THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

SENG2130 – Week 2
Introduction to UML
Use Case diagram

Dr. Joe Ryan

SENG2130 – Systems Analysis and Design

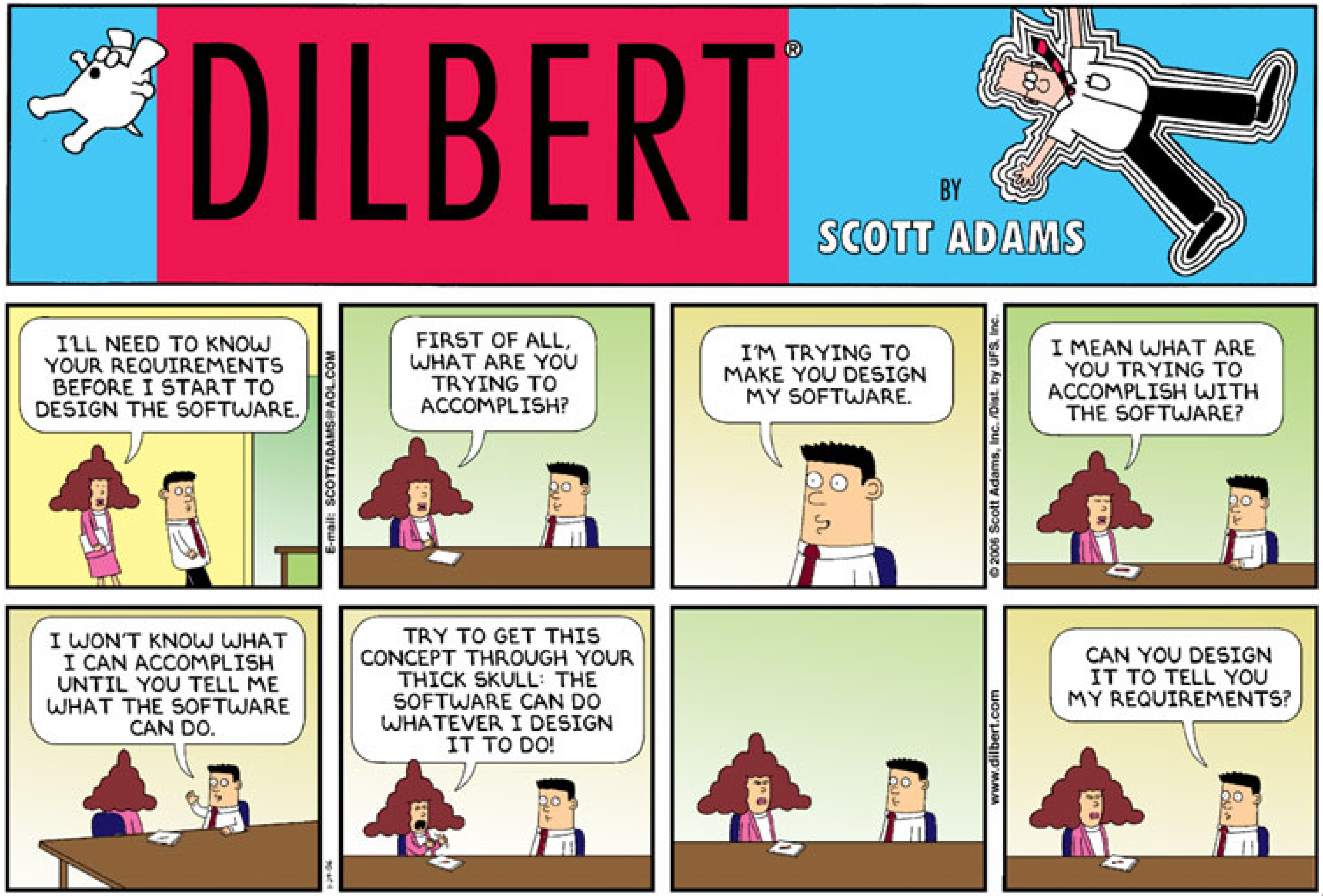
University of Newcastle

Outline of this class

- Requirements
  - (specification today, elicitation Week 3)
- What is UML?
- UML: a more detailed view on
  - Use case diagrams / Use case description
  - Activity diagrams (week 3)
  - Class diagrams (week 4)
  - Sequence diagrams (week 5)

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Requirements


DILBERT®
BY SCOTT ADAMS
I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?
E-mail: SCOTTADAMS@AOL.COM
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
www.dilbert.com

