INFO2000/INFO2004 Modelling System Requirements

Mitchell Hughes
Room CLM131
mitchell.hughes@wits.ac.za
@mitchell_hughes

Usage of Twitter

- You are welcome to follow @mitchell_hughes
- Tweet using hashtag #INFO2000 if you would like your tweet to be addressed and/or retweeted
- You are free to tweet anything relating to the course, including questions, praise, complaints, rants, raves etc.
- We are also trying to create a "community of inquiry" around our course
- "Geeky"/"techy" retweets and links of interest beyond the scope of the course are also most welcome
- Usage of Twitter is voluntary, but you never know what hints, tips etc. might crop up ☺

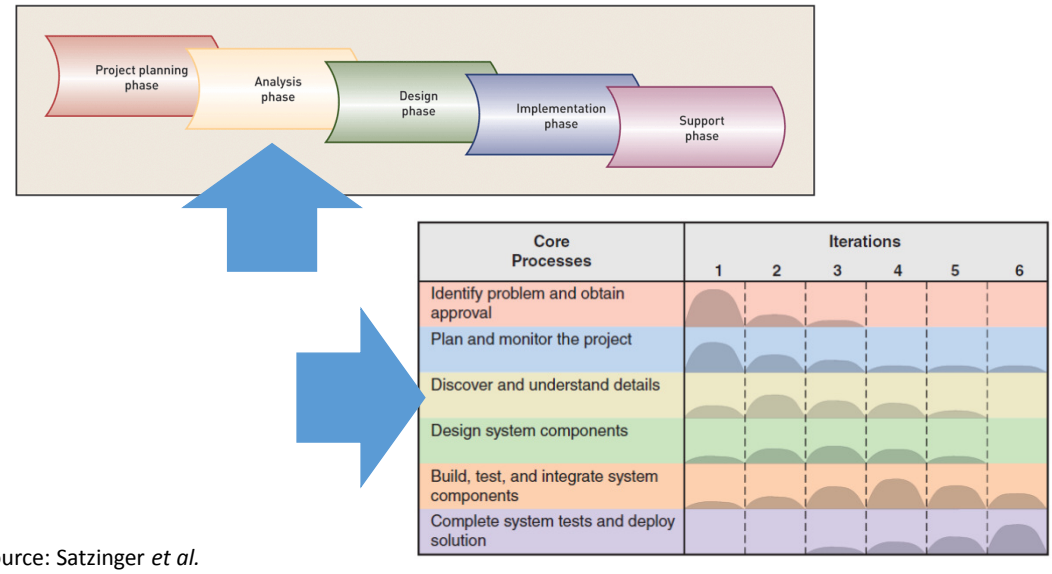
2

Lecture Materials - Credits

- These lecture materials are largely based on:
  - Satzinger, J., Jackson, R. & Burd, S. (2012). *Introduction to Systems Analysis and Design: An Agile, Iterative Approach*. (6th Ed.). Course Technology, Cengage Learning.
  - Lano, K. (2009). *Model-Driven Software Development with UML and Java*. Course Technology, Cengage Learning.
  - Larman, C. (2005). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* (3rd Ed.). Pearson.
  - Marakas, G. (2006). *Systems Analysis & Design: An Active Approach*. (2nd Ed.), Boston: McGraw-Hill.
- Images from Google Images and flaticon.com

3

Where we are



Source: Satzinger *et al.*

4

What do we do with the functional requirements we gather?

- Functional requirements are documented using a variety of models
- A model is a representation of an artefact (either existing or to be developed), used to analyse or design the artefact, i.e. they are the blueprints that guide the construction process
  - Expressed in precise graphical or textual notation, using a specific language of symbols
  - In systems development, we are concerned with the Unified Modelling Language (UML)
- Virtually all contemporary approaches to systems development begin requirements modelling with the concept of a *use case*

5

Defining use cases

- A use case shows an activity that the system performs, usually in response to a request by a user (Satzinger et al., 2012)
- A use case shows a service provided by the system and with which users/agents/actors this service interacts (Lano, 2009)
- Use cases provide an excellent picture of the system context, showing the boundary of the system, what lies outside of it and how it gets used (Larman, 2005)
- Use cases focus primarily on FUNCTIONAL REQUIREMENTS

6

Use cases…

- ... emphasise user goals and perspective, i.e. are highly user-centric
- ... represent the analyst's understanding of a particular business process
- … are used as a high-level communication tool between the analyst and users, stakeholders, management, team members etc.
- Has the analyst understood the process and modelled it correctly?
- Why do you think this is often done diagrammatically?
- Cliché alert: "A picture tells a thousand words"

7

Use cases… (cont.)

