# Systems Design and Security

UNIFIED MODELING LANGUAGE™

## Part 4: Requirements Modelling

http://staffwww.dcs.shef.ac.uk/people/A.Simons/

Home ⇒ Teaching ⇒ Lectures ⇒ COM2008/COM3008

The University Of Sheffield.

## Bibliography

- Unified Modelling Language
  - M Fowler, UML Distilled: A Brief Guide to the Standard Object Modelling Language, Addison-Wesley, 3rd ed., 2004.
  - S Bennett, S McRobb R Farmer, Object-Oriented Systems Analysis and Design using UML, 4th ed., McGraw-Hill, 2010.
- More on Use Cases
  - A Cockburn, Writing Effective Use Cases, Addison Wesley Longman, 2001.
  - A J H Simons, Use cases considered harmful, TOOLS-29 Europe, 1999, 194-203.

# Outline

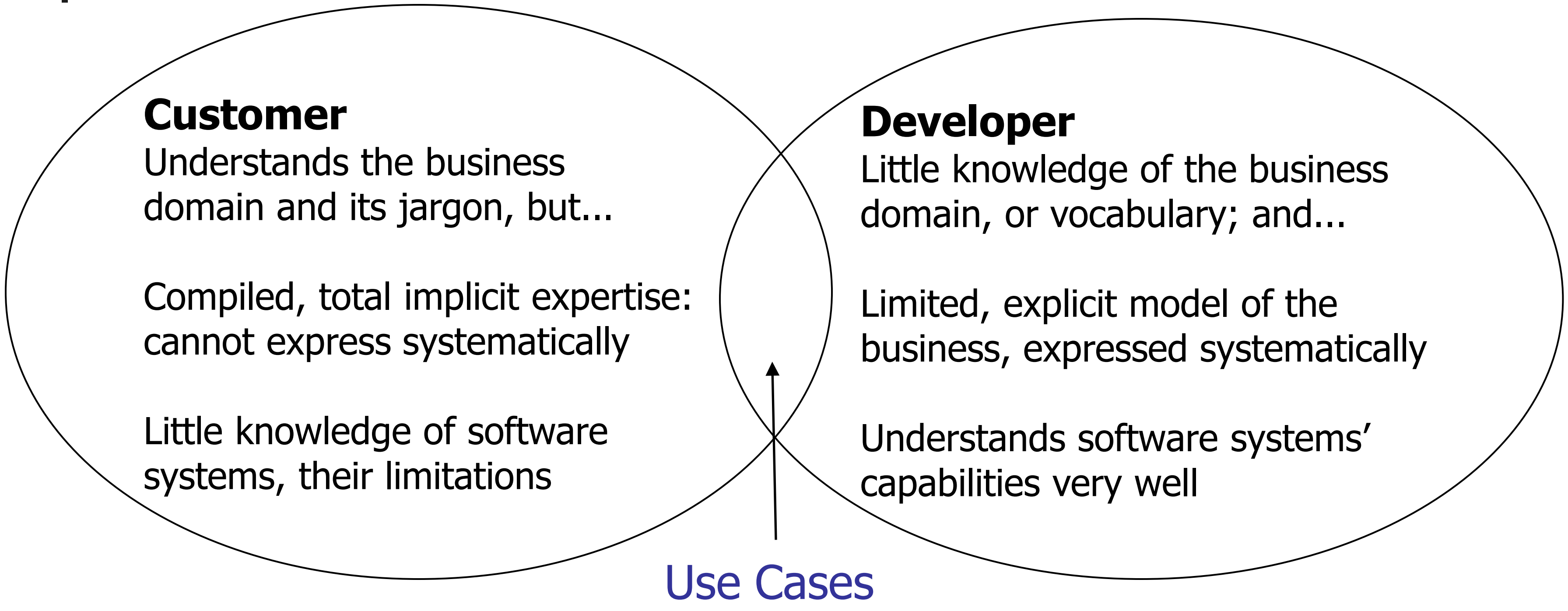
- Use case modelling
- Actors and use cases
- Use case diagrams
- Modelling event flows
- Structuring use cases
- Choosing system options
- Testing from scenarios

Reading: Bennett, et al. chapter 6;
Fowler, chapter 9;
Simons article

## Use Case Modelling

- A user-centric approach [Jacobson, 1992]
  - identify the typical users of the system
  - identify the typical tasks they perform
  - capture functional requirements incrementally
- Rationale for use case modelling
  - difficult to specify all system requirements top-down
  - easier to identify what single users expect to be able to do
  - collect stories of typical user interactions, until done
  - structure the requirements model gradually
    - collect obvious, normal flows of events first
    - collect alternative, exceptional flows of events later
  - the final specification is the collection of use cases

# Helps Communication



**Customer**
Understands the business domain and its jargon, but...

Compiled, total implicit expertise: cannot express systematically

Little knowledge of software systems, their limitations

**Developer**
Little knowledge of the business domain, or vocabulary; and...

Limited, explicit model of the business, expressed systematically

Understands software systems' capabilities very well

Use Cases

allow customer to tell stories
help to decompile expertise

allow developer to collect
structured info incrementally

## Explore Requirements

- Customer may only have a general business vision
  - Opportunity-driven: an idea to capture a new market
  - Necessity-driven: a solution to avert business failure
- Customer may supply initial requirements
  - Can be too abstract, high-level, imprecise, incomplete
  - Can have a biased viewpoint, make tacit assumptions
  - Can focus narrowly on one make-or-break issue
- Developer must compensate for this
  - Wider scope may deliver many additional business benefits
  - Analysis must therefore include the whole business context

## Modelling What?

- Model a system, or a business
  - identify user interactions with a system: use case modelling [Jacobson, 1992]
  - model stakeholder interactions within a business: business process re-engineering [Jacobson, 1994]
- Strong and weak points
  - good at capturing "what" a system does
  - not good at expressing the control logic
- Central place in UML
  - UML sees the use case view as the central viewpoint
  - other models must comply with the use case model

## Actors and Use Cases

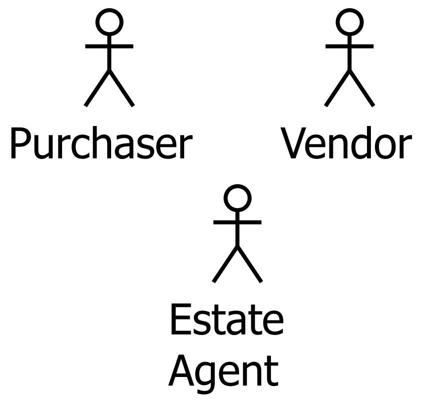
- UML
  - refers to *actors*, *use cases* in a technical sense
- Actor
  - a type of external user interacting with the system
    - eg: *cashier, customer, bank*
  - not the individual person, but the role they play
    - the same individual could play many roles
- Use Case
  - a type of interaction of an actor with the system
    - eg: *inspect balance, withdraw cash, order statement*
  - a use case is a related collection of event flows
    - a single event flow is called a scenario in UML

## Definition of ‘Actor’

- An actor [Jacobson, 1994]:
  - “defines a role ... that someone or something in the environment can play in relation to the system.”
- Emphasis on the role
  - a “typical user” of the (intended) system
  - a person acting in a role, not the individual
  - also, any external computer system
  - positive examples:
    - *Employee, Manager, Husband, Bank* (roles)
  - negative examples:
    - *John Smith, Jane Doe* (individuals)