review

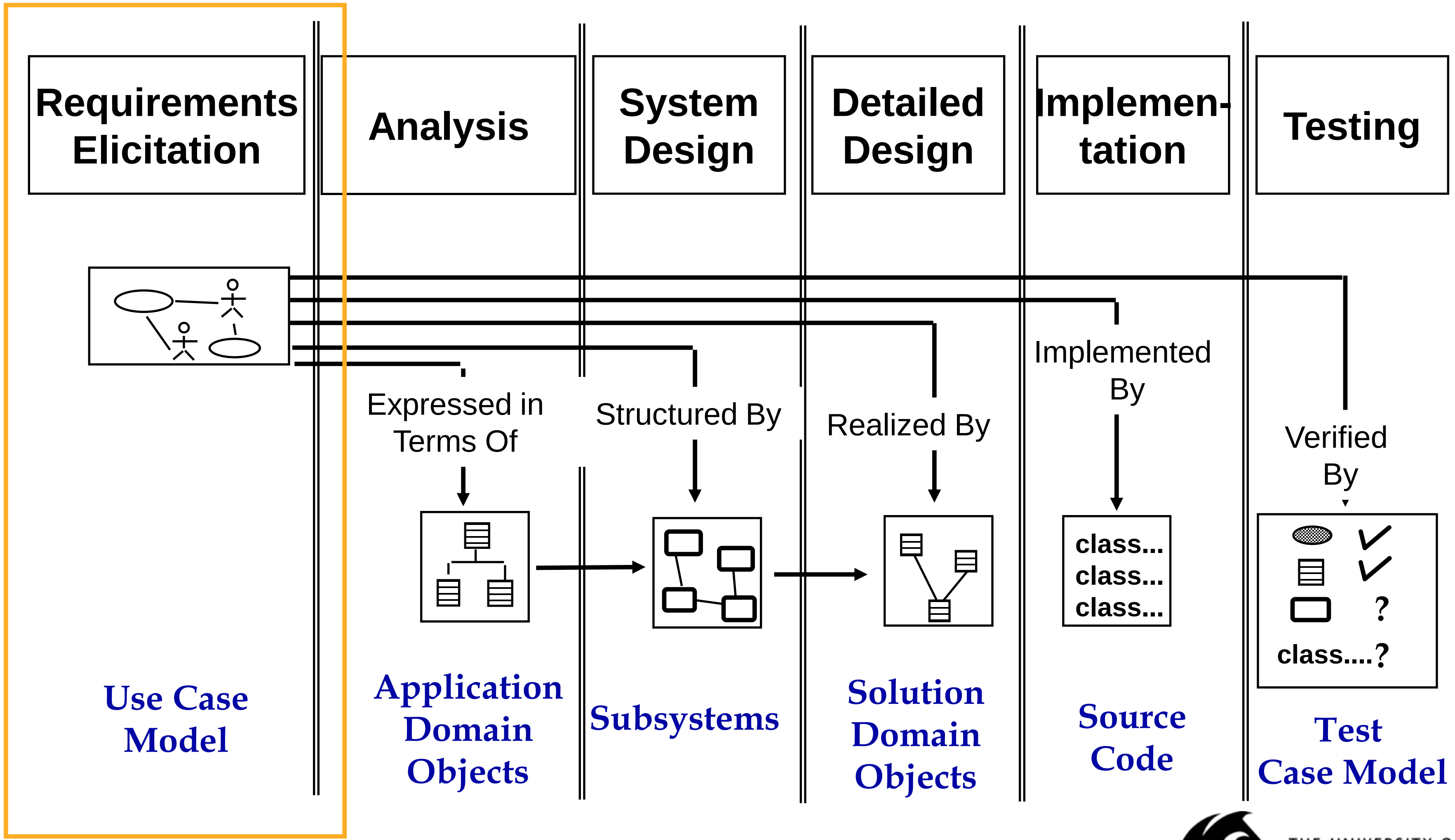
What is the best Software Lifecycle?

- Answering this question was the topics of last week's lecture on software lifecycle modeling
- For now we assume we have a set of predefined activities:

review

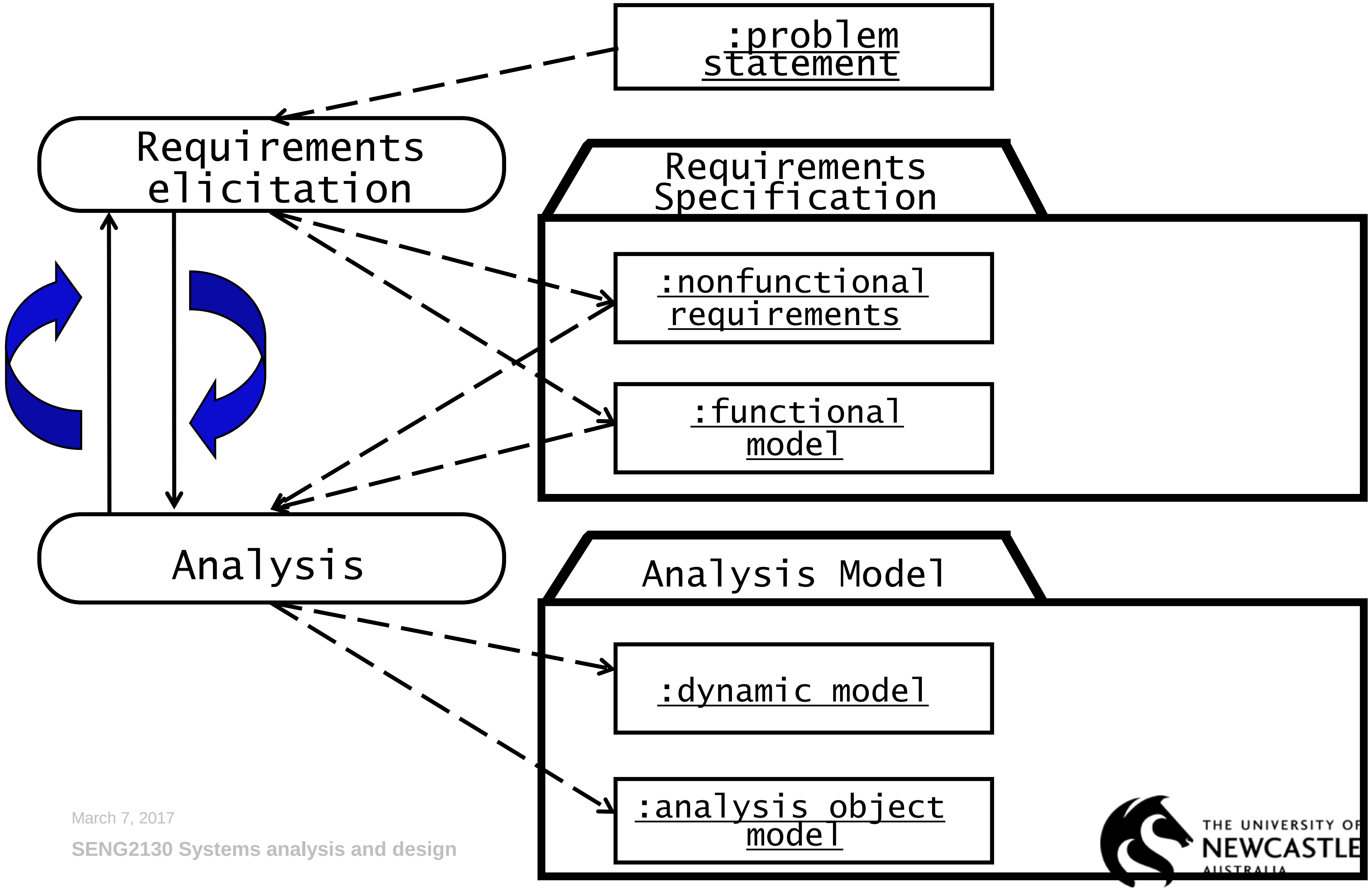
Software Lifecycle Activities...and their models

Today we focus on the activity requirements



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Requirements Process



What is Requirement Engineering?

