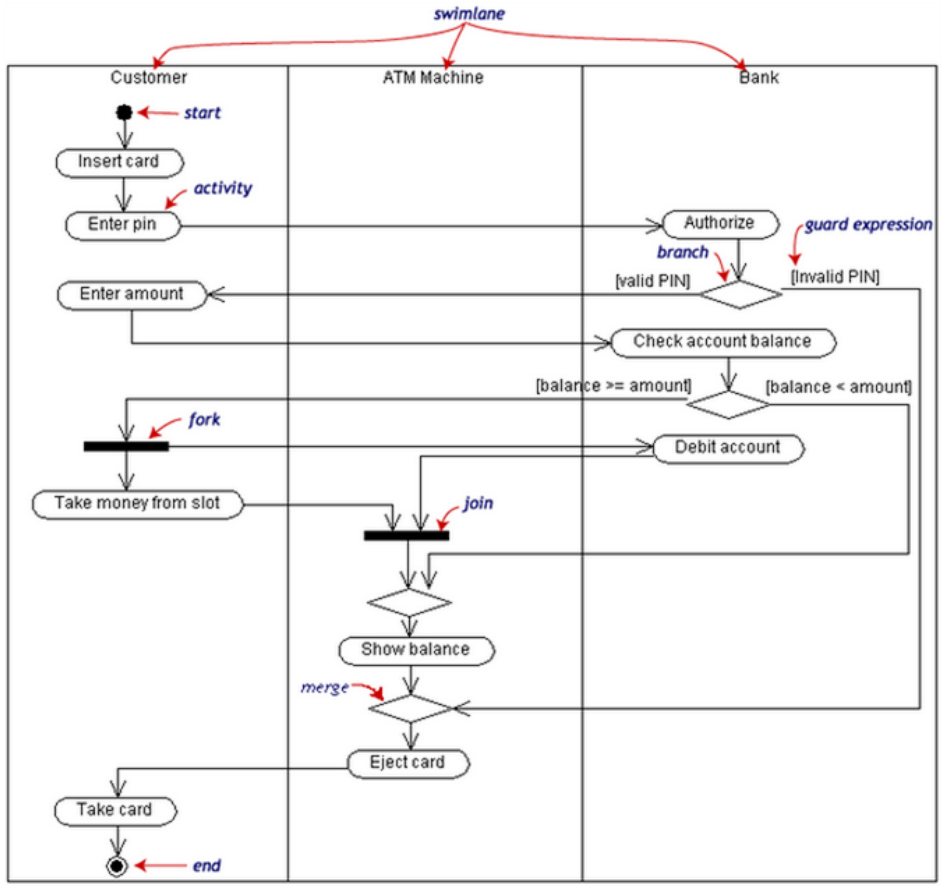
"Withdraw money from a bank account through an ATM."

The three involved classes (people, etc.) of the activity are **Customer**, **ATM**, and **Bank**. The process begins at the black start circle at the top and ends at the concentric white/black stop circles at the bottom. The activities are rounded rectangles.

Hide image



Requirements engineering

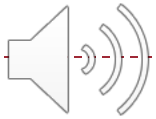
- Requirements elicitation
- Requirements modelling
- Use case analysis
  - Modelling technique: use-case diagrams and descriptions
- Requirements validation (will do in the class this week)

Dr Cunjin Luo, School of Computer Science and Electronic Engineering, University of Essex

Requirements

- Requirements engineering: making decisions about "what are we going to build?"
  - Focus on WHAT (the user needs), not HOW (to achieve it)
- **Requirement:**
  - An expression of desired behaviour ('feature')

  or
  - A constraint which must be satisfied
- A requirement exists either because —
  - The product demands certain functions
  - The client demands certain features
  - External (legal, organizational, financial etc.) demands
    - For example: the legal requirement from Website to be accessible to users with sight disabilities



Reminder: why Are Requirements Important?

- Basili and Perricone report
  - 48% of the faults observed in a medium-scale software project were attribute to "incorrect or misinterpreted functional specification or requirements"
- Top factors that caused projects to fail
  - Incomplete requirements
  - Lack of user involvement
  - Unrealistic expectations
  - …
- Requirements errors can be expensive if not detected early
  - It is much cheaper to 'fix' the requirements than the implementation

Basili and Perricone (1984). Software Errors and Complexity: An Empirical Investigation. Communications of the ACM, pp42-52, Jan 1984.

Types of requirements

- **Functional requirement**: describes the functionality of the system: what it 'does'
  - 'The program should be able to send email'
  - 'The program must guide the Mars Rover from point A to point B'
- **Non-functional requirement**: all other requirements
  - In particular: requirements that contribute to the internal qualities of software (safety, security, maintainability, …)
  - Examples:
    - Hardware requirements ('it must run on mobile phones with 1 MB memory')
    - Operational environment requirements ('it must run on the Android operating system')
    - Process requirements ('it must conform to level 3 in the CMM or above')
    - Human-Computer Interaction requirements ('it must use the Windows Vista native graphical user interface')
    - Design and architectural requirements ('it must be structured by the Client-Server architectural style')
- **Constraints** or **pseudo-constraints**: legal, organizational, financial etc. Constraints
  - **Example: "The Data Protection Act requires all personal data to be removed after client's relationship with the company ends"**

Requirement elicitation

User Requirements

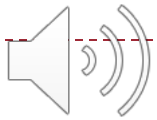
- Need to understand how the organization operates at present
- What are the problems with the current system?
- What are the requirements users have of a new system that are not in the current system?