# Actor Examples

- Actors are named by the role they play in the system
- The same individual may play multiple roles



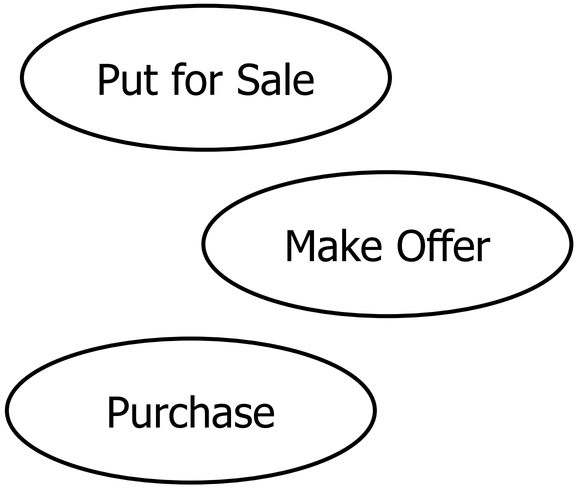
- Eg: in a Real Estate Trading system
- The same person might be
  - a Vendor when selling a house
  - a Purchaser when buying a house
  - the Estate Agent when managing the trading system

# Definition of 'Use Case'

- A use case [Jacobson, 1994]:
  - "is a sequence ... of related transactions between an actor and a system that yields a result of measurable value."
- Emphasis on a suitable size
  - upper bound: a "sequence of related transactions"
    - accomplishes a single objective, not multiple goals
  - lower bound: a "result of measurable value"
    - accomplishes a business objective, not a code instruction
  - positive examples:
    - *deliver goods, pay invoice, process order* (have objective)
  - negative examples:
    - *enter password, increase productivity, quit menu* (don't)

## Use Case Examples

- Use cases represent business tasks, not system actions
- They must have a measurable, business-level objective



- Eg: in a Real Estate Trading system
- Tasks with a business objective include:
  - Put for Sale – house is available
  - Make Offer – make a bid for house
  - Purchase – buy the house

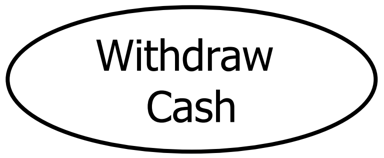
## Identify Use Cases

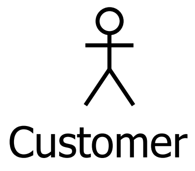
- Starting point
  - idea is to elicit the required system functionality
  - no prior knowledge of the system is assumed
- Interviewing technique
  - identify the primary actors – main users who benefit
  - identify main use cases – main tasks done by actors
  - identify secondary actors – who support the system
  - identify extra use cases – tasks done by supporters
- Documenting technique
  - model the actors and use cases in a use case diagram
  - link actors to the use cases in which they participate

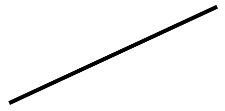
## Study: ATM Cash Machine

- Purpose
  - to automate the job of cashiers (bank *tellers*)
- Actors involved
  - *Customer* – primary, benefits from the system
  - *Bank, Cashier* – secondary, support the system
- Identify use cases
  - main: *Inspect Balance, Withdraw Cash, Order Statement*
  - extra: *Check Level, Reload ATM*
- Sketch diagram
  - assume all use cases will be implemented in the system

## Diagram Syntax

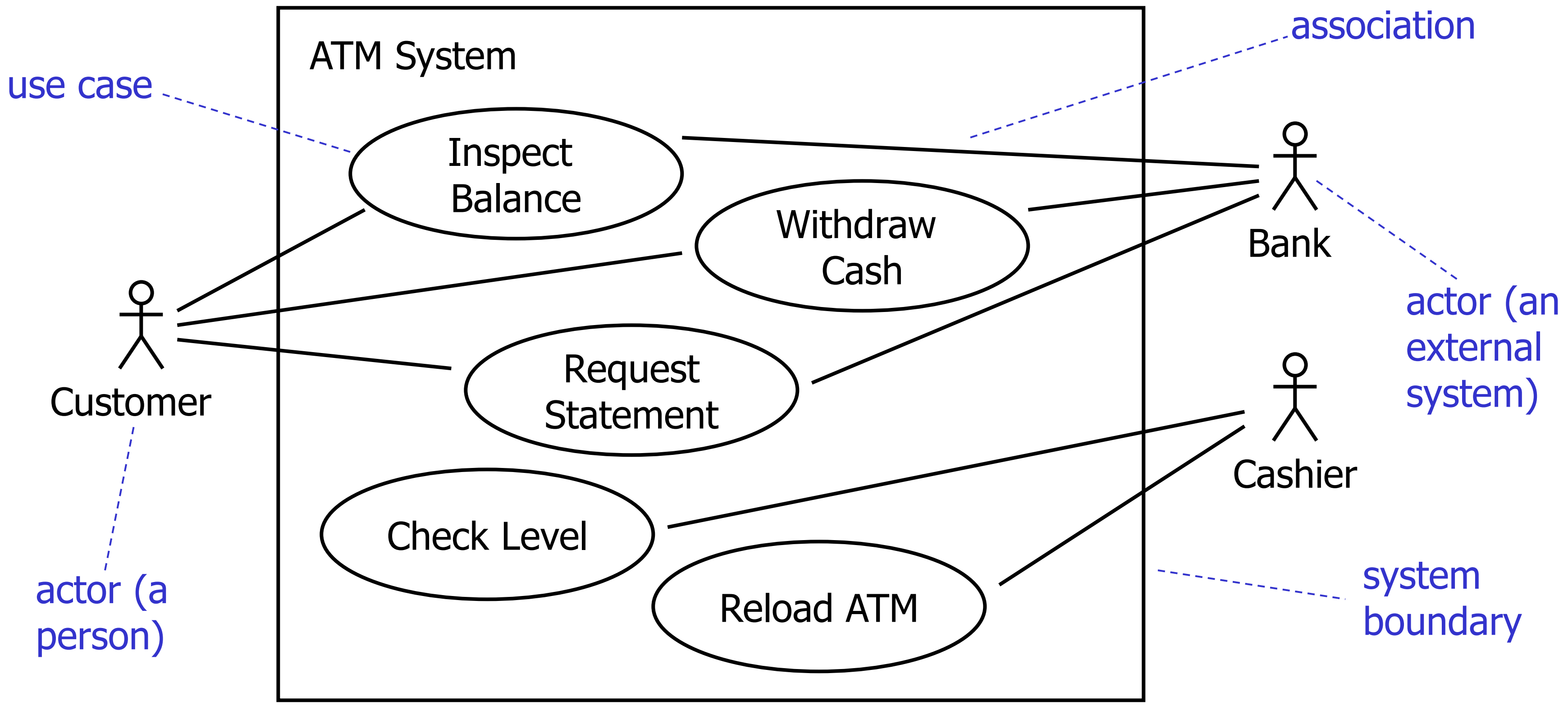






- Use cases
  - draw each use case as a labelled ellipse
  - use verb-expressions to label each use case
  - use cases can be within, or outside, the boundary
- Actors
  - draw each actor as a labelled stick-figure
  - name each actor according to the role it plays
  - external systems can also be actors
- Associations
  - draw associations as lines, not quite touching
  - link actors to use cases in which they participate

# Use Case Diagram



## Lab 1: Online Trading

- Purpose
  - to provide an online trading system which interfaces with external payment and delivery services
- Actors involved
  - *Purchaser, Bank, Vendor, Shipping Agent*
  - Which actors are primary and which secondary?
- Identify participation in use cases
  - *Add Item, Remove Item, Checkout, Give Address, Give Bank Details, Ship Goods, Debit Account, Credit Account*
- Sketch diagram
  - which use cases will be implemented in the system?

