INFO2000/INFO2004 Modelling System Requirements

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

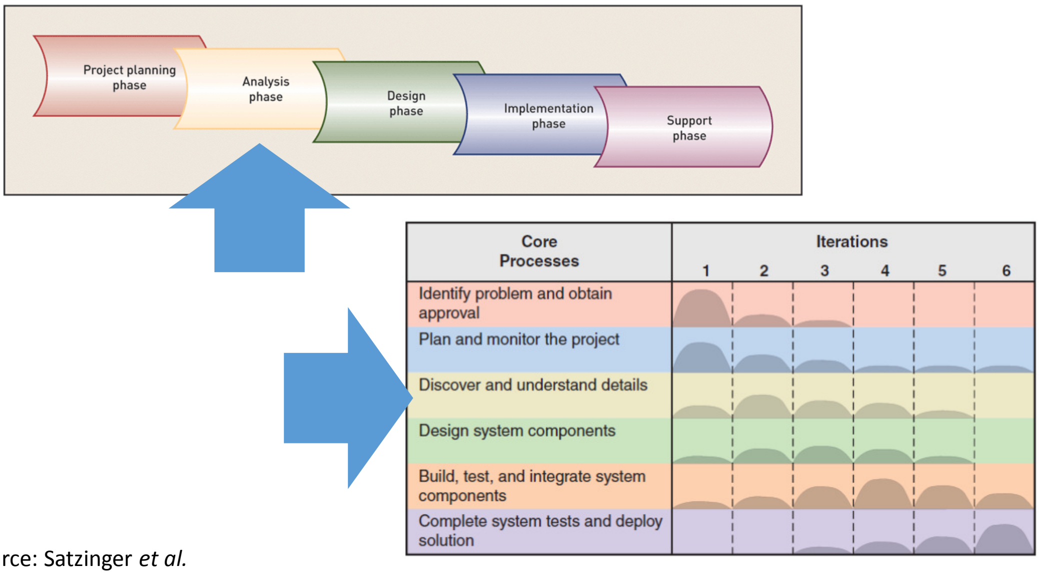
Usage of Twitter

- 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 ☺

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

Where we are



Core Processes	Iterations					
	1	2	3	4	5	6
Identify problem and obtain approval						
Plan and monitor the project						
Discover and understand details						
Design system components						
Build, test, and integrate system components						
Complete system tests and deploy solution						

Source: Satzinger *et al.*

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*

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

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”

⚠

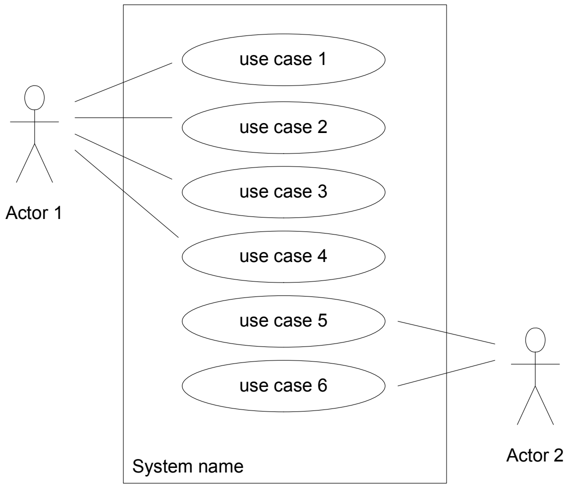
Use cases… (cont.)

