# Génie Logiciel et Gestion de Projets

## Software Requirements Engineering

# Software Requirements Engineering

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

# Roadmap

- Software Requirements
  - User requirements versus system requirements
  - Functional and non-functional Requirements
  - The software requirements document
- Requirements Engineering Processes
  - Feasibility studies
  - Requirements elicitation and analysis
  - Requirements validation
  - Requirements management

# Roadmap continued

- Models
  - Behavioural models
  - Data models
  - Object models

# Software Requirements

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

# Roadmap

- Software Requirements
  - User requirements versus system requirements
  - Functional and non-functional Requirements
  - The software requirements document

# Requirements Engineering

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

# What is a Requirement?

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.
- This is inevitable as requirements may serve a dual function
  - May be the basis for a bid for a contract -> must be open to interpretation;
  - May be the basis for the contract itself -> must be defined in detail;

# User vs System Requirements

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- System requirements
  - set out the system's functions, services and operational constraints in detail. The system requirements document should be precise. It should define exactly what is to be implemented. It may be part of a contract between client and the software developers.

# Functional requirements

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.

# Functional req. examples

- The user shall be able to search either all of the initial set of databases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier which the user shall be able to copy to the account's permanent storage area.

# Functional req. problems

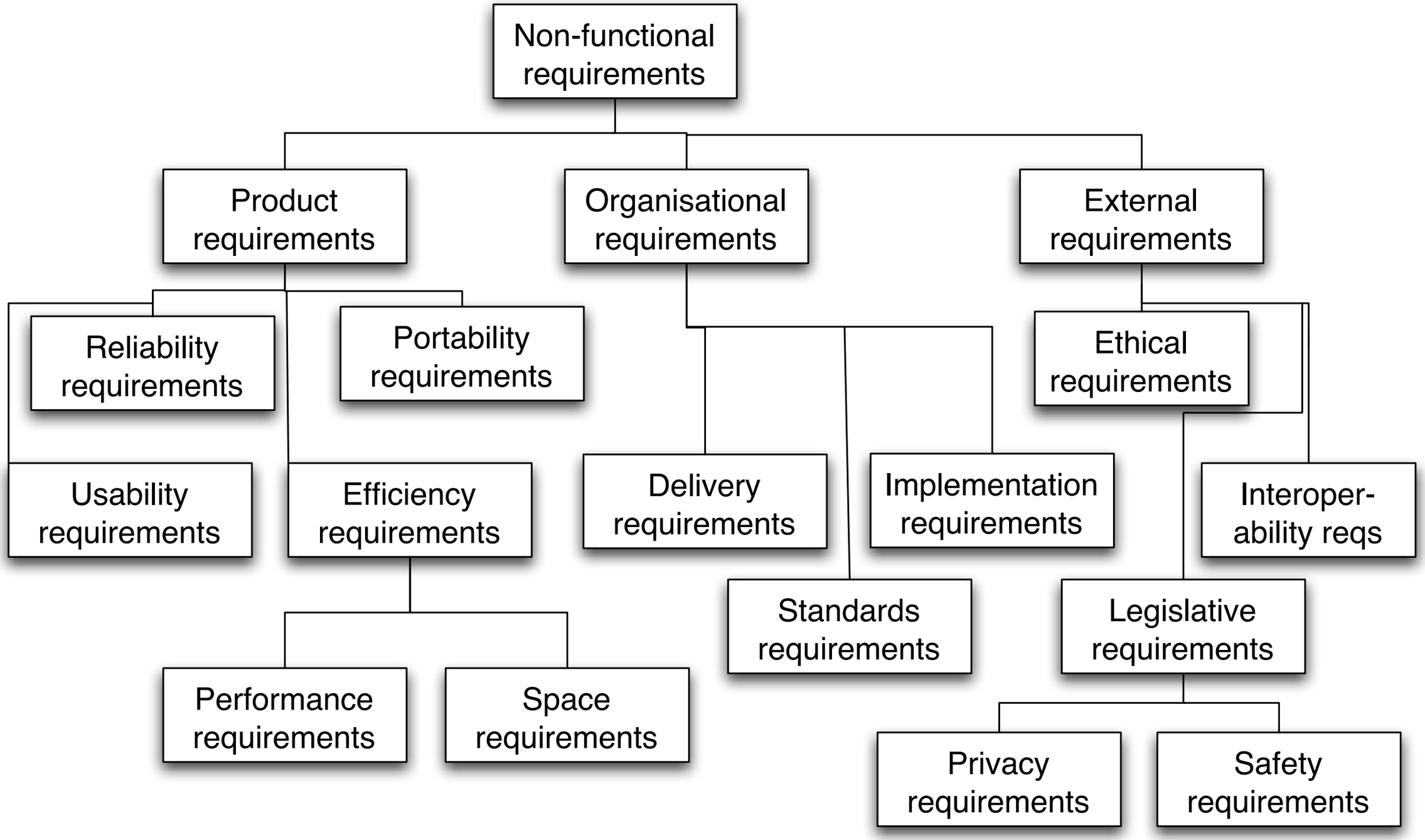
- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- In principle, requirements should be both complete and consistent.
  - Complete
    - They should include descriptions of all facilities required.
  - Consistent
    - There should be no conflicts or contradictions in the descriptions of the system facilities.
  - In practice, it is impossible to produce a complete and consistent requirements document.

# Non-functional requirements

- are constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

# Non-functional classification

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.




© Ian Sommerville 2004

# Non-functional examples

- Product requirement
  - The user interface for the library system shall be implemented as simple HTML without frames or Java applets.
- Organisational requirement
  - The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.
- External requirement
  - The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.

# Non-functional req. problems

- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
- Non-functional requirements often conflict and interact with other non-functional or functional requirements. These conflicts are common in complex systems.

# Example

- A system goal
  - The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised.
- A verifiable non-functional requirement
  - Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

# Requirements measures

| Property | Measure |
|---|---|
| Speed | Processed transactions/second<br>User/Event response time<br>Screen refresh time |
| Size | M Bytes<br>Number of ROM chips |
| Ease of use | Training time<br>Number of help frames |
| Reliability | Mean time to failure<br>Probability of unavailability<br>Rate of failure occurrence<br>Availability |
| Robustness | Time to restart after failure<br>Percentage of events causing failure<br>Probability of data corruption on failure |
| Portability | Percentage of target dependent statements<br>Number of target systems |

# Domain requirements

- Requirements that come from the application domain of the system and that reflect characteristics of that domain.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

# Domain req. examples

- There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.
- Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

# Domain req. problems

- Understandability
  - Requirements are expressed in the language of the application domain;
  - This is often not understood by software engineers developing the system.
- Implicitness
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

# User req. revisited

- Should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
- User requirements are defined using natural language, tables and diagrams as these can be understood by all users.
- Problems with natural language
  - Lack of clarity
  - Requirements confusion
  - Requirements amalgamation

# Guidelines for writing requirements

- Invent a standard format and use it for all requirements.
- Use language in a consistent way. Use *shall* for mandatory requirements, *should* for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.

# The requirements document

- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

# Users of a requirements document

| | |
|---|---|
| System Customers | Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements. |
| Managers | Use the requirements document to plan a bid for the system and to plan the system development process |
| System Engineers | Use the requirements to understand what system is to be developed |
| System Test Engineers | Use the requirements to develop validation tests for the system |
| System Maintenance Engineers | Use the requirements to understand the system and the relationship between its parts. |

# IEEE Requirements standard

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

# SRS document structure

- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

# Obstacles to good RE practice

- effort needs to be spent before the project contract is signed, without the guarantee that a contract will be signed
- heard to measure cost savings and benefits of using RE technology
- requirements doc can be felt as big and complex. Documentation gets outdated
- stronger concerns and pressure on tight schedules

# Wrap-up (1)

- Requirements set out what the system should do and define constraints on its operation and implementation.
- Functional requirements set out services the system should provide.
- Non-functional requirements constrain the system being developed or the development process.
- User requirements are high-level statements of what the system should do. User requirements should be written using natural language, tables and diagrams.

# Wrap-up (2)

- System requirements are intended to communicate the functions that the system should provide.
  - A software requirements document is an agreed statement of the system requirements.
  - The IEEE standard is a useful starting point for defining more detailed specific requirements standards.

