WIA2002: Software Modelling
Semester 1, Session 2016/17

Lecture 4: Requirements Analysis - UML Use Case Diagrams

Learning Objectives

- Know what is UML and their diagrams.
- Understand the purpose of use case diagrams.
- Understand the process used to identify business processes and use cases.
- Understand the notation of use case diagrams.
- Be able to draw use case diagrams.
- Be able to write use case descriptions.
- Understand the use of prototyping with use case modelling.

What is UML?

- UML - Unified Modeling language.
- UML is a modeling language, not a methodology or process.
- Fuses the concepts of the Booch, OMT, OOSE methods.
- Developed by Grady Booch, James Rumbaugh and Ivar Jacobson at Rational Software.
- Accepted as a standard by the Object Management Group (OMG), in 1997.

More on UML…

UML is a modeling language for visualising, specifying, constructing and documenting the artifacts of software systems.

More on UML…

- **Visualising** - a picture is worth a thousand words; a graphical notation articulates and unambiguously communicates the overall view of the system (problem-domain).

- **Specifying** - UML provides the means to model precisely, unambiguously and completely, the system in question.

- **Constructing** - models built with UML have a “design” dimension to it; these are language independent and can be implemented in any programming language.

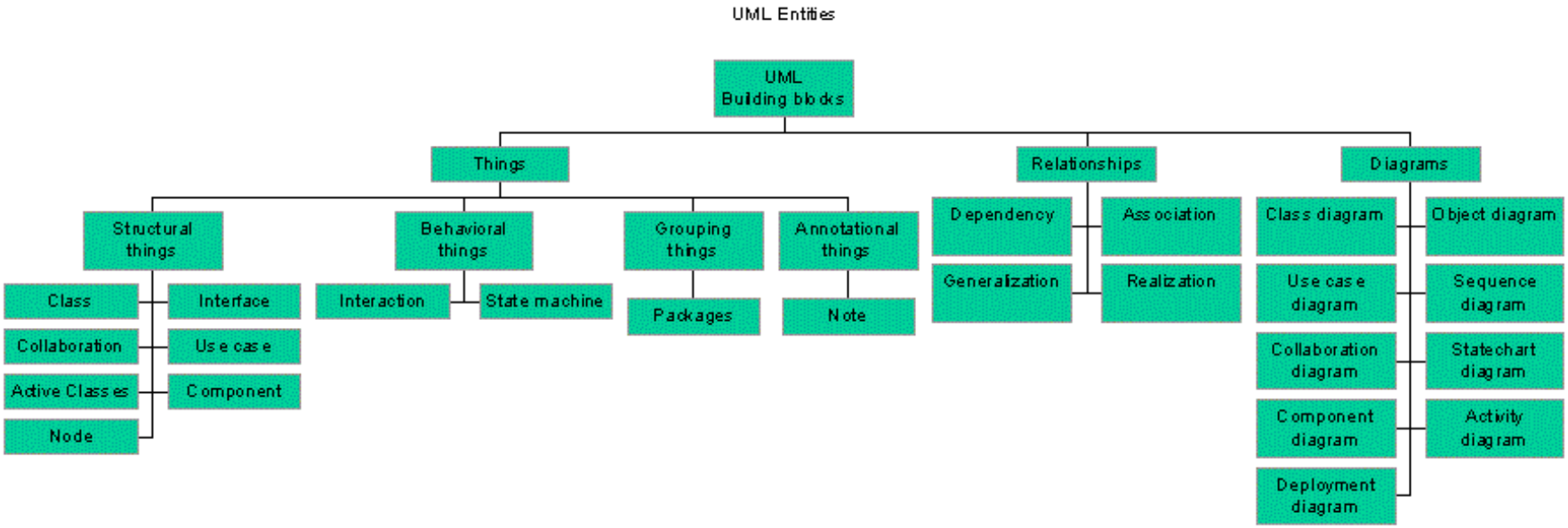
- **Documenting** - every software project involves a lot of documentation - from the inception phase to the deliverables.

3 steps to understanding UML

Understand:

1. The UML's basic building block.
2. The rules that dictate how those building blocks may be put together.
3. Some common mechanisms that apply throughout the UML.