- … 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 ☺

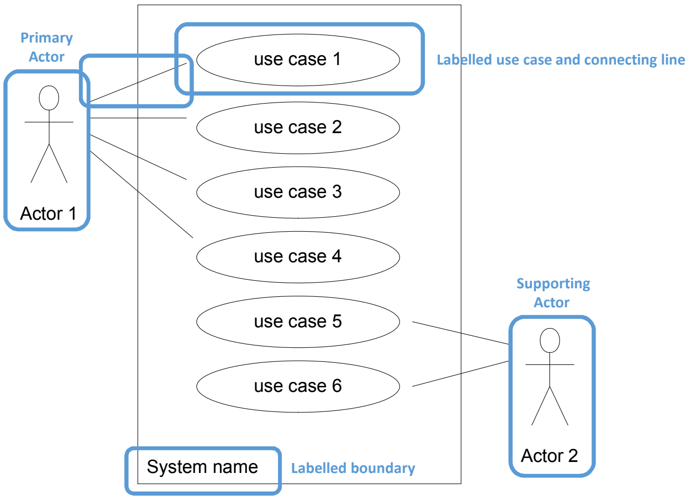
Use case terminology and notation

- 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

Generic use case diagram for a system



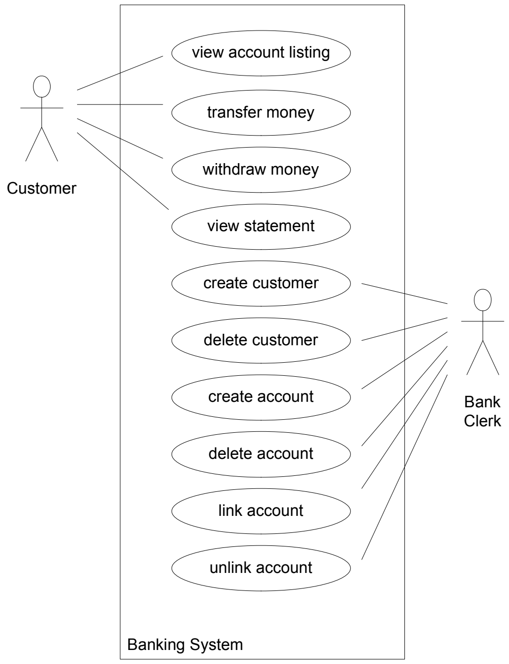
Generic use case diagram for a system



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

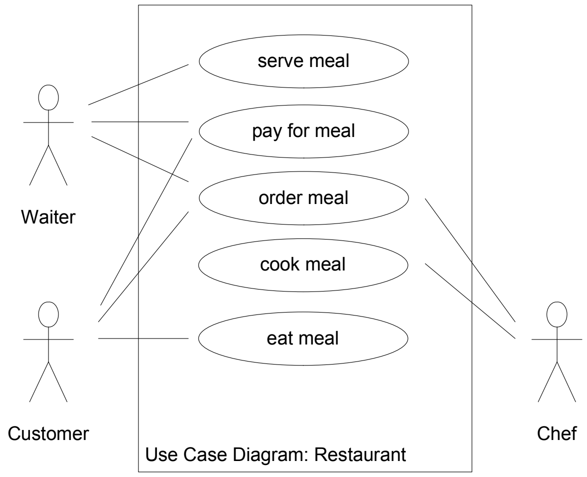
Use Case Diagram: Banking System



Lecture Exercise 1

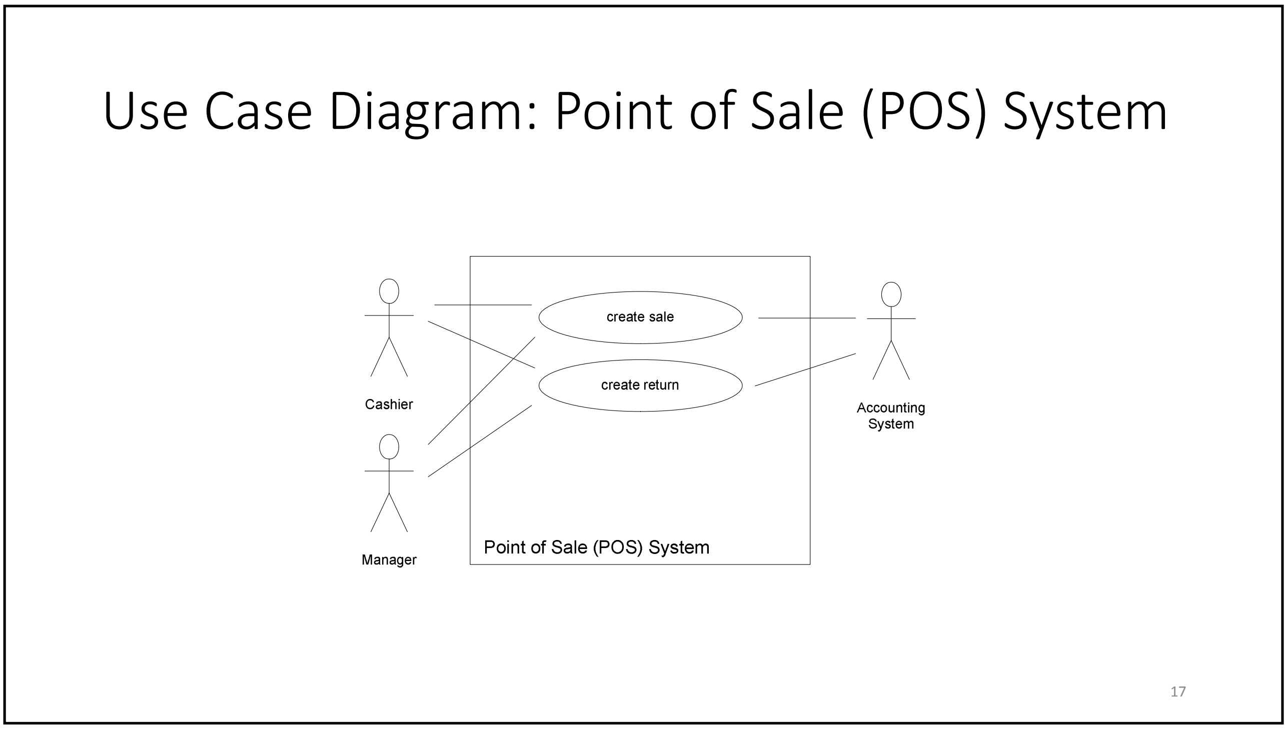
- Construct a use case diagram for the following restaurant scenario:
  - A customer can order a meal from a waiter, eat the meal and pay for the meal through a waiter
  - A waiter takes the order, serves the meal and handles the payment for the meal
  - A chef cooks the meal based on the order

Use Case Diagram: Restaurant

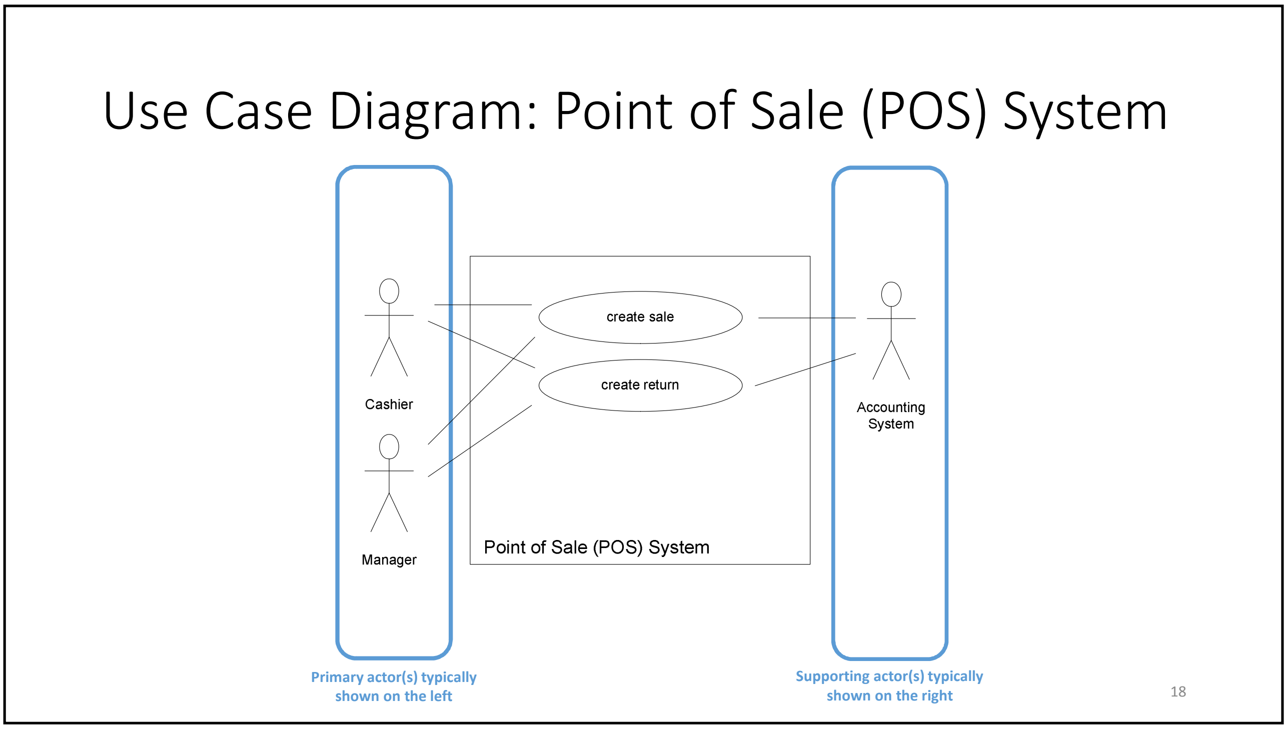


Lecture Exercise 2

- 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.