The *Agile* approach

- Advocates of Agile methods focus on developing the new system and not on extensive analysis of the existing system
- In the Agile Manifesto they state that they value working software over comprehensive documentation

Requirements elicitation

- **Purpose**: To gather requirements from clients
- **Challenges**: bridging the gap between non-technical (users) and technical (software engineers) people
  - Customers do not always understand/aware of their needs
  - Customers have trouble articulating their needs
  - Customers do not like to be observed
  - Inconsistencies
- **Stakeholders**
  - **Clients**: pay for the software to be developed
  - **Customers**: buy the software after it is developed
  - **Users**: use the system
  - **Domain experts**: familiar with the problem
  - **Market researchers**: determine future trends & potential customers
  - **Lawyers or auditors**
  - **Software engineers** or other technology experts



Fact Finding Techniques

- Background Reading
- Interviewing
- Observation
- Document Sampling
- Questionnaires

Background Reading

- Aim is to understand the organization and its business objectives
- Includes:
  - reports
  - organization charts
  - policy manuals
  - job descriptions
  - documentation of existing systems

Interviewing

- Aim is to get an in-depth understanding of the organization's objectives, users' requirements and people's roles
- Includes:
  - managers to understand objectives
  - staff to understand roles and information needs
  - customers and the public as potential users

Interviewing

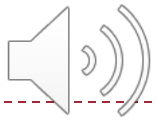
- Advantages:
  - personal contact allows the interviewer to respond adaptively to what is said
  - it is possible to probe in greater depth
  - if the interviewee has little or nothing to say, the interview can be terminated

Interviewing

- Disadvantages:
  - can be time-consuming and costly
  - notes must be written up or tapes transcribed after the interview
  - can be subject to bias
  - if interviewees provide conflicting information this can be difficult to resolve later

Observation

- Aim is to see what really happens, not what people say happens
- Includes:
  - seeing how people carry out processes
  - seeing what happens to documents
  - obtaining quantitative data as baseline for improvements provided by new system
  - following a process through end-to-end
- Can be open-ended or based on a schedule



Observation

- Appropriate situations:
  - when quantitative data is required
  - to verify information from other sources
  - when conflicting information from other sources needs to be resolved
  - when a process needs to be understood from start to finish

Document Sampling

- Aims to find out the information requirements that people have in the current system
- Also aims to provide statistical data about volumes of transactions and patterns of activity
- Includes:
  - obtaining copies of empty and completed documents
  - counting numbers of forms filled in and lines on the forms
  - screenshots of existing computer systems

Questionnaires

- Aims to obtain the views of a large number of people in a way that can be analysed statistically
- Includes:
  - postal, web-based and email questionnaires
  - open-ended and closed questions
  - gathering opinion as well as facts

YES/NO Questions

Do you print reports from the existing system? YES NO 10
(Please circle the appropriate answer.)

Multiple Choice Questions

How many new clients do you obtain in a year? 11
(Please tick one box only.)

a) 1–10 ☐
b) 11–20 ☐
c) 21–30 ☐
d) 31 + ☐

Scaled Questions

How satisfied are you with the response time of the stock update?
(Please circle one option.)

1. Very satisfied 2. Satisfied 3. Dissatisfied 4. Very dissatisfied 12

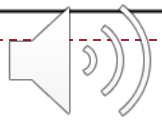
Open-ended Questions

What additional reports would you require from the system?

_____

_____

_____



Questionnaires

- Appropriate situations:
  - when views of large numbers of people need to be obtained
  - when staff of organization are geographically dispersed
  - for systems that will be used by the general public and a profile of the users is required

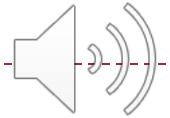
Requirements modelling & specification

Use cases

Object-oriented analysis

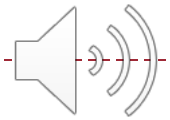
Requirements languages and notations

- Purpose: to represent the requirements in an explicit, structured form
- Input: a report (in natural language) on elicited requirements
- Output:
  - **Formal requirements specifications languages** use mathematical languages to represent or model functional specifications
    - (Logic, Z, OCL)
    - Statecharts
  - **Informal** requirement specification notations
    - Use-case diagrams
    - Class diagrams
    - Type diagrams
    - (sequence diagrams, interaction diagrams)



Requirements Specification

- Should include (as a minimum):
  - Problem Definition – what is the problem being addressed?
  - Viewpoint Structure – what is the scope and who is involved?
  - Functional Requirements – what should the system do?
  - Non-functional requirements – how should the system behave?
- Standards for Requirements Specification:
  - Software requirements specification IEEE 830
  - IEEE Recommended Practice for Software Requirements Specifications
  - http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=392555&sortType%3Dasc_p_Sequence%26filter%3DAND%28p_Publication_Number%3A3114%29



Functional and non-functional requirements

- The requirements specification should contain a **list** of functional and non-functional requirements
- A functional requirement defines **a function of a system** or its component. A function is described as a set of inputs, the behavior, and outputs.
  - Eg. The system shall print an invoice for the work completed
- A non-functional requirement is a requirement that specifies criteria that can be used to **judge the operation of a system**, rather than specific behaviors
  - Eg. the system should be implemented in Java
- All requirements should be **traceable** to their source, and if possible **prioritized** by their level of importance

