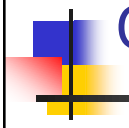


Lecture 1
Overview of OOAD and UML

Dr Peter T. Popov
Centre for Software Reliability

27th September, 2018

Outline of the lecture

- What is OOAD
- System complexity and the role of models
- Overview of UML
  - History
  - UML structure
- OO Analysis Recap
  - Robustness analysis
  - Consistency between UML diagrams
    - Some consistency rules

2

What is OOAD?

Object-oriented (OO) → the paradigm we use to model the software system

Analysis & Design → *what* the system needs to do

Design → *how* the system shall do it

- OO – represents the world as interacting objects
- OOAD ***specifies*** software systems in *sufficient* detail so that they can be built
- We do this by creating *models* of software systems
  - model - a representation of something that captures ***important details*** from a particular ***perspective***
  - Unified Modeling Language (UML) - a ***visual*** syntax for object models
  - "… all models are wrong, but some are useful" (George E. P. Box)

 3

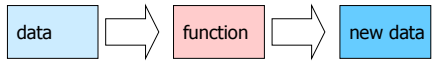
What is OO?

- A way of modeling and building software systems
- OO models systems as sets of objects that
  - Encapsulate data and function
  - Interact with each other by sending messages
- The objects *should* map directly onto things found in the problem domain:
  - E.g. in the banking domain, things such as BankAccount, Person, Money etc.
  - Principle of ***Convergent Engineering***

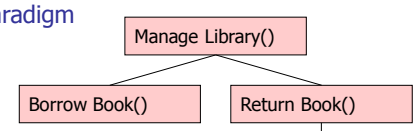
 4

Different paradigms...

Procedural paradigm



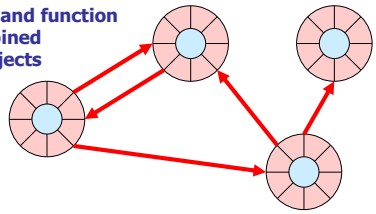
Data and function separate



Top-down functional decomposition, entity relationship etc.

Object-oriented paradigm

Data and function combined in objects



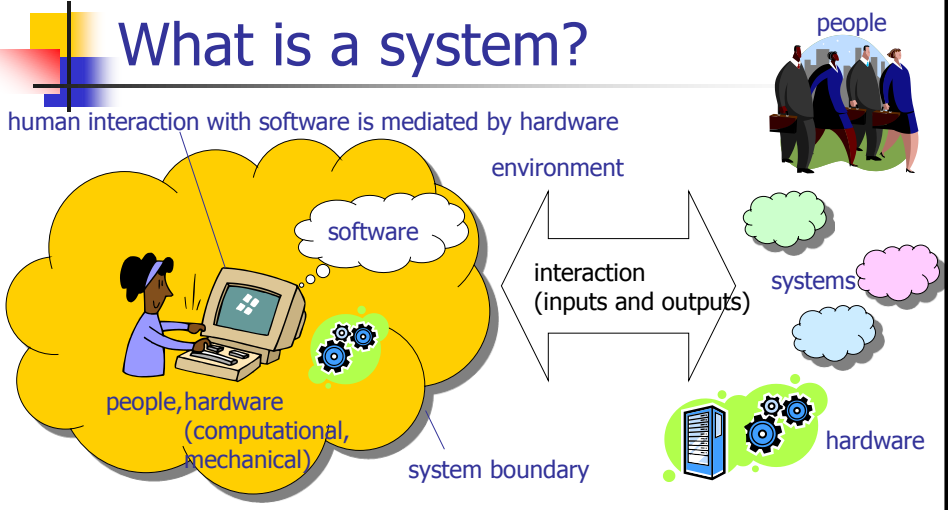
Object Oriented Analysis & Design (OOAD)

 5

Modelling System

 6

What is a system?



- A system is a bounded physical/virtual entity consisting of ***interacting elements*** operating in an environment to achieve defined objectives
- A system needs to interact with its ***external environment*** to achieve its objectives
- The choice of system boundary is arbitrary – it depends on the problem you are trying to solve!

 7

Software system complexity

complex (adj.):

"Consisting of many different and interconnected parts."

- Examples:
  - retail banking, scheduled airline services, web retailing
- Problems:
  - developed by a team in a lengthy process
  - impossible for an individual to comprehend fully
  - difficult to document and test
  - potentially inconsistent or incomplete
  - subject to change
  - no fundamental laws to explain phenomena and approaches

 8

Reasons for system complexity

- Grady Booch identifies four reasons for the complexity of software-intensive systems:
  - Nature of the problem domain:
    - complex requirements
    - decay of systems
  - Complexity of process:
    - management problems
    - need for simplicity
  - Software flexibility is a double-edged sword:
    - "Software is flexible and expressive and thus encourages highly demanding requirements, which in turn lead to complex implementations which are difficult to assess"
  - Characterising behaviour of discrete systems
    - numerous possible states
    - difficult to explore all states

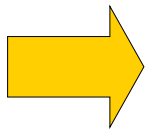
 9

Complex systems are hard to understand

- Hampered by human limitations:
  - cognitive limitations of individuals
  - poor communication between individuals