Run a Poll

## Document Use Cases

- Starting point
  - describe the required system functionality in detail
  - collect the different scenarios incrementally
- For each use case
  - document the main success scenario first (normal flow)
  - document the various extensions later (variant flows)
    - each extension diverges at some point from the main flow
    - each complete path through the UC is called a scenario
- Writing technique
  - write each scenario as a sequence of events
  - style as a dialogue between the actor and system

## Writing Style

- UML specifies no written format for use cases
  - different contexts may require more/less formality
  - usually written in “structured” natural language
    - itemise the steps, express system states
- Possible levels of formality [Cockburn, 2001]
  - brief use case
    - short description to fit a spreadsheet cell
  - casual use case
    - several paragraphs written in natural language
  - fully dressed use case
    - written according to a structured template

## Describe: Inspect Balance

- What happens?
  - user inserts a bank card, then types a PIN code, then requests to see their bank balance
- Suggest using this template
  - state name, purpose of use case
  - write down the enabling condition – what must be true before the use case can proceed?
  - write down flow of events as a dialogue – what the user does, what the system does in response
  - write down the success condition – what must be true when the use case has finished?

## Main Success Scenario

| | |
|---|---|
| **Use Case:** | Inspect Balance |
| **Purpose:** | Allow the customer to view the balance of his/her account. |
| **Require:** | The customer has a bank card, and knows a secret PIN code. The ATM is displaying a welcome screen. |
| **Event Flow:** | 1. The customer inserts the bank card for the account.<br>2. The ATM recognises the account and asks for the PIN.<br>3. The customer inputs a four-digit PIN code.<br>4. The ATM confirms the PIN code and displays a menu.<br>5. The customer selects the "inspect balance" option.<br>6. The ATM displays the balance for the account.<br>7. The ATM returns the card after a while. End use case. |
| **Success:** | The customer has the bank card and has seen the balance. The ATM is displaying the welcome screen. |

## Extensions

- A complete use case has:
  - a main success scenario (normal flow), written first
  - various extensions (variant flows), written afterwards
- Types of extension [Simons, 1999]
  - optional extension – optional inserted behaviour that is sometimes executed, then resumes the normal flow
  - alternative extension – another successful path through the use case, which may resume the normal flow, or end
  - failure extension – an exception that causes the use case to fail, without reaching its objective
    - must state the failure condition – what is true after a failure

## Extend: Inspect Balance

- What else can happen?
  - user can ask optionally to have a printed balance slip
  - user may mistype their PIN code and have to re-enter it
  - user's bank card may be unreadable, or out of date
- Suggest using this template
  - state name of the extension and the extension point – where in the main flow it branches out
  - write the enabling condition – what triggers this flow?
  - write down flow of events as a dialogue, as before.
  - write the success condition or the failure condition – if the extension accomplishes something extra/different

## Optional Extension

| | |
|---|---|
| **Use Case:** | Inspect Balance |
| … | |
| **Extension:** | Print Balance |
| **Point:** | Inspect Balance, after step 6 |
| **Require:** | The customer selects the "print balance" option immediately after viewing the balance. |
| **Event flow:** | 1. The ATM prints the current balance on a slip of paper. Resume use case. |
| **Success:** | The customer also has a printed balance slip. |

An optional extension is for optional behaviour that is only executed if the enabling condition is satisfied. When finished, resume the main flow in the success scenario at the same point as you left it (here, from step 7).

# Alternative Extension

…

|  |  |
|---|---|
| **Extension:** | Re-enter PIN |
| **Point:** | Inspect Balance, after step 3 |
| **Require:** | The customer inserted an incorrect PIN code, but has done this less than three times in succession. |
| **Event flow:** | 1. The ATM rejects the incorrect PIN code and displays a warning message.<br>2. After a few seconds the ATM asks the customer to re-enter the PIN code. Resume use case at step 3. |

An alternative extension is for an alternative path executed when the enabling condition is satisfied. When finished, may resume the main flow at a specified point (as here), or end the use case with a success condition.

# Failure Extension

| | |
|---|---|
| **Extension:** | Reject Card |
| **Point:** | Inspect Balance, after step 1 |
| **Require:** | The ATM cannot read the bank card, or the card is out of date. |
| **Event flow:** | 1. The ATM cannot read the bank card and displays a warning message.<br>2. After a few seconds the ATM returns the bank card and displays the welcome screen. End use case. |
| **Failure:** | The customer has the bank card. The ATM is displaying the welcome screen. |

A failure extension describes an exception triggered when the enabling condition is satisfied. The exception typically resets the system, or cleans up the failure, but must end the use case and supply a failure condition.

# Incremental Approach

- Strong Points
  - the business stakeholder will focus on successful outcomes first – psychologically relevant
  - the analyst asks later what could go wrong after each step – helps to cover all possibilities
  - easy to add variant flows after each main use case
- Weak Points
  - not all applications have a “distinguished” normal flow
  - by default, UML treats all extensions as optional insertions
  - labelled “resume” resembles poorly-structured programs with goto statements [Simons, 1999]

## Structuring Use Cases

- Initial Stage
  - collected all the success scenarios and extensions
  - possible repetition in the descriptions of use cases
- Review Stage
  - identify use cases with common sub-sequences
  - extract the common sub-sequences as extra use cases
  - simplify the main use cases by including extra use cases
- Use Case Diagram
  - revise the diagram using the UML «include» dependency to link main use cases to included use cases
  - rewrite the use case descriptions accordingly

# Overlapping Descriptions

| | |
|---|---|
| **Use Case:** | Inspect Balance … |
| **Event Flow:** | 1. The customer inserts the bank card for the account. 2. The ATM recognises the account and asks for the PIN. 3. The customer inputs a four-digit PIN code. 4. The ATM confirms the PIN code and displays a menu. … |

Common subsequence is to validate the Account and User

| | |
|---|---|
| **Use Case:** | Withdraw Cash … |
| **Event Flow:** | 1. The customer inserts the bank card for the account. 2. The ATM recognises the account and asks for the PIN. 3. The customer inputs a four-digit PIN code. 4. The ATM confirms the PIN code and displays a menu. … |

## The Included Case

| | |
|---|---|
| **Name:** | Validate Customer |
| **Purpose:** | Read the bank card and validate the PIN code. |
| **Require:** | The customer has a bank card, and knows a secret PIN code. The ATM is displaying a welcome screen. |
| **Event Flow:** | 1. The customer inserts the bank card for the account. 2. The ATM recognises the account and asks for the PIN. 3. The customer inputs a four-digit PIN code. 4. The ATM confirms the PIN code and displays a menu. |
| **Success:** | The account has been recognised and the PIN code is valid. The ATM is displaying an options menu. |

Find a suitable name and purpose for the common subsequence, so that it may be described as a use case. Identify the enabling condition and success condition.