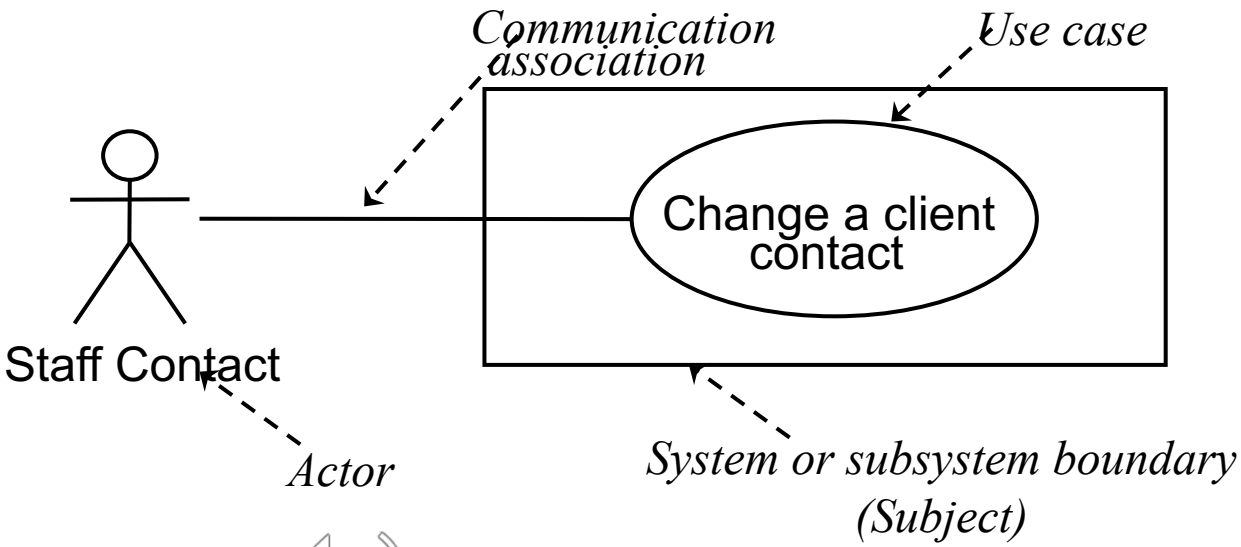
Requirements elicitation: Use case diagrams

Drawing Use Case Diagrams

- Purpose
  - document the functionality of the system from the users' perspective
  - document the scope of the system
  - document the interaction between the users and the system using supporting use case descriptions (behaviour specifications)
  - Do NOT specify the flow of the implementation
  - provide a starting point for capturing and analysing requirements
  - **One of the first activities that you should do**

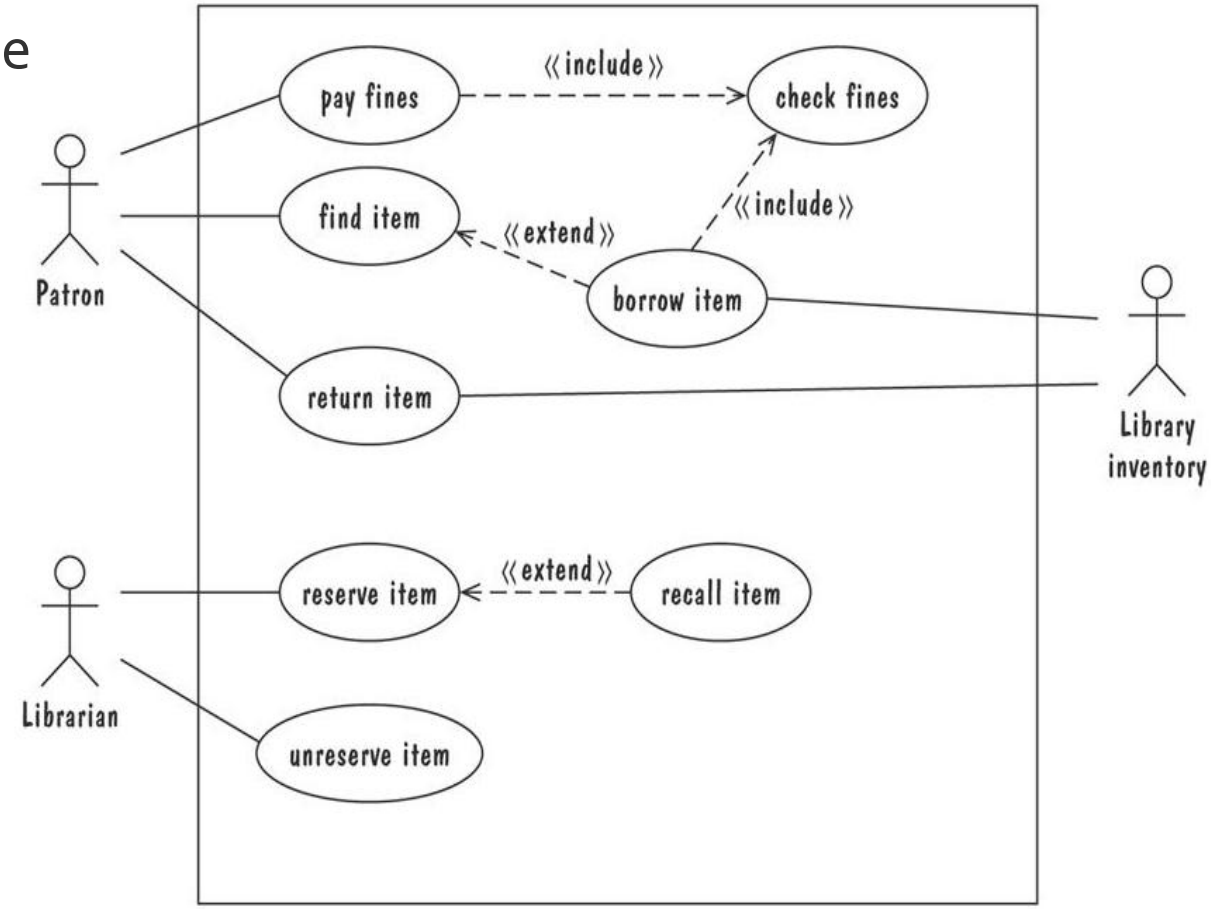
Modelling requirements technique: use-case diagrams

- Represent the results of use-case analysis
- Vocabulary:
  - Box: system boundary
  - Stick figures (outside the box): actors, human and systems
  - Oval (inside the box): a use case
    - Represents some major required functionality and its variant
    - Heuristic: individual menu item often corresponds to a use case
  - Line between actor—use case : the actor participates in the use case