1. UML Building Blocks



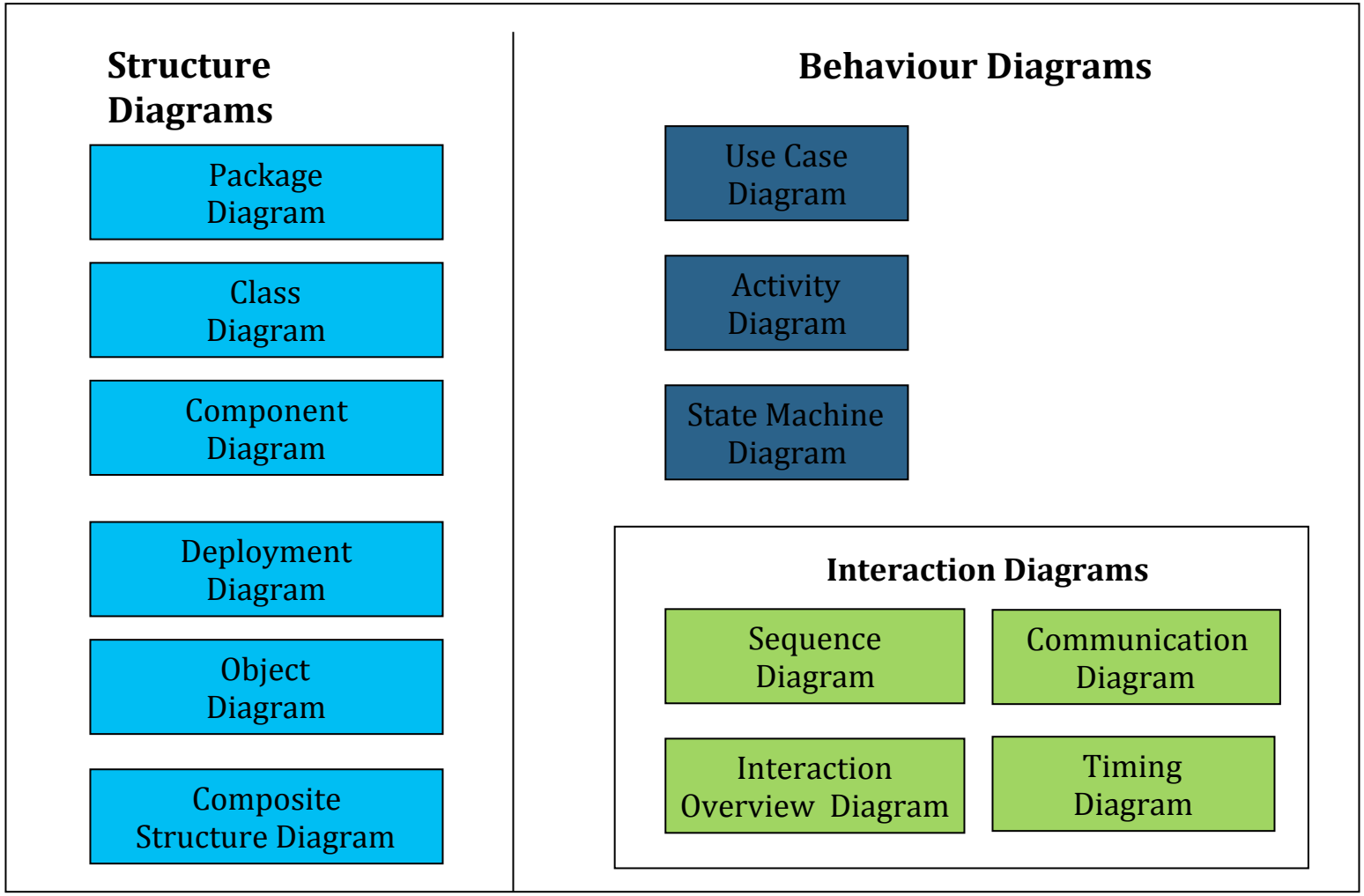
Diagrams

- The graphical presentation of the model. Represented as a connected graph - vertices (things) connected by arcs (relationships).

- UML 2.0 includes the following diagrams:
  - Class Diagram
  - Object Diagram
  - Use Case Diagram
  - Interaction Diagrams
    - Sequence Diagram
    - Communication Diagram
    - Timing diagram
    - Interaction overview diagram
  - State Machine Diagram
  - Activity Diagram
  - Component Diagram
  - Deployment Diagram
  - Package Diagram
- Each capturing a different dimension of a software-system.

Diagrams

- The taxonomy of structure and behaviour diagrams



2. Rules of the UML

- The UML has rules that ***specify how*** the building blocks can be put together in order to build a ***well-formed*** model.

- A well-formed model is one that is semantically self-***consistent*** and ***in harmony*** with all its related models.

- The UML has semantic rules for: -
  - Names : what we call things, relationships and diagrams
  - Scope : the context that gives specific meanings to names
  - Visibility : how those names can be seen and used by others
  - Integrity: how things properly and consistently relate to others.
  - Execution : What it means to run or simulate a dynamic model.

2. Rules of the UML

- The rules encourages you to address the most important issues in analysis, design and implementation in order to achieve a well-formed model.

3. Common mechanisms in the UML

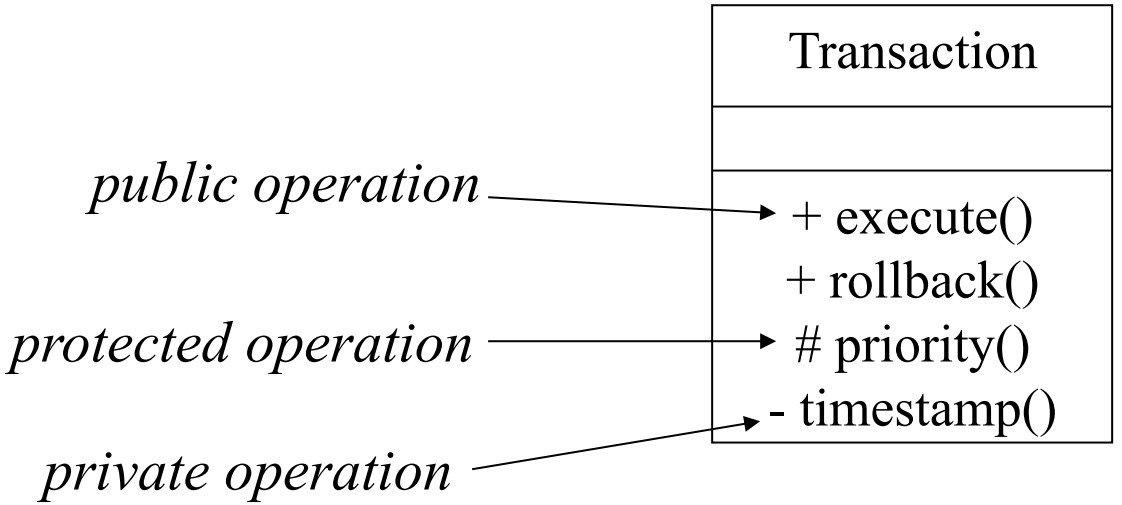
- UML is made simpler by the presence of 4 common mechanisms:
    a) Specifications
    b) Adornments
    c) Common divisions
    d) Extensibility mechanisms

a) Common Mechanism: Specifications

- Behind every part of UML's graphical notation, there is a specification that provides a textual statement of the syntax and semantics of that building block
  - UML's graphical notation is used to visualise a system.
  - UML's specification is used to state the system's details.

b) Common Mechanism: Adornments

- Most elements in UML have a unique and direct graphical notation that provides representation of the most important aspects of the element.
- Every element in the UML notation starts with a basic symbol, to which can be added a variety of adornments specific to that symbol.
- They are just ***added details.***

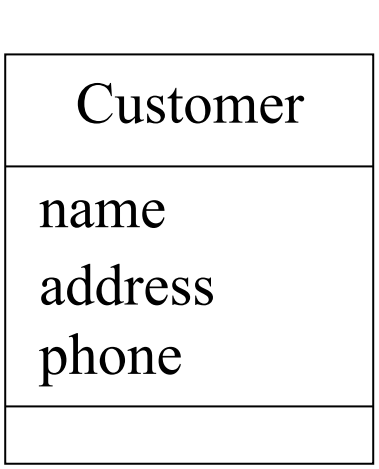


c) Common Mechanism: Common divisions

- In modeling object-oriented systems, the world often gets divided into a number of ways: -
    a) A division between class and object
    b) A separation between an interface and implementation

c) Common Mechanism: Common divisions

- A division between class and object
  - A class is an abstraction.
  - An object is one concrete manifestation of that abstraction.
  - UML distinguishes an object by using the same symbol as its class but by underlying the object's name.