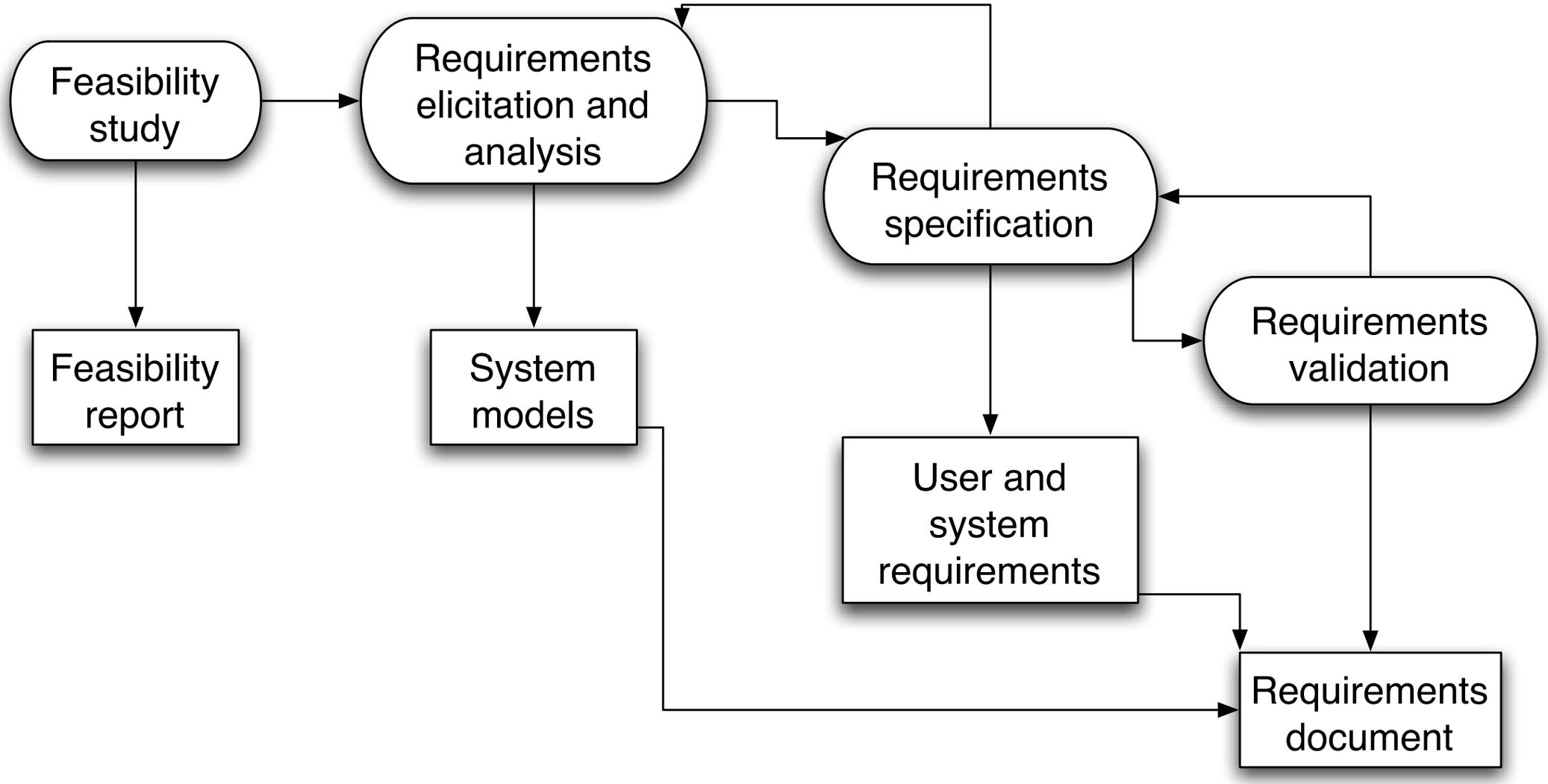
# Requirements Engineering Processes

# Roadmap

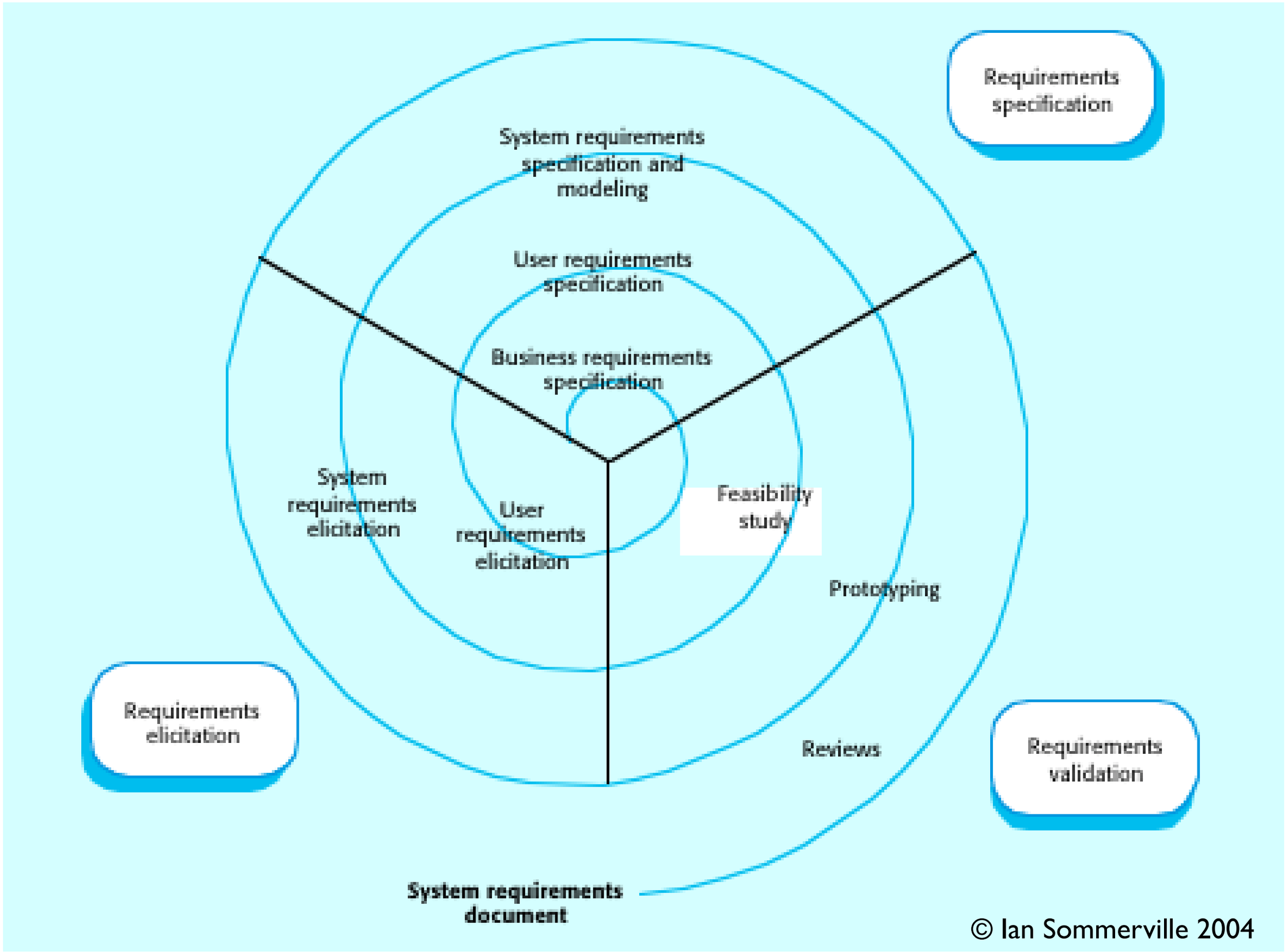
- Requirements Engineering Processes
  - Feasibility studies
  - Requirements elicitation and analysis
  - Requirements validation
  - Requirements management