- … are designed to be simple so that users and other stakeholders can actively contribute to their development
- … provide a "black box" view of the functionality of the system, i.e. they omit the detail of how the functionality is actually carried out
- Why do you think this is so?
- Because systems analysis is concerned with the "what" and not the "how"
- This is particularly important to bear in mind for those that usually code first and ask questions later ☺

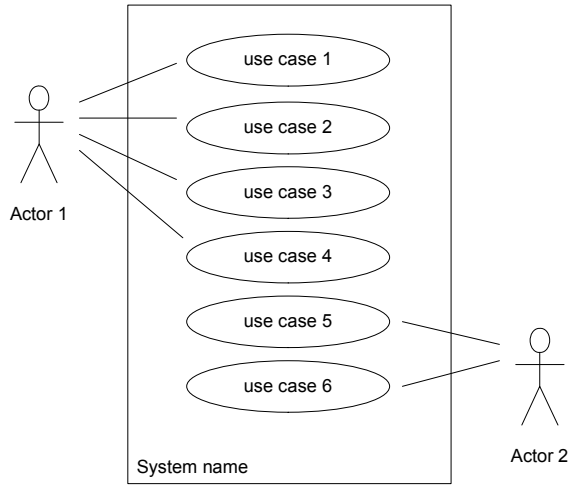
8

Use case terminology and notation

- Labelled ovals represent use cases
  - Use cases are labelled using descriptive verb-noun phrase notation
- Labelled stick figures represent actors/users/agents
  - An actor is something with behaviour
  - Represents a role (e.g. customer, user etc.) and not a specific person
  - Specific people could play several roles, each of which will require a separate use case
  - Actors can also be other systems
  - Actor names are always SINGULAR
- Connecting lines join use cases to their actors
- Labelled rectangles represent the automation (or system) boundary, i.e. defines the scope of the system being represented
  - The actor's communication with the use case crosses the boundary