Class

Jan - *a* Customer *object*

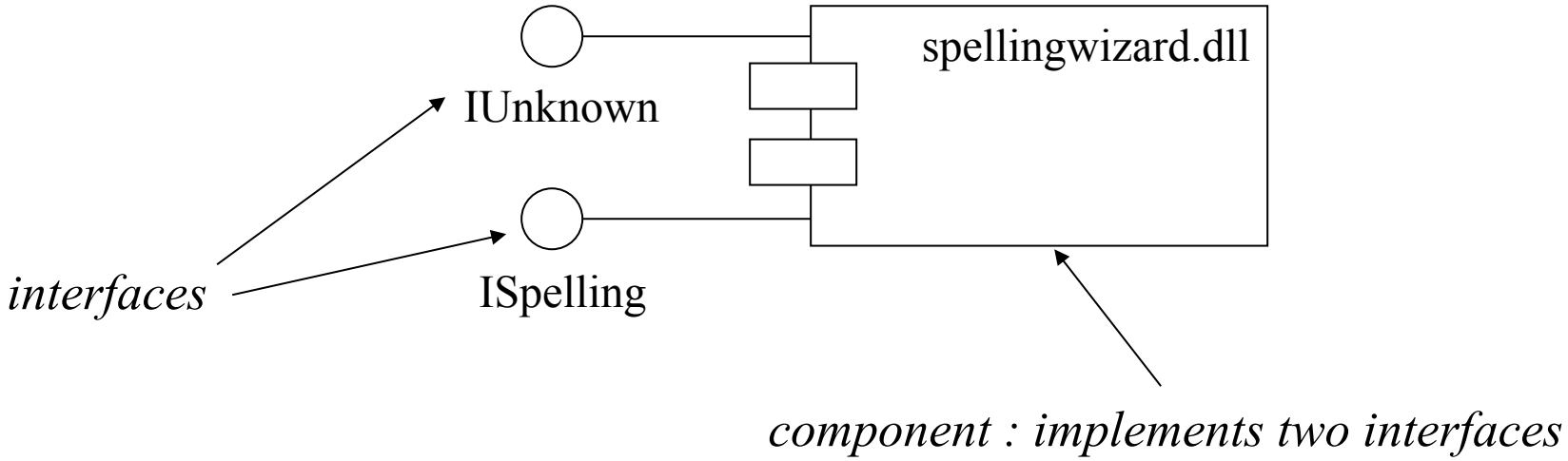
an anonymous Customer *object*

marked as a Customer *object in the specification*

Object

c) Common Mechanism: Common divisions

- Separation of interface and implementation
  - An interface declares a contract.
  - An implementation represents one concrete realization of that contract (is responsible for carrying out the interface's complete semantics).

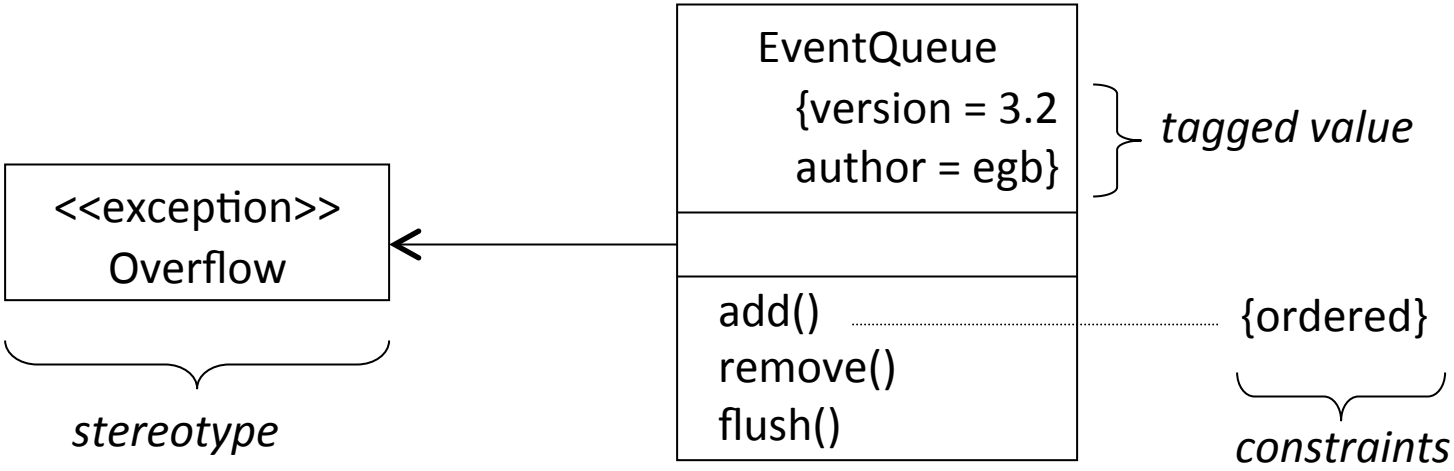


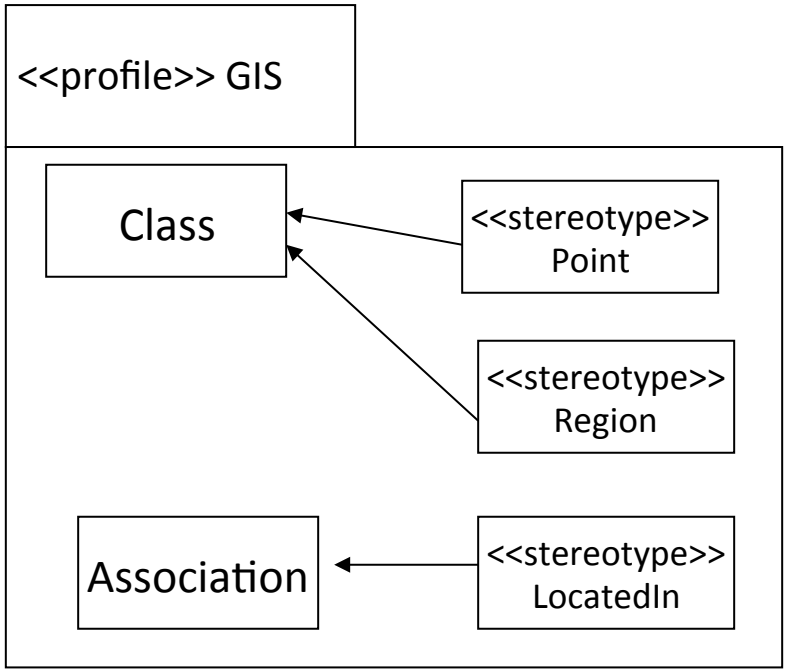
d) Common Mechanism: Extensibility mechanisms

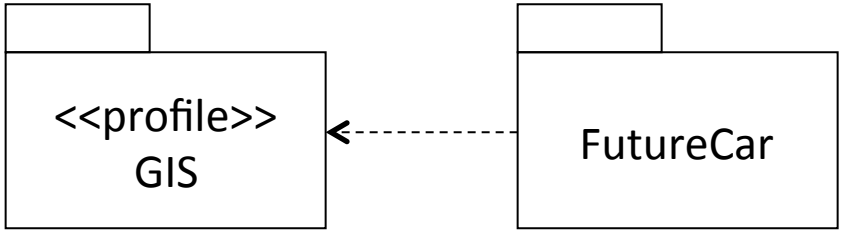
- UML is an open-ended language
  - Making it possible to extend the language in a controlled way.