- = Requirements elicitation (week 3)
  + Requirements specification (this week)

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.

- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

Requirements specification

- The requirements specification is the official statement of what is required of the system developers

- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it

Stakeholders of a Requirements specification

System customers	• Specify the requirements and read them to check that they meet their needs. They specify changes to the requirements.
Managers	• Use the requirements specification to plan a bid for the system and to plan the system development process.
System developers	• Use the requirements to understand what system is to be developed.
System testers	• Use the requirements to develop validation tests for the system.
System maintainers	• Use the requirements to help understand the system and the relationship between its parts.

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Types of Requirements

- User requirements and System requirements
- Functional and non-functional requirements

User vs. System requirements

User Requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints.
- Should describe **functional** and **non-functional** requirements in such a way that they are understandable by system users who do not have detailed technical knowledge.

System Requirements

- More detailed specifications of system functions, services and constraints than user requirements.
- Defines what should be implemented so may be part of a contract between client and contractor.
- **Functional** Requirements
  - What the system does
- **Non-functional** Requirements
  - How well the system does it

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Functional vs. non-functional requirements

Functional Requirements	Non-functional Requirements
• Describe user tasks that the system needs to support • Phrased as actions • “Advertise a new league” • “Schedule tournament” • “Notify an interest group”	• Describe constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, etc. • Phrased as constraints • “All user inputs should be acknowledged within 1 second” • “A system crash should not result in data loss”.

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.

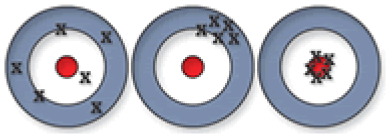
Functional requirements example

Online searchable document database

1. The user shall be able to search either all of the initial set of databases or select a subset from it.

2. The system shall provide appropriate viewers for the user to read documents in the document store.

3. Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area.



Requirements imprecision

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term 'appropriate viewers'
  - User intention
    - special purpose viewer for each different document type.
  - Developer interpretation
    - Provide a text viewer that shows the contents of the document.

Non-functional requirements categories

Usability

Easy to use

- A user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
- Must be measurable
- e.g. Specification of the number of steps – the measure! - to perform a internet-based purchase with a web browser

Reliability

Robustness, Safety

- Robustness: The ability of a system to maintain a function
  - even if the user enters a wrong input
  - even if there are changes in the environment
- Safety: The ability of the system to operate without catastrophic failure

Nonfunctional requirements categories

Performance

Response time, Throughput, Availability

- Response time: how quickly the system reacts to a user input
- Throughput: how much work the system can accomplish within a specified amount of time
- Availability: The ratio of the expected uptime of a system to the aggregate of the expected up and down time
- e.g. the system is down not more than 5 minutes per week.

Supportability

Adaptability, Maintainability, Portability

- Adaptability: the ability to change the system to deal with additional application domain concept
- Maintainability: the ability to change the system to deal with new technology or to fix defects
- Portability: the ease with which a system or component can be transferred from one hardware or software environment to another

Non-functional requirements examples

- "Spectators must be able to watch a match without prior registration and without prior knowledge of the match."
  - ➢ *Usability Requirement*
- "The system must support 10 parallel tournaments"
  - ➢ *Performance Requirement*
- "The operator must be able to add new games without modifications to the existing system."
  - ➢ *Supportability Requirement*

Requirements interaction

- Conflicts between different non-functional requirements are common in complex systems.

- Example: Spacecraft system
  - To minimise weight, the number of separate chips in the system should be minimised.
  - To minimise power consumption, lower power chips should be used.
  - However, using low power chips may mean that more chips have to be used.
  - Which is the most critical requirement???

  Prioritizing requirement: High, Medium, Low priority

Requirements Validation

Requirements validation is a quality assurance step, usually performed after requirements elicitation or after analysis

- Correctness: The requirements represent the client's view
- Completeness: All possible scenarios, in which the system can be used, are described
- Consistency: There are no requirements that contradict each other.
- Clarity: Requirements can only be interpreted in one way
- Realism: Requirements can be implemented and delivered
- Traceability: Each system behavior can be traced to a set of functional requirements

Requirements Validation

- Problems with requirements validation:
  - Requirements change quickly during requirements elicitation
  - Inconsistencies are easily added with each change
  - Tool support is needed!
    - One of the requirements validation techinques
    - Reviews and inspections
      - Walkthroughs
      - Formal inspections
      - Checklists

IEEE requirements standard

- Defines a generic structure for a requirements document that must be instantiated for each specific system.
  - Introduction.
  - General description.
  - Specific requirements.
  - Appendices.
  - Index.

Key points

- Requirements set out what the system should do and define constraints on its operation and implementation.
- Functional requirements set out services the system should provide.
- Non-functional requirements constrain the system being developed or the development process.
- The IEEE standard is a useful starting point for defining more detailed specific requirements standards.
- A software requirements specification (this week) is an agreed statement of the system requirements.
- A software requirements elicitation (next week) is achieved by using UML diagram.

UML

Why UML?

- Requirements Documentation detailing description of system
  - Document forms “contract” with client
  - Discussions focus upon document
- Result:
  - Large legalistic documents
  - Easy to misinterpret
  - Changes hard to manage
  - Easy to miss / omit requirements
- Modern approach – Model using UML
  - Use cases are used to capture functional requirements (next week)

So, we use Models to describe software systems

- Object model: What is the structure of the system?
- Functional model: What are the functions of the system?
- Dynamic model: How does the system react to external events?