# Requirements Engineering Processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.



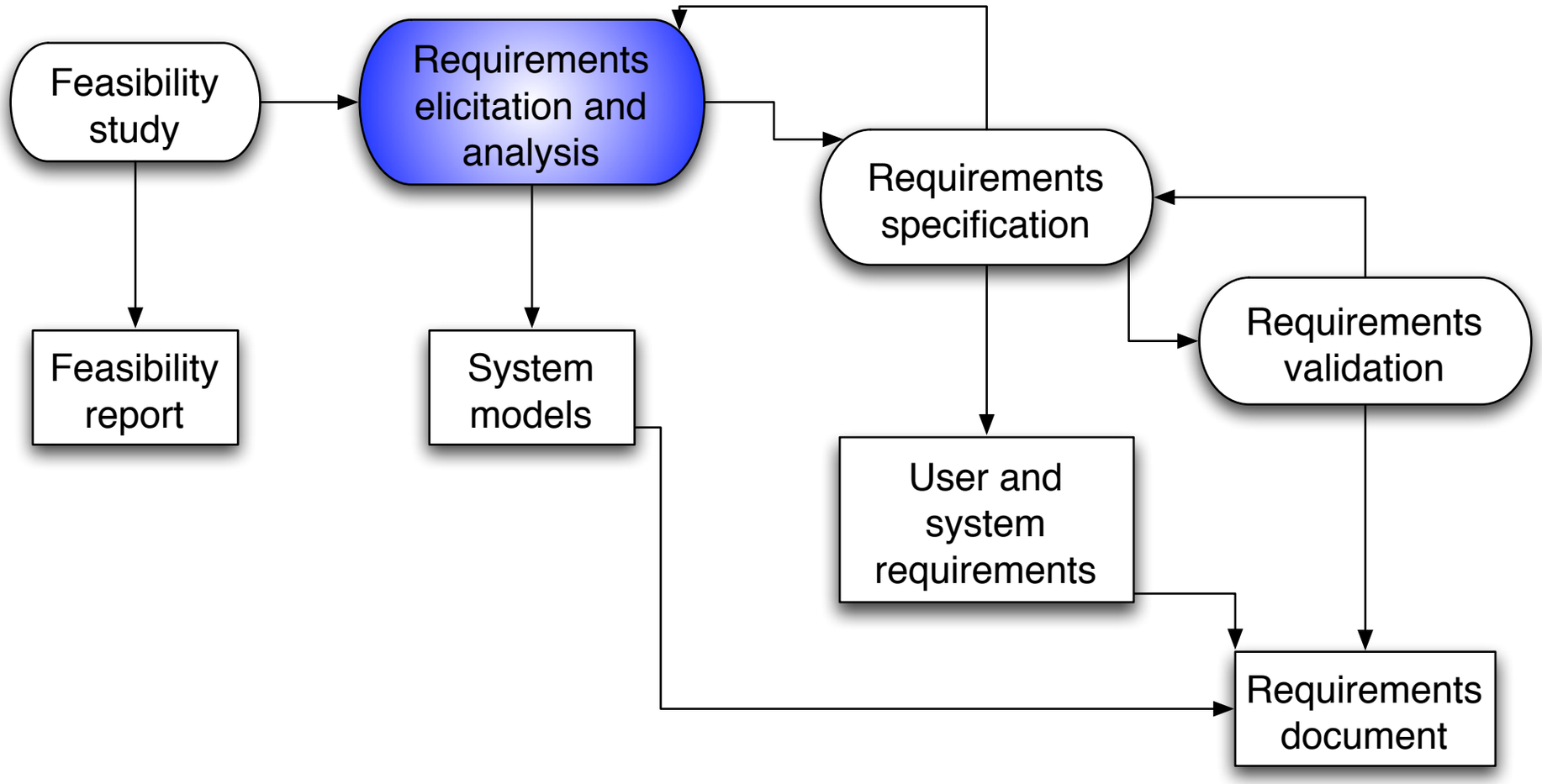

© Ian Sommerville 2004


Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
System requirements elicitation
User requirements elicitation
Feasibility study
Prototyping
Requirements elicitation
Reviews
Requirements validation
System requirements document
© Ian Sommerville 2004

# Feasibility studies

# Feasibility study

- A feasibility study decides whether or not the proposed system is worthwhile.
- A short focused study that checks
  - If the system contributes to organisational objectives;
  - If the system can be engineered using current technology and within budget;
  - If the system can be integrated with other systems that are used.




© Ian Sommerville 2004

# Requirements Elicitation and Discovery

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

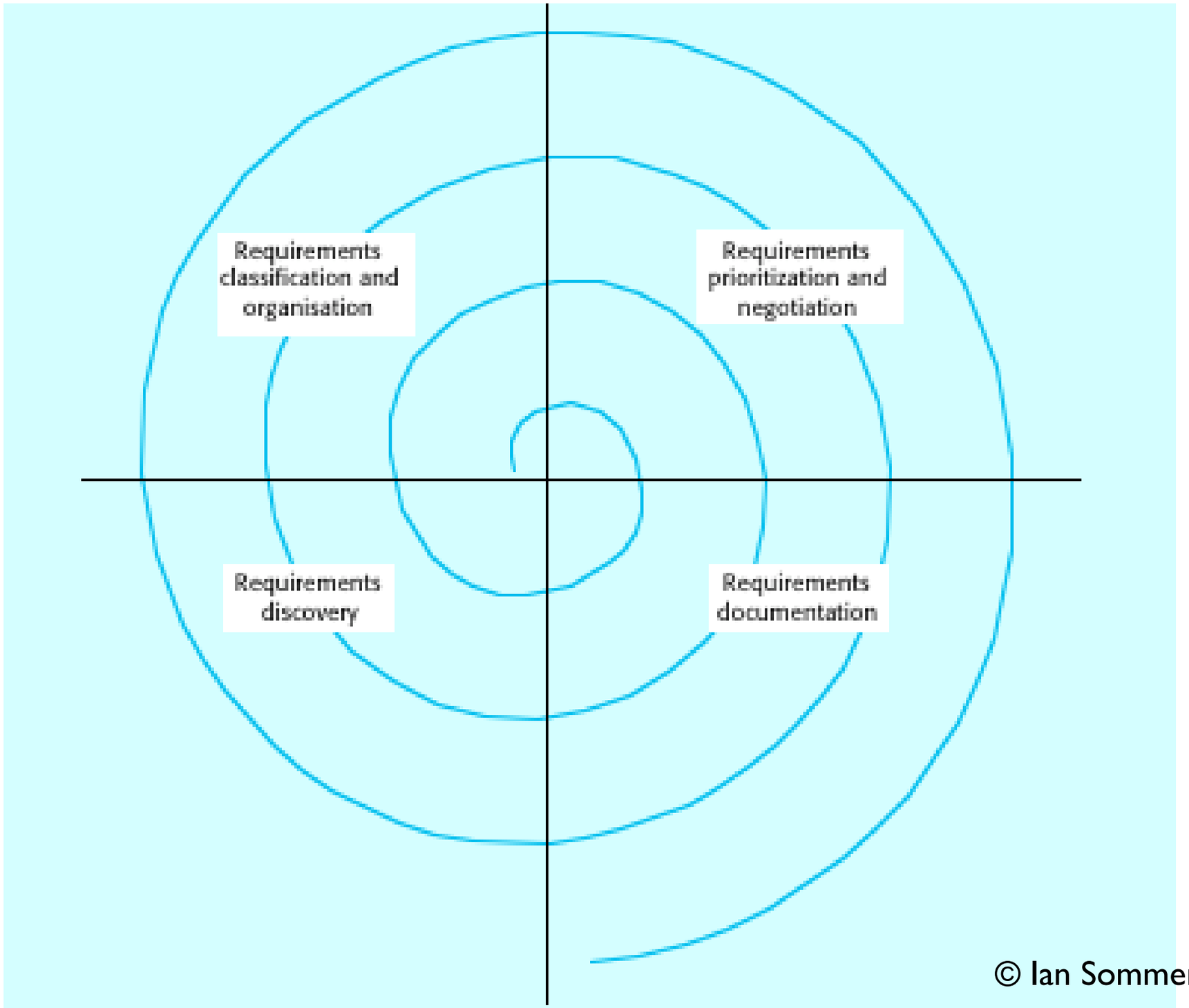
# Elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called stakeholders.

# Problems of Requirements Analysis

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment change.

# The requirements spiral




© Ian Sommerville 2004

# Process activities

- Requirements discovery
  - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
- Requirements classification and organisation
  - Groups related requirements and organises them into coherent clusters.
- Prioritisation and negotiation
  - Prioritising requirements and resolving requirements conflicts.
- Requirements documentation
  - Requirements are documented and input into the next round of the spiral.

# Requirements Discovery

# Requirements discovery

- The process of gathering information about the proposed and existing systems and distilling the user and system requirements from this information.
- Sources of information include documentation, system stakeholders and the specifications of similar systems.

# Viewpoints

- Viewpoints are a way of structuring the requirements to represent the perspectives of different stakeholders. Stakeholders may be classified under different viewpoints.
- This multi-perspective analysis is important as there is no single correct way to analyse system requirements.

# Types of viewpoints

- Interactor viewpoints
  - People or other systems that interact directly with the system. In an ATM, the customer's and the account database are interactor VPs.
- Indirect viewpoints
  - Stakeholders who do not use the system themselves but who influence the requirements. In an ATM, management and security staff are indirect viewpoints.
- Domain viewpoints
  - Domain characteristics and constraints that influence the requirements. In an ATM, an example would be standards for inter-bank communications.

# Interviewing

- In formal or informal interviewing, the RE team puts questions to stakeholders about the system that they use and the system to be developed.
- There are two types of interview
  - Closed interviews where a pre-defined set of questions are answered.
  - Open interviews where there is no pre-defined agenda and a range of issues are explored with stakeholders.

# Interviews in practice

- Normally a mix of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- Interviews are not good for understanding domain requirements
  - Requirements engineers cannot understand specific domain terminology;
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.