- Four types of UML's extensibility mechanism: -
  - Profile
  - Stereotypes
  - Tagged values
  - Constraints

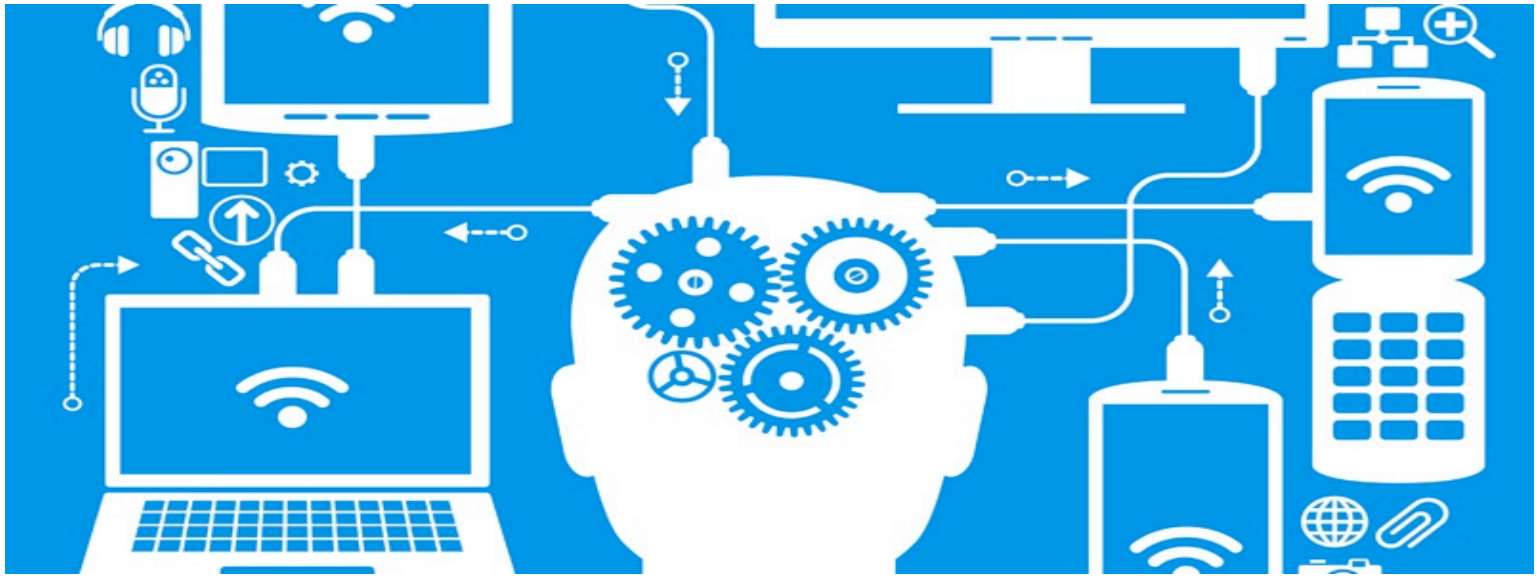
- Collectively, these four extensibility mechanisms allow the shaping and growing of the UML according to a project's needs


EventQueue
{version = 3.2
author = egb}
tagged value
<<exception>>
Overflow
add()
remove()
flush()
{ordered}
stereotype
constraints


<<profile>> GIS
Class
<<stereotype>>
Point
<<stereotype>>
Region
Association
<<stereotype>>
LocatedIn


<<profile>>
GIS
FutureCar

Any question so far..?



Use case Diagram

- Shows a set of use cases and actors and their relationship.
- It addresses the static use case view of a system.
- Important in modeling the behavior of a system

Purpose of Use Case Diagrams

- Document the functionality of the system from the users' perspective.
- Document the scope of the system.
- Document the interaction between the users and the system using supporting use case descriptions (behaviour specifications)

Use Case Diagram

- An analyst can employ use cases and the use-case diagram to better understand the functionality of the system at a very high level.

- A use-case diagram provides a simple, straightforward way of communicating to the users exactly what the system will do.

- A use-case diagram is drawn when gathering and defining requirements for the system
  - Can encourage the users to provide additional high-level requirements.

Use Case Diagram

- All object-oriented systems development approaches are use-case driven, architecture-centric, and iterative and incremental:
  - From a **practical perspective**, use cases represent the entire basis for an object-oriented system. Use cases can document the current system or the new system being developed.
  - From an **architecture-centric perspective**, use-case modeling supports the creation of an external or functional view of a business process in that it shows how the users view the process.
  - All object-oriented systems development approaches are **developed in an incremental and iterative manner.**

Use Case Diagram

- A use-case diagram illustrates in a very simple way the main functions of the system and the different kinds of users that will interact with it.

- Given that object-oriented systems are use-case driven, use cases also form the foundation for testing and user-interface design.

Types of Requirements

1. Functional Requirements

- Describe what a system must do.
- Include:
  - processes
  - interfaces with users and other systems
  - what the system must hold data about

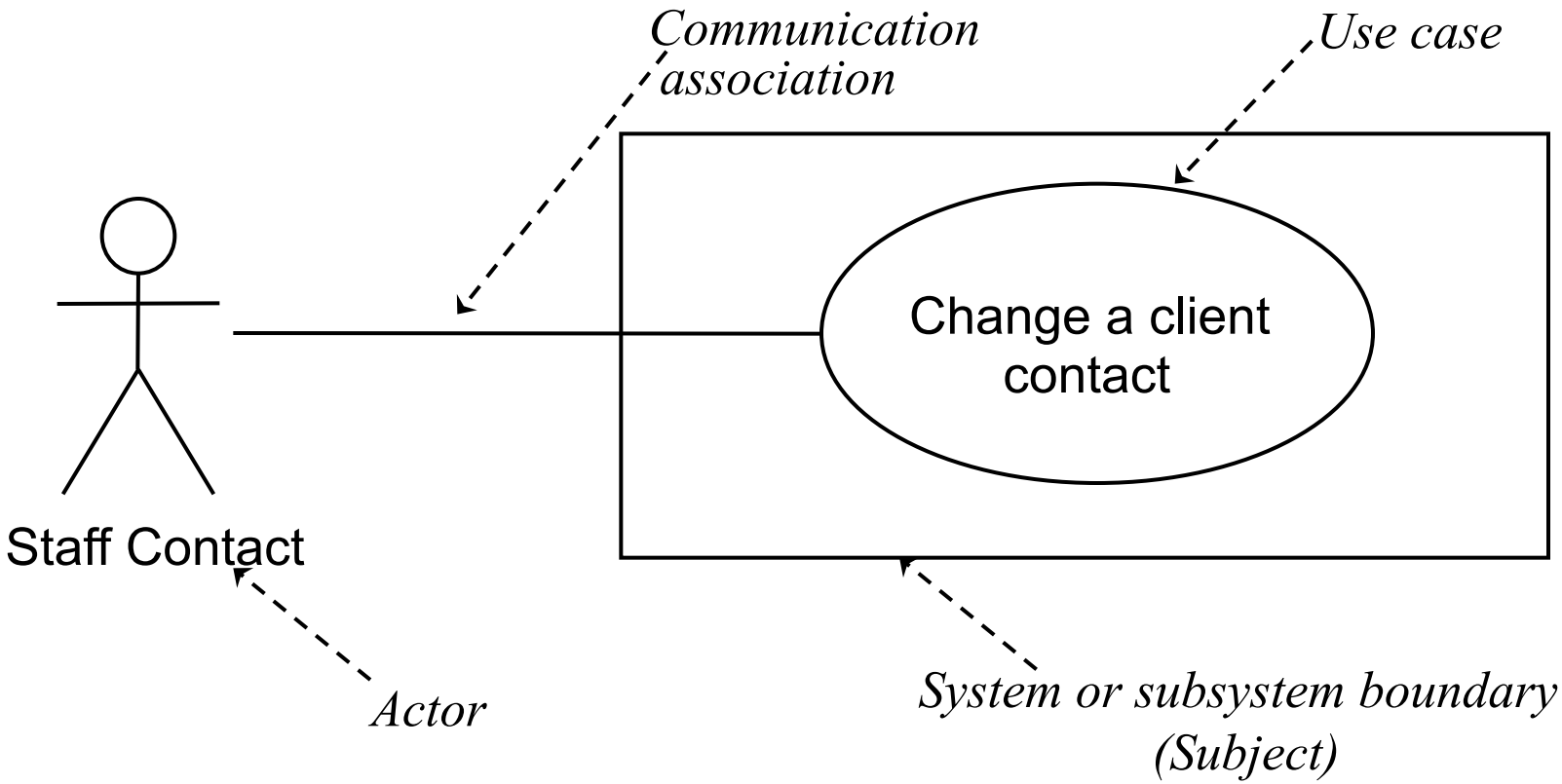
- Modelled with Use Case Diagrams. Later will be modelled with other kinds of diagrams that show the structure of the system (Class Diagrams) and its behaviour (Interaction Diagrams and State Machines)

