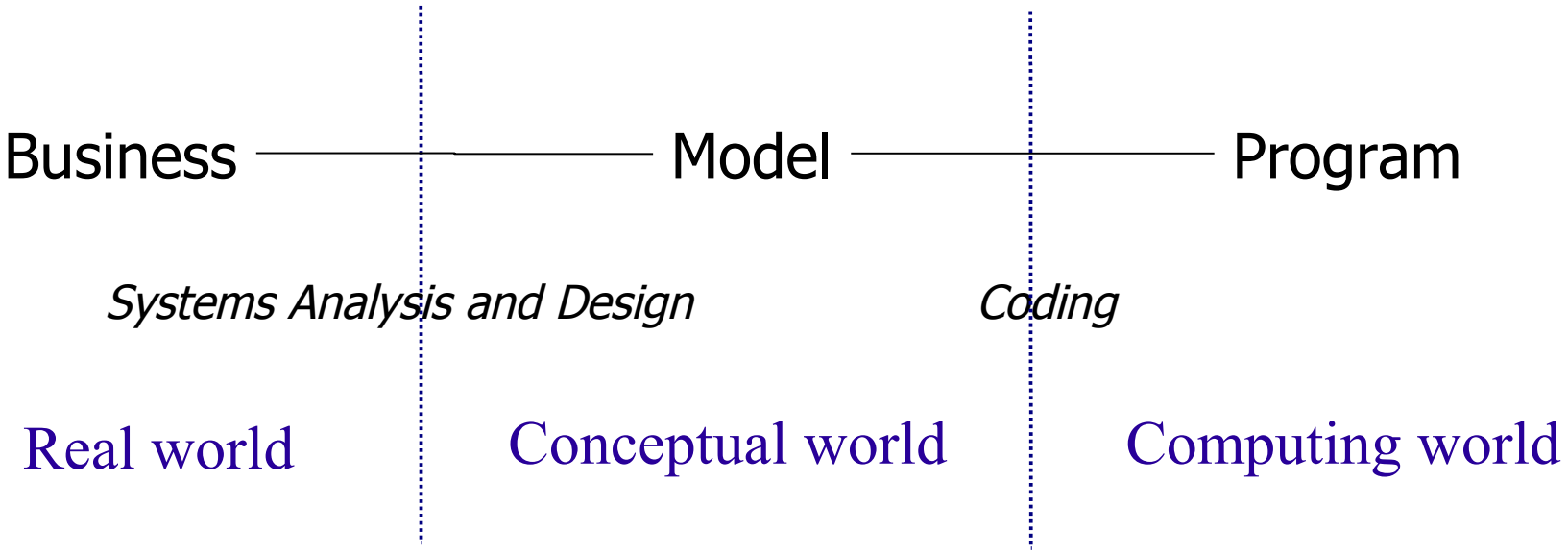


What is a Model?

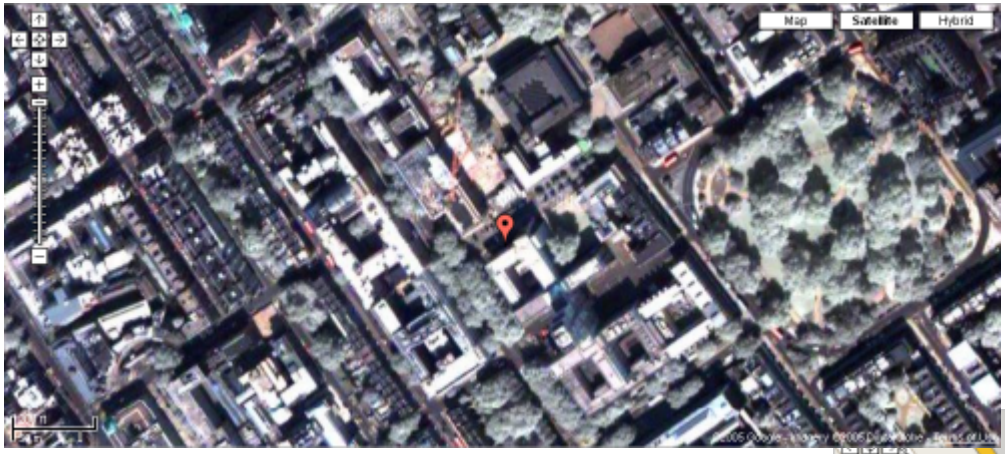
- "A model captures a ***view*** of a physical system. It is an ***abstraction*** of the physical system, with a certain ***purpose***. This purpose determines what is to be included in the model and what is irrelevant. Thus the model completely describes those aspects of the physical system that are ***relevant*** to the purpose of the model, at the relevant ***level of detail***."