Use Case Diagram: Point of Sale (POS) System



Use Case Diagram: Point of Sale (POS) System

Primary actor(s) typically shown on the left

Supporting actor(s) typically shown on the right

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.

— — —<<includes>>- — — →

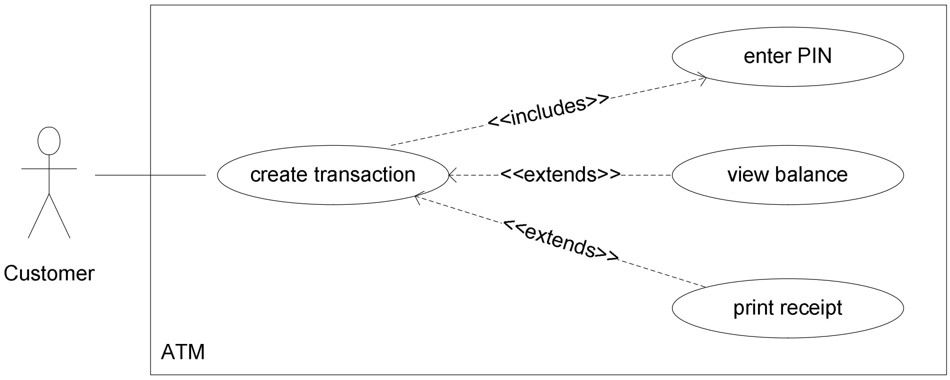
- Sometimes a use case might have optional or supplementary functionality that is not necessarily used (or invoked) every time
- This referred to as an *extends* relationship (note the reversed arrow)

← — —<<extends>>- — — –

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.

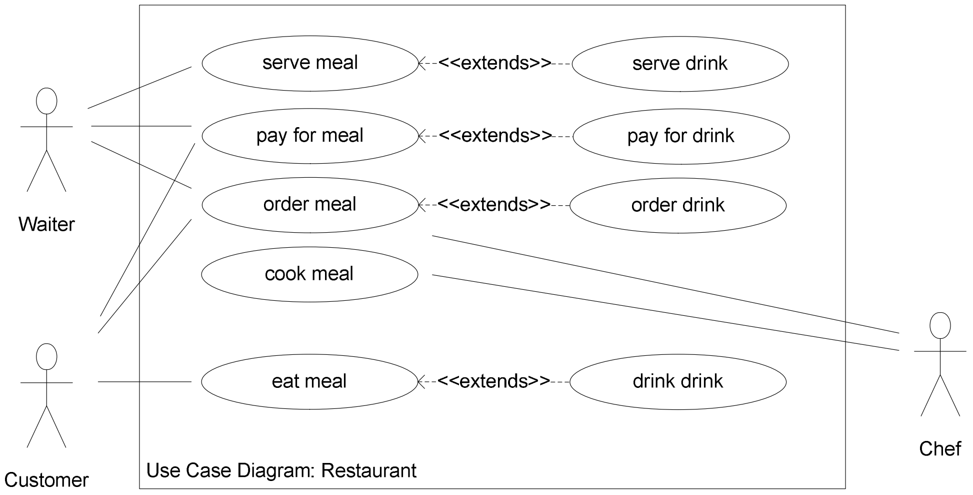
Use Case Diagram: Simple ATM



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

Use Case Diagram: Restaurant



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

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

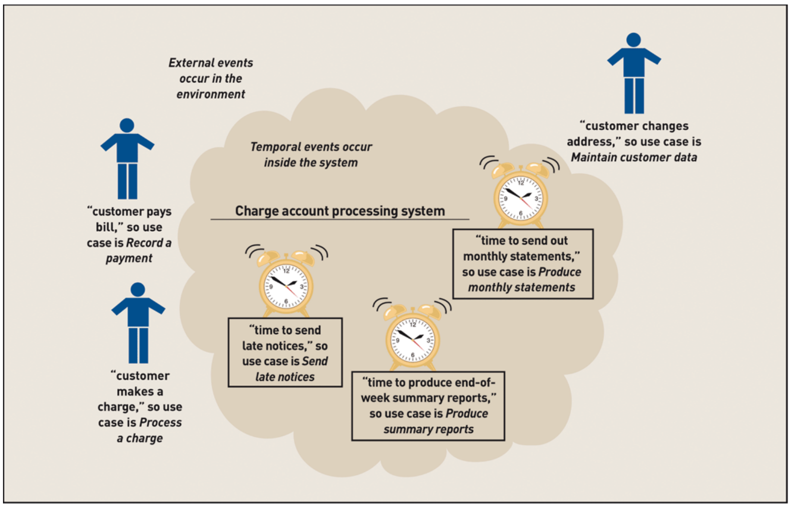
Types of events

- *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.

The event decomposition technique

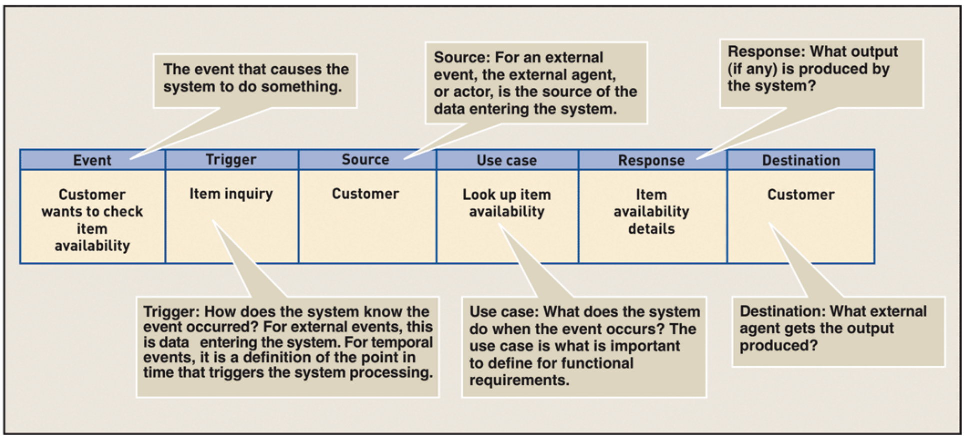
1. Identify external events that require a response from the system
2. For each external event, identify and name the use case
3. Identify temporal events that require a response from the system
4. For each temporal event, identify and name the use case AND the point in time that triggers the use case
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*

Identifying use cases from events


External events occur in the environment
"customer changes address," so use case is *Maintain customer data*
Temporal events occur inside the system
"customer pays bill," so use case is *Record a payment*
Charge account processing system
"time to send out monthly statements," so use case is *Produce monthly statements*
"time to send late notices," so use case is *Send late notices*
"customer makes a charge," so use case is *Process a charge*
"time to produce end-of-week summary reports," so use case is *Produce summary reports*

Source: Satzinger *et al.*, 2008

We can also construct and use event tables



The event that causes the system to do something.