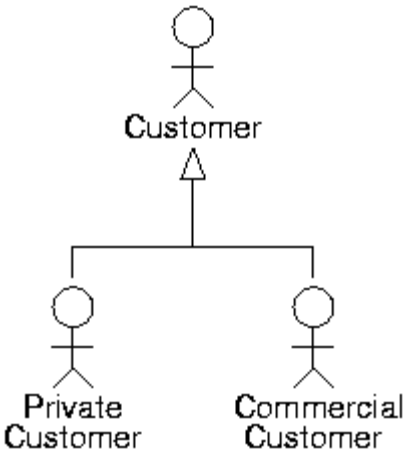
Use -case diagrams: example

- Library use cases:
  - Borrowing a book
  - Returning a borrowed book
  - Paying a library fine
  - Reserving items
  - Unreserving items



Actors

- A coherent set of roles that users of use cases play when they interact with use cases
  - Typically represents the role that a human, hardware device or another system plays with the system
  - Actors are not part of the system
  - As an actor is a class, it can be generalized
- Examples:
  - *Registrar*: maintain the curriculum
  - *Billing System*: receive billing information from registration



Notation of Use Case Diagrams

- Use cases
  - drawn as ellipses with a name in or below each ellipse
  - describe a sequence of actions that the system performs to achieve an observable result of value to an actor
  - the name is usually an active verb and a noun phrase

Notation of Use Case Diagrams

- Communication associations
  - line drawn between an actor and a use case
  - represent communication link between an instance of the use case and an instance of the actor. These are **NOT** labelled.

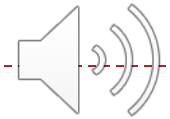
Notation of Use Case Diagrams

- Subjects (subsystems)
  - drawn as a rectangle around a group of use cases that belong to the same subject
  - in a CASE tool, use cases for different subjects are usually placed in separate use case diagrams

Notation of Use Case Diagrams

- Dependencies
  - **Extend** and **Include** relationships between use cases
  - shown as stereotyped dependencies
  - stereotypes are written as text strings in guillemets: «extend» and «include»

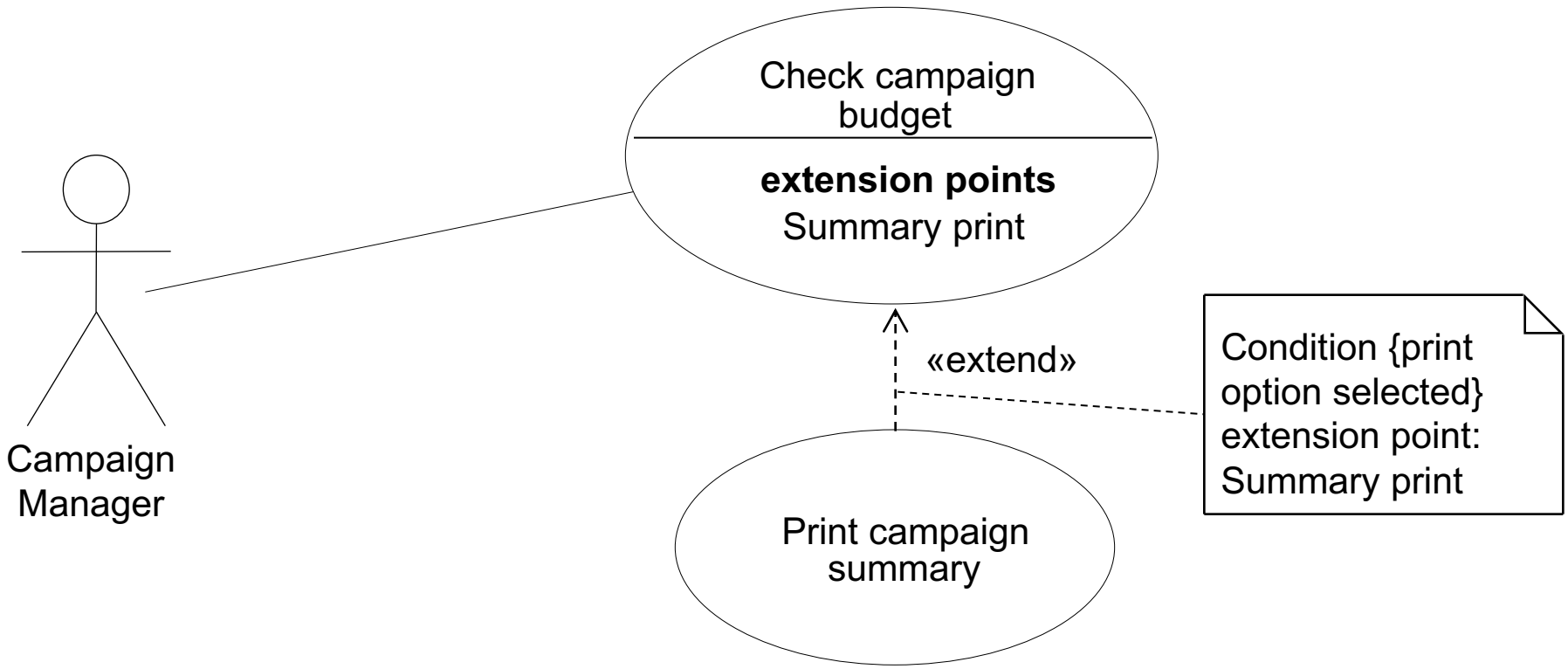
- Generalisation
  - To show that one use case is a **type** of another use case
  - Show which**specific** use case is the **generalised** type and which are the types
  - Use <<generalise>> or inheritance notation



Notation of Use Case Diagrams

- Extend relationship
  - used when one use case provides additional functionality that **may** be required in another use case
  - there may be multiple ways of extending a use case, which represent variations in the way that actors interact with the use case
  - extension points show when the extension occurs
  - a condition can be placed in a note joined to the dependency arrow (Note that it is not put in square brackets, unlike conditions in other diagrams.)