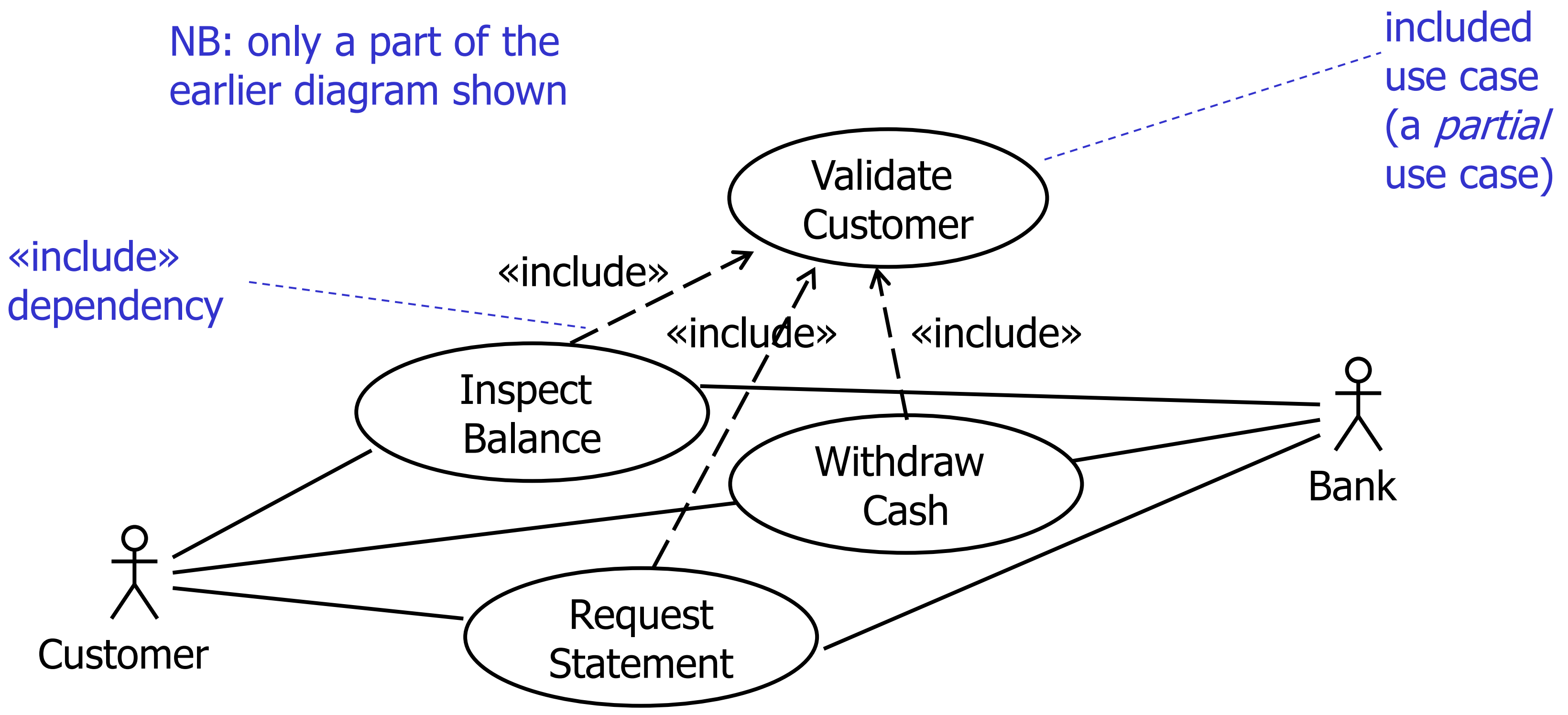
## The Including Case

| | |
|---|---|
| **Name:** | Inspect Balance |
| **Purpose:** | Allow the customer to view the balance of his/her account. |
| **Require:** | The customer has a bank card, and knows a secret PIN code.<br>The ATM is displaying a welcome screen. |
| **Event Flow:** | 1. Include Validate Customer.<br>2. The customer selects the "inspect balance" option.<br>3. The ATM displays the balance for the account.<br>4. The ATM returns the bank card after a few seconds. |
| **Success:** | The customer has the bank card and has seen the balance.<br>The ATM is displaying the welcome screen. |

Simplify the descriptions of the main use cases that use the common sequence, so that they include the common use case as one of their steps.

# Including Use Cases

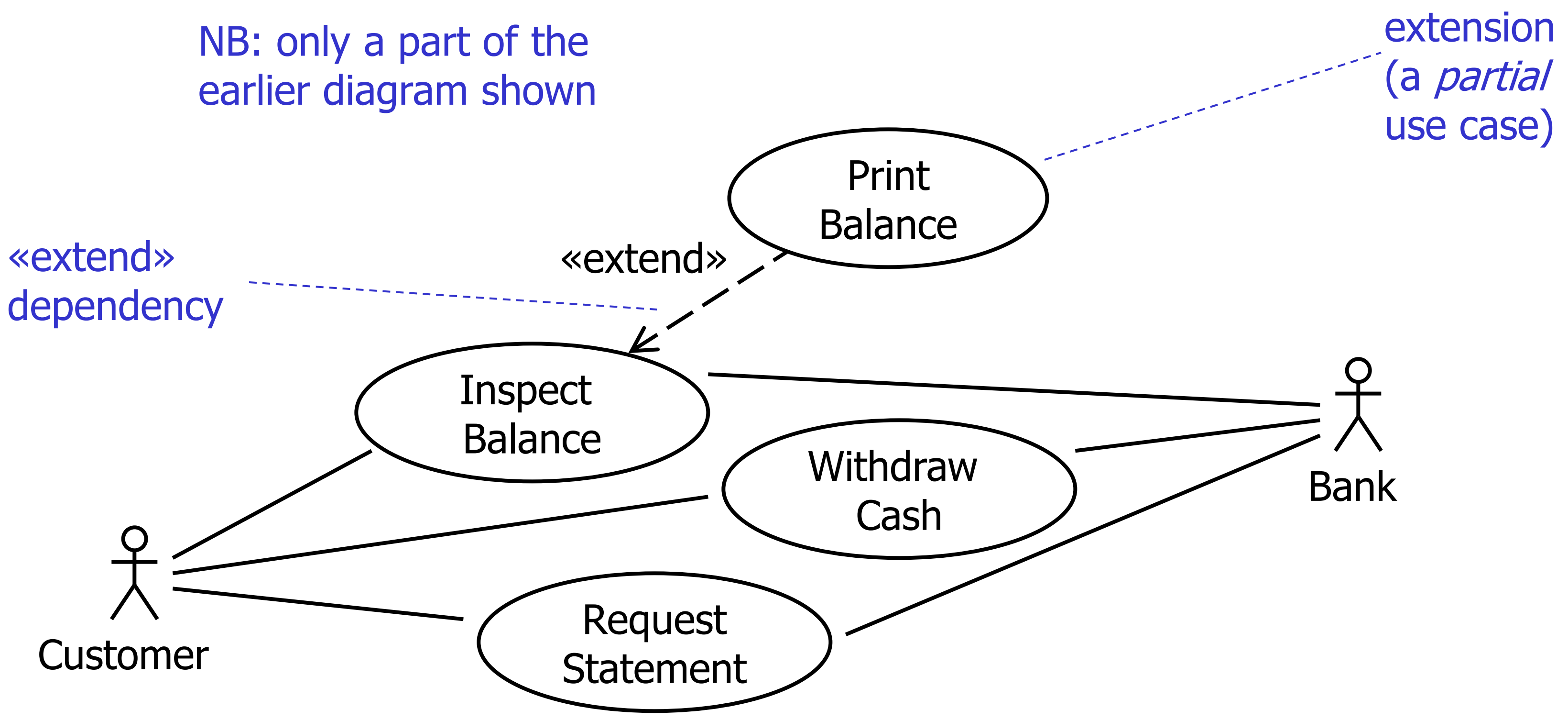
NB: only a part of the earlier diagram shown