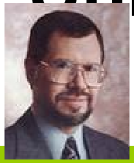
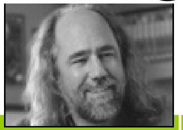
- System Model: Object model + functional model + dynamic model

UML

What is UML? Unified Modelling Language

- UML
  - Is a modelling language for documenting Object Oriented analysis and design
    - Determine what to do
    - Extract Objects (modules)
    - Determine how to build

What is UML? Unified Modelling Language

	James Rumbaugh	Grady Booch	Ivar Jacobson
Born	1941	1955	1939
Degree	Physics (MIT) Astronomy (Caltech) Computer science (MIT)	Electronic engineering	Electronic engineering
Title	Object Methodologist	Software Engineer	Computer Scientist
	Object Modelling Technique (OMT) – General Electric +	Booch Method- Rational software +	Object Oriented Software Engineering (OOSE) – Objectory AB
Rational Unified Process (RUP) - Rational software	1994-2002	~2002	1995-2002
Unified Modelling Language (UML) -- IBM	2003-2006	2003 - 2008	2003 - 2004

UML

- Notation standard defined by the Object Management Group - OMG
- Current Version: UML 2.5
  - Information at the OMG portal http://www.uml.org/
- Commercial tools:
  - Rational Rose (IBM),Together (Borland), Visual Architect (Visual Paradigm), Enterprise Architect (Sparx Systems)
- Open Source tools http://www.sourceforge.net/
  - ArgoUML, StarUML, Umbrello (for KDE)
- Example of research tools: Unicase, Sysiphus
  - Based on a unified project model for modeling, collaboration and project organization
  - http://unicase.org

UML

- is a standard language for writing software blueprints.
  - It may be used to visualize, specify, construct and document the artefacts of a software-intensive system.
  - Used to enable to articulate complex ideas succinctly and precisely, in projects involving participants with different technical and cultural backgrounds,
  - thus reducing the chances of miscommunication.

UML

- UML provides a wide variety of notations for modeling many aspects of software systems
- Our course concentrates on:
  - Functional model: Use case diagram
  - Object model: Class diagram, Object diagram
  - Dynamic model: Sequence diagram, Activity diagram
- Now we go into a little bit more detail…

UML

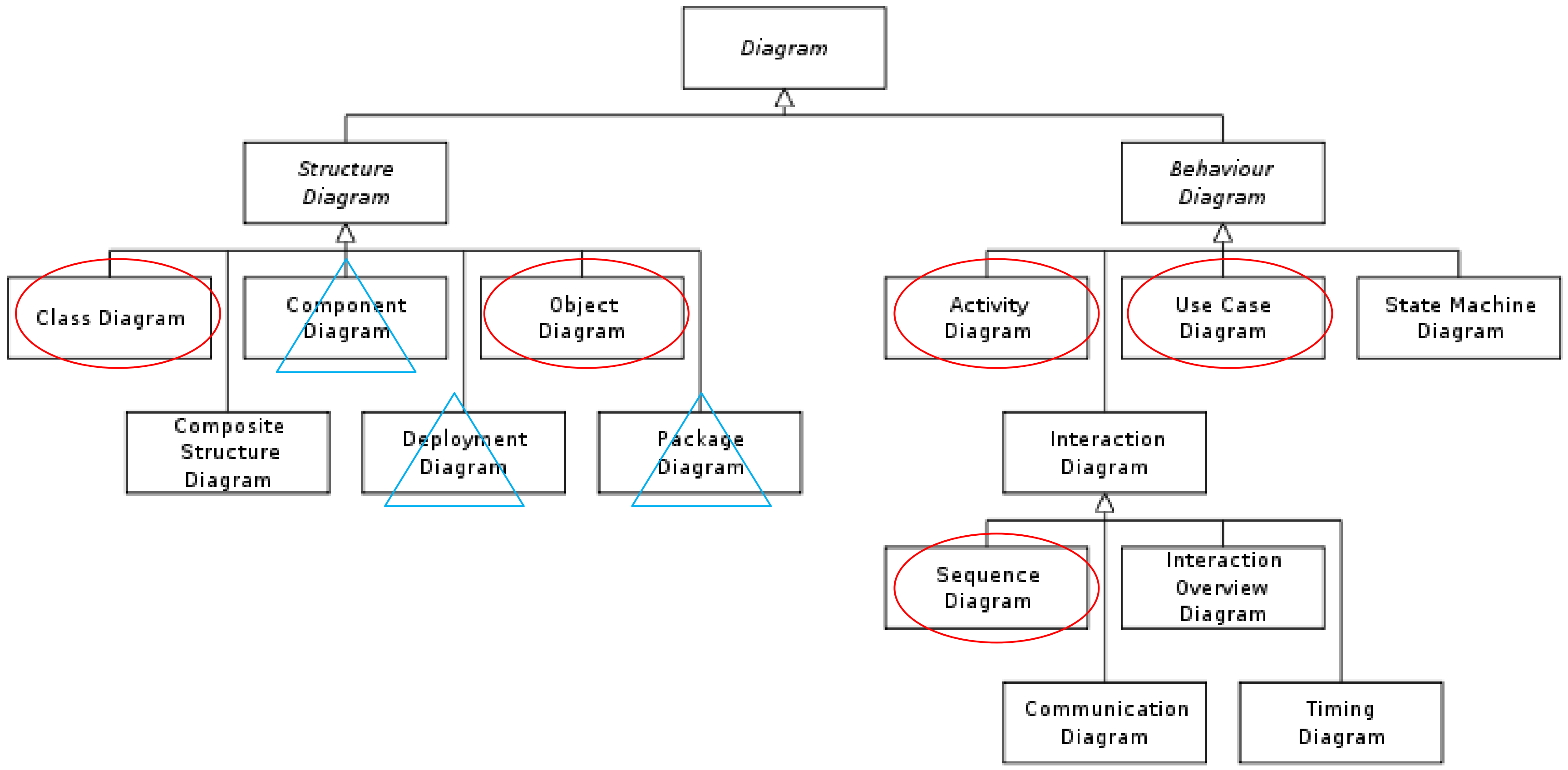


Diagram
Structure Diagram
Behaviour Diagram
Class Diagram
Component Diagram
Object Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Interaction Overview Diagram
Communication Diagram
Timing Diagram

UML

- The vocabulary of the UML encompasses three kinds of building blocks:
  - Things (4) – structural, behavioural, grouping, annotation
    - are the abstractions that are first-class citizens in a model
  - Relationships (4) – tie these *things* together
    - A relationship is a connection between two or more UML model elements (*things*) that adds semantic information to a model
  - Diagrams (9) – Use case, Activity, Class, Sequence, Object, Collaboration, Statechart, Component, Deployment
    - group interesting collection of *things*