Extend relationship

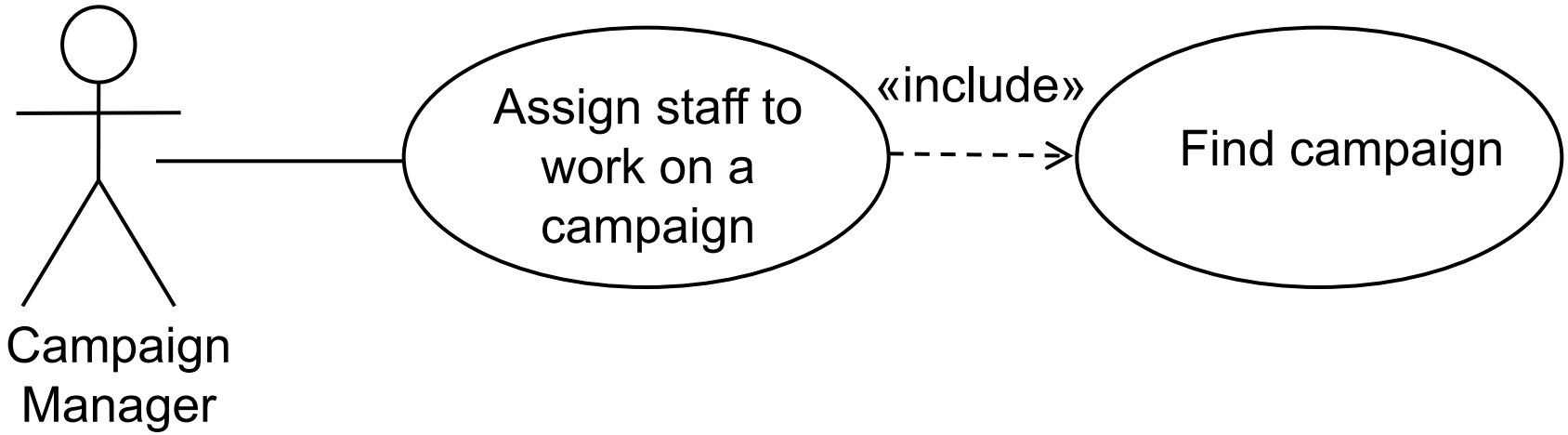


Arrow direction goes from sub-use-case towards the larger use-case

Notation of Use Case Diagrams

- Include relationship
  - used when one use case **always** includes the functionality of another use case
  - a use case may include more than one other
  - can be used to separate out a sequence of behaviour that is used in many use cases
  - should not be used to create a hierarchical functional decomposition of the system