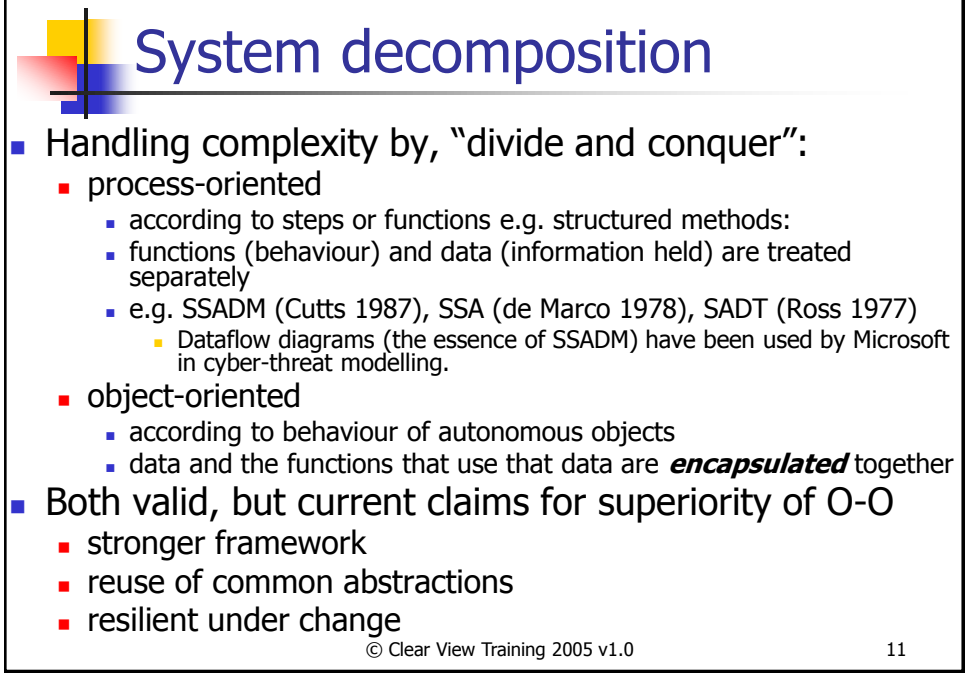
Secrets of OOAD

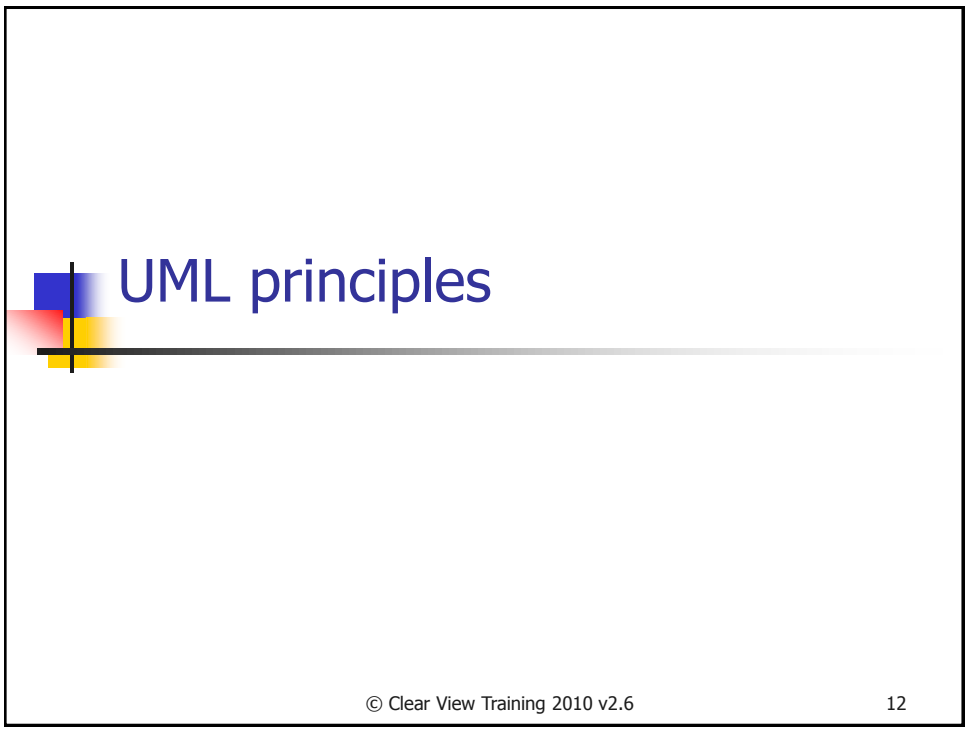
- ***Abstraction*** (i.e. modelling) helps people to understand information and ideas:
  - grouping
  - generalising
  - chunking



- identification of components and subsystems
- manageable model of the system

10





1.2

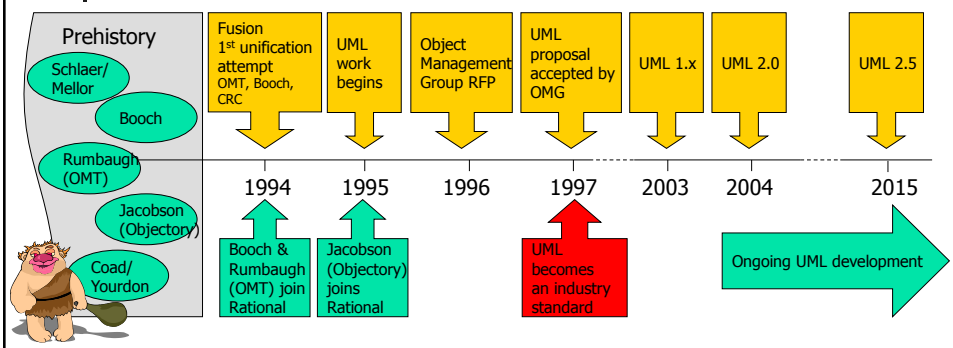
What is UML?