Types of Requirements

2. Non-Functional Requirements

- Concerned with how well the system performs
- Example:
  - response times
  - volumes of data
  - security considerations
- Documented in Requirements List or in Use Case Model (for requirements that can be linked to specific use cases)

Notation of Use Case Diagrams



Notation of Use Case Diagrams

- An actor
  - Drawn as stick people with a name.
  - Shows the roles that people, other systems or devices take when communicating with a particular use case or use cases.
  - Not the same as job titles or people.
  - People with one job title may play the roles of several actors
    - one actor may represent several job titles.
  - Placed outside the subject boundary.

Notation of Use Case Diagrams

- A use case:
  - Drawn as ellipses with a name in or below each ellipse.
  - Represents a major piece of system functionality.
  - Describes a sequence of actions that the system performs to achieve an observable result of value to an actor.
  - The name is usually an active verb and a noun phrase.
  - Placed inside the system boundary.

Notation of Use Case Diagrams

- A communication/relationship association
  - A line drawn between an actor and a use case.
  - Links an actor with the use case(s) with which it interacts.

- A subject (system or subsystem) boundary
  - Drawn as a rectangle around a group of use cases that belong to the same subject.
  - Represents the scope of the subject, e.g., a system or an individual business process.
  - In a CASE tool, use cases for different subjects are usually placed in separate use case diagrams.

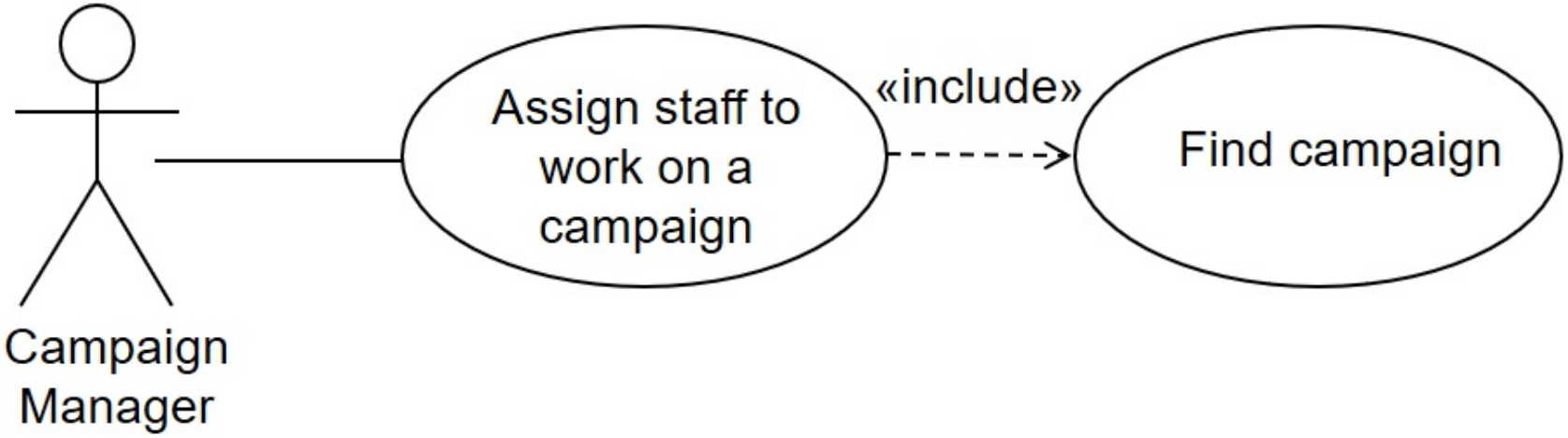
Notation of Use Case Diagrams

- Dependencies
  - Extend and Include relationships between use cases.
  - Shown as stereotyped dependencies.
  - Stereotypes are written as text strings in guillemets: «extend» and «include».

Notation of Use Case Diagrams

- Include relationship
  - Used when one use case **always** includes the functionality of another use case.
  - Represents the inclusion of the functionality of one use case within another.
    - a use case may include more than one other use cases.
  - 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.
  - Has an arrow drawn from the base use case to the used use case.