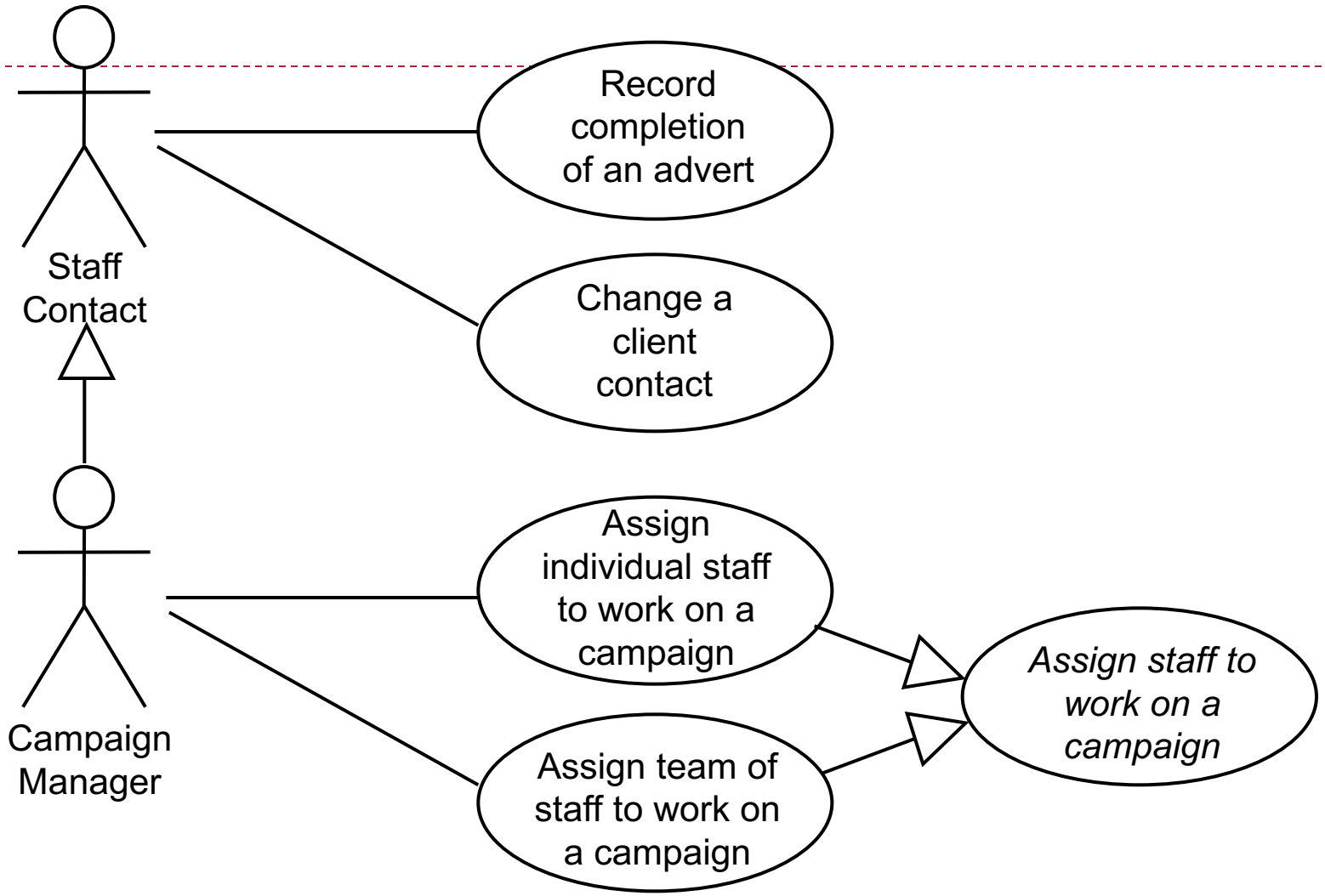
Include Relationship



Arrow direction goes from the larger use-case towards sub-use-case (different to <<extend>>)

Notation of Use Case Diagrams

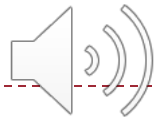
- Generalization
  - shows that one use case provides all the functionality of the more general use case and some additional functionality
  - shows that one actor can participate in all the associations with use cases that the more general actor can plus some additional use cases



Arrow direction goes from the child use-case towards parent use-case

Summary of relations between use cases

- <<include>>: one use case incorporates the behaviour of another use case (eg. where behaviour is used frequently in a number of use cases, to avoid repetition)
  - 'borrow item' includes 'check fine'
  - 'pay fine' include 'check fine'
- <<extend>>: One use case adds to another (ie. provides additional/optional functionality)
  - 'Recall Item' extends 'Reserve Item': It involves reserving an item and adds additional steps to it
- <<generalize>>: One use case is special case of (is-kind-of) another
  - 'Enter book details' is a generalization of 'scan book details'
  - 'Enter book details' is a generalization of 'type book number'



Drawing Use Case Diagrams

- Identify the actors and the use cases
- Prioritize the use cases
- Develop each use case, starting with the priority ones, writing a description for each
- Add structure to the use case model: generalization, include and extend relationships and subsystems

Requirements elicitation: Use case descriptions

Use Case Descriptions

- Very similar to a scenario
- Based on predicted uses of the new system
- Should aim to capture use-case descriptions for all scenarios
- Ideally there should be a **use-case description** for every use case bubble in the **use-case diagrams**
- Each use-case description may only be a simple paragraph …
  Assign staff to work on a campaign
  - *The campaign manager wishes to record which staff are working on a particular campaign. This information is used to validate timesheets and to calculate staff year-end bonuses.*

Requirement analysis technique: scenario analysis

- Jennifer is standing in front of the lift on floor 3 in the Computer Science building.
- She presses a lift request button for upward direction.
- The lift arrives and the doors open.
- Jennifer enters the lift and presses the floor request button for floor 5A. The doors close.
- The lift travels to floor 5A and the doors open.
- Jennifer leaves the lift.

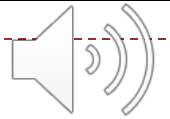
Use Case Descriptions

- It is more normal that it is a step-by-step breakdown of interaction between actor and system

Assign staff to work on a campaign

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

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



Use Case Descriptions

- Many projects use templates. Typical headings might be:
  - name of use case
  - pre-conditions
  - post-conditions
  - actors involved
  - purpose
  - description
  - alternative courses
  - errors

Use-case description (for illustration only)

- **Name of use case:** Lift operation
- **Pre-conditions:**
  - User standing in front of lift
  - Lift request button is enabled
- **Post-conditions:**
  - Lift request button is enabled on that floor
- **Actor:** Lift user
- **Purpose:** description of lift operation
- **Description:**
  - User presses lift request button
  - The direction indicators highlight the requested direction of travel. Disable lift request button on that floor.
  - Lift travels to the users floor
  - Etc .....