- Unified Modelling Language (UML) is a general purpose visual modelling ***language***
  - Can support ***all existing lifecycles***
  - Intended to be ***supported*** by CASE tools
- Unifies past OO modelling techniques and experience
- Incorporates current best practice in software engineering
- UML is *not* a methodology!
  - UML is a visual language

 13

1.3

UML history



- The last major upgrade to UML occurred at the end of 2003:
  - Greater consistency
  - More precisely defined semantics
  - New diagram types
  - Backwards compatible

 14

1.5

Why "unified"?

- UML is unified ***across several domains***:
  - Across historical methods and notations (documented in books)
  - Across application domains
    - banking, process control, energy, telecommunications, avionics, etc.
  - Across implementation languages and platforms
    - Many languages supported by UML tools, which allows for seamless development.
  - Across the development lifecycle and development processes
    - Unified Process (or Rational Unified Process) is typically used with UML, but other, more/less rigid processes can be used too (Waterfall, even Agile)
      - Some of the proponents of Agile manifesto (http://agilemanifesto.org/) are well known authors of books on UML (e.g. Martin Fowler)

15

1.6

Objects and the UML

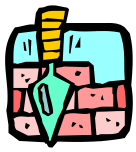
- UML models systems as collections of objects that interact to deliver benefit to outside users and they consist of:
  - Static structure
    - What kinds of objects are important
    - What are their relationships
  - Dynamic behaviour
    - Lifecycles of objects
    - Object interactions to achieve goals

16

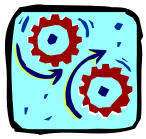
1.7

UML Structure

- In this section we present an overview of the structure of the UML
- The modelling elements mentioned here are discussed later, and in much more detail!



Building blocks



Common mechanisms

Architecture

1.8

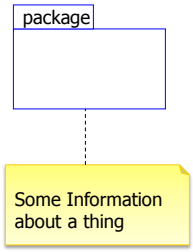
UML building blocks

- Things
  - Modelling elements (e.g. use-cases, classes, objects, etc.)
- Relationships
  - Tie things together (associations, generalisation)
- Diagrams
  - Views showing interesting collections of things
  - Are ***views*** of the model
    - Using many views allows the modeller to focus on different aspects of the system.

1.8.1

Things

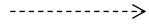
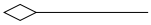
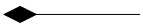
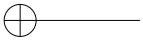
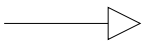
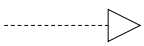
- Structural things – nouns of a UML model
  - Class, interface, collaboration, use case, active class, component, node
- Behavioural things – verbs of a UML model
  - Interactions, state machine
- Grouping things
  - Package
    - Models, frameworks, subsystems
- Annotational things
  - Notes
  - Tagged values



 19

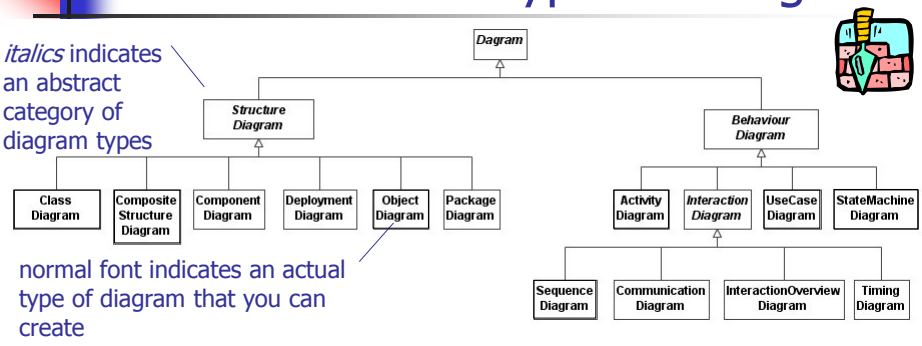
1.8.2

Relationships

relationship	UML syntax	brief semantics
dependency		The source element depends on the target element and may be affected by changes to it.
association		The description of a set of links between objects.
aggregation		The target element is a part of the source element.
composition		A strong (more constrained) form of aggregation.
containment		The source element contains the target element.
generalization		The source element is a specialization of the more general target element and may be substituted for it.
realization		The source element guarantees to carry out the contract specified by the target element

 20

UML 2.0 has 13 types of diagram

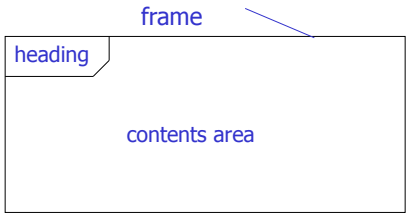


- Structure diagrams model the structure of the system (the static model)
- Behavior diagrams model the dynamic behavior of the system (the dynamic model)
- Each type of diagram gives a ***different view on the system.***

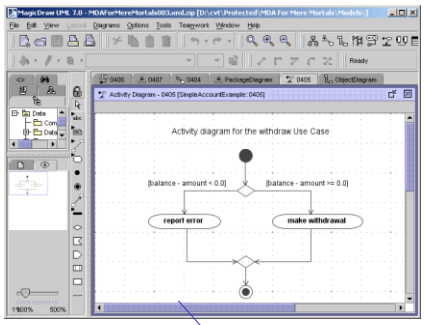
21

1.8.3

UML 2 diagram syntax



heading syntax: <kind> <name> <parameters>
N.B. <kind> and <parameters> are optional



implied frame

- The heading specifies the kind of diagram, it's name and any information (parameters) needed by elements in the diagram
- The frame may be implied by a diagram area in the UML tool

22

UML benefits

- What are the benefits of using diagrams at all?
- A couple of sceptical views about UML:
- Example 1: Is not software code sufficient to document software development?
  - Not really.
    - Take a large open source project, e.g. PostgreSQL (Firebird) database server and try to make sense of the 50-100 MB of source code.
  - Consider also the following:
    - Analysts tend to get paid more than programmers! Analysts use models!
    - Many pure programming jobs are outsourced to countries with lower wages.
    - A number of our alumni reported to me that the knowledge of UML has been a key part for them to progress in their professional carrier.
  - Model-driven development is a norm today in serious organisations in IT (e.g. IBM), leading engineering firms (e.g. Airbus, etc.)