# Scenarios

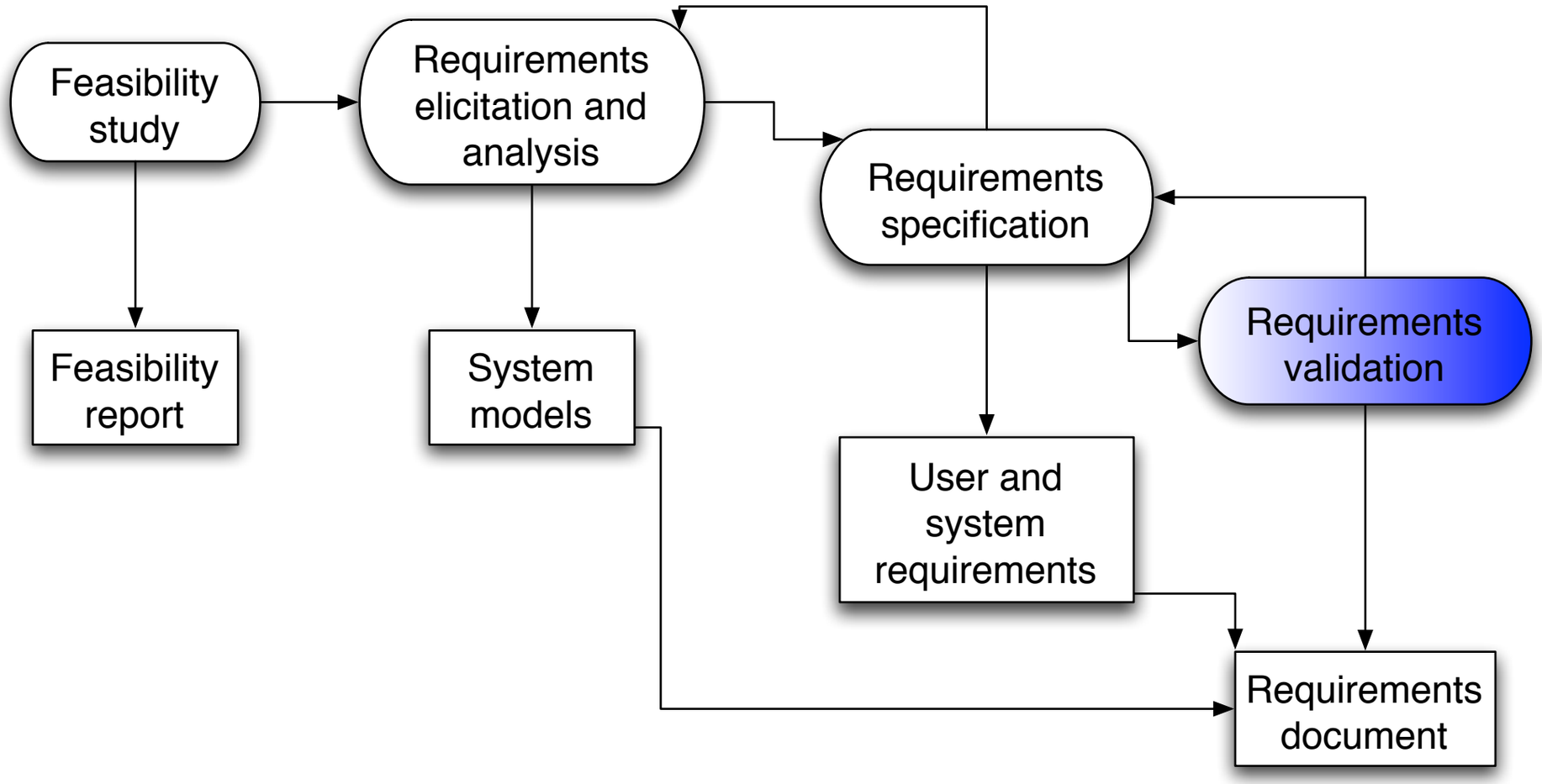
- Scenarios are real-life examples of how a system can be used.
- They should include
  - A description of the starting situation;
  - A description of the normal flow of events;
  - A description of what can go wrong;
  - Information about other concurrent activities;
  - A description of the state when the scenario finishes.

# Use cases

- Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself.
- A set of use cases should describe all possible interactions with the system.
- Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.

# Social and organisational factors

- Software systems are used in a social and organisational context. This can influence or even dominate the system requirements.
- Social and organisational factors are not a single viewpoint but are influences on all viewpoints.
- Good analysts must be sensitive to these factors but currently no systematic way to tackle their analysis.




© Ian Sommerville 2004

# Requirements Validation

# Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

# Requirements checking

- **Validity**. Does the system provide the functions which best support the customer's needs?
- **Consistency**. Are there any requirements conflicts?
- **Completeness**. Are all functions required by the customer included?
- **Realism.** Can the requirements be implemented given available budget and technology?
- **Verifiability.** Can the requirements be checked?

# Requirement validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using an executable model of the system to check requirements.
- Test-case generation
  - Developing tests for requirements to check testability.

# Requirements Management

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

# Requirements Management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed;
  - Different viewpoints have different requirements and these are often contradictory.

# Requirements management planning

- During the requirements engineering process, you have to plan:
  - Requirements identification
    - Each requirement must be uniquely identified so that it can be cross-referenced by other requirements and so that it can be used in traceability assessments.
  - A change management process
    - Set of activities that assess impact and cost of changes.

# Requirements management planning

- Traceability policies
  - The amount of information about requirements relationships that is maintained;
    - Source traceability
      - Links from requirements to stakeholders who proposed these requirements;
    - Requirements traceability
      - Links between dependent requirements;
    - Design traceability
    - Links from the requirements to the design;
- CASE tool support
  - The tool support required to help manage requirements change;

# Requirements change management

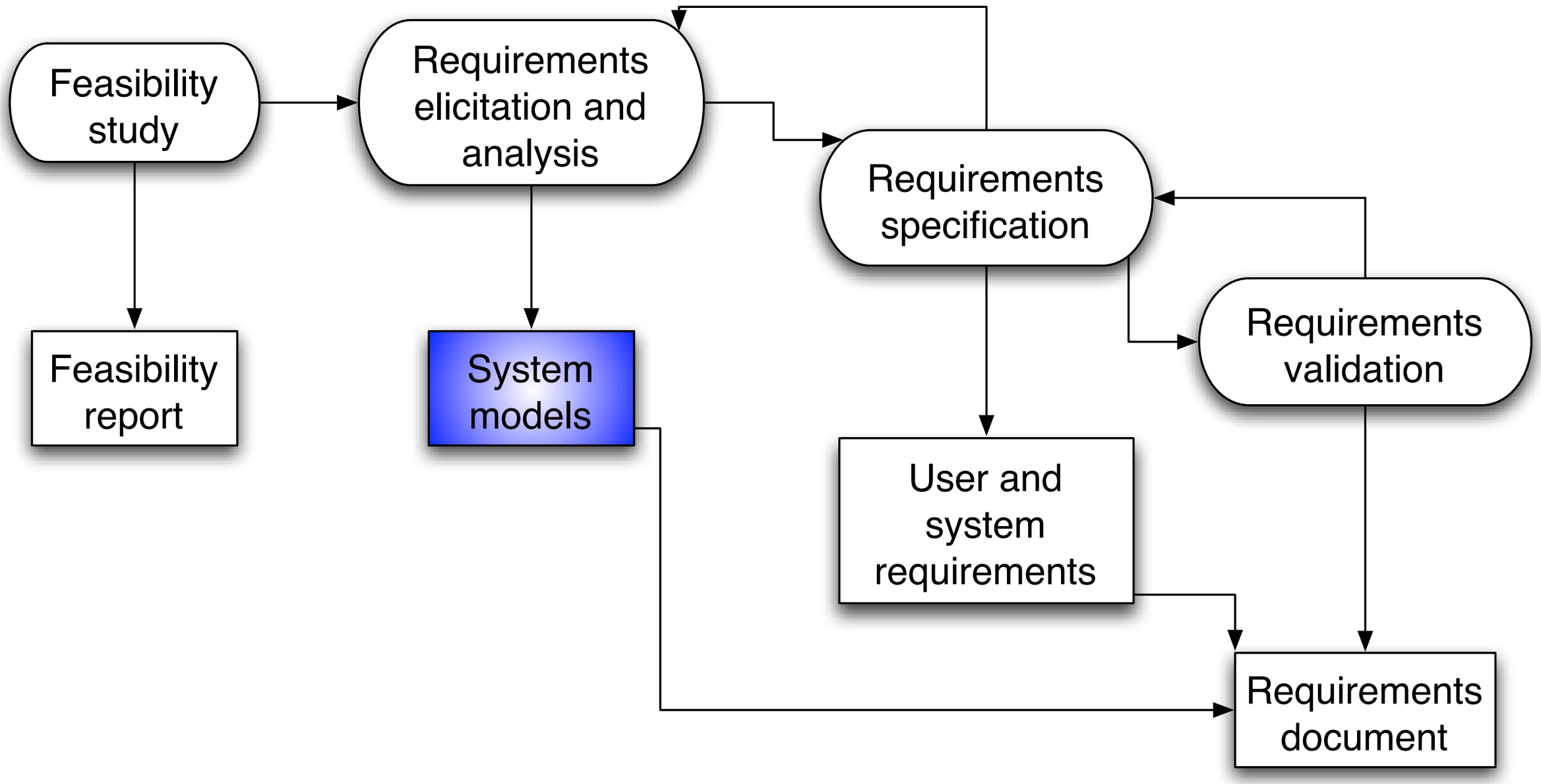
- Should apply to all proposed changes to the requirements.
- Principal stages
  - Problem analysis. Discuss requirements problem and propose change;
  - Change analysis and costing. Assess effects of change on other requirements;
  - Change implementation. Modify requirements document and other documents to reflect change.