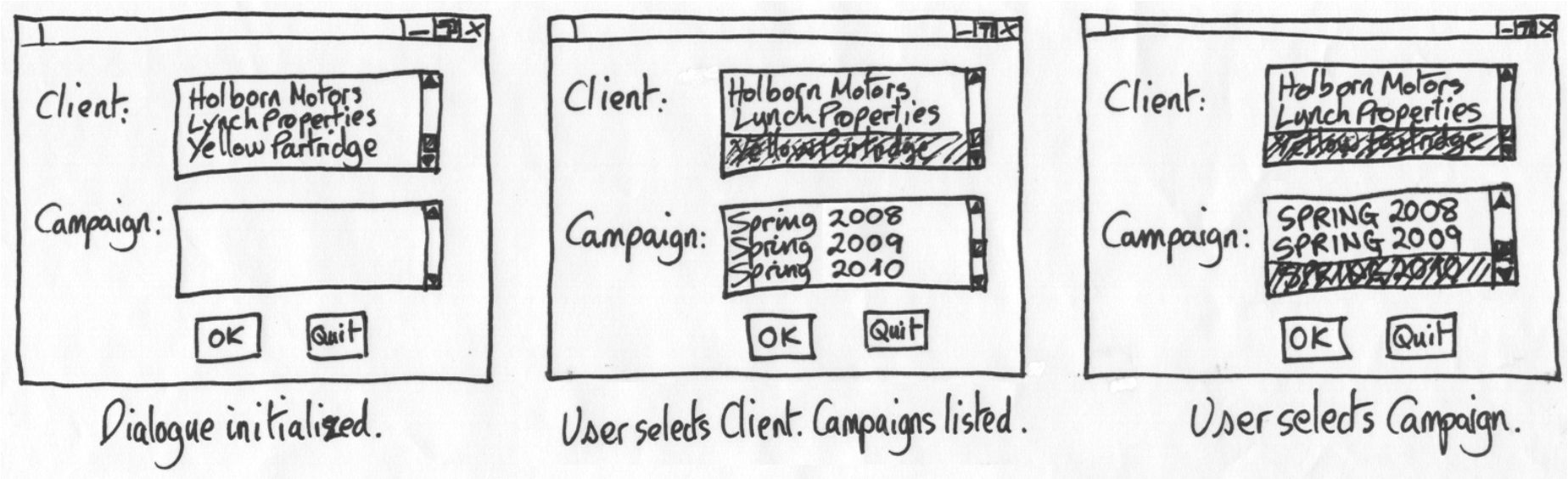
Requirements elicitation: Prototyping

Prototyping

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

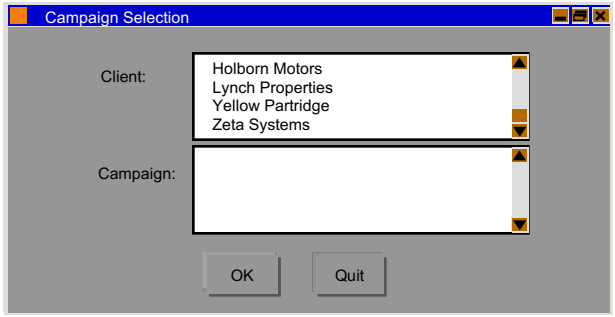
Prototyping

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

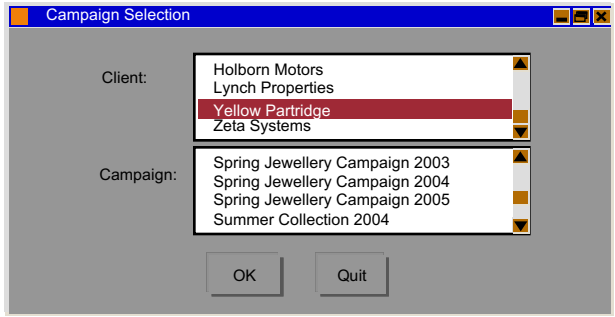


Prototyping

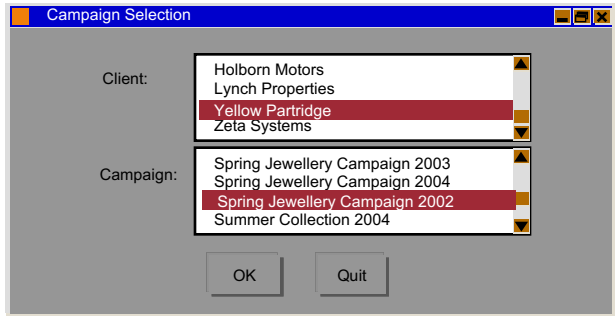
- User interface prototypes can be implemented using languages other than the one that the system will be developed in



Dialogue initialized.



User selects Client. Campaigns listed.



User selects Campaign.

In this lecture we have looked at:

- Requirements elicitation
- Requirements modelling & specification
  - Formal and informal specifications
  - Types of requirements: functional & non-functional
  - The requirements specification
- Use case analysis
  - Scenarios and use case descriptions
  - Modelling technique: use-case diagrams
  - The link between use-case diagrams and use-case descriptions
  - Use of prototyping
- Requirements validation – class exercise (next)

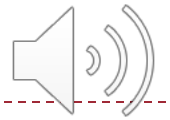
Class exercise: Requirements validation

Requirements verification & validation

- Verification: compare one document to another
- Validation: check requirements with customer

Requirements Validation

- A critical step in the development process
- Requirements validation criteria:
  - Correctness:
    - The requirements represent the client's view.
  - Completeness:
    - All possible scenarios, in which the system can be used, are described, including exceptional behavior by the user or the system
  - Consistency:
    - There are functional or nonfunctional requirements that contradict each other
  - Realism:
    - Requirements can be implemented and delivered
  - Traceability:
    - Each system function can be traced to a corresponding set of functional requirements



Example: requirement from a word-processor

- Requirements should be clear, unambiguous, understandable

> Selecting is the process of designating areas of the document that you want to work on. Most editing and formatting actions require two steps: first you select what you want to work on, such as text or graphics; then you initiate the appropriate action.

- Question: does a selected area need to be continuous?

Example: requirement from a real safety-critical system

- Precise, unambiguous, clear

> The message must be triplicated. The three copies must be forwarded through three different physical channels. The receiver accepts the message on the basis of a two-out-of-three voting policy.

- **Question**: can a message be accepted as soon as we receive 2 out of 3 identical copies of the message or do we need to wait for receipt of the 3rd?

Example: requirement from a word-processor

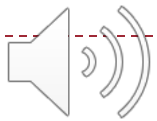
- Consistent

> The whole text should be kept in lines of equal length. The length is specified by the user. Unless the user gives an explicit hyphenation command, a carriage return should occur only at the end of a word.

- **Question**: What if the length of a word exceeds the length of the line?