23

UML benefits (2)

Example 2: "Real programmers do not do modelling!"

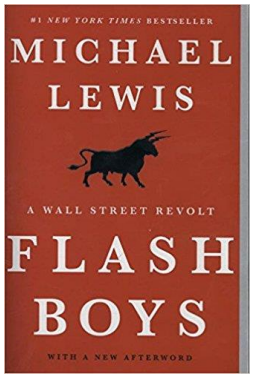
This view is simplistic and often simply ***wrong***! The current trend in developing application software shifts towards model driven development! Read about Model Driven Architecture (http://www.omg.org/mda/)

- Consider also the following aspects:
  - With tool support switching between models and code is really very simple. Many tools make it trivial to keep models (e.g. class model) and the source code consistent.
  - Many tools support ***reverse engineering*** of existing source code, e.g. your own code or the code by a 3rd party (Visual Paradigm will do it for a number of languages).
    - As an experiment, take a large piece of open-source and try to "make sense" of it. Then reverse engineer the source code and check whether the resulting diagrams (class and possibly sequence) helped you understand the structure of the software product better than the code itself.
  - I have done this exercise many times in the past with a few student submissions to help me understand the structure of their code.
    - some have separated the concerns very well (quite clear from the class diagram)
    - others decided to have ***functoids***, i.e. mixed together the core functionality of the problem domain classes with the GUI functionality, a messy solution.

24

UML in industrial practice

- Model-driven development is a reality in the development of ***high-integrity*** software
  - Airbus makes it mandatory for all its subcontractors to use tools in development (typically UML based)
    - Papyrus (https://eclipse.org/papyrus/)
    - CHESS (https://www.polarsys.org/chess/)
- UML diagrams are used to develop public specifications for interoperable services
  - AMI (Advanced Metering Infrastructure), e.g. http://www.corniceengineering.com/Pubs/AMIUseCaseReportVersion1.0Final.pdf
- Not documenting software development leads to software which is difficult to maintain.
  - For examples from financial industry read "Flash Boys: A Wall street revolt".
    - One of the points made in the book is that the code used for ***fast computer trading*** oved the years has becomes very hard to maintain (spaghetti code) due to the lacks of documentation.



25

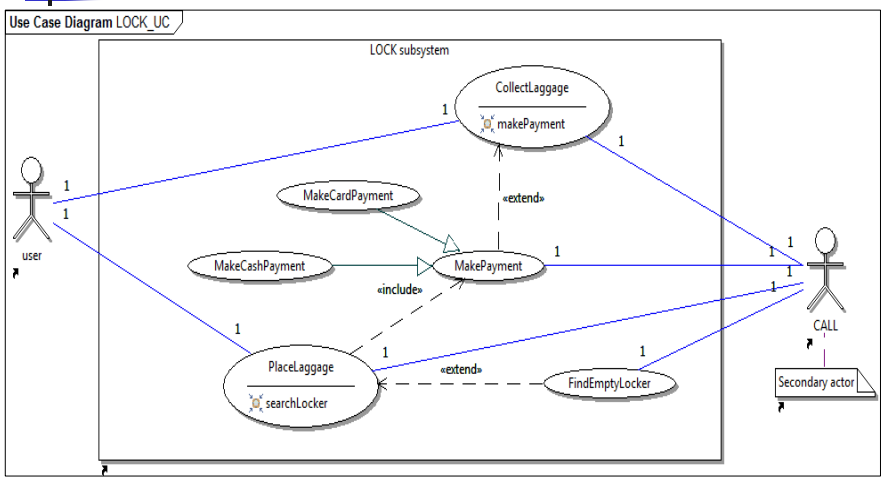
Examples of diagrams

A set of important UML diagrams follows:

- You are already familiar with some of these diagrams
  - ***Use case diagram*** shows how the system can be used by external actors.
- You are already familiar with other diagrams, but we will refine these and add details in design
  - **Class diagram** – presents the structure of the code concisely, hiding many details;
  - ***Sequence diagram*** : shows the interaction between objects (what messages get exchanged between the objects),
- New diagrams
  - ***State machine diagram*** : i) shows how the data held by objects gets changed as a result of external stimuli; ii) communication protocols
  - ***Deployment diagram*** – shows how software gets deployed on physical hardware.
    - (not illustrated in this lecture, but we will cover it in the module)
  - ***Component diagram*** –shows software architecture (high level design), i.e. how software components (e.g. off-the-shelf software) get integrated in a system and how their interfaces are coupled (defined and used).