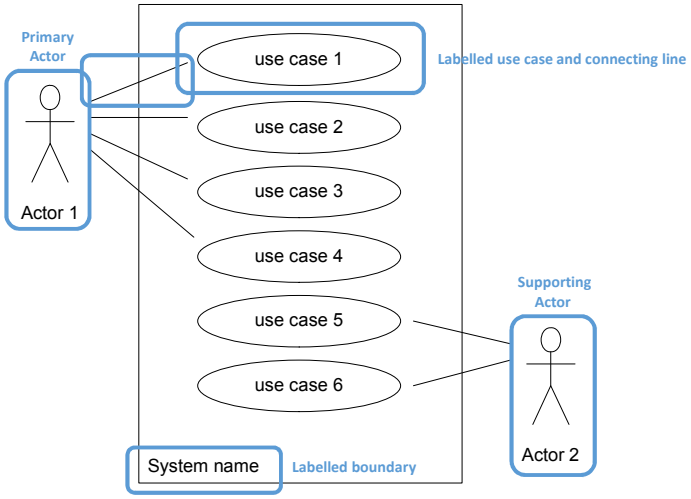
9

Generic use case diagram for a system



10

Generic use case diagram for a system



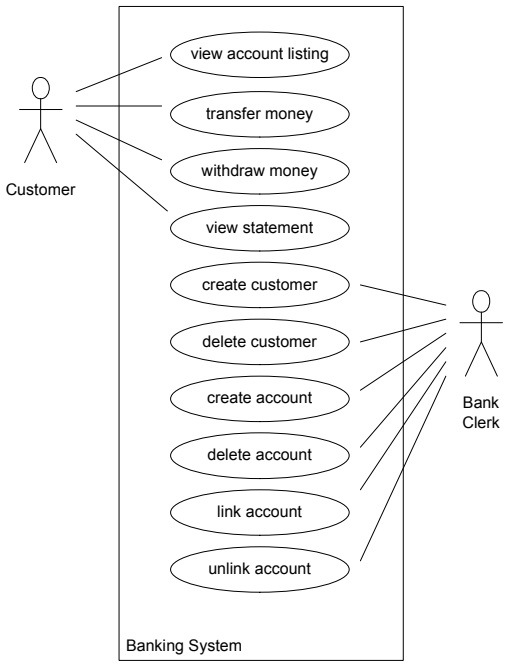
11

A simple use case example…

- … for a banking system
- There are two (2) users/clients/actors, namely:
  - A Customer, who can view a list of his/her accounts, transfer money, withdraw money and view a statement
  - A Bank Clerk (or Bank Employee), who can create or delete customers, create or delete accounts and link or unlink customers from accounts

12

Use Case Diagram: Banking System



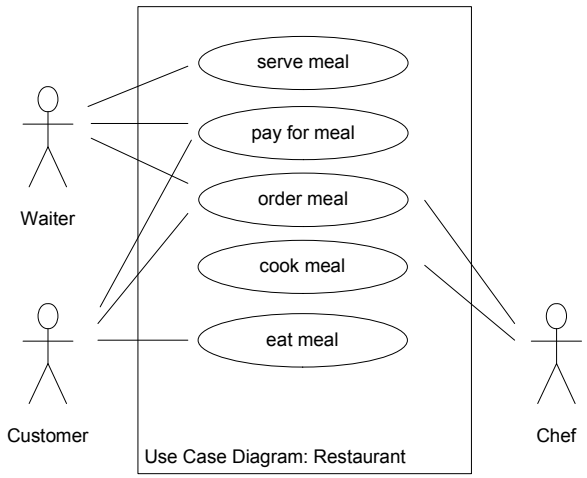
13

Lecture Exercise 1

- Construct a use case diagram for the following restaurant scenario:
  - A customer can order a meal from a waiter, eat the meal and pay for the meal through a waiter
  - A waiter takes the order, serves the meal and handles the payment for the meal
  - A chef cooks the meal based on the order

14

Use Case Diagram: Restaurant



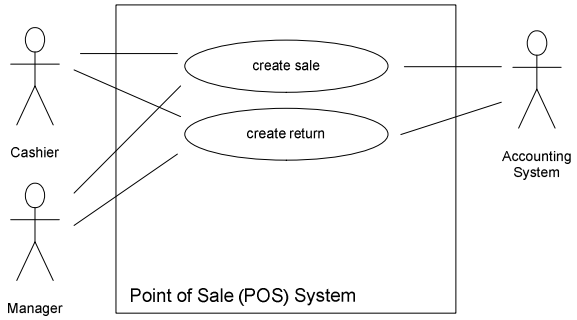
15

Lecture Exercise 2

- Construct a use case diagram for a simple cash only point of sale (POS) system
- Sales are captured by a Cashier. They are also recorded by an internal Accounting System. Cashiers also capture returns, which must first be approved by a Manager and, once approved, are also recorded in the Accounting System. During peak trading hours, managers may also capture sales and returns as if they were cashiers.

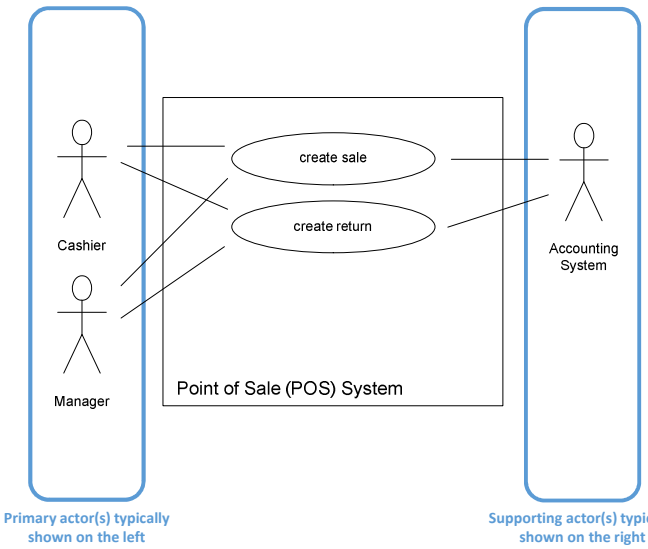
16

Use Case Diagram: Point of Sale (POS) System



17

Use Case Diagram: Point of Sale (POS) System



18

Adding more detail

- Sometimes a use case might need to use (or invoke) the functionality of another use case in order to perform its function, i.e. it is required, not optional
- This referred to as an *includes* relationship
- Guillemet (or angle quote) notation is used, together with a dashed arrow, i.e.