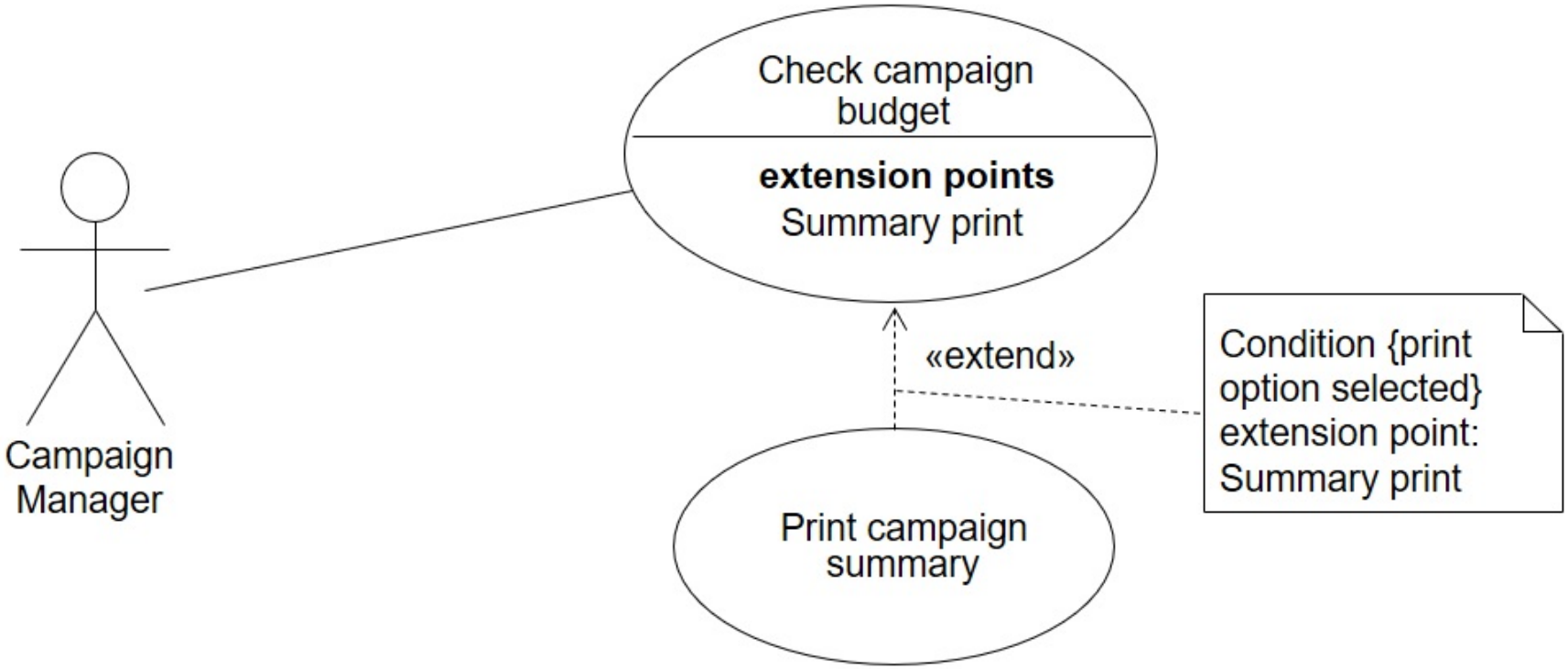
Include relationship



Notation of Use Case Diagrams

- Extend relationship
  - Used when one use case provides additional functionality that **may** be required in another use case.
  - Represents the extension of the use case to include *optional behaviour.*
  - There may be multiple ways of extending a use case, which represent variations in the way that actors interact with the use case.
  - Extension points show when the extension occurs.
  - A condition can be placed in a note joined to the dependency arrow.
  - Has an arrow drawn from the extension use case to the base use case.

Extend relationship



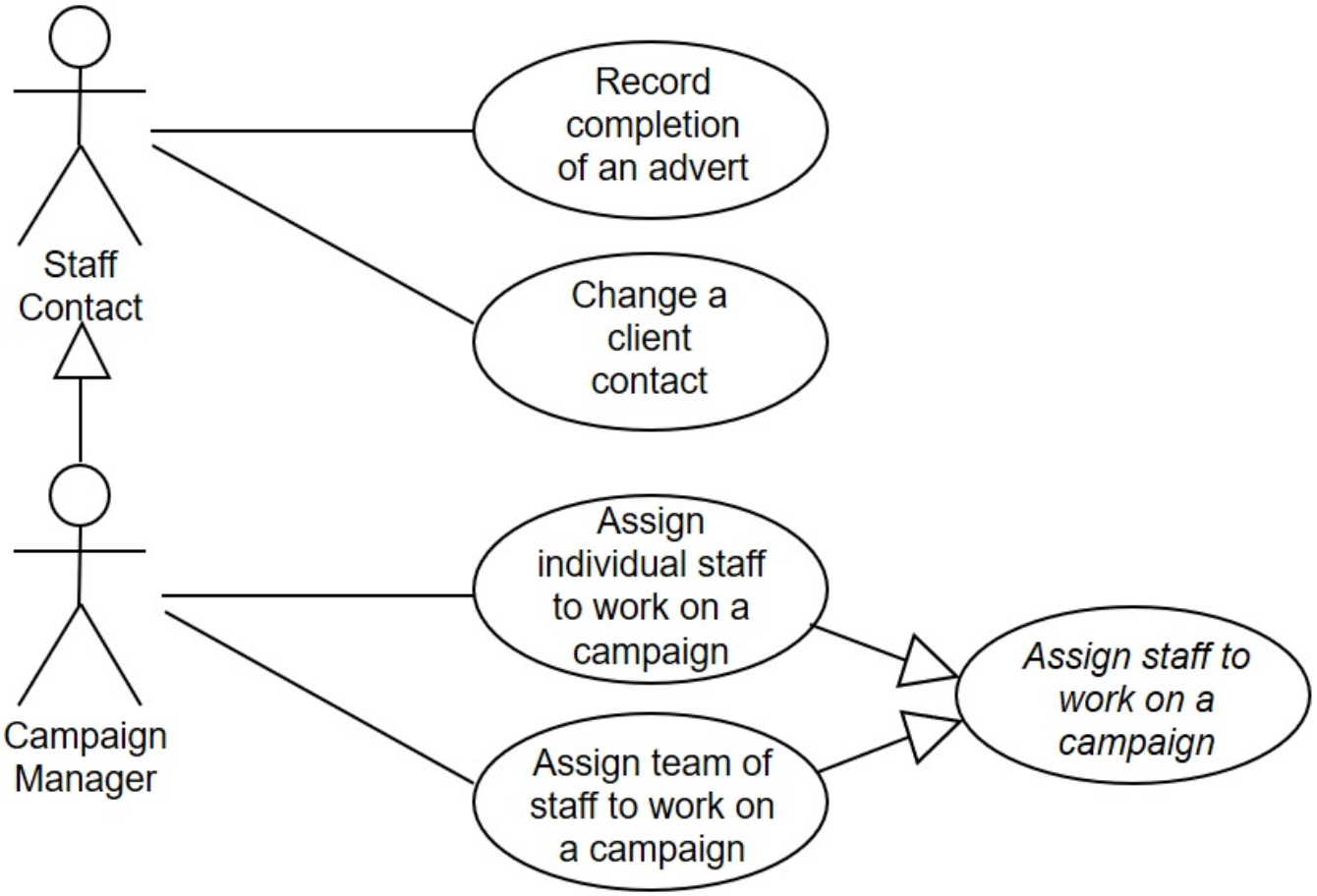
Notation of Use Case Diagrams

- **Generalization** - using a specialization/ superclass association.

- A use case can be associated with other use cases using generalization:
  - to show that one use case provides all the functionality of the more general use case and some additional functionality.

- An actor can be associated with other actors using generalization
  - To show that one actor can participate in all the associations with use cases that the more general actor can plus some additional use cases.

Generalization relationship



Use Case Descriptions

- Can be a simple paragraph.

- Example:

  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

- Can be a step-by-step breakdown of interaction between actor and system
- Example:

Assign staff to work on a campaign

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

Alternative Courses
Steps 1–3. The actor knows the campaign name and enters it directly.

Use Case Descriptions

- Many projects use templates
  - name of use case
  - pre-conditions
  - post-conditions
  - purpose
  - description
  - alternative courses
  - errors

Creating Use Case Descriptions

1. Choose a use case to document with a use case description.
2. Create an overview description of the use case.
   - e.g. primary actor, set the type for the use case, list all of the identified stakeholders, level of importance, brief description, trigger information, relationship
3. Describe the normal flow of events.
4. Check the normal flow of events (not too long/complex).
5. Identify alternative or exceptional flows.
6. Review the use case description.