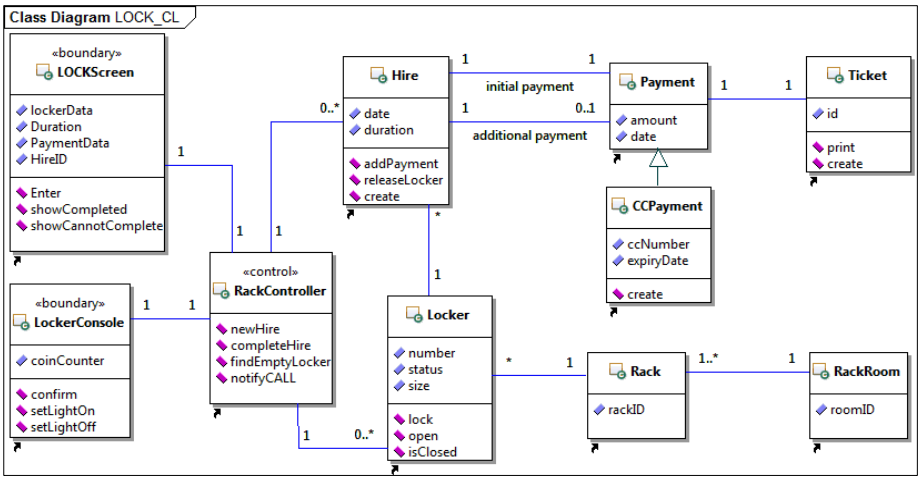
26

Use-case diagram



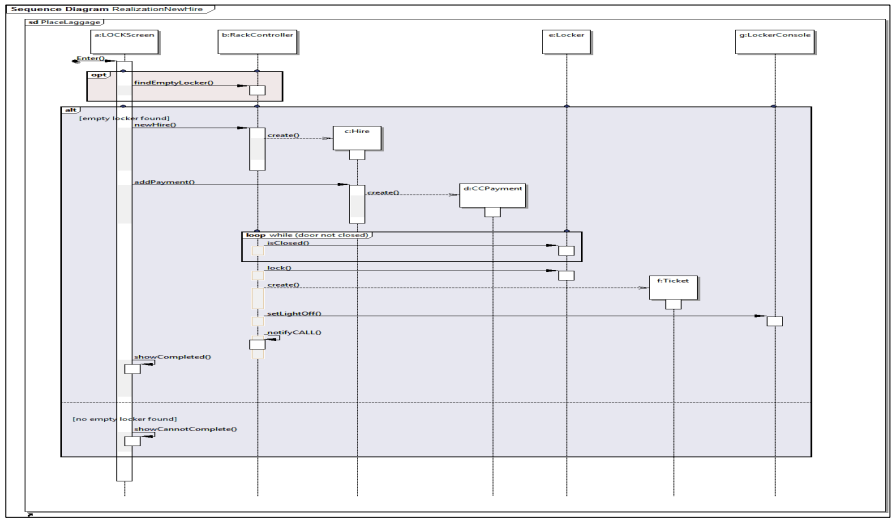
27

Class diagram



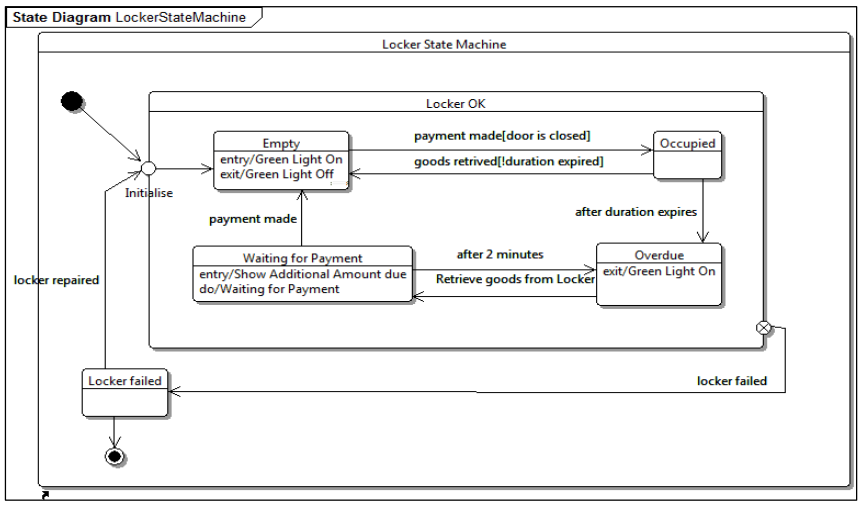
28

Sequence Diagram



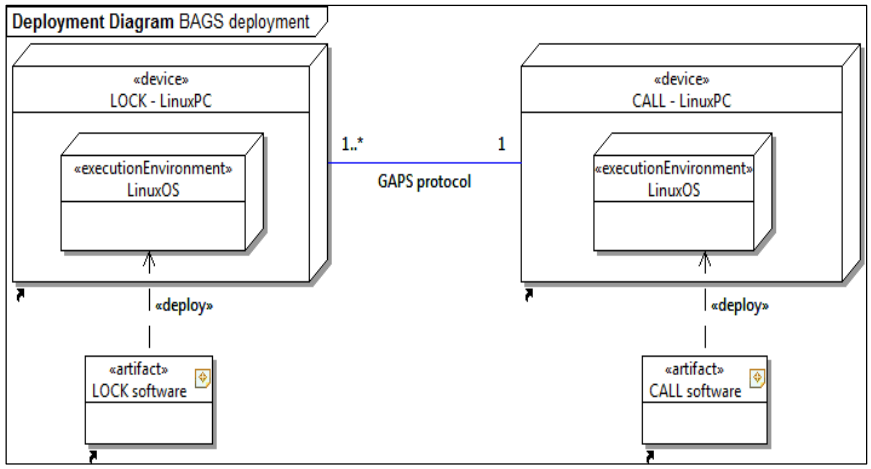
29

State machine diagram



30

Deployment Diagram

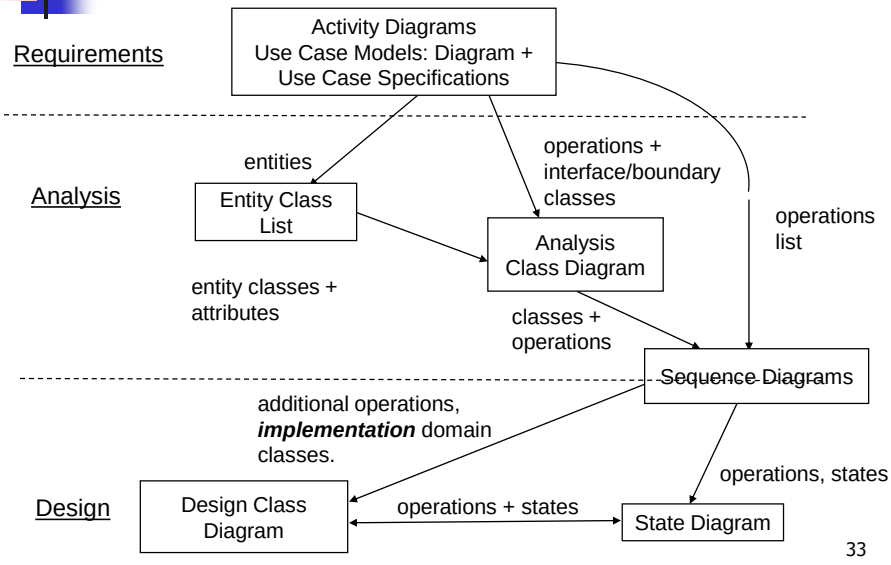


31

Why so many diagrams?

32

UML Analysis Models

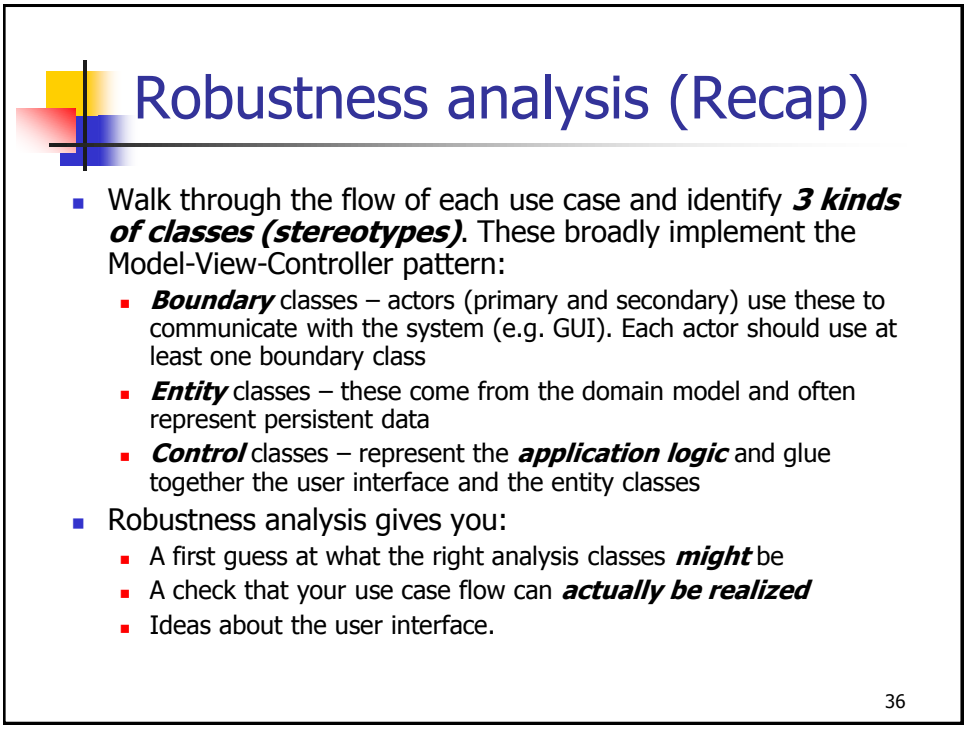


33

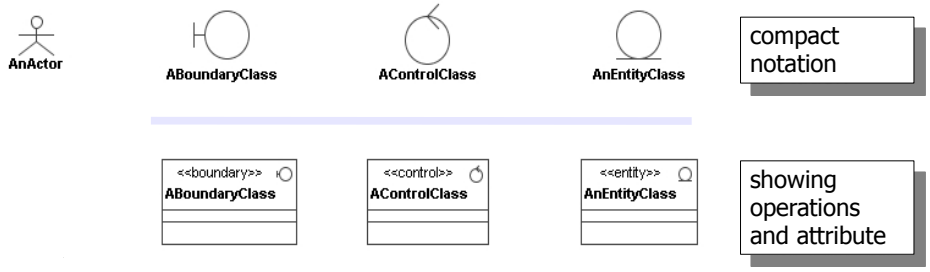
OOAD benefits

- In Software Engineering module (Year 1) you learned:
  - how to write a ***requirements specification*** and
  - how to analyse the ***problem domain*** by creating analysis models:
    - a use-case model (diagram and specifications),
    - a class diagram, and
    - a sequence diagrams
- We will build on this knowledge and will proceed to software design and testing the implemented software code.
  - The implementation itself (i.e. constructing software using a programming language) is ***not in the scope of the module***.
  - The UML diagrams, related to implementation, however, will be introduced. These are used to communicate to stakeholders (e.g. developers):
    - software architecture (component diagrams), and
    - software deployment on real hardware (deployment diagrams).

34



Robustness analysis notation



- ***Stereotypes*** such as «boundary» extend the UML meta model by introducing new modelling elements based on existing ones
  - Each stereotype can have its own icon
  - They are one of the UML extensibility mechanisms

37

Robustness analysis rules

- The different types of classes can only be ***associated*** as shown.