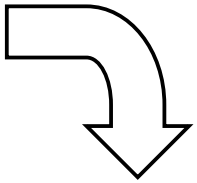
(OMG, 2009)

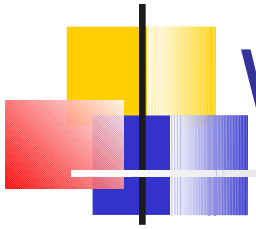
What is a Model?



Abstraction



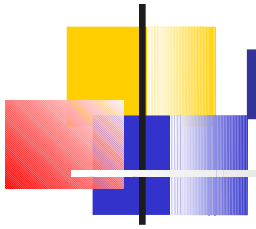




Why use a model?

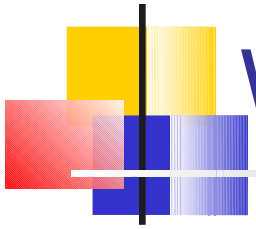
- A model is quicker and easier to build
- A model can be used in a simulation
- A model can evolve as we learn
- We can choose which details to include in a model
- A model can represent real or imaginary things from any domain

- A model allows us to talk, or reason, about the real thing without **actually building it**
- Much of software development involves creating and refining models, rather than writing lines of code



Requirements model

- describes *what the software should do*
- represents *people, things and concepts* important to understand what is going on
- shows *connections and interactions* among these people, things and concepts
- shows the business situation in enough detail to evaluate possible designs
- must be organized so as to be useful for designing the software



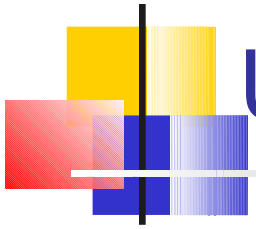
What is a Diagram?

- A diagram is a *graphical* representation of a set of elements in the model of the system
- Models v Diagrams
  - A diagram illustrates *some* aspect of a system
  - A model provides a *complete* view of a system at a particular stage and from a particular perspective
    - Most IS models today are in the form of diagrams, with supporting textual descriptions and logical or mathematical specifications
    - A model usually contains many diagrams – related to one another in some way

Why use a diagram?

- Natural language is often too ambiguous to be used for modeling
  - Communication + Ambiguity = Confusion !

A large object with one trunk and four legs.