Behaviour Specifications

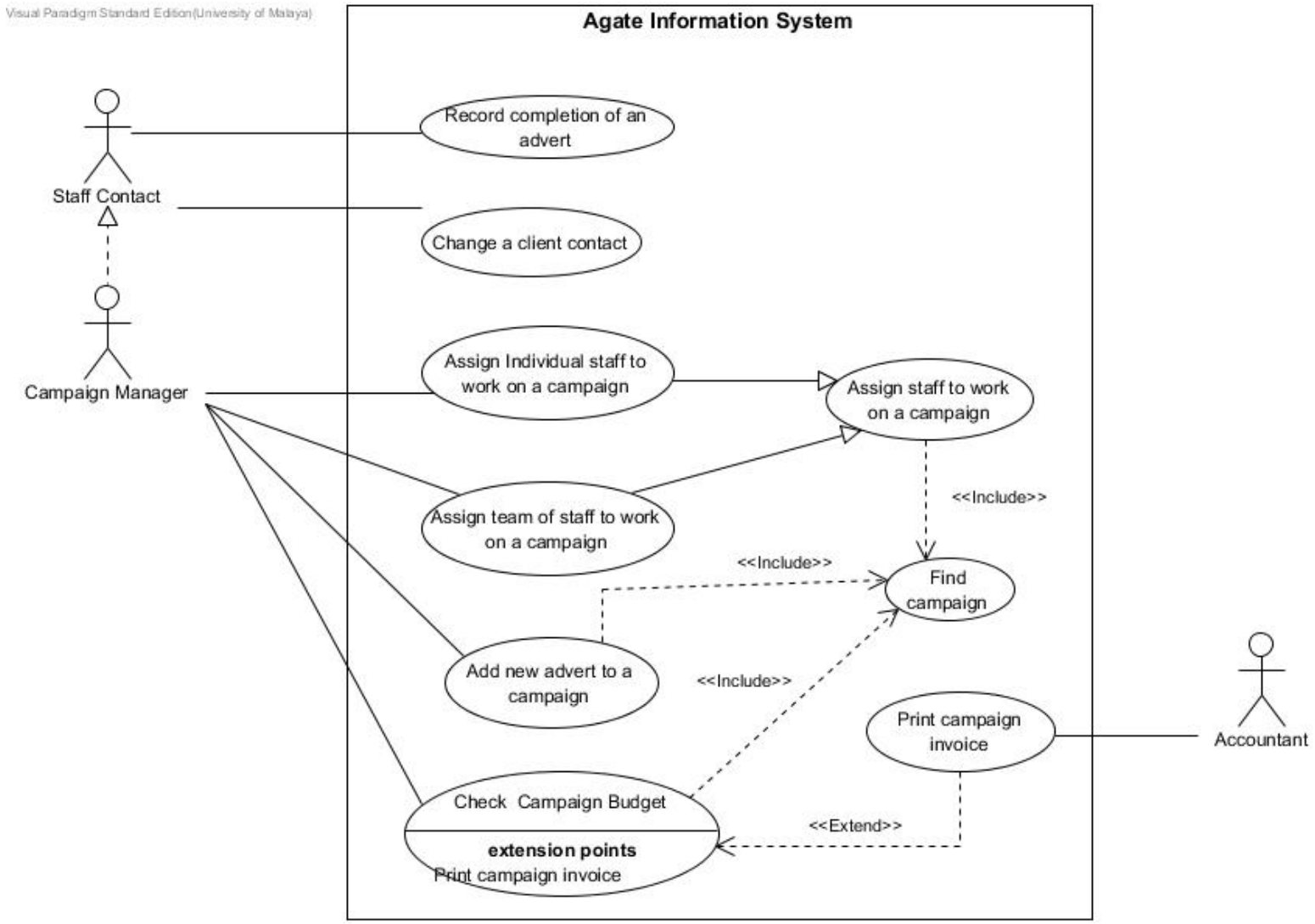
- Rather than (or as well as) using text, a use case can be linked to another diagram that specifies its behaviour.

- Typically a Communication Diagram, a Sequence Diagram, a State Machine or more than one of these.

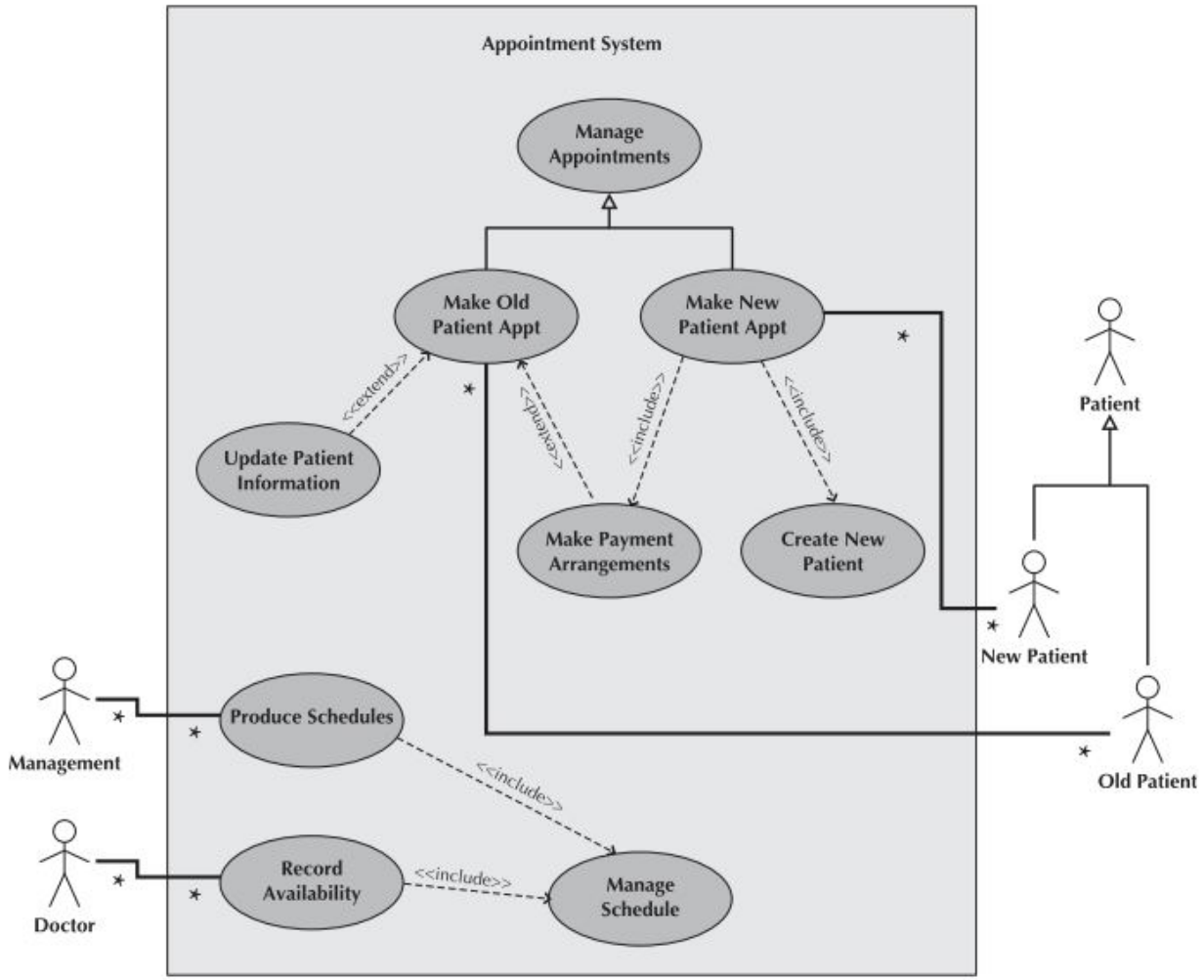
Drawing Use Case Diagrams

1. Review requirements definition
2. Identify subject's boundaries
3. Identify the primary actors and the use cases
4. Prioritize the use cases
5. Develop each use case, starting with the priority ones, writing a description for each
6. Add structure to the use case model: generalization, include and extend relationships and subsystems