	AnActor	ABoundaryClass	AControlClass	AnEntityClass
AnActor		→		
ABoundaryClass			→	
AControlClass		→	→	→
AnEntityClass			→	→

38

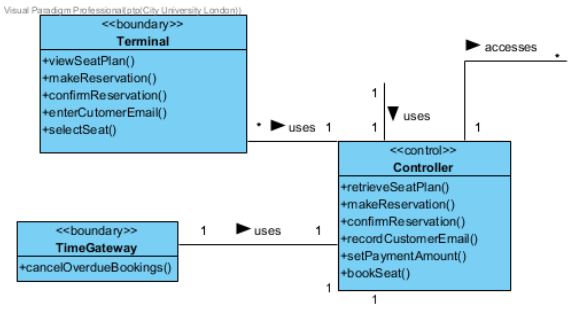
Robustness analysis in practice

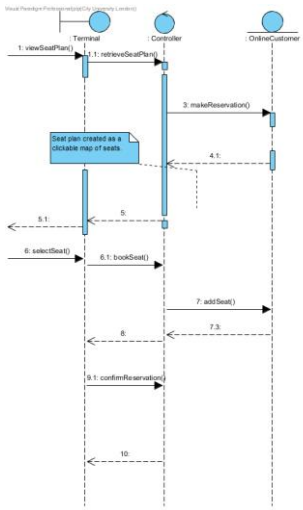
- Analyse each use case specification for:
  - Nouns
    - If the noun describes something that the system must keep information about it indicates an entity class
      - Some nouns may indicate attributes of entity classes
      - Some nouns may indicate relationships between classes
    - If the noun describes something an actor interacts with, it indicates a ***boundary class***.
  - Verbs
    - Describe things ***the system does*** – indicate controller classes
    - May imply relationships between classes
    - May indicate operations of a class

39

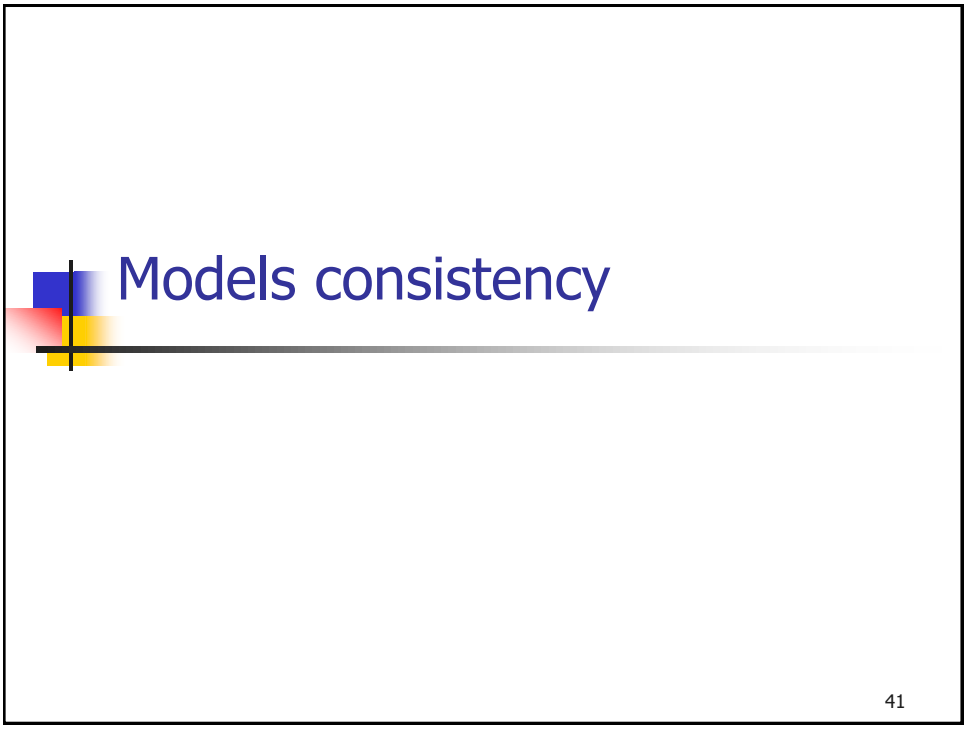
Examples of Analysis Diagrams

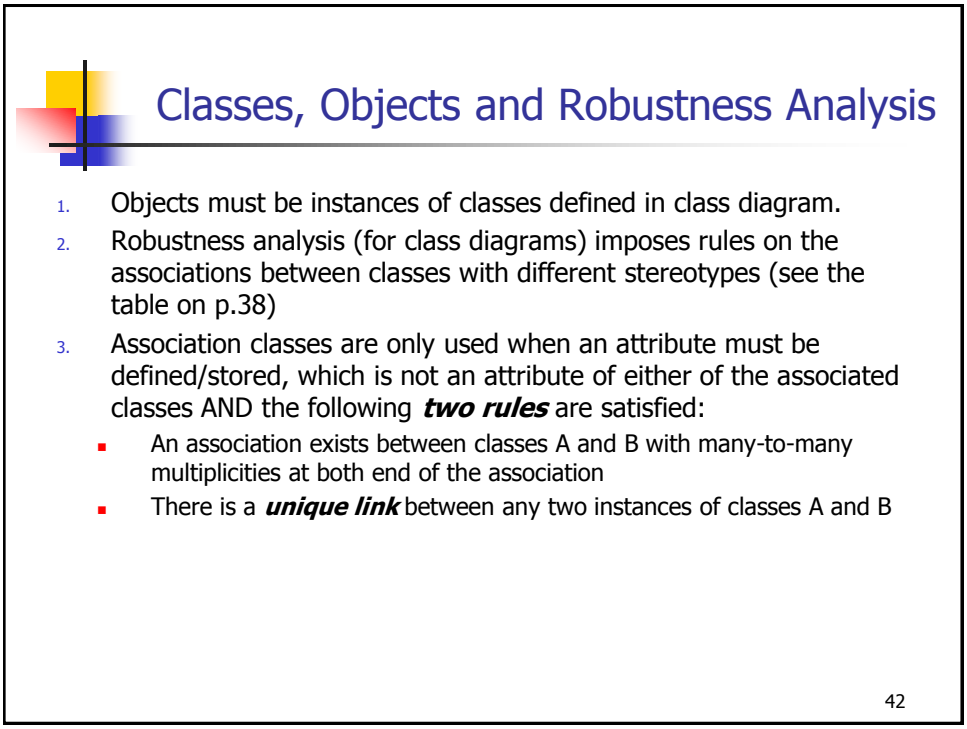
- <start INM312-2014 CW model>
  - Robustness Analysis implications for Class and Sequence diagrams