# Wrap-up (1)

- The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification and requirements management.
- Requirements elicitation and analysis is iterative involving domain understanding, requirements collection, classification, structuring,  prioritisation and validation.
- Systems have multiple stakeholders with different requirements.

# Wrap-up (2)

- Social and organisation factors influence system requirements.
- Requirements validation is concerned with checks for validity, consistency, completeness, realism and verifiability.
- Business changes inevitably lead to changing requirements.
- Requirements management includes planning and change management.




© Ian Sommerville 2004

# System Models In RE

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

# Roadmap

- Models
  - System models???
  - The Unified Modeling Language
  - Use Cases for describing requirements
  - Behavioural models
  - Structural models

# System models??

# System models

- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.
- System models leave out detail.
- Different models present the system from different perspectives
  - *External perspective* showing the system's context or environment;
  - *Behavioural perspective* showing the behaviour of the system;
  - *Structural perspective* showing the system or data architecture.

# Unified Modeling Language

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

# The UML

- The Unified Modeling Language (UML)
  - unifies the notations of Booch, Rumbaugh (OMT) and Jacobson (OOSE).
  - is approved as a standard by the OMG and has become the de facto modelling language.
  - captures knowledge at different abstraction levels
  - different diagram types:
    - class diagrams, object diagrams, use case diagrams, deployment diagrams, component diagram , state machine diagram, activity diagram, state machine diagram, activity diagram, ....

# Ways of using the UML(1)

- As a sketch -> selectivity, exploration
  - High level diagrams to help communicate some aspects of a system
  - Can be used in 2 directions:
    - Forward engineering: draw UML diagram before writing code, discuss and communicate ideas and alternatives about the software that needs to be written
    - Reverse engineering: build diagram from existing code, use sketches to explain how some part of the software works
  - Sketches are informal and dynamic, on whiteboard or with lightweight drawing tool
- UML diagrams  in books are typically sketches

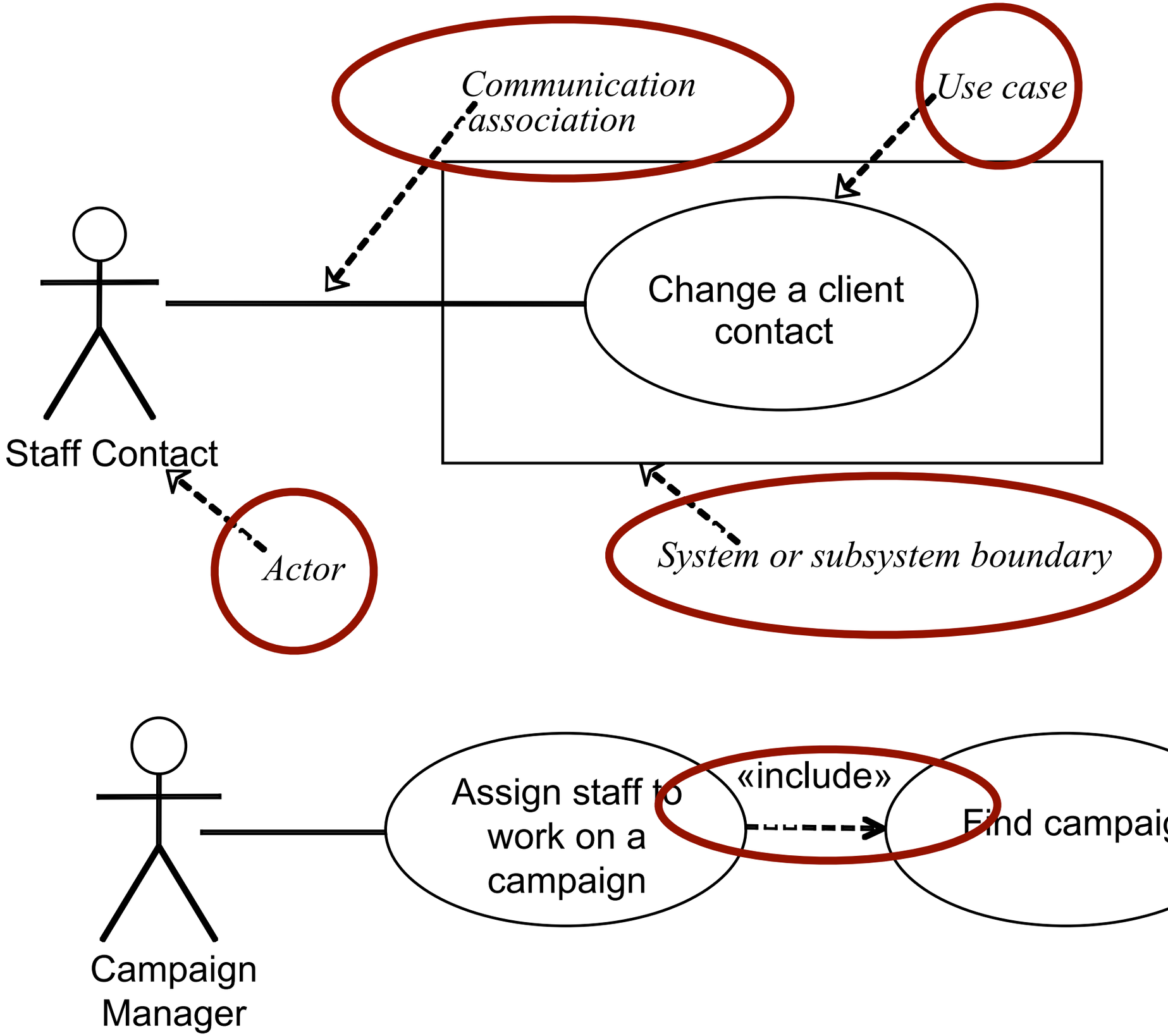
# Ways of using the UML(2)

- As a blueprint -> completeness, definition
  - Complete detailed design with all design decisions laid out so that a programmer can code from it straightforwardly
  - Can also be used in 2 directions
  - Requires more sophisticated tools
    - Forward engineering tools support diagram drawing and back it up with a repository to hold the information
    - Reverse engineering tools read source code, interpret from it into the repository and generate diagrams
    - Round-trip tools can do both forward and reverse engineering