- Sometimes a use case might have optional or supplementary functionality that is not necessarily used (or invoked) every time
- This referred to as an *extends* relationship (note the reversed arrow)

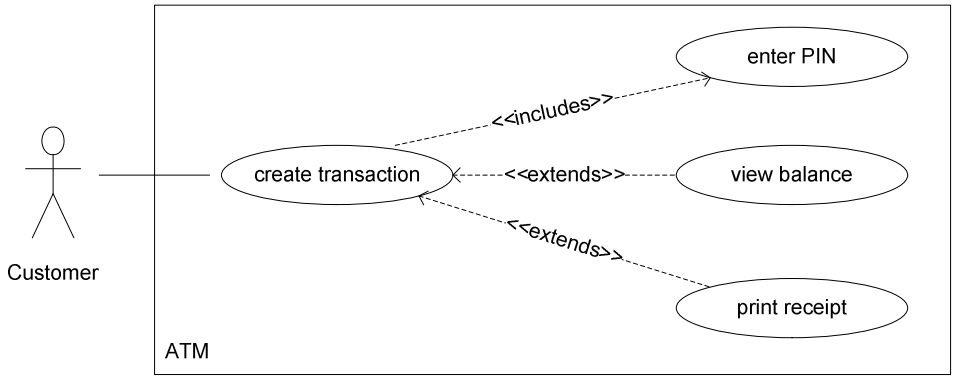
19

An <<includes>> and <<extends>> example…

- … for a simple ATM
- There is one (1) actor, a Customer, who can withdraw money from the ATM. This is known as creating a transaction and always involves the entering of his/her PIN.
- During the transaction, he/she may also view his/her balance after the withdrawal and/or print a paper receipt for the transaction.

20

Use Case Diagram: Simple ATM



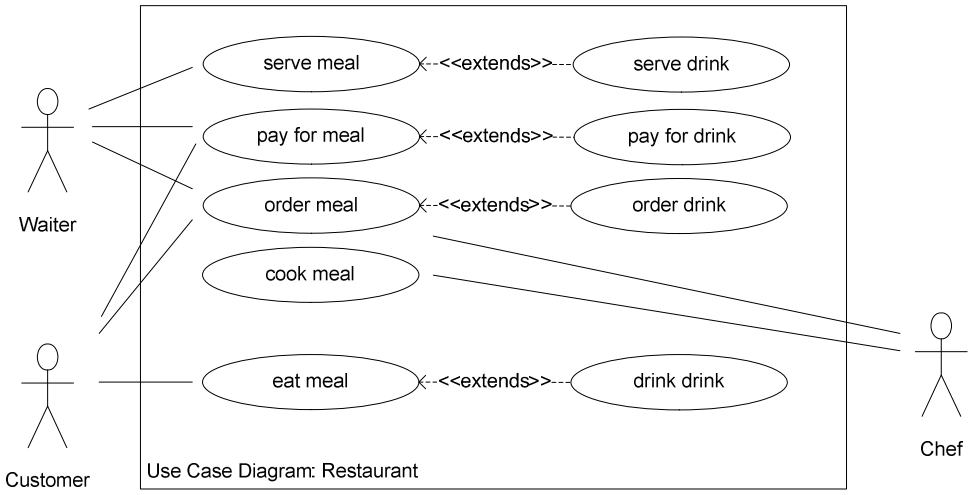
21

Lecture Exercise 3

- In the restaurant scenario, a customer may order a drink, which will also have to be served by the waiter and be paid for
- Extend your solution to Lecture Exercise 1 to include these new business rules

22

Use Case Diagram: Restaurant



23

How to identify use cases (1): The user goal approach

- Use cases are defined to meet the goals/objectives of the primary actor(s), so…
  1. Define the system boundary
  2. Identify the primary actor(s), i.e. those that have goals fulfilled through using the system
  3. Identify the goals for each primary actor
  4. Define the use cases that satisfy user goals
- Usually a *one use case per user goal* rule