Source: For an external event, the external agent, or actor, is the source of the data entering the system.

Response: What output (if any) is produced by the system?

Event	Trigger	Source	Use case	Response	Destination
Customer wants to check item availability	Item inquiry	Customer	Look up item availability	Item availability details	Customer

Trigger: How does the system know the event occurred? For external events, this is data entering the system. For temporal events, it is a definition of the point in time that triggers the system processing.

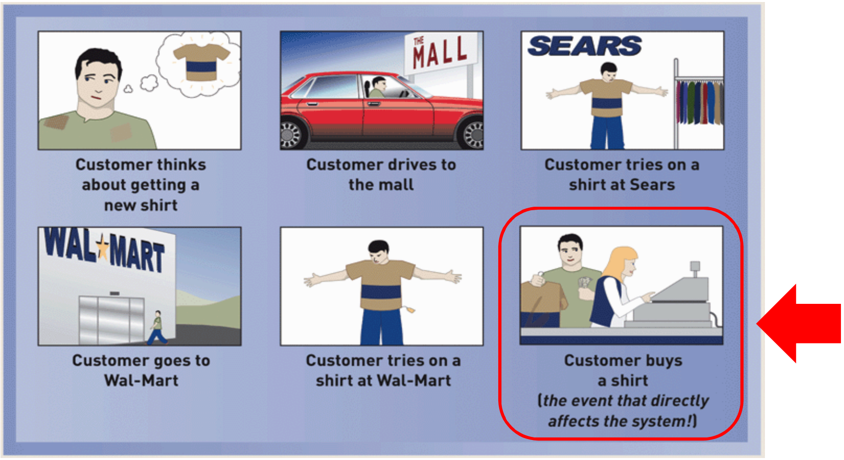
Use case: What does the system do when the event occurs? The use case is what is important to define for functional requirements.

Destination: What external agent gets the output produced?

Source: Satzinger *et al.*, 2008

29

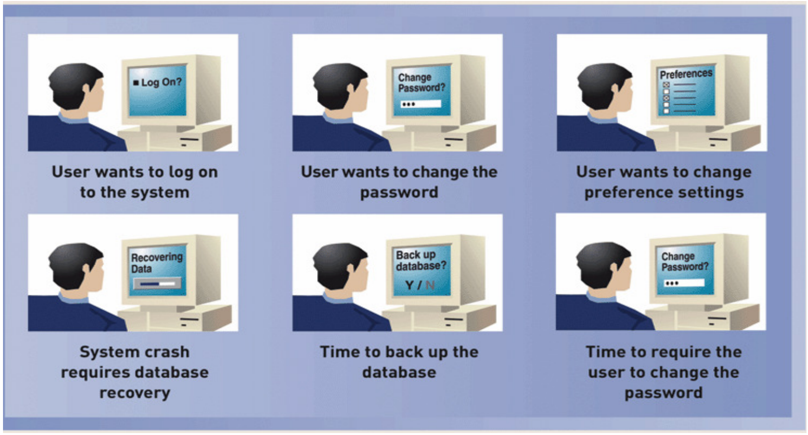
Separating events from things that happen before (triggers) and after (responses)


THE MALL
SEARS
WAL★MART
Customer thinks about getting a new shirt
Customer drives to the mall
Customer tries on a shirt at Sears
Customer goes to Wal-Mart
Customer tries on a shirt at Wal-Mart
Customer buys a shirt
(*the event that directly affects the system!*)

Source: Satzinger *et al.*, 2008

30

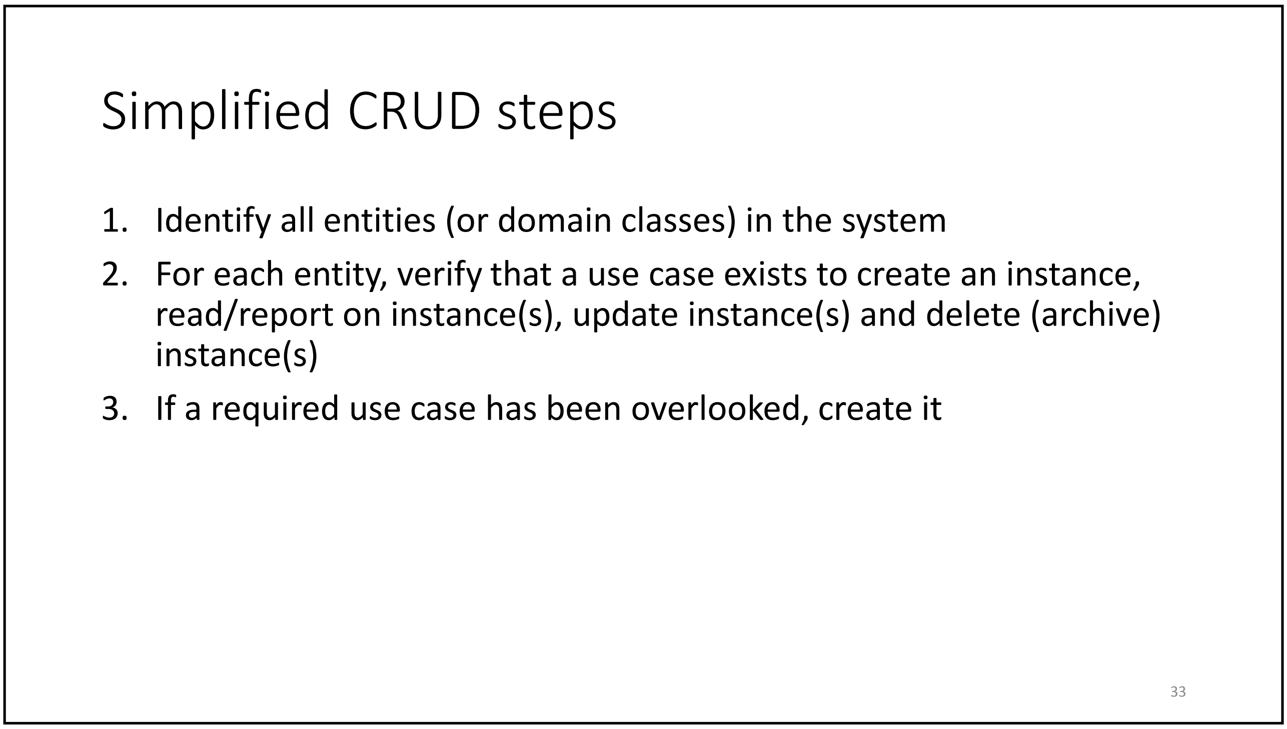
We also leave some events to the design phase… why?