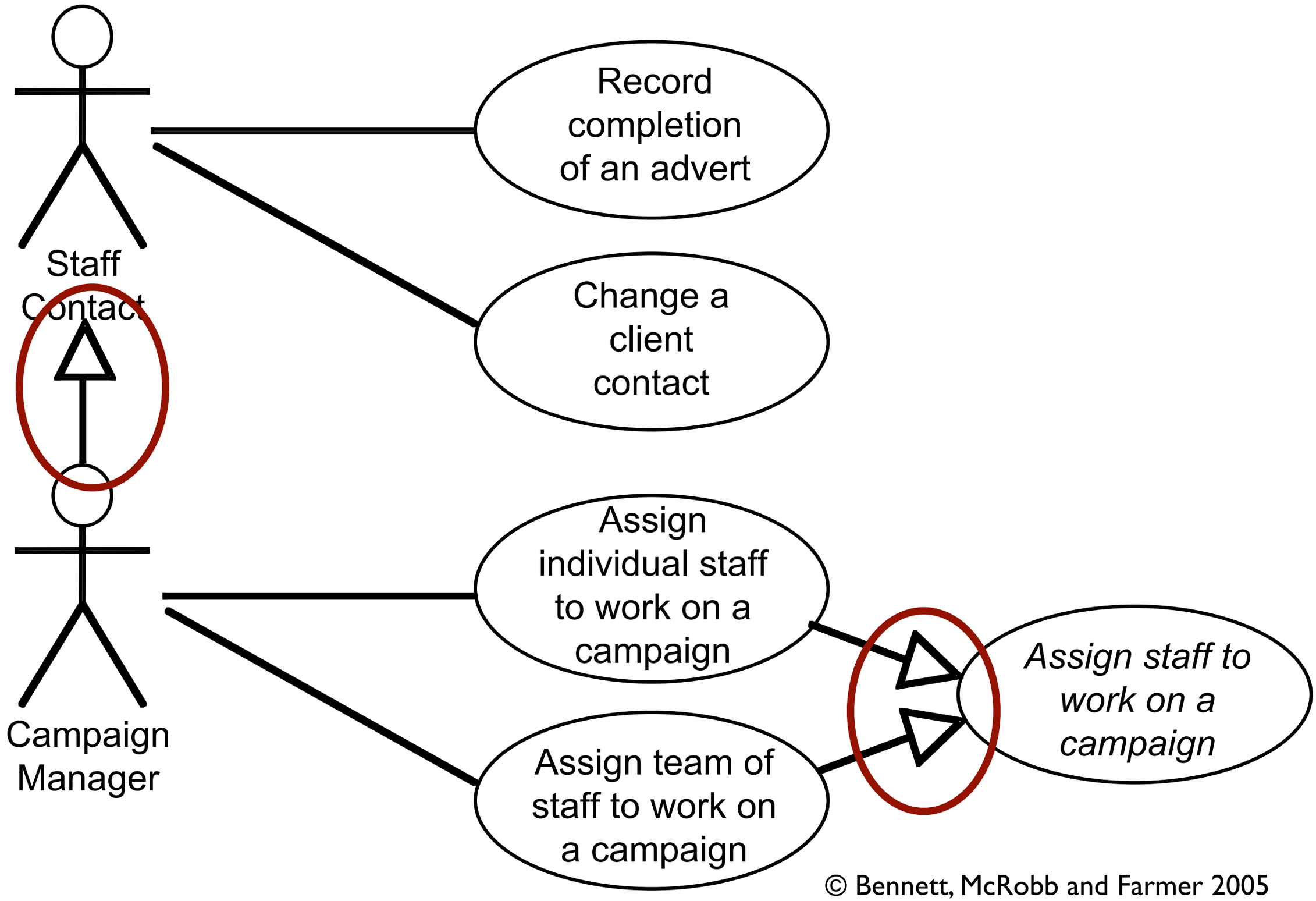
# Use Cases for Requirements

# Use Cases

- Use cases are a technique for capturing the functional requirements of a system.
- Use cases work by describing the typical interactions between the users of the system and the system itself.
- A use case is
  - *a set of scenarios tied together by a common user goal.*
- No standard way to write the content of a use case, different formats work in different cases.




© Bennett, McRobb and Farmer 2005




© Bennett, McRobb and Farmer 2005

# Use Case Content Description

- Example of definition of a use case:
  - *Name*: name used to refer to the use case
  - *Summary*: a short description
  - *Actors*: all actors involved
  - *Preconditions*: condition of the system at the start of the use case
  - *Description*: the complete description
  - *Exceptions*: special cases
  - *Result*: condition of the system at the end of the use case

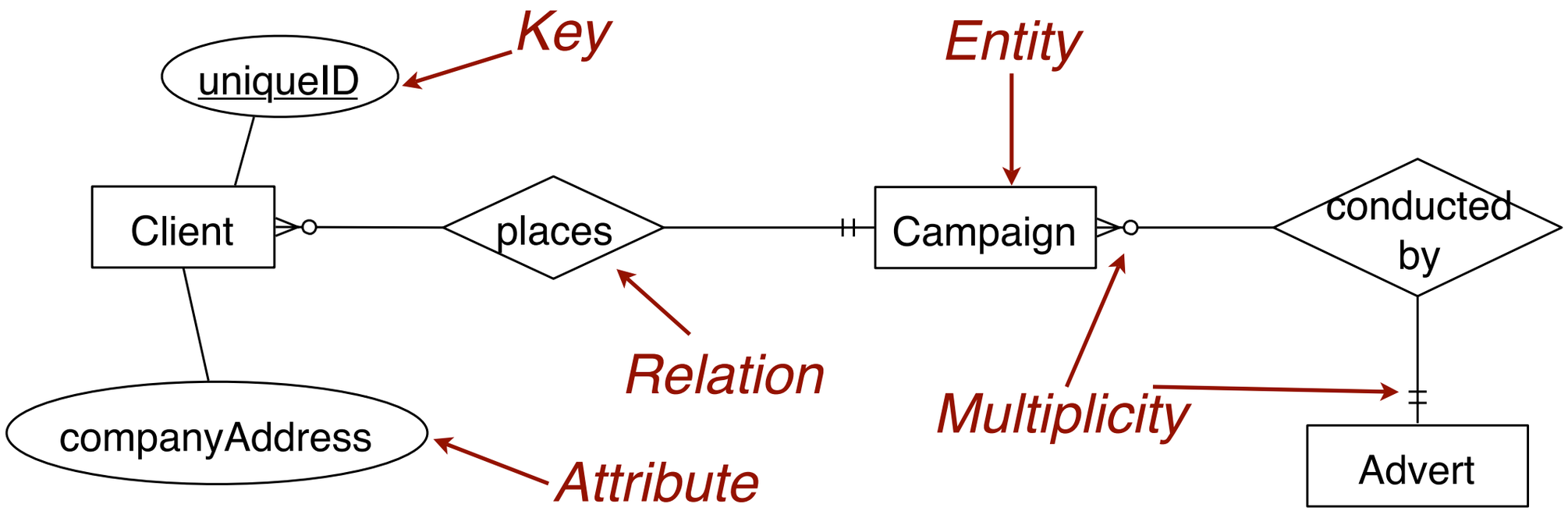
# Structural Models in RE

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

# Roadmap Structural Models in RE

- Data models
  - Entity-relationship diagram (ERD)
  - Data dictionary (DD)
- Object models
  - class diagrams

# Entity-relationship diagram




- An entity-relation-attribute  model sets out the entities in the system, the relationships between these entities and the entity attributes
- Used to describe the logical structure of data processed by the system.
- Widely used in database design. Can readily (after normalization) be implemented using relational databases.
- ERD is not part of UML!! !!!See Database Management!!!

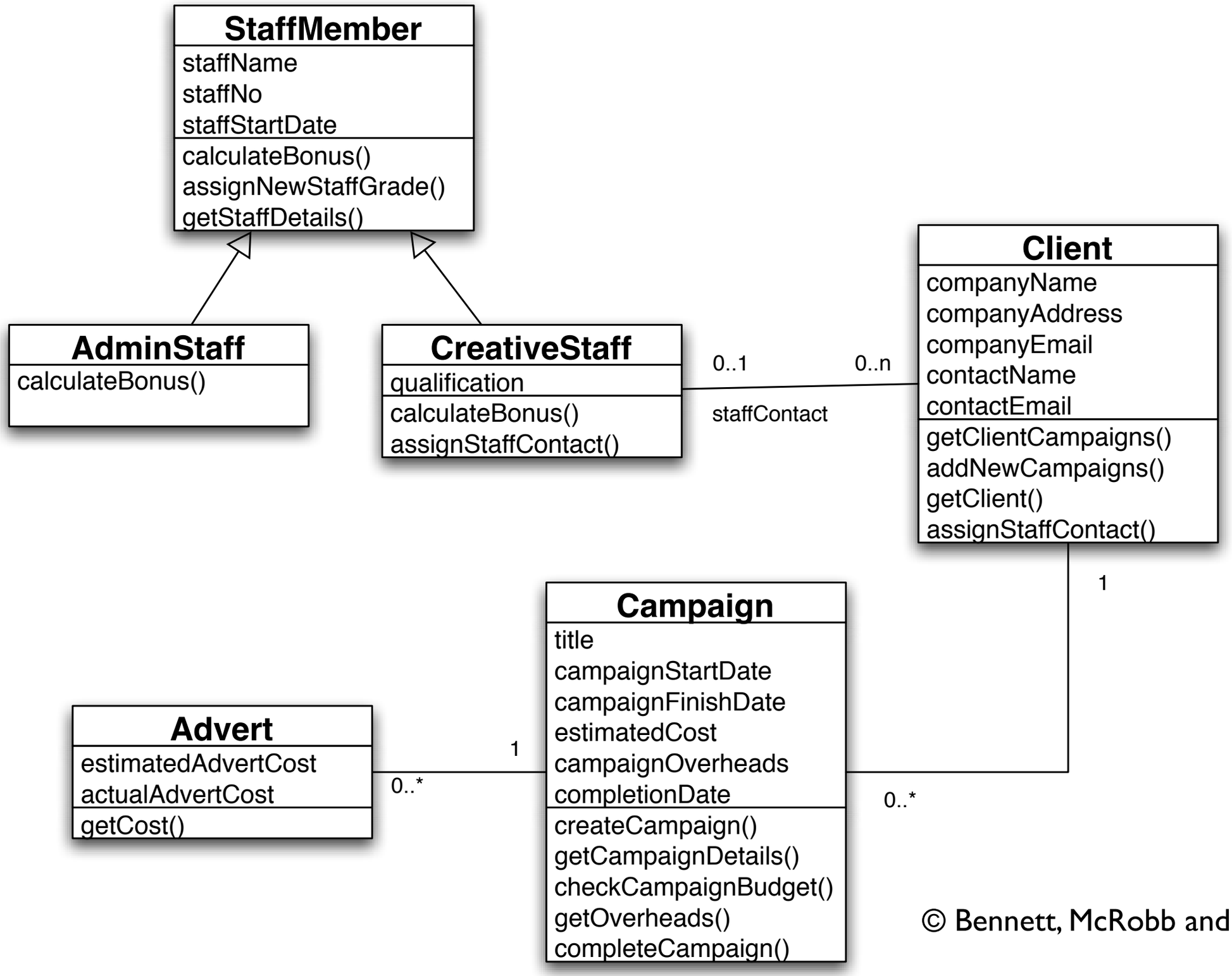
# Data dictionary

- Data dictionaries are lists of all of the names used in the system models. Descriptions of the entities, relationships and attributes are also included.
- Advantages
  - Support name management and avoid duplication;
  - Store of organisational knowledge linking analysis, design and implementation.