40





Consistency Between Structure and Behaviour Models

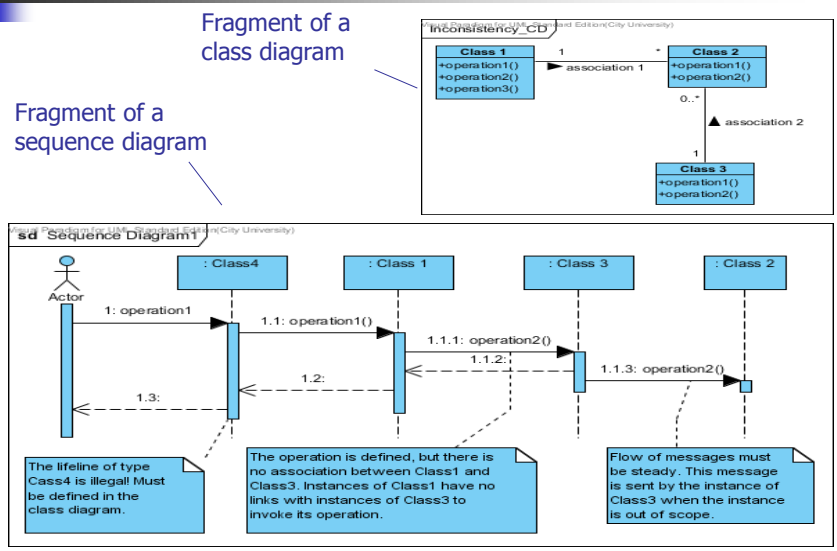
- Each lifeline (represents an object) in a sequence diagram must be an instance of a class ***defined*** in the class diagram
  - System cannot be a lifeline!
  - Database is NOT part of problem domain and should not appear in analysis class diagram. It is part of the ***implementation*** domain and is added in design.
- All ***call messages*** sent to a lifeline must refer to an operation defined for the class, an instance of which the lifeline is.
- A message between two lifelines in the sequence diagram ***requires a link*** between the objects:
  - This in turn implies that the corresponding classes in the class diagram ***must be associated***.
  - When you develop sequence diagrams, you may discover that a message must be used between lifelines which are not linked (i.e. the corresponding classes are not associated).
  - In this case the class diagram must be ***updated*** by adding an association between the respective classes.
- Object nodes in an Activity diagram are instances of the classes defined in the class diagram.

43

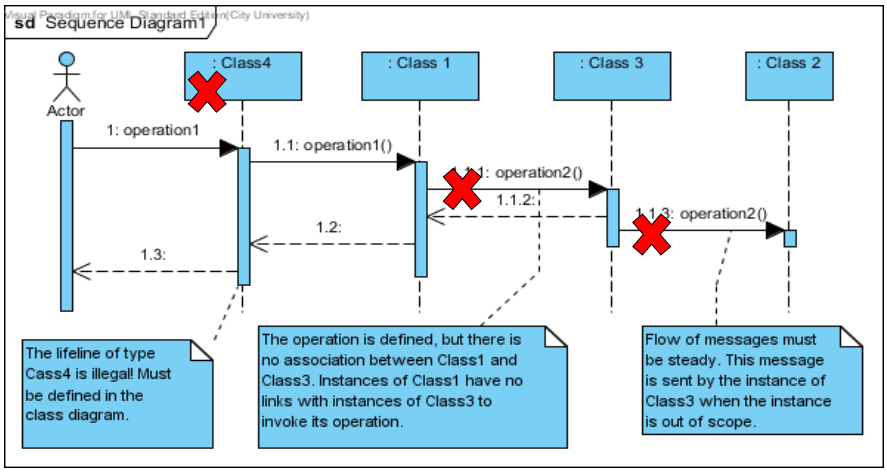
Examples of inconsistency

Fragment of a class diagram

Fragment of a sequence diagram



Examples of inconsistency (2)



Tutorial this week

- Tutorial 1 will be on a realisation (i.e. a sequence diagram) of a use case of the BAPPERS system.
- You will be provided with:
  - a use case diagram
  - specifications of several related use–cases, and
  - a 1st - cut class diagram.
- You are expected to:
  - develop a sequence diagram, a complete realisation of a set of related use cases (with <<include>>/<<extend>> relationships), which show:
    - the flow of messages between "lifelines" (i.e. objects of classes defined in the 1st – cut class diagram) as defined in the ***main*** and the ***alternative flows*** of the use-case specifications.
  - The class diagram ***may require changes***, e.g. adding operations to existing classes, associations between classes, and even new classes.
- A similar assignment will be included in the CW for the module.

 46

1.11

Summary

- We use models to deal with software complexity.
- The Unified Modelling Language (UML) is a general purpose visual modelling language, not a methodology
- UML is composed of building blocks:
  - Things
  - Relationships
  - Diagrams
- There are 13 different kinds of diagrams in UML 2.0
- Further reading:
  - Chapters 1, 4-5, 7-10 and 12-13 of the main text (i.e. Arlow's book) cover analysis comprehensively.

 47