Log On?
User wants to log on to the system
Change Password?
User wants to change the password
Preferences
User wants to change preference settings
Recovering Data
System crash requires database recovery
Back up database?
Y / N
Time to back up the database
Change Password?
Time to require the user to change the password

Source: Satzinger *et al.*, 2008

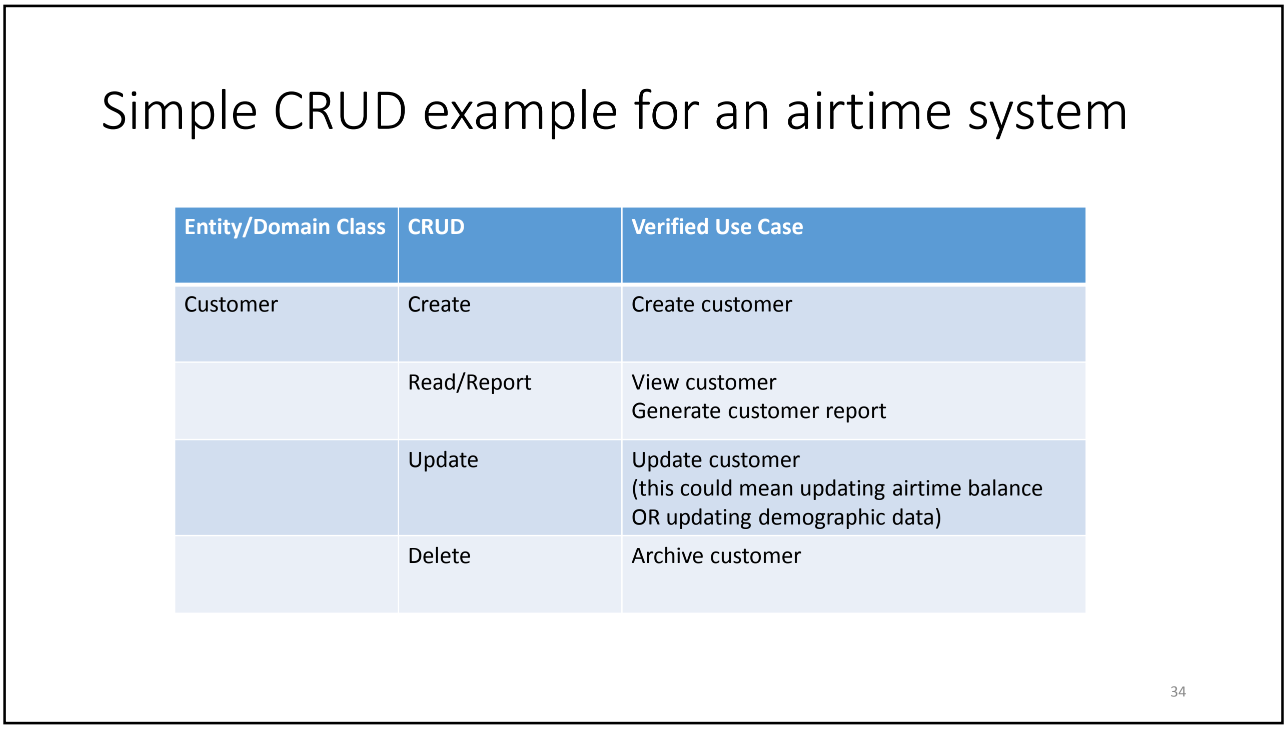
Validating and refining use cases

- 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)



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



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

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

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

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

The true power of use cases

- Diagramming is the easy part ☹
- Many regard use case diagrams as secondary in use case work
- Use cases are text documents, which means doing use case work means to write text ☠
- These text documents (or text models) are known as *use case descriptions*
- Use case descriptions have three varying levels of detail, i.e.
  - Brief, intermediate, detailed
  - Sometimes called brief, casual and fully dressed ☺

Use case formats

- Brief
  - Terse/concise/succinct, one paragraph summary
  - Covers a single, usually well-understood success scenario
  - Done during early requirements analysis for a quick sense of scope and requirements, i.e. only a first step
- Intermediate/Casual
  - Multiple paragraph format containing more detail, also covering potential variations/exceptions
  - Still used as above, i.e. early in the process
- Detailed/Fully dressed
  - All steps (including potential variations/exceptions) written in detail for fuller understanding
  - Include supporting sections such as pre- and post-conditions, exceptions etc.
  - Only a few significant and high-value use cases are written in detail (sometimes only around 10%) before design and programming starts, especially in "newer", more agile approaches to systems development

Brief use case example

Create sale: A Customer arrives at a checkout point with one or more items to purchase. The Cashier indicates the start of a new sale. As each item is added to the sale, the system displays a brief description of that item and a running total. After all items have been added, the Cashier indicates the end of the sale and the system displays the total cost. The Cashier captures the Customer's payment on the system and the system updates inventory to reflect the changes in stock levels. The system generates and prints a receipt which the Cashier hands over to the Customer.