# Data dictionaries entries

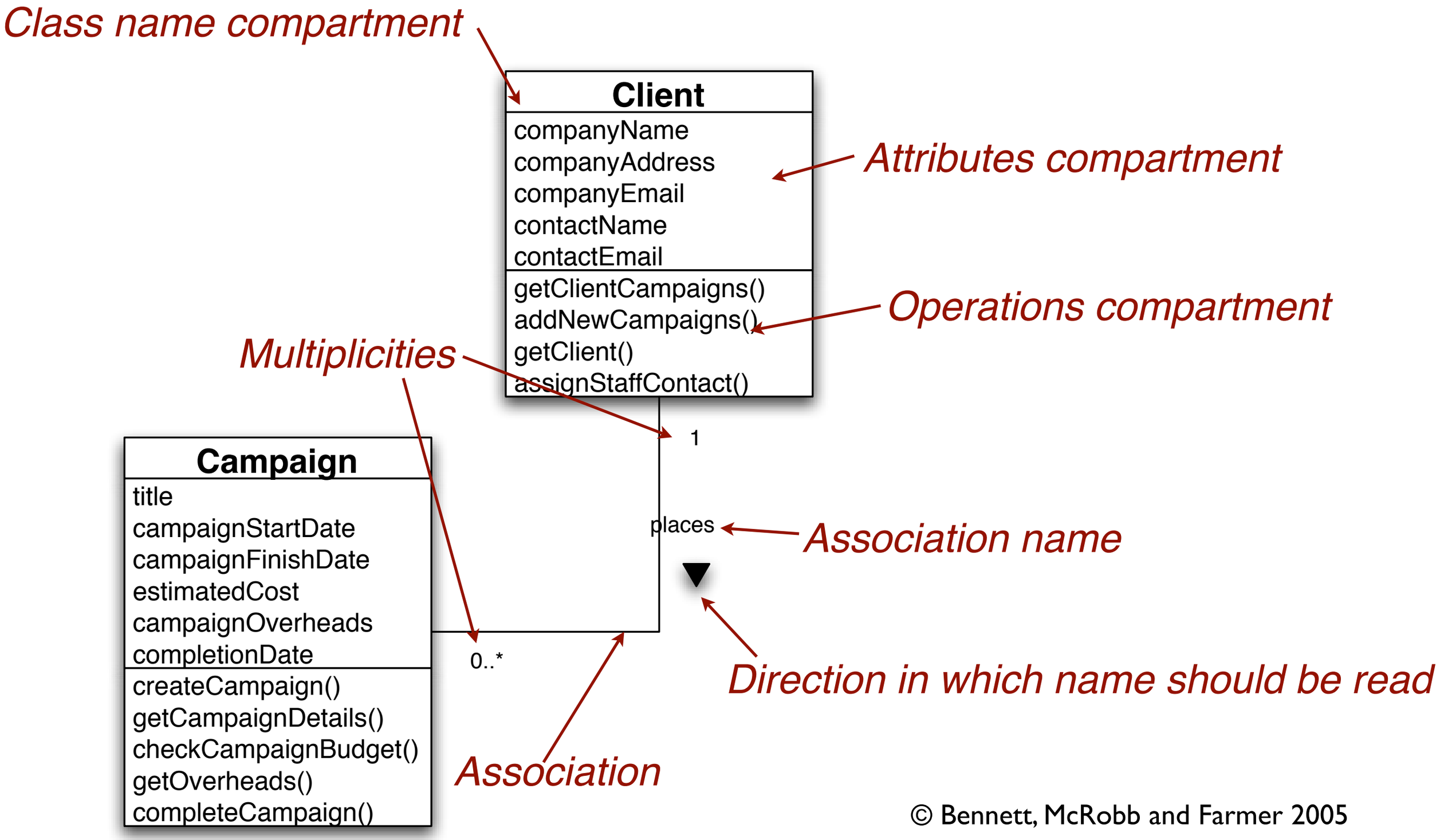
| Name | Description | Type |
|---|---|---|
| Client | Agate deals with other companies that it calls clients. | Entity |
| companyName | name of the client company | Attribute |
| places | A 1:n relationship between Client and Campaign placed by the Client. | Relation |
| Campaign | advertising campaign developed by Agata | Entity |

# Class Diagrams




© Bennett, McRobb and Farmer 2005

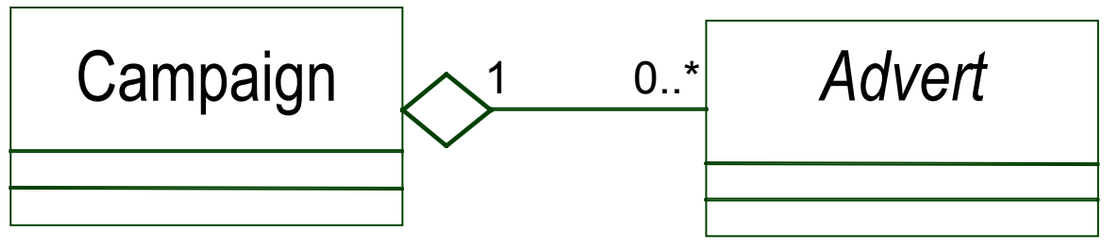
# Classes and Associations




© Bennett, McRobb and Farmer 2005

# Aggregation and Composition

- Special types of association, both sometimes called whole-part
- Aggregation is essentially any whole-part relationship
  - Semantics can be very imprecise
- Composition is ‘stronger’:
  - Each part may belong to only one whole at a time
  - When the whole is destroyed, so are all its parts

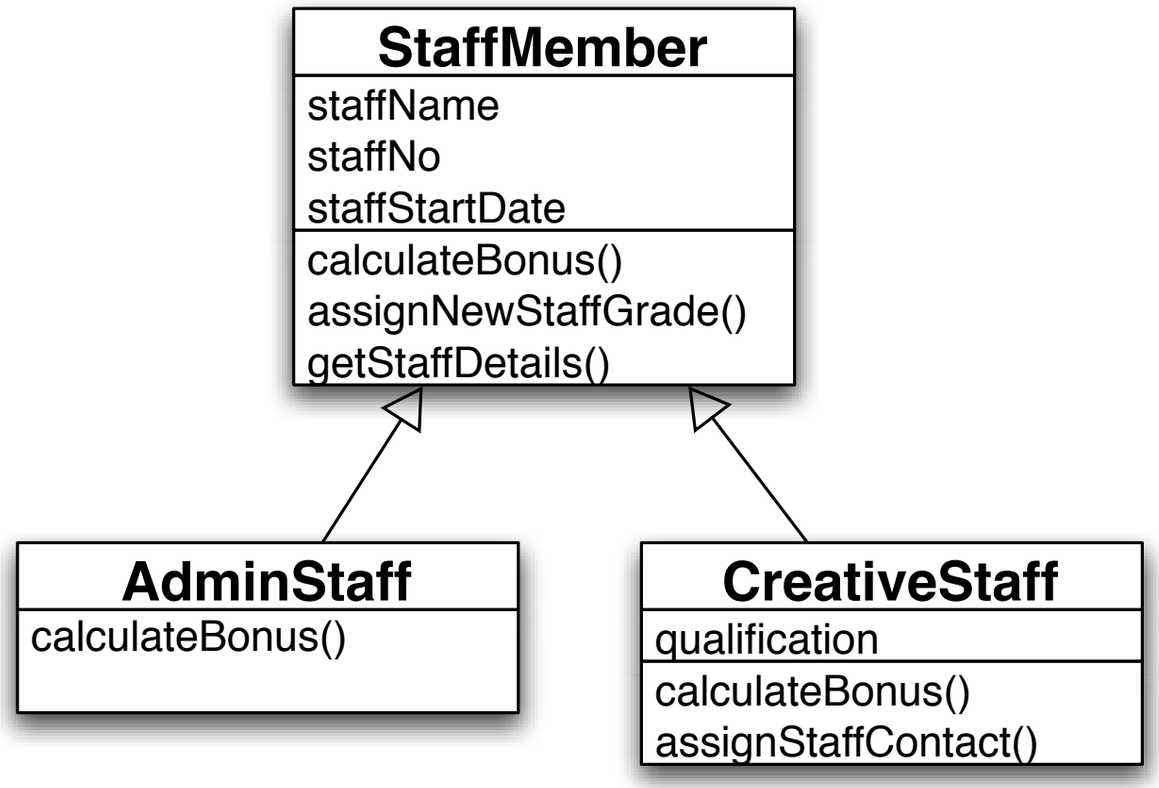




© Bennett, McRobb and Farmer 2005

# Inheritance

- Add generalisation structures when:
  - Two classes are similar in most details, but differ in some respects
  - May differ:
    - In behaviour (operations or methods)
    - In data (attributes)
    - In associations with other classes