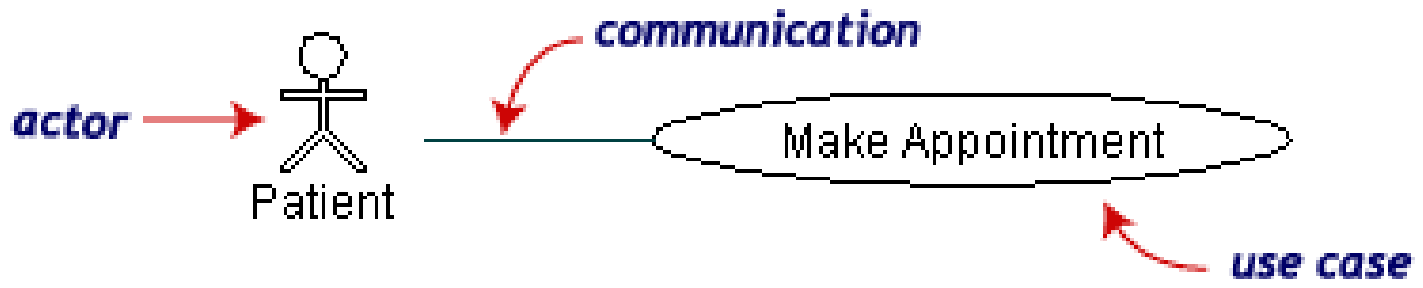
Why so many diagrams?

- The collection of diagrams make it possible to examine a system from a number of viewpoints
  - Not all diagrams must appear in every UML model
- Different stakeholders have different views
- UML provides a common language and mapping between stakeholders, and their viewpoints, in order to define accurate requirements and support good design

Use Case Diagrams

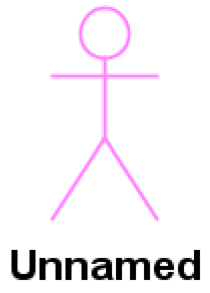
Use case diagrams

- A use case diagram describes what a system does?
  - Typically related to 'users'
- Useful to build what client wants
- A collections actors, use cases, and their communications.
- Actors are stick figures, Use cases are ovals



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Use case diagrams - actors



- A direct external user of a system
- Actors can be people, devices or other systems
- Multiple actors if it has different facets to its behaviour
- A patient calls the clinic to make an appointment for checkup. The receptionist schedules the appointment for that time slot. As per investigation, doctor give medicine to patient and at the same time patient have to pay bill. Patient can also cancel an appointment.

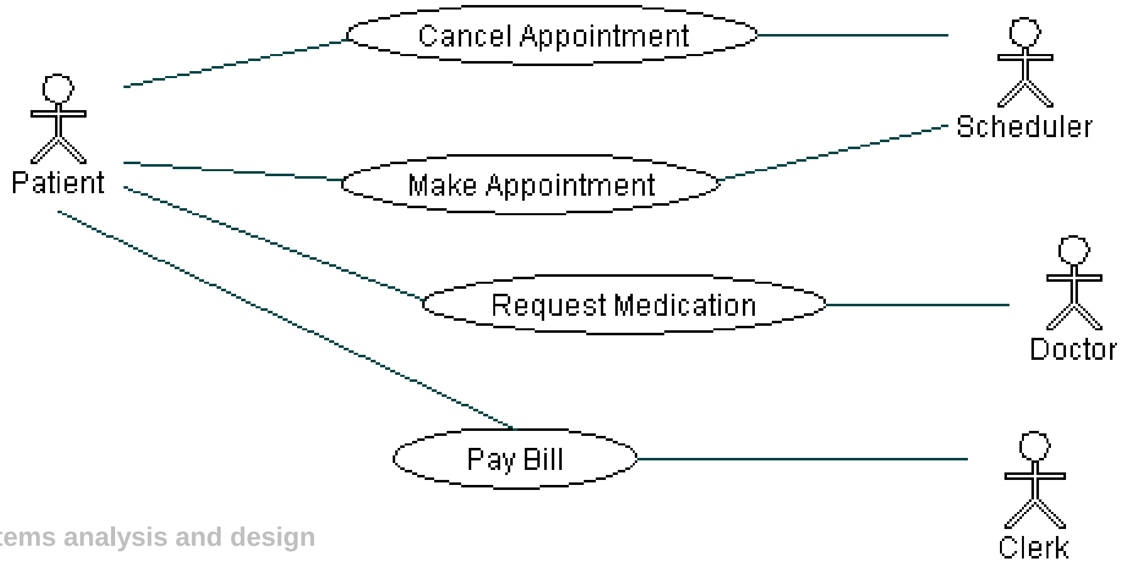
Anyone else?

Use case diagrams – use case

- Every way that an ‘actor’ uses a system is called a Use Case
- Each use case represents a piece of functionality that a system provides to a user
- Typically named with a verb than a noun
  - Do something to something