Techniques of validating requirements

Validation	Walkthroughs Readings Interviews Reviews Checklists Models to check functions and relationships Scenarios Prototypes Simulation Formal inspections
Verification Checking	Cross-referencing Simulation Consistency checks Completeness checks Check for unreachable states of transitions Model checking Mathematical proofs

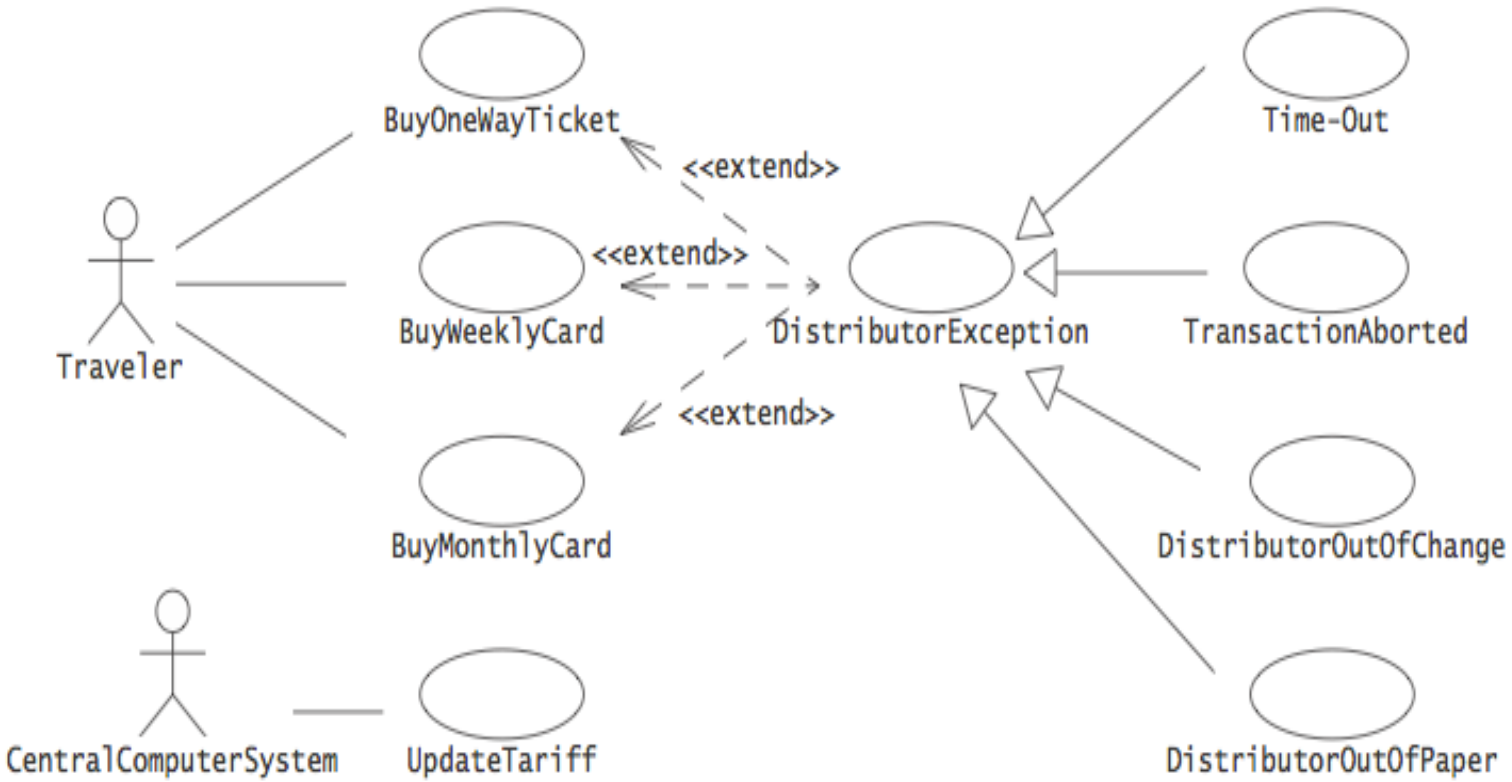


Exercise (1)

- Below are examples of non-functional requirements. Specify which of these requirements can be validated and which cannot.
  - "The system must be usable."
  - "The system must provide visual feedback to the user within 1 second of issuing a command."
  - "The availability of the system must be above 95%."
  - "The user interface of the new system should be similar enough to the old system such that users familiar with the old system can be easily trained to use the new system."

Exercise (2)

- Draw a use case diagram for a ticket distributor for a train system. The system includes two actors: a traveller, who purchases different types of tickets, and a central computer system, which maintains a reference database for the tariff.
- Use cases should include: BuyOneWayTicket, BuyWeeklyCard, BuyMonthlyCard, UpdateTariff.
- Also include the following exceptional cases: Time-Out (i.e., traveler took too long to insert the right amount), TransactionAborted (i.e., traveler selected the cancel button without completing the transaction), DistributorOutOfChange, and DistributorOutOfPaper.


BuyOneWayTicket
Time-Out
<<extend>>
<<extend>>
Traveler
BuyWeeklyCard
DistributorException
TransactionAborted
<<extend>>
BuyMonthlyCard
DistributorOutOfChange
CentralComputerSystem
UpdateTariff
DistributorOutOfPaper

Exercise 3: Divide by functional/nonfunctional

The allocation of staff to production lines should be mostly automated. A process will be run once a week to carry out the allocation based on the skills and experience of operatives. Details of holidays and sick leave will also be taken into account. A first draft Allocation List will be printed off by 12.00 noon on Friday for the following week. Only staff in Production Planning will be able to amend the automatic allocation to fine-tune the list. Once the amendments have been made, the final Allocation List must be printed out by 5.00pm. The system must be able to handle allocation of 100 operatives at present, and should be capable of expansion to handle double that number.

Examples of functional requirements are:

- the need for a process to be run that allocates staff to lines based on their skills and experience, holidays and sick leave;
- printing out an allocation list;
- amending the allocation list.

Examples of non-functional requirements include:

- printing the allocation list by 12.00 noon;
- the need to handle 200 operatives' details.

Further reading

- Basili and Perricone (1984). Software Errors and Complexity: An Empirical Investigation. Communications of the ACM, pp42-52, Jan 1984.
- Brooks, Frederick P. (1987). *No Silver Bullet: Essence and Accidents of Software Engineering*. (Reprinted in the 1995 edition of *The Mythical Man-Month*)
- SUNA - http://www.jisc.ac.uk/media/documents/programmes/eframework/scenario_based_design_chris_fowler.pdf