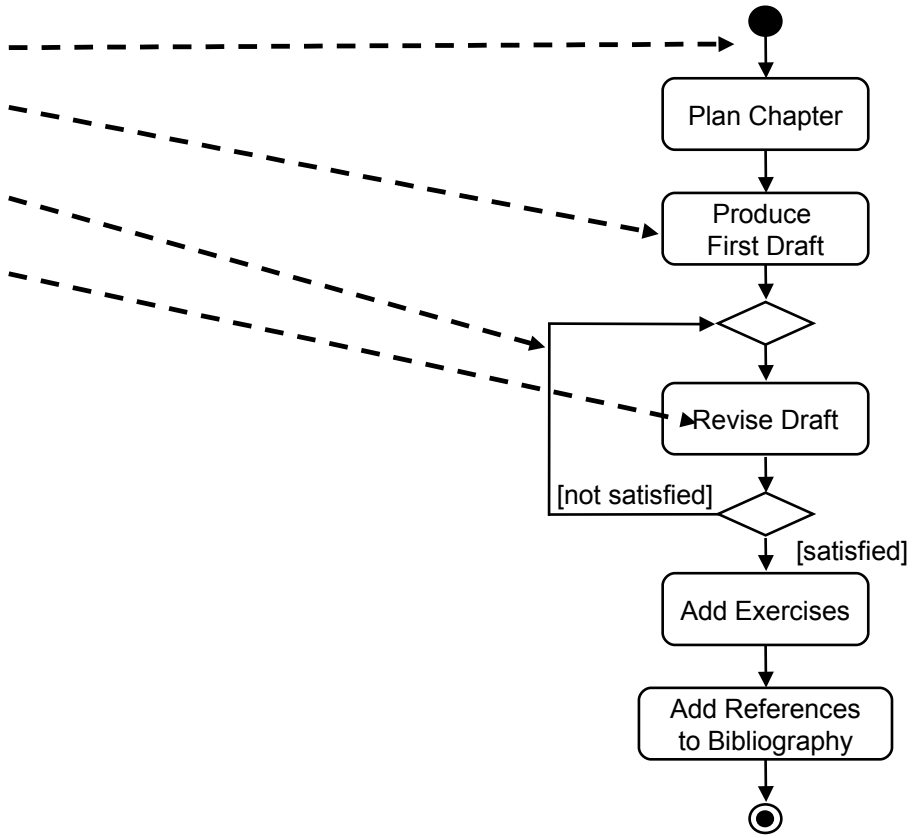
UML Diagrams

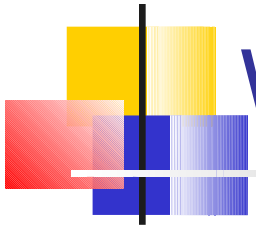
- UML 2 defines 13 types of diagrams
  - Structure
    - **Class Diagram**, Object Diagram
    - Component Diagram, Package Diagram
    - Composite Structure Diagram, Deployment Diagram
  - Behaviour
    - **Use Case Diagram**
    - **Activity Diagram,** State Machine Diagram
  - Interaction
    - **Sequence Diagram**, Communication Diagram
    - Timing Diagram, Interaction Overview Diagram

See Also: Appendix A – Notation Summary

UML diagrams notation

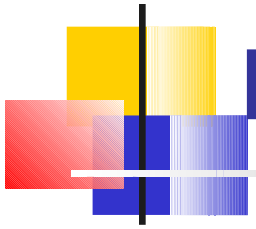
- A UML diagram usually consists of:
  - icons
  - symbols
  - paths
  - strings





What models/diagrams are good?

- Accurate
  - *unambiguous*, following rules or standards
- Concise
  - showing *only* what needs to be shown
- Complete
  - showing *all* that needs to be shown
- Consistent
  - internally and among each other
- Hierarchical
  - breaking the system down into different levels of details

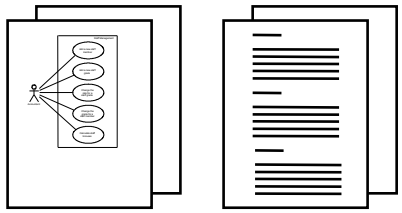


Developing models

- During the life of a project using an iterative lifecycle, models change along the dimensions of:
  - abstraction — they become more concrete
  - formality — they become more formally specified
  - level of detail — additional details are added

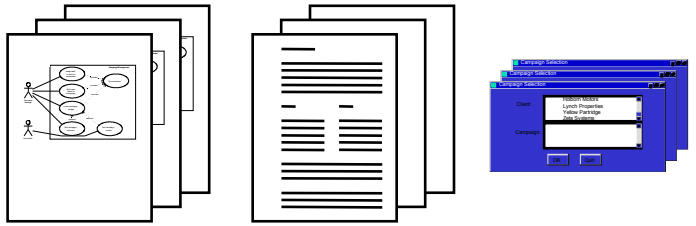
Example: Development of a Use Case model

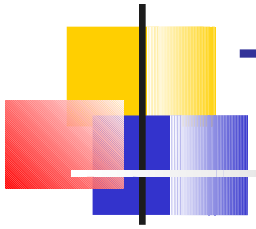
Iteration 1
Obvious use cases
Simple use case descriptions



Iteration 2
Additional use cases
Simple use case descriptions
Prototypes

Iteration 3
Structured use cases
Structured use case descriptions
Prototypes





Take Home Messages

- Models
- Diagrams