Detailed/fully dressed use case template

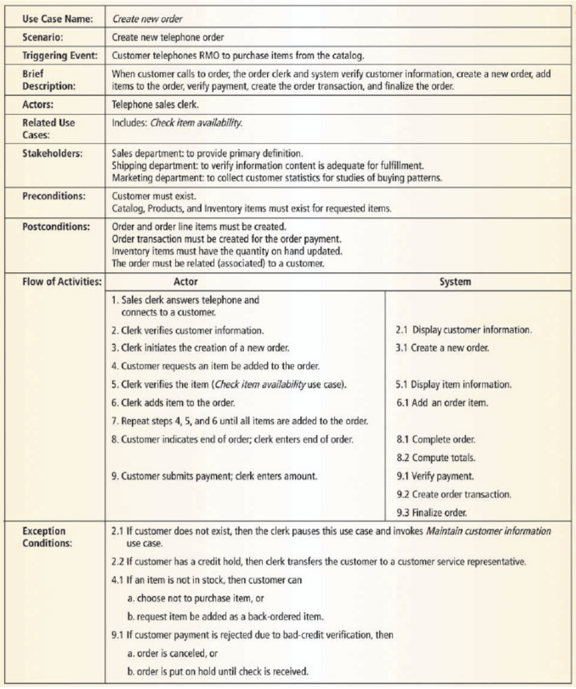
Use case name:	Verb-noun phrase format
Scope:	Name of the system or sub-system of which this use case is a part
Triggering event:	What sets the use case in motion?
Brief description:	One paragraph summary (i.e. the brief use case description)
Actor(s):	Primary and supporting actors (classified)
Related use cases:	Use cases that will/may be invoked
Stakeholders and interests:	Who cares about this use case and why?
Pre-conditions:	What must be true for the use case to begin?
Post-conditions:	What must be true after successful execution?
Flow of activities: (sometimes called **Main success scenario:** or **Basic flow:**)	List the process in numbered sequence, assuming a successful outcome and using the perfect technology assumption **Note:** Sometimes called the “happy path” ☺
Extensions: (sometimes called **Alternative flows:**)	List alternative scenarios of success or failure **Note:** Alternative/branching/exception statements are deferred to this section

Flow of activities section

Expressed in a two-column, numbered, sequential format that shows actor and system responsibilities, e.g.

Flow of activities:	**Actor**	**System**
	1. A new customer requests to create an account	1.1 Prompts customer to enter demographic data
	2. Customer enters demographic data	2.1 Records demographic data 2.2 Prompts customer to enter credit card details
	3. Customer enters credit card details	3.1 Records credit card details 3.2 Verifies credit card details with external credit bureau 3.3 Creates a new customer account using captured customer data 3.4 Confirms new account creation

A full example



Use Case Name:	*Create new order*	
Scenario:	Create new telephone order	
Triggering Event:	Customer telephones RMO to purchase items from the catalog.	
Brief Description:	When customer calls to order, the order clerk and system verify customer information, create a new order, add items to the order, verify payment, create the order transaction, and finalize the order.	
Actors:	Telephone sales clerk.	
Related Use Cases:	Includes: *Check item availability.*	
Stakeholders:	Sales department: to provide primary definition. Shipping department: to verify information content is adequate for fulfillment. Marketing department: to collect customer statistics for studies of buying patterns.	
Preconditions:	Customer must exist. Catalog, Products, and Inventory items must exist for requested items.	
Postconditions:	Order and order line items must be created. Order transaction must be created for the order payment. Inventory items must have the quantity on hand updated. The order must be related (associated) to a customer.	
Flow of Activities:	**Actor**	**System**
	1. Sales clerk answers telephone and connects to a customer.	
	2. Clerk verifies customer information.	2.1 Display customer information.
	3. Clerk initiates the creation of a new order.	3.1 Create a new order.
	4. Customer requests an item be added to the order.	
	5. Clerk verifies the item (*Check item availability* use case).	5.1 Display item information.
	6. Clerk adds item to the order.	6.1 Add an order item.
	7. Repeat steps 4, 5, and 6 until all items are added to the order.	
	8. Customer indicates end of order; clerk enters end of order.	8.1 Complete order. 8.2 Compute totals.
	9. Customer submits payment; clerk enters amount.	9.1 Verify payment. 9.2 Create order transaction. 9.3 Finalize order.
Exception Conditions:	2.1 If customer does not exist, then the clerk pauses this use case and invokes *Maintain customer information* use case. 2.2 If customer has a credit hold, then clerk transfers the customer to a customer service representative. 4.1 If an item is not in stock, then customer can a. choose not to purchase item, or b. request item be added as a back-ordered item. 9.1 If customer payment is rejected due to bad-credit verification, then a. order is canceled, or b. order is put on hold until check is received.	

Source: Satzinger *et al.*, 2008

Lecture Exercise 5a

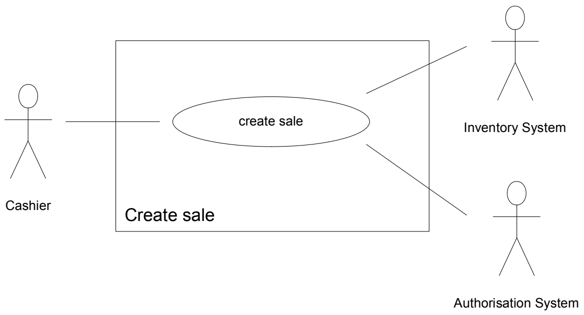
- … for a CASH ONLY point of sale (POS) system
- Write a fully dressed use case description for the *Create sale* use case based on the following scenario:
- A Customer arrives at a checkout point with one or more items to purchase. The Cashier indicates the start of a new sale. As each item is scanned and added to the sale, the system displays a brief description of that item and a running total (including VAT at 14%). After all items have been added, the Cashier indicates the end of the sale and the system displays the total cost. The Cashier captures the Customer's payment on the system, including calculating change if necessary. The system also updates an Inventory System to reflect the changes in stock levels. Finally, the system generates and prints a receipt which the Cashier hands over to the Customer. The Customer then leaves with the items.
- Suggested solution via hand-out

Lecture Exercise 5b

- Now extend the use case description to include credit card payments, which are authorised using an external Authorisation System.
- You may assume that credit card payments are handled by the Cashier swiping the card (there is no need for the Customer to enter a PIN) and that all payments are successful.
- Finally, draw the use case diagram for the extended *Create sale* use case.
- Suggested solution via hand-out

45

Use Case Diagram: Create sale



Note:

- How little detail appears in the use case diagram compared to the fully dressed use case description
- AND the value of constructing the text model BEFORE the diagram, which then becomes very simple to draw

46

Think about other possible extensions

- Customer asks Cashier to remove item from sale
- Customer asks Cashier to cancel sale
- Customer presents discount card to Cashier
- Customer asks Cashier to change purchase/sale type
- etc.
- Note: These are just a few extensions/alternative scenarios… this is the reality of the complexity of a “real world” business process in a “real world” client environment
- We cannot run from the detail, we must embrace it

A question to ponder

- Who is the primary actor in cases like this?
- Is it the Customer or the Cashier?
- One of the most common controversies in Systems Analysis
- It depends on where you draw the boundary
- When looking specifically at the POS system, the actor is definitely the Cashier
- When taking a broader view (e.g. the retail organisation as a whole), the actor becomes the Customer as the Cashier would be considered part of the system

A final reminder about use cases

- Text models should ALWAYS come BEFORE diagrams… why?
- Diagrams are exercises in translation from a text model, NOT creation from scratch
- Extensions normally comprise the majority of the text… why?
- Capturing the true complexity of a “real world” business process in a “real world” client environment
- With thorough use case writing, the combination of both “happy path” AND extensions should satisfy the functional interests of most stakeholders
- Note: Some interests are best handled as non-functional requirements

Where do use cases fall short?

