Outline

- Use Case Modelling
  - Section 6.6 – 6.6.2 (pp. 154 – 159)

- Prototyping
  - Section 3.3.2 (pp. 73 – 76)
  - Section 6.6.3 (pp. 159 – 161)
Use Cases

- Use Cases – descriptions of the **functionality** of the system *from the users’ perspective*.
  - Use Case diagrams
    - show which users will communicate with the system
    - define the scope of the system
  - Use Case descriptions
    - specify the **interaction** between the users (actors) and the system for each use case *as the users see it*
    - could be further elaborated by communication/sequence diagrams.

*What are the key activities that make this business work?*
Use Case Diagrams

Staff Contact

Communication association

System or subsystem boundary

Actor

Use case

Change a client contact
Use Case Diagrams: Actor

- describe the role that people, other systems or devices take when communicating with a particular use case or use cases
  - not the same as job title or specific person
    - one job title may play the roles of several actors
    - one actor may represent several job titles
  - drawn as a stick figure with a name

Lecturer Dell Zhang  Role: Admission Tutor

Professor Mark Levene  Role: Research Tutor
Use Case Diagrams: Use Case

- describe a sequence of actions that the system performs to achieve an observable result of value to an actor
- drawn as a *bubble* (ellipse) with a name in or below
  - the name is usually an active verb and a noun phrase

Change a client contact
Use Case Diagrams: Communication Association

- describes the communication link between an instance of the use case and an instance of the actor
- drawn as a *line* between the actor and the use case

Staff Contact -> Change a client contact
Use Case Diagrams: Sub-system

- drawn as a rectangle around a group of use cases that belong to the same sub-system
  - When use cases for different sub-system are placed in separate use case diagrams, such rectangles are redundant

Change a client contact
Use Case Diagrams: Extend and Include relationships

The `extend` and `include` relationships between Use Cases are shown as stereotyped dependencies:

«extend» and «include» (text strings in guillemets)
Use Case Diagrams: «extend»

- One use case provides *additional* functionality that may be required in another use case.
- There may be multiple ways of extending a use case, which represent *variations* in the way that actors interact with the use case.
- The *extension points* show when the extension occurs.
- A *condition* can be placed in a note joined to the dependency arrow (Note that it is not put in square brackets, unlike conditions in other diagrams.)
Use Case Diagrams: «extend»

Campaign Manager

Check campaign budget

extension points
Summary print

«extend»

Print campaign summary

Condition {print option selected} extension point: Summary print
Use Case Diagrams: «include»

- One use case *always* includes the functionality of another use case.
- A use case may include more than one other.
- Can be used to separate out a sequence of behaviour that is used in many use cases.
- Should not be used to create a hierarchical functional decomposition of the system.
Use Case Diagrams: «include»

Campaign Manager

Assign staff to work on a campaign

«include»

Find campaign
Use Case Diagrams: Generalization

- Between use cases: shows that one use case provides all the functionality of the more specific use case and some additional functionality.
- Between actors: shows that one actor can participate in all the associations with use cases that the more specific actor can plus some additional use cases.
Use Case Diagrams: Generalization

Staff Contact

- Record completion of an advert
- Change a client contact
- Assign individual staff to work on a campaign
- Assign team of staff to work on a campaign

Campaign Manager

Assign staff to work on a campaign
Use Case Diagrams: Exercise

- Vending machine
Use Case Descriptions

- Using a simple paragraph

Assign staff to work on a campaign

The campaign manager wishes to record which staff are working on a particular campaign. This information is used to validate timesheets and to calculate staff year-end bonuses.
## Use Case Descriptions

- Using a step-by-step breakdown of interaction between actor and system

### Assign staff to work on a campaign

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The actor enters the client name.</td>
<td>2. Lists all campaigns for that client.</td>
</tr>
<tr>
<td>3. Selects the relevant campaign.</td>
<td>4. Displays a list of all staff members not already allocated to this campaign.</td>
</tr>
<tr>
<td>5. Highlights the staff members to be assigned to this campaign.</td>
<td>6. Presents a message confirming that staff have been allocated.</td>
</tr>
</tbody>
</table>

**Alternative Courses**

Steps 1–3. The actor knows the campaign name and enters it directly.
Use Case Descriptions

- Using a template
  - name of use case
  - pre-conditions
  - post-conditions
  - purpose
  - description
  - alternative courses (routes)
  - errors
Use Case Descriptions

- Case Study Problem 7.A
  - Use Case Description Example

**Stop line**
When the production line stops for a routine reason, e.g., for a break, to restock or to reload equipment, the line supervisor records the time the run stopped and the reason.
Use Case Descriptions

- Case Study Problem 7.A
  - Use Case Description Example

Record employee leaving the line
Normally employees are recorded as leaving the line when they clock off at the end of a working shift. Although there are breaks in the operation of the line during a shift (e.g., lunch breaks and downtime due to faults), when this happens employees are not normally recorded as leaving the line.
In all circumstances, date, time and location are recorded.
Prototyping

- A prototype is a system or a partially complete system that is built quickly to explore some aspects of the system requirements and that is not intended as the final working system.

- Prototyping can be used to support use case modelling
  - Help elicit *functional requirements*
  - Test out system architectures based on the use cases in order to meet the *non-functional requirements*
Prototyping

1. Initial analysis
2. Define objectives
3. Specify
4. Evaluate
5. Construct
6. Prototyping completed
Advantages of prototyping

- Early demonstrations of system functionality help identify any misunderstandings between developer and client
- Client requirements that have been missed can be identified
- Difficulties in the interface can be identified
- The feasibility and usefulness of the system can be tested, even though, by its very nature, the prototype is incomplete
Disadvantages of prototyping

- The client may perceive the prototype as part of the final system
- The prototype may divert attention from functional to solely interface issues
- Prototyping requires significant user involvement
- Managing the prototyping life cycle requires careful decision making
User Interface prototypes

- For user interface prototypes, storyboarding can be used with hand-drawn designs.
User Interface prototypes

- User interface prototypes can be implemented using languages other than the one that the system will be developed in, for example, Visual Basic.

Dialogue initialized.  
User selects Client. Campaigns listed.  
User selects Campaign.
Take Home Messages

- Use Case Modelling
  - Use Case Diagrams
  - Use Case Descriptions

- Prototyping