Systems Concepts and Modelling

Lesson Objectives

• Understand basic systems concepts
• Understand the basic concepts of design and modelling that underpin everything in this course
  – Component decomposition
  – Trade-off between coupling and cohesion
• Understand the rationale for UML as language for systems modelling
• Be aware of the basic types of UML diagrams
A System: Car

A System: Company Payroll

- Key Inputs: employee information
- Key outputs: payslips, cheques, cash
- Physical components: people, paper computers
- Conceptual components: pension fund
A System: London Underground

- Key inputs: passengers, money
- Key outputs: weary and frustrated passengers, fumes
- Physical components: trains, drivers, track, signals
- Conceptual components: routes, timetables

A System: General Properties

- Made up of components, both physical and conceptual
- Receives inputs and transforms these into outputs
- Exists within an environment
- Boundary divides things inside the system from things outside
- Exhibits behaviour
- Fulfils some specific purpose which varies according to particular viewpoints
Systems: Definition

An assembly of components, connected together in an organised way and separated from its environment by a boundary. This organised assembly has an observable purpose which is characterized by how it transforms inputs from the environment into outputs to the environment.

A system with no inputs or outputs is closed.

Software components

- Files
- Subroutines
- Library functions
- Classes
- Packages
Component dependency

- Component A depends on B
- A change to B may require a change to A
- Many types of component dependency

Coupling: Definition

The level of interaction between modules in a system.

'loosely' coupled system  'tightly' coupled system
Cohesion: Definition

The cohesion of a module is the extent to which the module has a single clear purpose or function.

Coupling and Cohesion: Getting the Right Balance

As far as possible systems should be decomposed at each relevant level of abstraction into modules which individually have high cohesion and such that there is a low level of overall coupling.
Modelling Systems

Different viewpoints require different models

Software Modelling Methods

• 1970s: flowcharts, data flow diagrams, Jackson structured design
• 1980s: formal methods
• 1990s: object oriented methods
• 2000s: OO methods consolidated in UML
Rigour and Expressiveness of Methods

- Natural Language
- Pseudocode
- Flowcharts
- DFDs
- Object Oriented
- Formal Methods

Object Oriented Methods

- Based on identifying:
  - objects
  - attributes
  - classes
  - members
  - relationships between objects
- OO technology applies to specification, design and programming
Concepts and Phenomena

Class

- An abstraction in the context of object-oriented languages
- Encapsulates both state (variables) and behavior (methods)

<table>
<thead>
<tr>
<th>Class</th>
<th>Watch</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td></td>
</tr>
<tr>
<td>setDate(d)</td>
<td></td>
</tr>
</tbody>
</table>
What is UML?

- Unified Modeling Language
- Convergence of three leading OO methods:
  - OMT (James Rumbaugh)
  - OOSE (Ivar Jacobson)
  - Booch (Grady Booch)
- Supported by several CASE tools (e.g. Together)

UML and This Course

- You can model 80% of most problems by using about 20% UML
- In this course, we teach you those 20%
Why UML?

- Now the industry standard method for software engineering (design and documentation)
- When applied properly using the tool support it makes software engineering possible (‘round-trip engineering’)
- All design/documentation and implementation can really be integrated

UML: The basic diagrams

- Use Case diagrams
- Class diagrams
- Sequence diagrams
- Statechart diagrams
Use case diagrams represent the functionality of the system from user's point of view.

Class diagrams represent the structure of the system.
Sequence diagrams represent the behavior as interactions.

Statechart Diagrams describe dynamic behaviour of an individual object as a finite state machine.
Other UML Notations

- Activity diagrams
- Implementation diagrams
  - Component diagrams
  - Deployment diagrams
- Object Constraint Language (OCL)

Lesson Summary

- Good design is all about sensible system decomposition
- Crucial to get the right balance between low coupling and high cohesion
- UML provides a wide variety of notations for representing many aspects of software development
- We can concentrate only on a subset of the UML notations