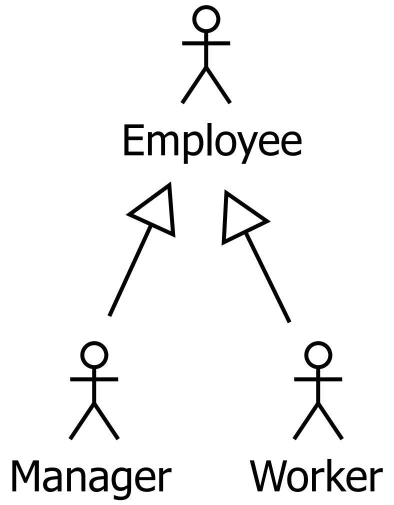
## Showing Custom Options

- Initial Stage
  - collected all the success scenarios and extensions
  - all extensions treated as variant flows of the use case
- Review Stage
  - some extensions are still just simple variant flows
  - some extensions are custom options for the system
  - make custom options explicit in the use case diagram
- Use Case Diagram
  - revise diagram using the UML «extend» dependency to link significant extensions to the main use cases
  - don’t do this for every extension, just for the custom options that you want to make explicit

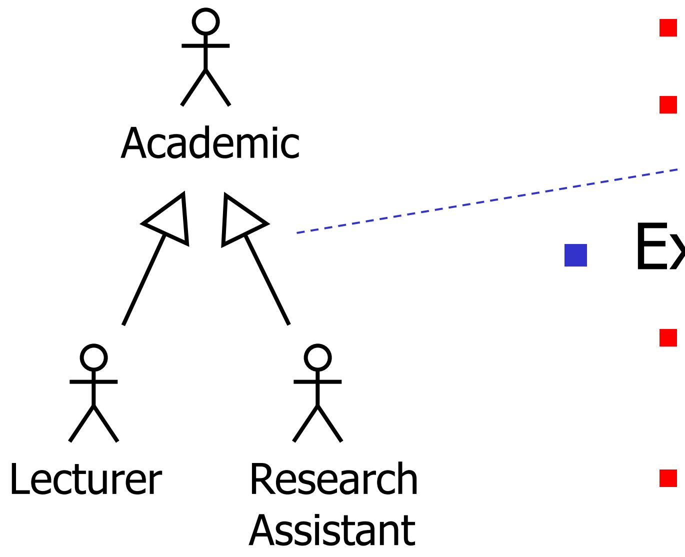
# Extending Use Cases



# Actor Hierarchy

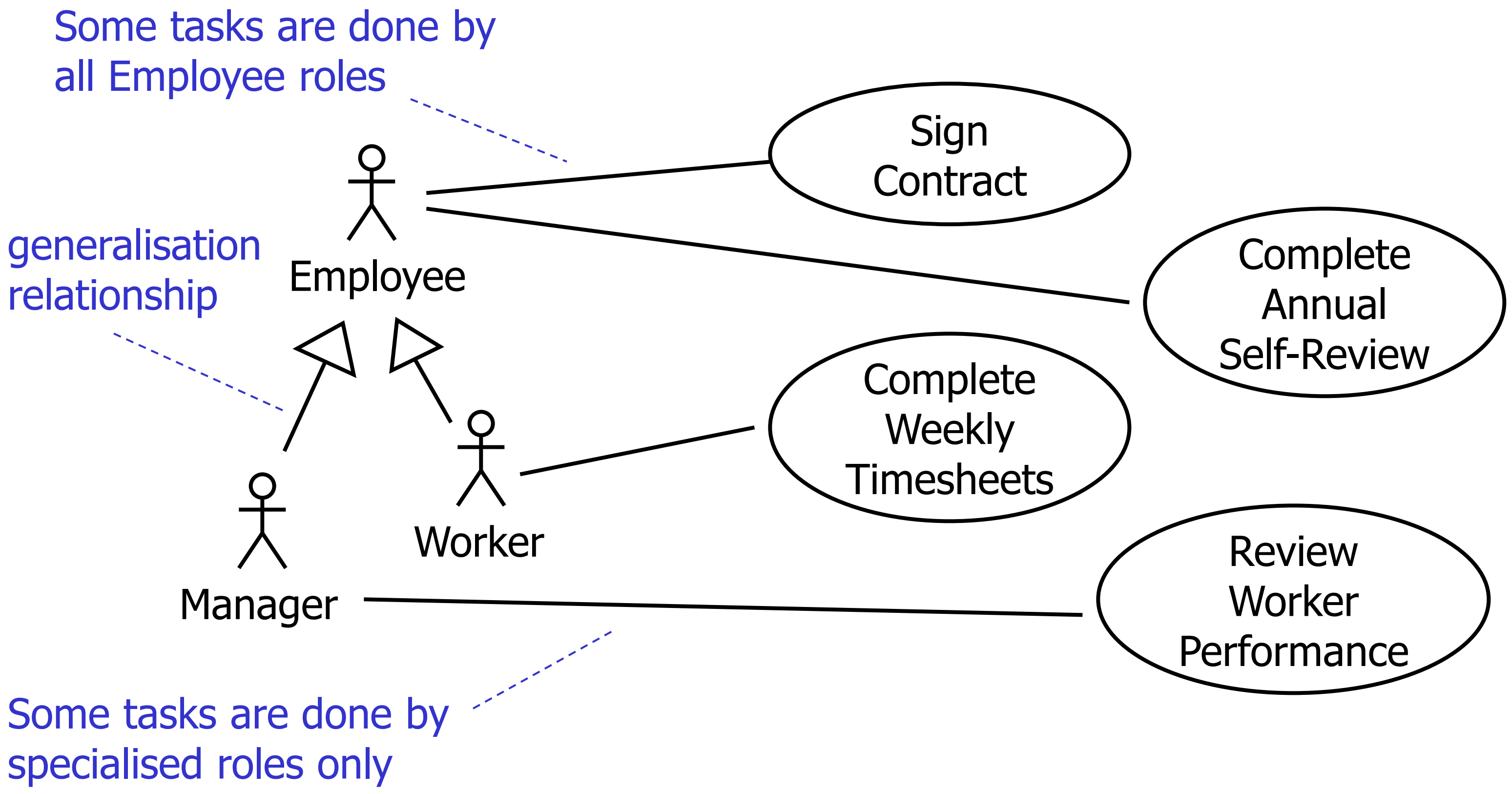


Identify subgroups that have different roles



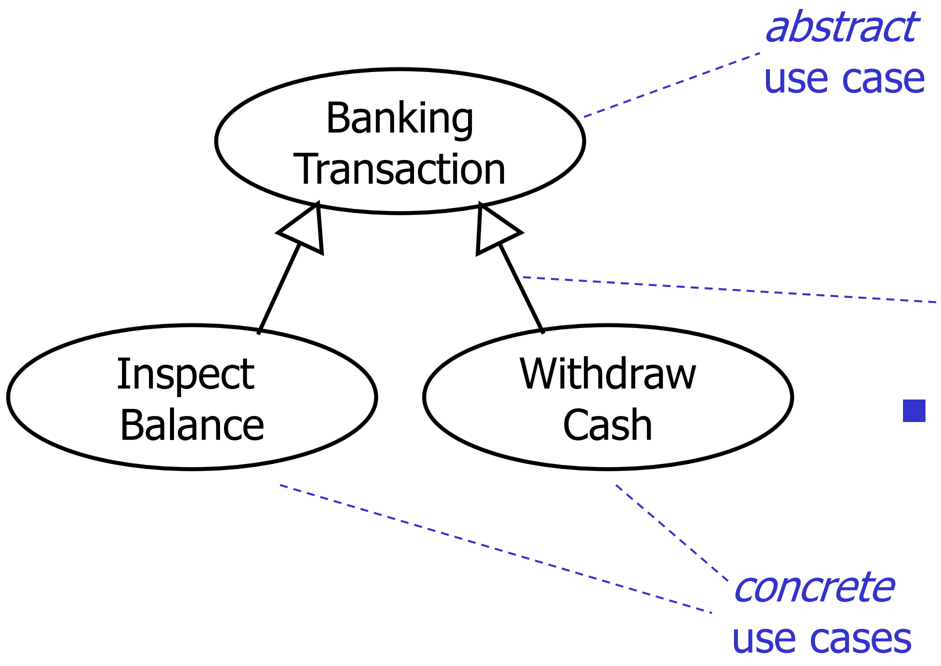
- Generalisation
  - an important semantic relationship in UML
  - indicates "a kind of"
  - draw using the UML generalisation arrow
- Examples
  - a Manager is a kind of Employee
  - a Lecturer is a kind of Academic

# Actor Generalisation


Some tasks are done by all Employee roles
Employee
Sign Contract
Complete Annual Self-Review
generalisation relationship
Complete Weekly Timesheets
Worker
Manager
Review Worker Performance
Some tasks are done by specialised roles only