- Identify flow of activities, BUT…
- … do not explicitly identify inputs and outputs, i.e. the “conversation” for us

System sequence diagrams (SSDs)

- Derived from use case descriptions
  - Cannot be developed without a detailed description of the use case
- Represent a single use case/scenario
- Show the time-based sequence of interactions between an actor and the system during the use case/scenario
  - Interactions are expressed in the form of messages between the actor and the system
- Show inputs, outputs and flow of data/information
- System shown as “black box”, i.e. “what”, not “how”

SSD notation

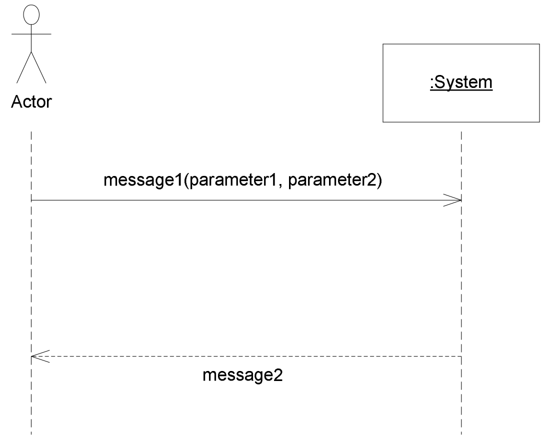
- A labelled stick figure in the top left represents the actor
- A rectangle labelled :System in the top right represents the system (Note: the colon and underlining)
- Vertical dashed lines, called “lifelines” are drawn under each

SSD notation cont.

- Labelled arrows represent messages
- A message is labelled to describe its purpose and may include parameters/arguments in brackets
- Messages follow the verb-noun naming syntax
- Input messages are drawn as horizontal solid arrows
  - Input messages are generally named after the method/activity to be performed by the system
- Output messages (or “returns” or “return values”) are drawn as horizontal dashed arrows
  - Output messages generally take the form of returned data/information or a return message

53

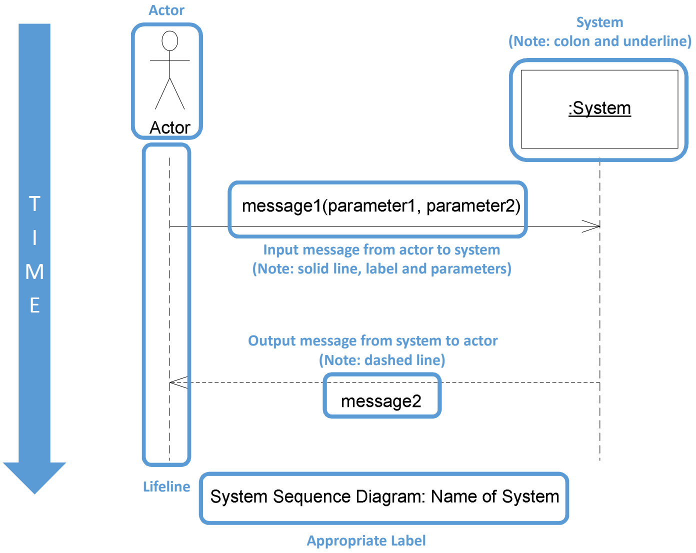
Generic SSD



System Sequence Diagram: Name of System

54

Generic SSD



Actor

Actor

System
(Note: colon and underline)

:System

T I M E

message1(parameter1, parameter2)

Input message from actor to system
(Note: solid line, label and parameters)

Output message from system to actor
(Note: dashed line)

message2

Lifeline

System Sequence Diagram: Name of System

Appropriate Label

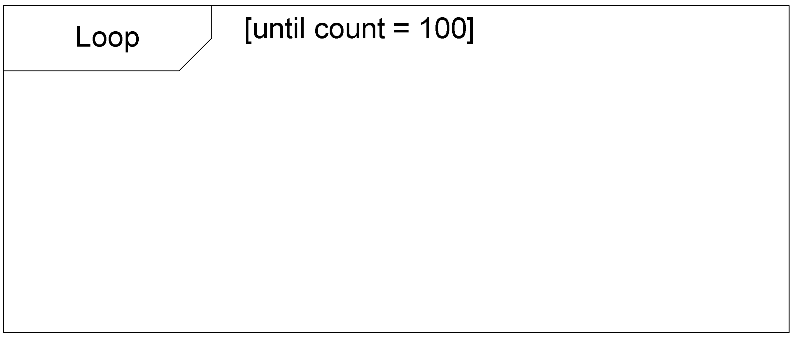
How to “read” a SSD

- Read horizontally (across) to see the actor and system participating in the use case
- Read vertically (down) to see when events/activities are happening (the further down you read, the further along in time things are happening)

More SSD notation

- Processing logic is implemented using interaction frames:
  - For looping/repetition, use a Loop frame
  - For optional (true/false), use an Opt frame
  - For alternative (if-then-else), use an Alt frame
- Whatever is inside the interaction frame will only execute until (Loop) or if (Opt, Alt) a certain condition is met
- This condition is called a *guard condition*
- Guard conditions are indicated by square brackets, i.e. [condition]

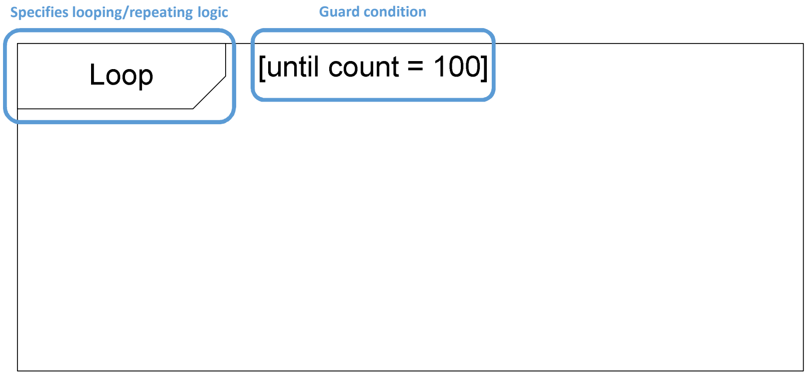
Loop frame example



Loop

[until count = 100]