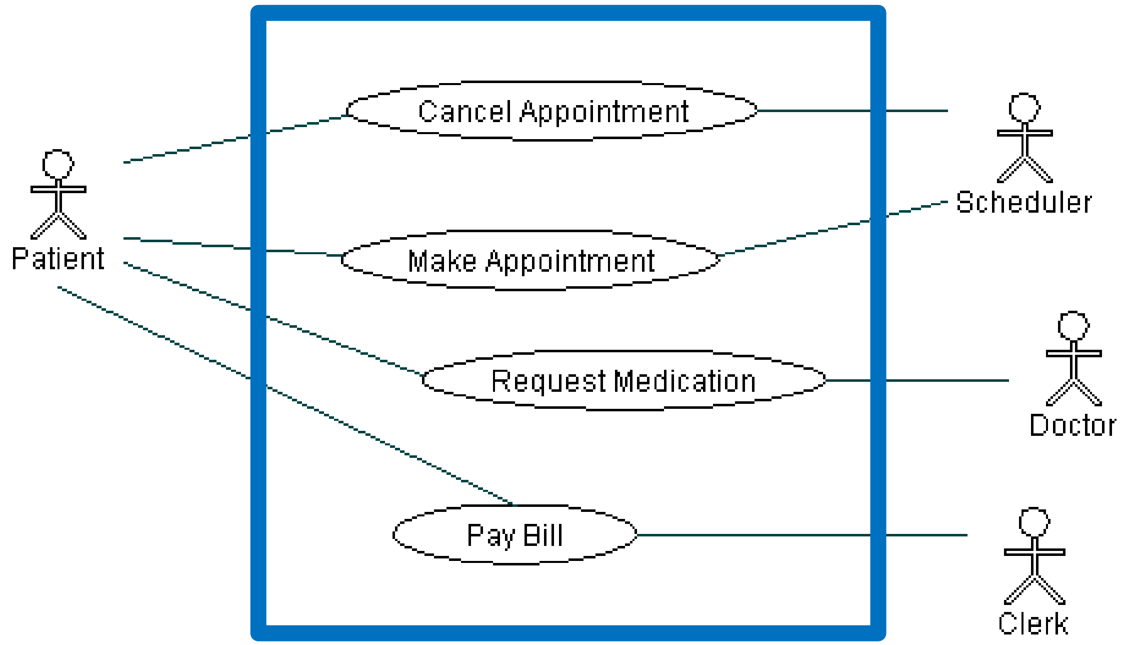
Use case diagrams – use case

- A patient calls the clinic to make an appointment for checkup. The receptionist schedules the appointment for that time slot. As per investigation, doctor give medicine to patient and at the same time patient have to pay bill. Patient can also cancel an appointment.



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

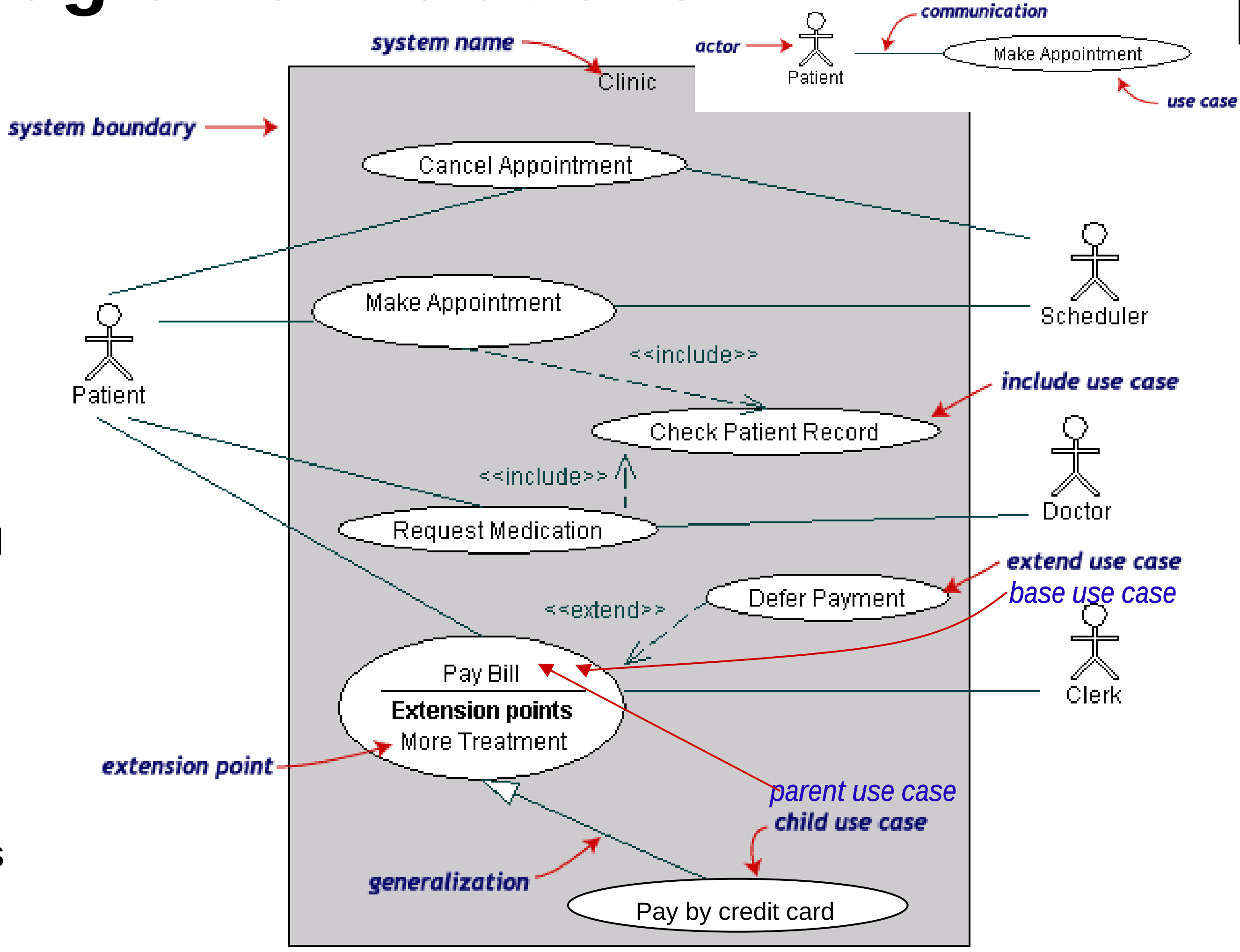
Use case diagrams – system boundary



Use case diagrams - relationship

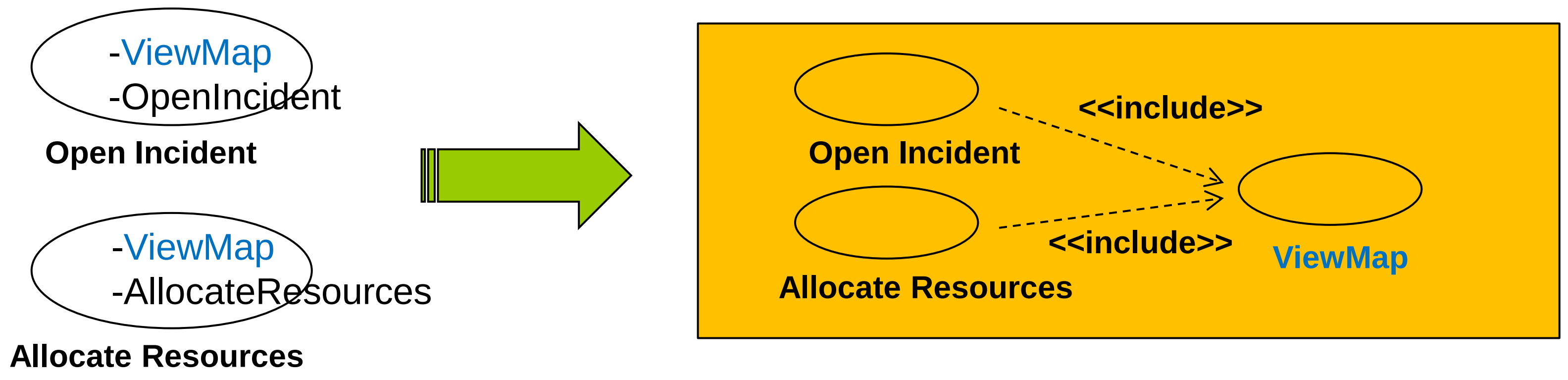
Communication

- Actors and use cases exchange information
- Include (or use)
  - A Use case calls the services of a common subroutine, where the common subroutine itself becomes an additional use case
- Extend
  - A use case can extend another by adding events. A typical application of extend relationships is the specification of exceptional behaviour
- Generalization
  - A use case specializes a more general one by adding more detail