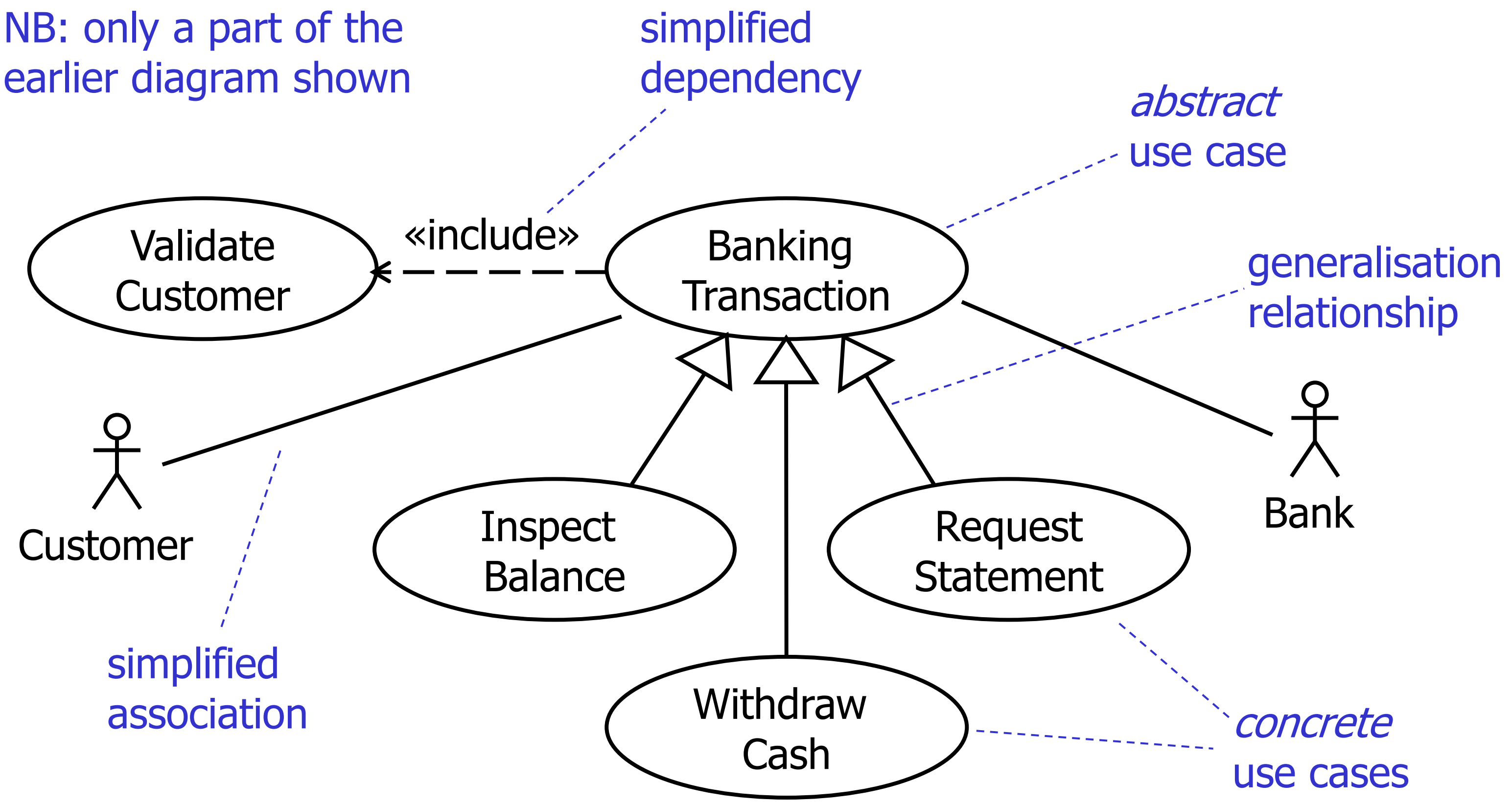
# Use Case Hierarchy

Identify use cases that share similar abstract behaviour

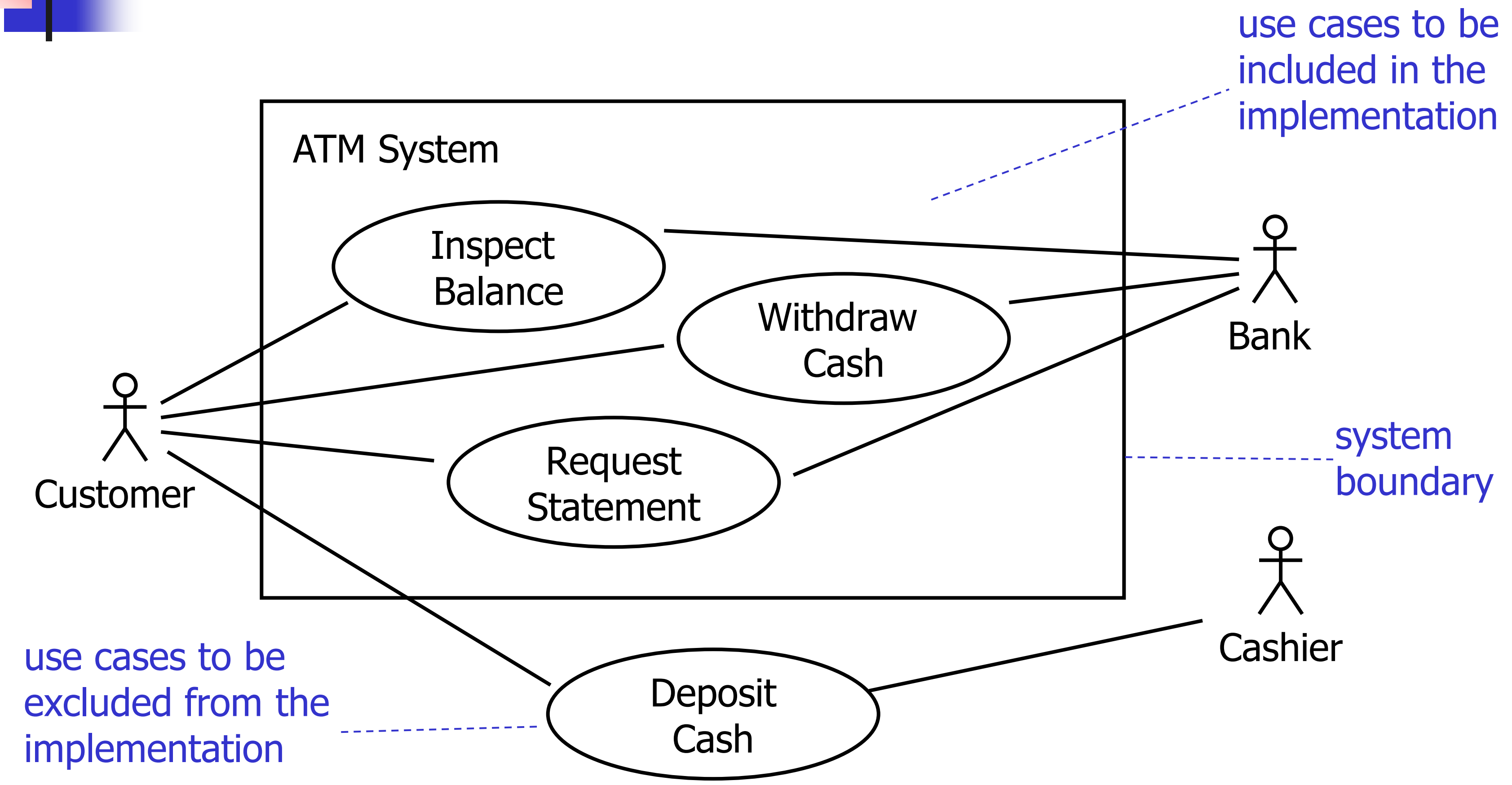


- Similar use cases
  - can be viewed as the same abstract use case
  - discover generalisations among use cases
  - draw using the UML generalisation arrow
- Example
  - Withdraw cash, inspect balance are "kinds of" banking transaction