Loop frame example



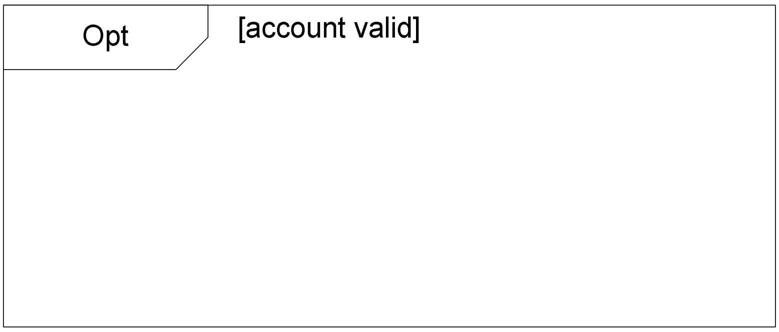
Specifies looping/repeating logic

Guard condition

Loop

[until count = 100]

Opt frame example



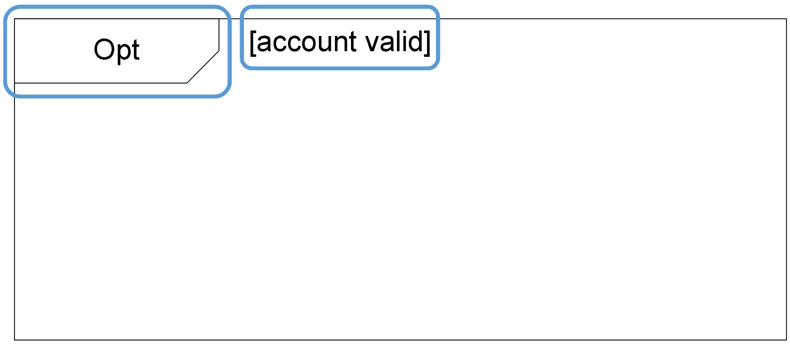
Opt

[account valid]

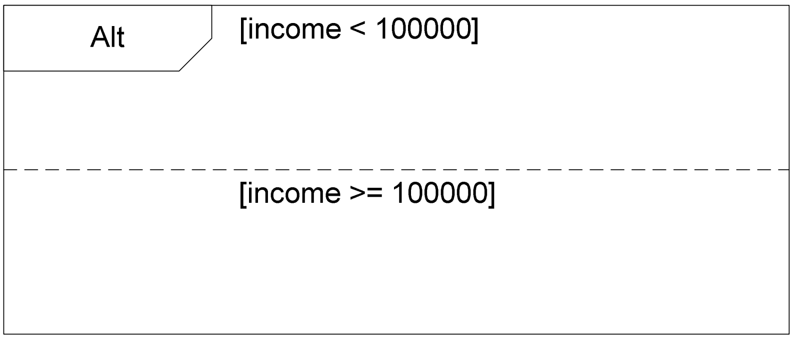
Opt frame example

Specifies optional processing, usually based on True/False

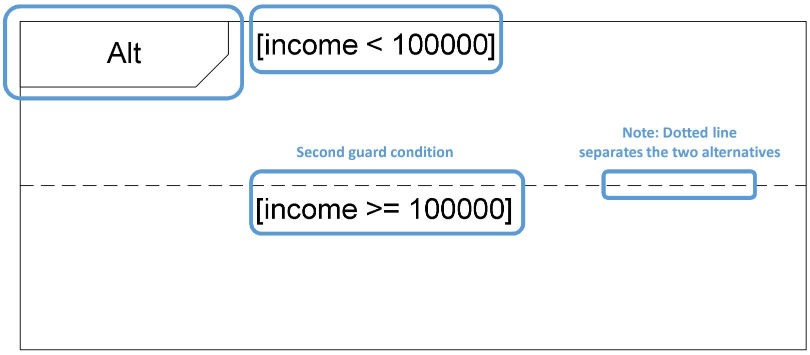
Guard condition



Alt frame example



Alt frame example



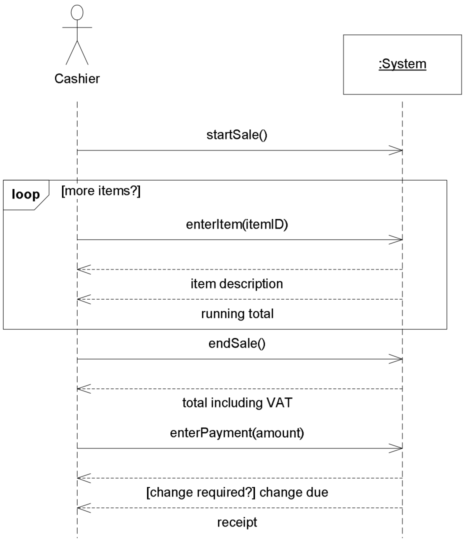
Specifies alternative logic, usually based on if-then-else

First guard condition

Second guard condition

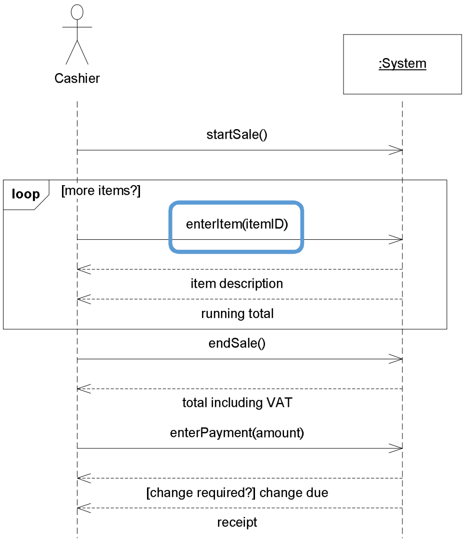
Note: Dotted line separates the two alternatives

Example: A cash only sales use case



System Sequence Diagram: Create sale

Example: A cash only sales use case



Cashier

:System

startSale()

loop [more items?]

enterItem(itemID)

item description

running total

endSale()

total including VAT

enterPayment(amount)

[change required?] change due

receipt

System Sequence Diagram: Create sale

Note:

- Why not scanItem(itemID)?
- Because using a scanner indicates “how” not “what”
- We are interested in the act of entering an itemID into the system, not how we do it

Lecture Exercise 6

- For the point of sale (POS) system
- Construct a system sequence diagram for the full *Create sale* use case, i.e. including both cash AND credit card payments
- Suggested solution via hand-out