Sample Use Case Diagram



Sample Use Case Diagram



Prototyping

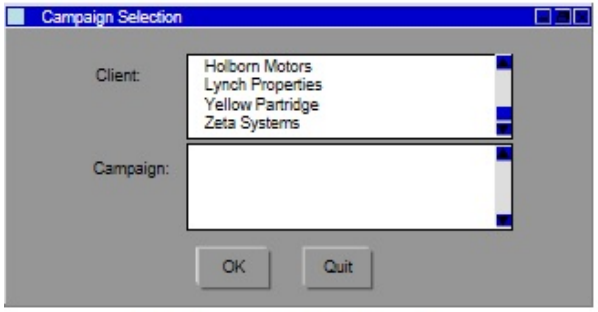
- Use case modelling can be supported with prototyping.
- Prototypes can be used to help elicit requirements.
- Prototypes can be used to test out system architectures based on the use cases in order to meet the non-functional requirements.

Prototyping

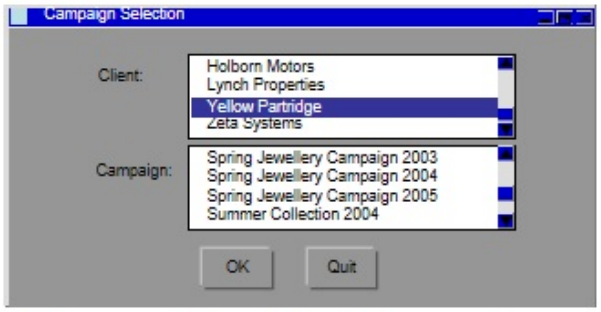
- For user interface prototypes, storyboarding can be used with hand-drawn designs

Prototyping

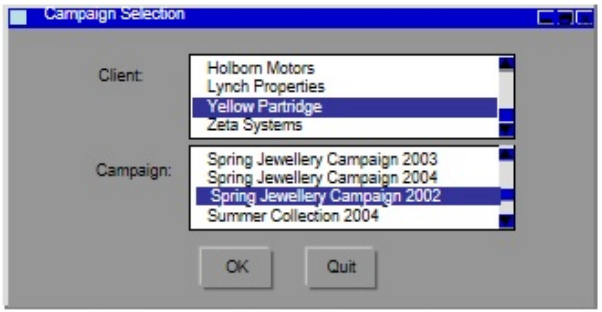
- User interface prototypes can be implemented using languages other than the one that the system will be developed in



Dialogue initialized.



User selects Client. Campaigns listed.



User selects Campaign.

Key points

- UML is a modeling language for visualising, specifying, constructing and documenting the artifacts of software systems.

- Understand:
  1. The UML's basic building block.
  2. The rules that dictate how those building blocks may be put together.
  3. Some common mechanisms that apply throughout the UML.

- Use case diagram is used to capture functionality, scope and interaction between users of the system.

Key points

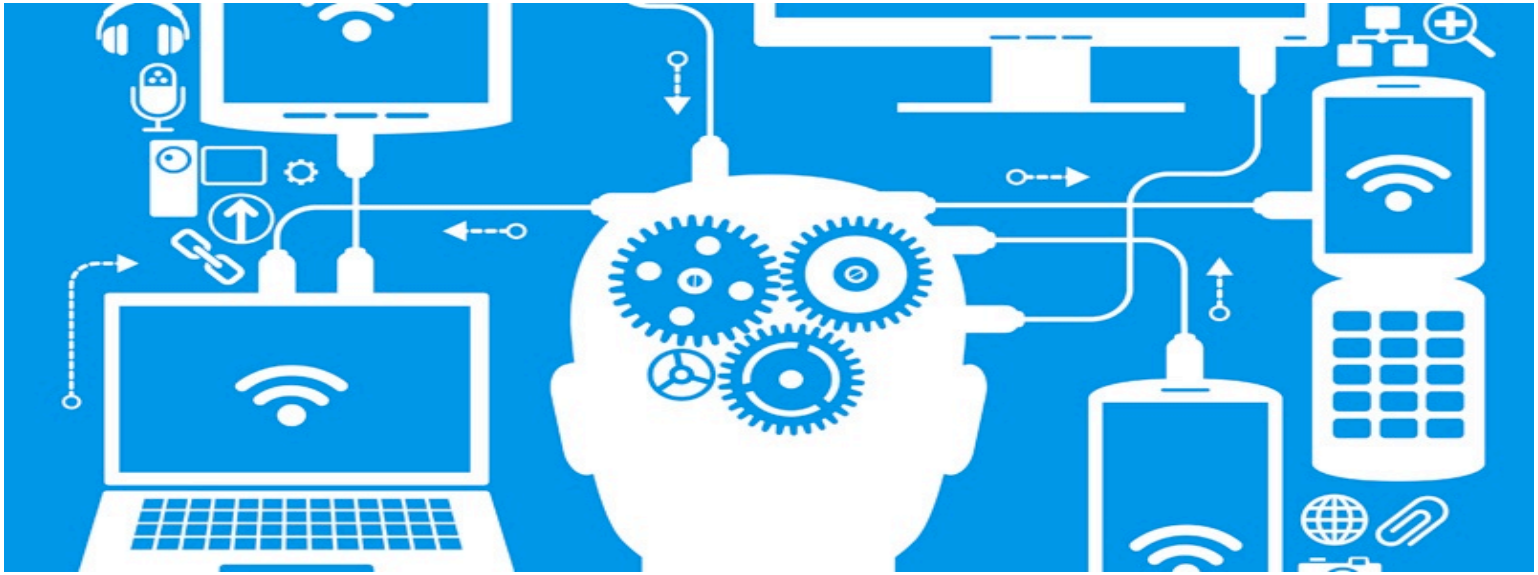
- The notation of use case diagrams consist of actors, communication associations, use cases and system/ subsystem boundary.
- Use case diagrams also includes dependencies (include/ extend) and generalization.
- Use case descriptions is used to describe use case.
- Prototyping can be used with use case modelling to help elicit requirements.

References

- Alan Dennis, Barbara Haley Wixom & David Tegarden. 2015. Systems Analysis and Design with UML, 5th edition, Wiley.

- Simon Bennett, Steve McRobb & Ray Farmer. 2010. Object Oriented Systems Analysis and Design using UML 4th Edition, McGraw-Hill.

In the next lecture..



Lecture 5: UML Class Diagrams