24

How to identify use cases (2): The event decomposition approach

- Use cases are identified by determining the business events to which the system must respond
- This is called *event decomposition* (or *event analysis*)
- An *event* is something that occurs at a specific time and place, can be precisely identified and must be remembered by the system
- It is usually something that produces, uses or modifies data within the system, i.e. affects the underlying database in some way

25

Types of events

- *External events* occur outside the system and are usually initiated by an external actor
  - e.g. order, request, update, view etc.
- *Temporal events* occur as a result of reaching a certain point in time
  - e.g. scheduled reporting, batch processing etc.
- *State events* occur when something happens inside the system that triggers some process
  - e.g. reorder level reached, account payment due etc.

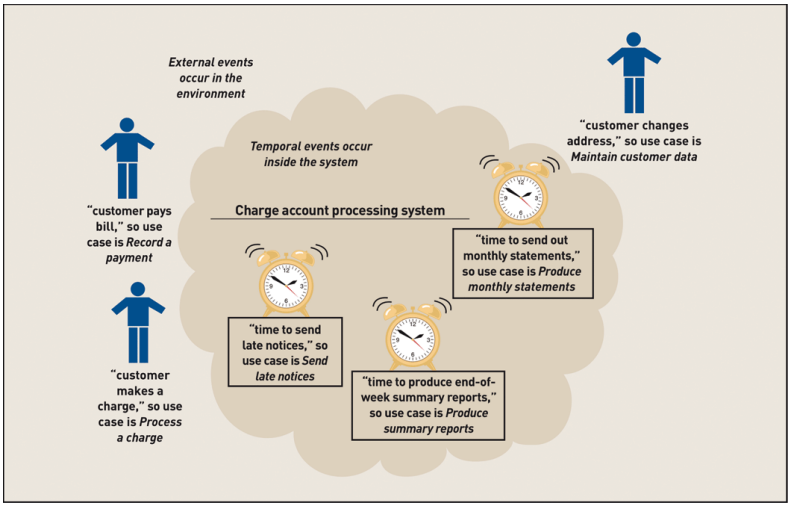
26

The event decomposition technique

1. Identify external events that require a response from the system
2. For each external event, identify and name the use case
3. Identify temporal events that require a response from the system
4. For each temporal event, identify and name the use case AND the point in time that triggers the use case
5. Identify state events that the system might respond to
6. For each state event, identify and name the use case AND define the state change
7. Verify that each use case is required by using the perfect technology assumption*