Use case diagrams - relationship

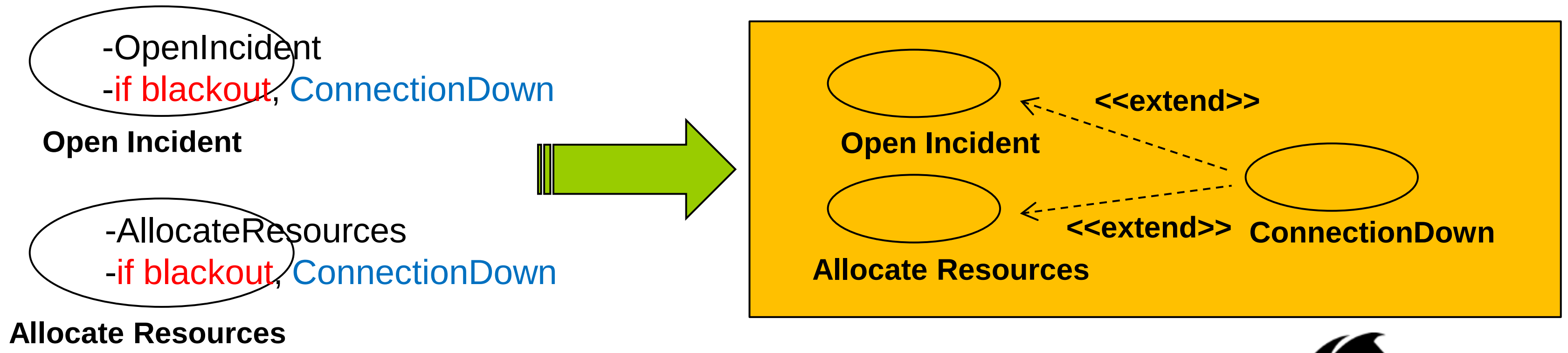
- Include (or use)
  - A Use case calls the services of a common subroutine, where the common subroutine itself becomes an additional use case
  - A dashed arrow from the source (including) to the target (included) use case.
  - The keyword <<include>> annotates the arrow



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

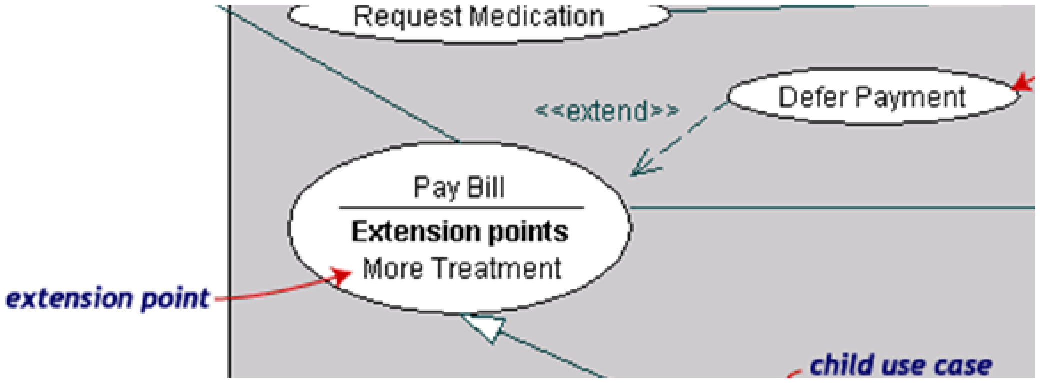
Use case diagrams - relationship

- Extend
  - Adds an “exceptional behaviour” to a base use case.
  - Base use case is meaningful on its own, it is independent of the extension.
  - Extension typically defines optional behaviour that is not necessarily meaningful by itself
  - A dashed arrow from the extension to the base use case.
  - The keyword <<extend>> annotates the arrow



Use case diagrams - relationship

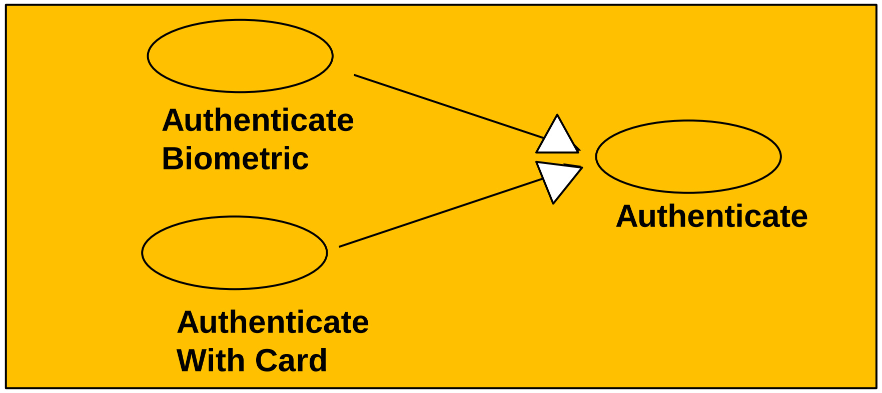
- Extend
  - Adds an “exceptional behaviour” to a base use case.
  - Base use case is meaningful on its own, it is independent of the extension.
  - Extension typically defines optional behaviour that is not necessarily meaningful by itself
  - A dashed arrow from the extension to the base use case.
  - The keyword <<extend>> annotates the arrow
  - Extension Points: specify the location at which the behaviour of the base use case may be extended. Extension points can have a condition attached. The extension behaviour occurs only if the condition is true.