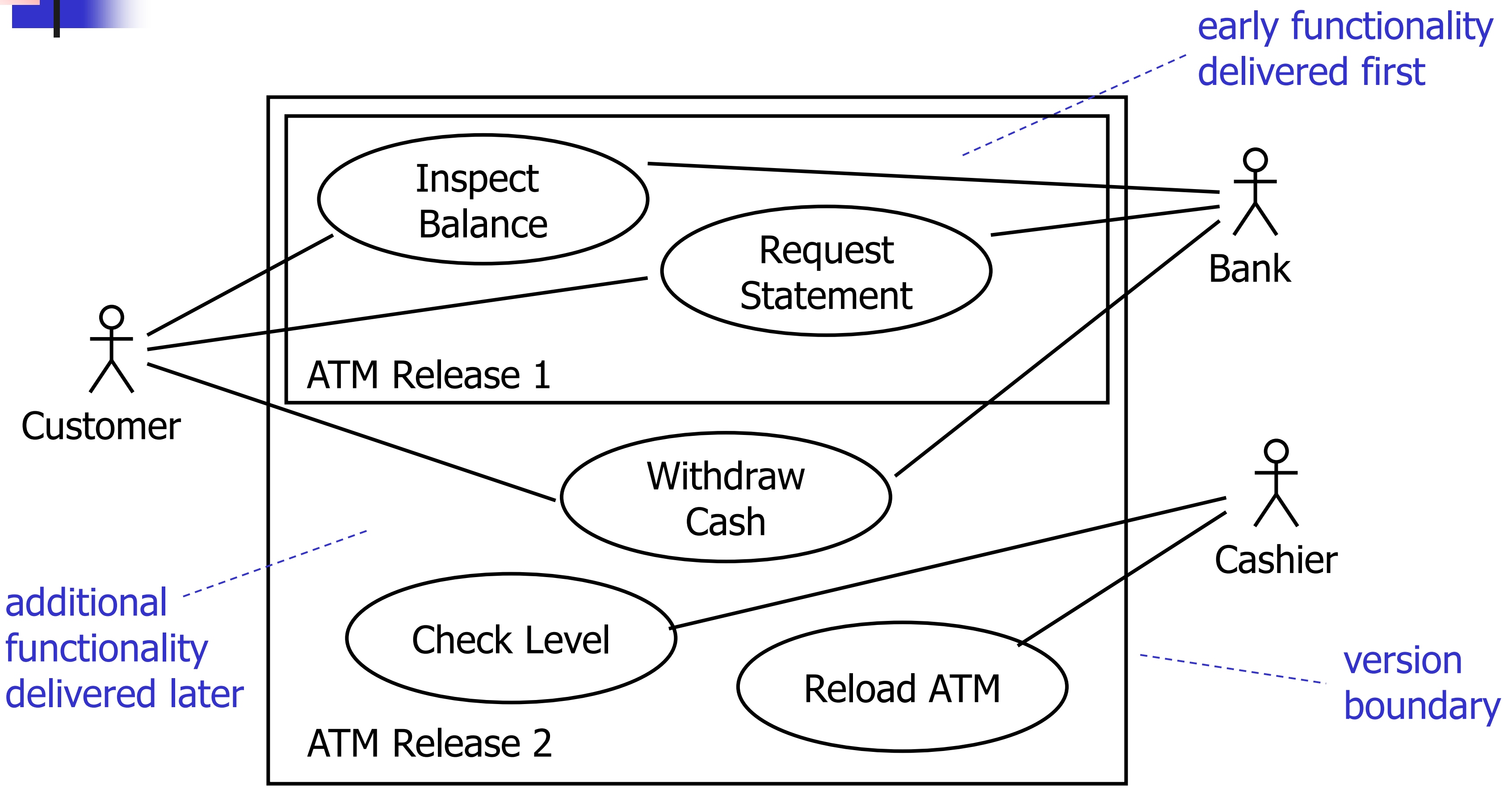
# Use Case Generalisation



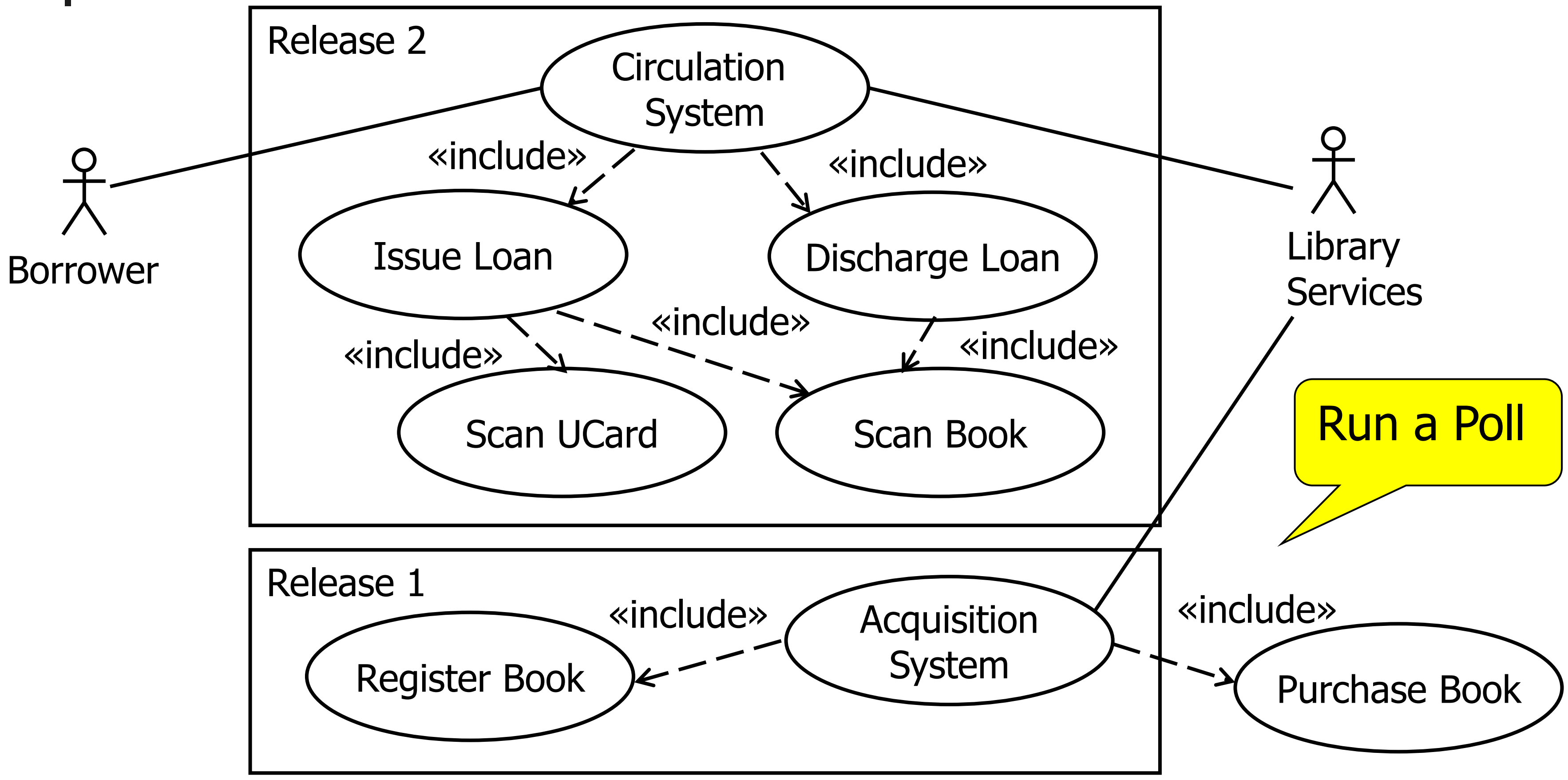
# System Boundary



# System Releases



# Lab 2: Right or Wrong?



Run a Poll

## Granularity

- Check granularity of each use case
  - a single complete interaction
  - delivers a result of business value
- Avoid modelling larger tasks
  - e.g. *Circulation System* is a collection of tasks!
  - accomplishes more than one business result
- Avoid modelling algorithm steps
  - e.g. *Scan UCard* is a step in *Issue Loan*
  - does not accomplish a business result by itself

## Structure

- Use Cases are not for structured design
  - not intended to capture function dependency
  - do not use for subroutine calls
  - do not use for conditional branching
  - results in horrible programs (see Simons article)
- Why the Use Case dependency notation?
  - only used to simplify writing requirements documents
  - only one level of dependency linking to sub-cases
  - if in doubt, don't use *«include», «extend»* at all!

## Purpose in UML

- Use Cases central in UML
  - collection of use cases are a kind of specification
  - everything else must be compliant with this
- Some correspondences
  - interaction diagrams – every message request corresponds to a step in a use case
  - collaboration diagrams – are cooperating objects that deliver a single use case
  - activity diagrams – use cases become high-level activity that is decomposed
  - state machine diagrams – use cases become high-level transitions in the GUI

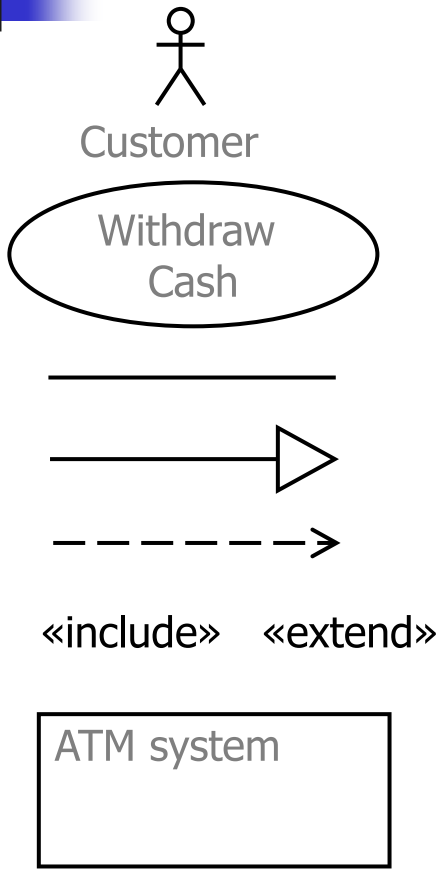
## Use Case Scenarios

- A use case
  - consists of the success scenario and all of its extensions
  - each extension point is a branch in the use case logic
  - you can trace the main flow, variant flows of events
- A scenario
  - is defined as one complete path traced through a use case ending in success or failure