27

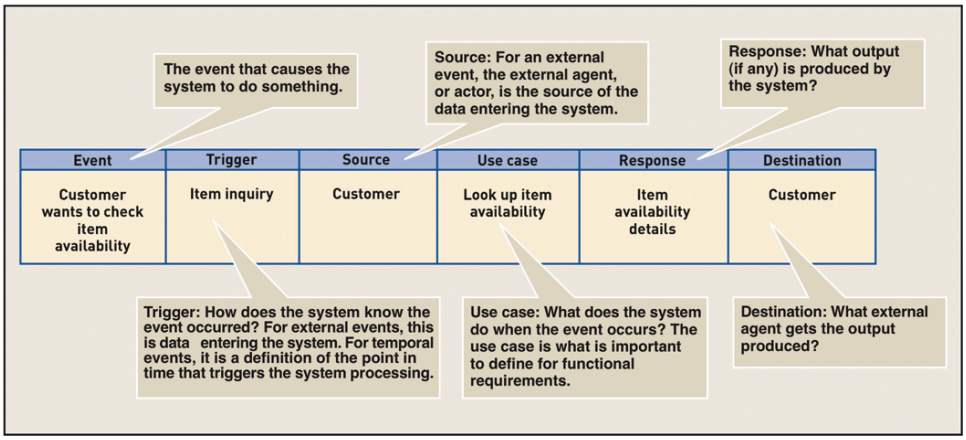
Identifying use cases from events



Source: Satzinger *et al.*, 2008

28

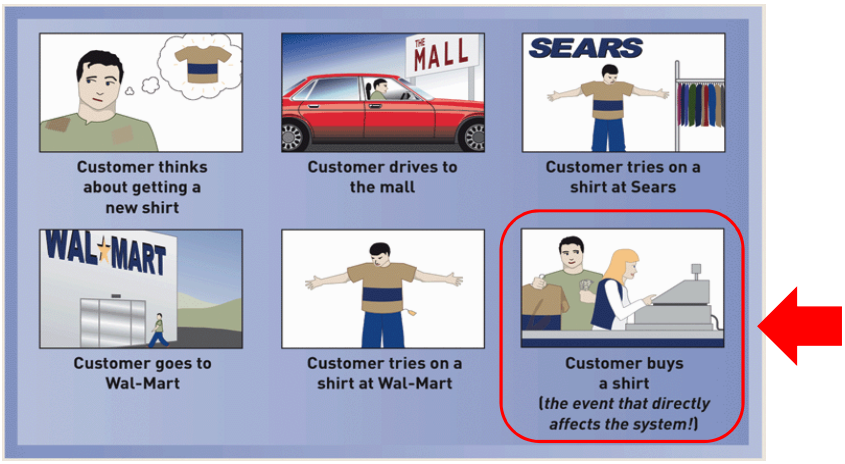
We can also construct and use event tables



Source: Satzinger *et al.*, 2008

29

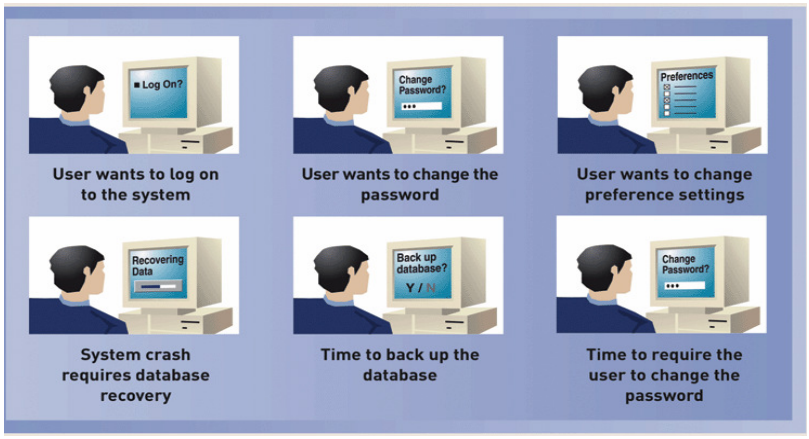
Separating events from things that happen before (triggers) and after (responses)



Source: Satzinger *et al.*, 2008

30

We also leave some events to the design phase… why?



Source: Satzinger *et al.*, 2008

31

Validating and refining use cases

- CRUD technique
  - Create a new instance of a thing
  - Read/Report on a thing
  - Update data relating to a thing
  - Delete an instance of a thing (usually ARCHIVE, not actually delete… why?)
- Based on what we might need to do with "things" in our environment
- Very focused on data and database design
- Often used as a cross-check along with the user goal approach (users focus on primary goals, whilst CRUD ensures that nothing is overlooked)

32

Simplified CRUD steps

1. Identify all entities (or domain classes) in the system
2. For each entity, verify that a use case exists to create an instance, read/report on instance(s), update instance(s) and delete (archive) instance(s)
3. If a required use case has been overlooked, create it

33

Simple CRUD example for an airtime system

Entity/Domain Class	CRUD	Verified Use Case
Customer	Create	Create customer
	Read/Report	View customer Generate customer report
	Update	Update customer (this could mean updating airtime balance OR updating demographic data)
	Delete	Archive customer

34

The deliverable: a use case set

- All identified use cases are compiled into a use case set
- Essentially a structured list
- Related use cases are grouped together under appropriate headings
- Key functionality must be made obvious

35

Use case set example

- Core business-related use cases:
  - Create customer
  - Create sale
  - Create return
- Maintenance-related use cases:
  - Update customer
  - Update staff member
- Reporting-related use cases:
  - Generate sales report
  - Generate commission report

36

Lecture Exercise 4

- Develop a user-related use case set for a social network, e.g. Twitter, Facebook or Instagram
- Verify your use case set using the CRUD technique

37