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Use case diagrams - relationship

- Generalization
  - A use case specializes a more general one by adding more detail
  - A parent use case represents a general behaviour
  - A child use case specializes the parent by inserting additional steps or by refining existing steps
  - The Child use case inherits the behaviour of the parent use case
  - An arrow with its tail on the child use case and a triangular arrowhead on the parent use case



Vending machine example

A customer can buy a beverage from a vending machine. The customer inserts money into the machine, makes a selection and, hopefully, receives a beverage. A repair technician can perform scheduled maintenance on a vending machine

Anyone else?
Anything else?

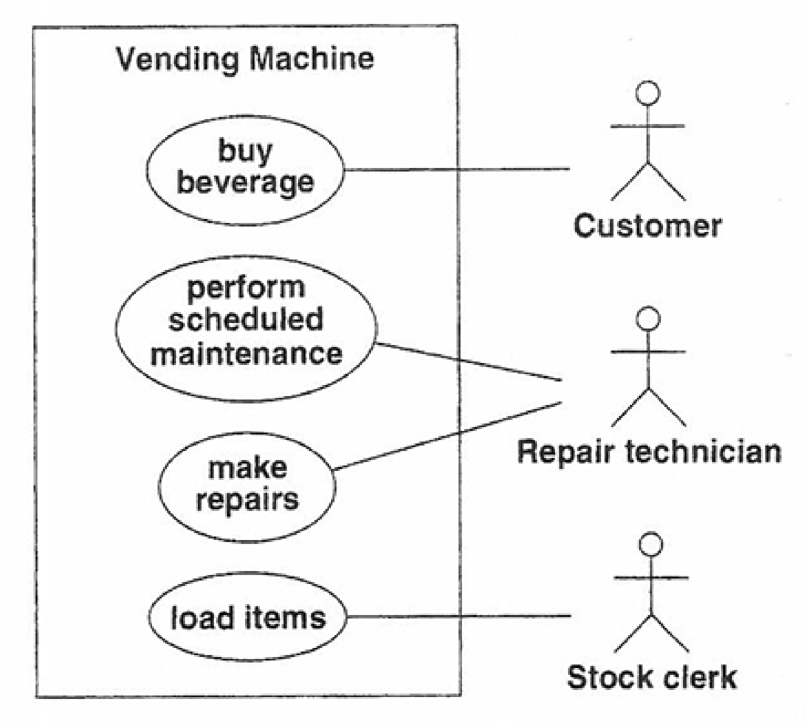
Vending machine: Use case diagram

❑ **Buy a beverage**. The vending machine delivers a beverage after a customer selects and pays for it.

❑**Perform scheduled maintenance.** A repair technician performs the periodic service on the vending machine necessary to keep it in good working condition

❑**Make repairs**. A repair technician performs the unexpected service on the vending machine necessary to repair a problem in its operation

❑**Load items**. A stock clerk adds items into the vending machine to replenish its stock of beverages.



Use case description

- Use case descriptions written at 3 level of detail
  - Brief description
    - Summary statement conjoined to activity diagram
  - Intermediate description
    - Expands brief description with internal flow of activities
  - Fully Developed Description
    - Expands intermediate description for richer view

Use case description – Brief Description

- **Use Case**: Buy a beverage

  The vending machine delivers a beverage after a customer selects and pays for it

Use case description – Intermediate description

Flow of activities for scenario of Buy a beverage

Main Flow:

1. Machine starts in the waiting state in which it displays the message “Enter coins”
2. Customer inserts coins into the machine.
3. Machine displays the total value of money entered
4. Machine lights up the buttons for the items that can be purchased for the money inserted.
5. Customer pushes a button.
6. Machine dispenses the corresponding item and makes change, if the cost of the item is less than the money inserted.

Exceptions:

1. If the customer presses the cancel button before an item has been selected,
   a. the customer’s money is returned and
   b. the machine resets to the waiting state.
2. If the customer presses a button for an item that costs more than the money inserted,
   a. the message “You must insert $*nn.nn* more for that item” is displayed, where *nn.nn* is the amount of additional money needed, and
   b. the machine continues to accept coins or a selection
3. If the customer has inserted enough money to buy the item but the machine cannot make the correct change
   a. the message “Cannot make correct change” is displayed, and
   b. the machine continues to accept coins or a selection.

Machine = System

Use case description – Fully developed description

Use Case Name	Buy a Beverage	
Brief Description	The vending machine delivers a beverage after a customer selects and pays for it	
Actors	customer	
Related Use cases	Include, extend, or generalization	
Entry condition	the waiting state in which it displays the message "Enter coins"	
Exit condition	the item is dispensed.	
Flow of Events	Actors	System
	1. Customer inserts coins into the machine. 2. Customer pushes a button.	1.1 Displays the total value of money entered 1.2 Machine lights up the buttons for the items that can be purchased for the money inserted. 2.1 Dispenses the corresponding item and make changes
Exception condition	2.1 If the customer presses a button for an item that costs more than the money inserted, a. the message "You must insert $*nn.nn* more for that item" is displayed, where *nn.nn* is the amount of additional money needed, and b. the machine continues to accept coins or a selection 2.2 If the customer has inserted enough money to buy the item but the machine cannot make the correct change a. the message "Cannot make correct change" is displayed, and b. the machine continues to accept coins or a selection. 2.3 If the customer presses the cancel button before an item has been selected, a. the customer's money is returned and b. the machine resets to the waiting state.	

Summary

- Requirements
  - The requirements are the descriptions of the system services and constraints
  - It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it
  - User requirements and System requirements
  - Functional
    - Tasks that the system need to support
  - Non-functional requirements
    - Usability, Reliability (Robustness, Safety), Performance (Response time, Throughput, Availability), Supportability (Adaptability, Maintainability, Portability)
  - The software requirements document

Summary

- What is UML (Unified Modelling Language)?
  - A standardized modelling language enabling developers to specify, visualize, construct and document artifacts of a software.
  - Object model: Class diagram, Object diagram
  - Functional model: Use case diagram
  - Dynamic model: Sequence diagram, Activity diagram
- Use case diagram
  - Actor (a direct external user of a system), Use case (a piece of functionality), Communication (Include/Extend/Generalization)

Next week

- Requirements elicitation
- Activity diagram