  - i.e. a trace from the start of the use case, via one or more branches, to the end of the use case
    - e.g. a withdrawal that succeeds in giving out £50 cash
    - e.g. a withdrawal that fails because the ATM has no banknotes

## Testing from Scenarios

- Basis for testing
  - the specification for the system is all of its scenarios
  - each scenario represents one complete behaviour of the system – is like a script for a test case
  - the system works as desired, if each of its scenarios can be executed without faults
- Test-driven development
  - UML use cases help you to prepare your tests while collecting system requirements
  - use scenarios to drive unit testing (of each use case)
  - use scenarios to drive user-acceptance testing (of system)

# Review of UML Syntax



**Actor** – participant in use case; a person or an external system acting in a role

**Use case** – related sequence of transactions satisfying a business objective

**Association** – indicating participation

**Generalisation** – "a kind of" relationship

**Dependency** – a relationship where the source depends on the target

**Stereotype** – a kind of label used to refine the meaning of a relationship

**System** or **subsystem boundary**

## Summary

- Elicit functional requirements by identifying actors and the use cases in which they participate
- An actor is a person or an external system interacting with the current system in a particular role
- A use case is a related sequence of transactions by an actor with a system that delivers a result of value
- Use case diagrams allow use cases to be documented visually, structured and related to the participating actors
- Templates for "fully dressed" use cases allow documentation of the details of each use case and its extensions
- Use cases consist of multiple scenarios, the normal/variant flows of events, which are used as test scripts