© Bennett, McRobb and Farmer 2005

# Behavioural Models in RE

Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009

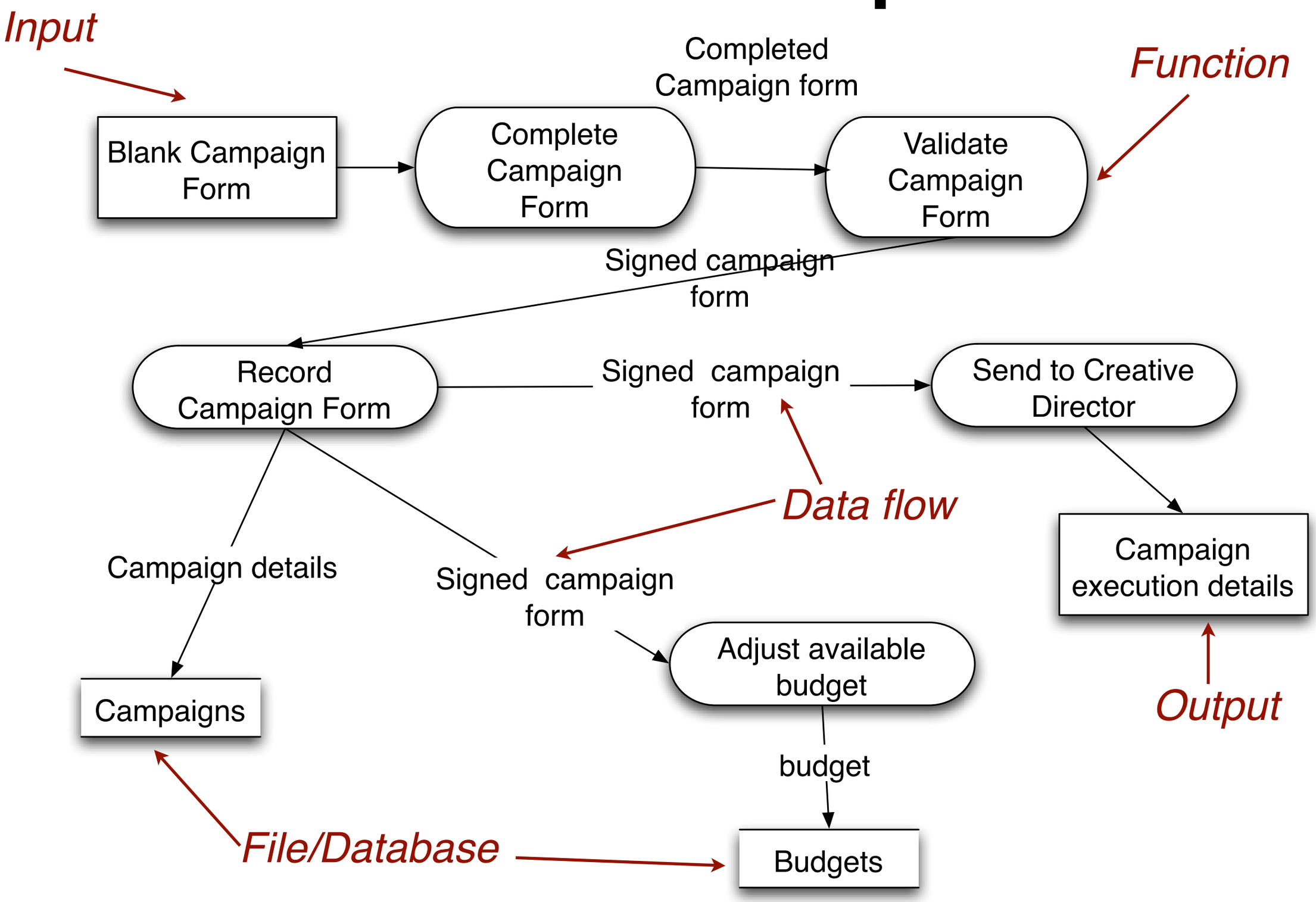
# Roadmap Behavioural Models in RE

- Data models
  - Data flow diagram (DFD)
- Object models
  - State machine
  - Sequence diagram
  - Activity diagram

# Data Flow Diagram

- Data flow diagrams (DFDs) may be used to model the system's data processing.
- These show the processing steps as data flows through a system.
- DFDs are an intrinsic part of many analysis methods.
- Simple and intuitive notation that customers can understand.
- Show end-to-end processing of data.

# DFD Example


Input
Completed
Campaign form
Function
Blank Campaign
Form
Complete
Campaign
Form
Validate
Campaign
Form
Signed campaign
form
Record
Campaign Form
Signed campaign
form
Send to Creative
Director
Data flow
Campaign details
Signed campaign
form
Campaign
execution details
Adjust available
budget
Campaigns
budget
Output
File/Database
Budgets

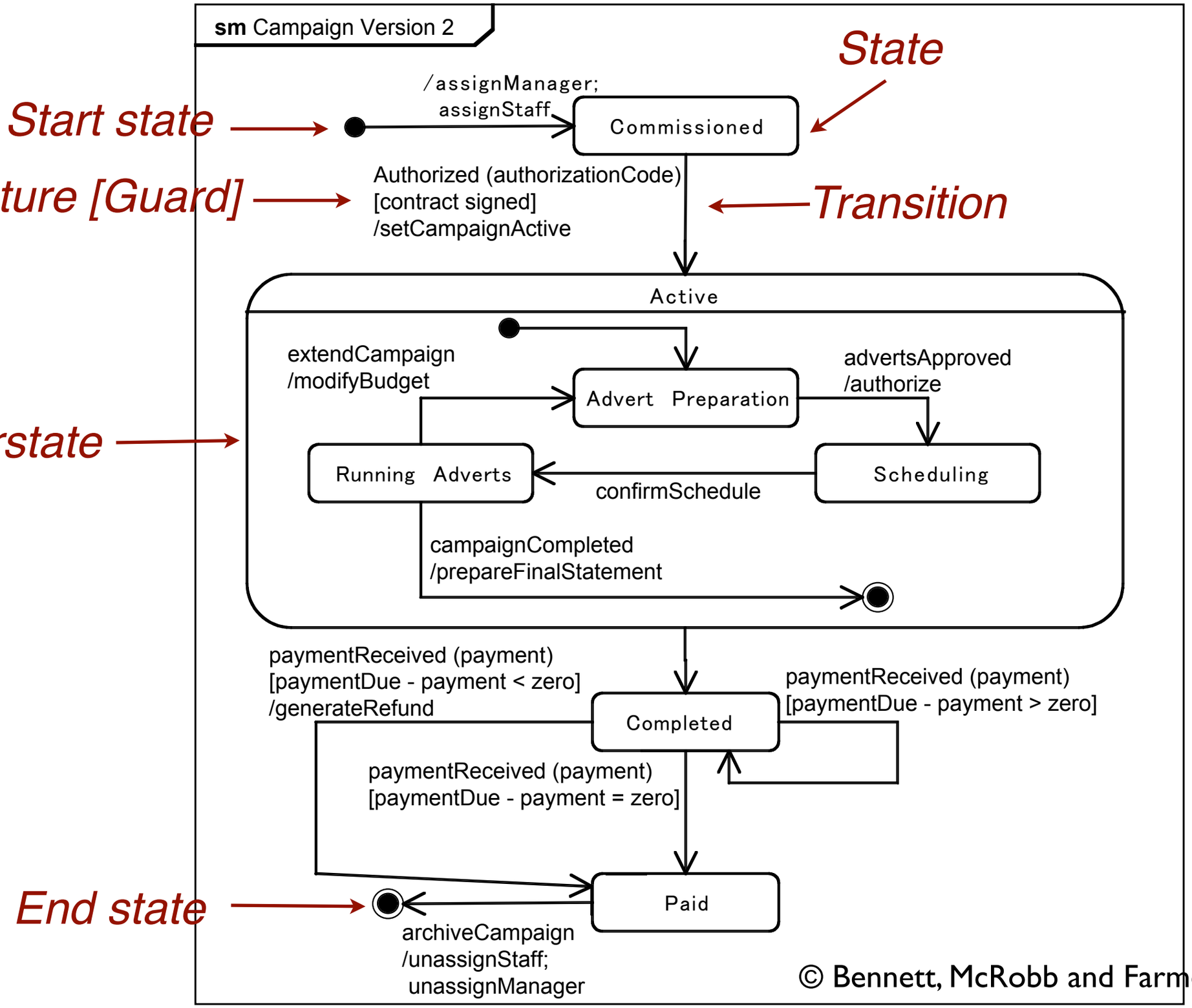
# Data Flow Diagrams

- DFDs model the system from a functional perspective.
- Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system.
- Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment.

# State Machine Models

- These model the behaviour of the system in response to external and internal events.
- They show the system's responses to stimuli.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.
- State machine diagrams are an integral part of the UML.

# UML State machine

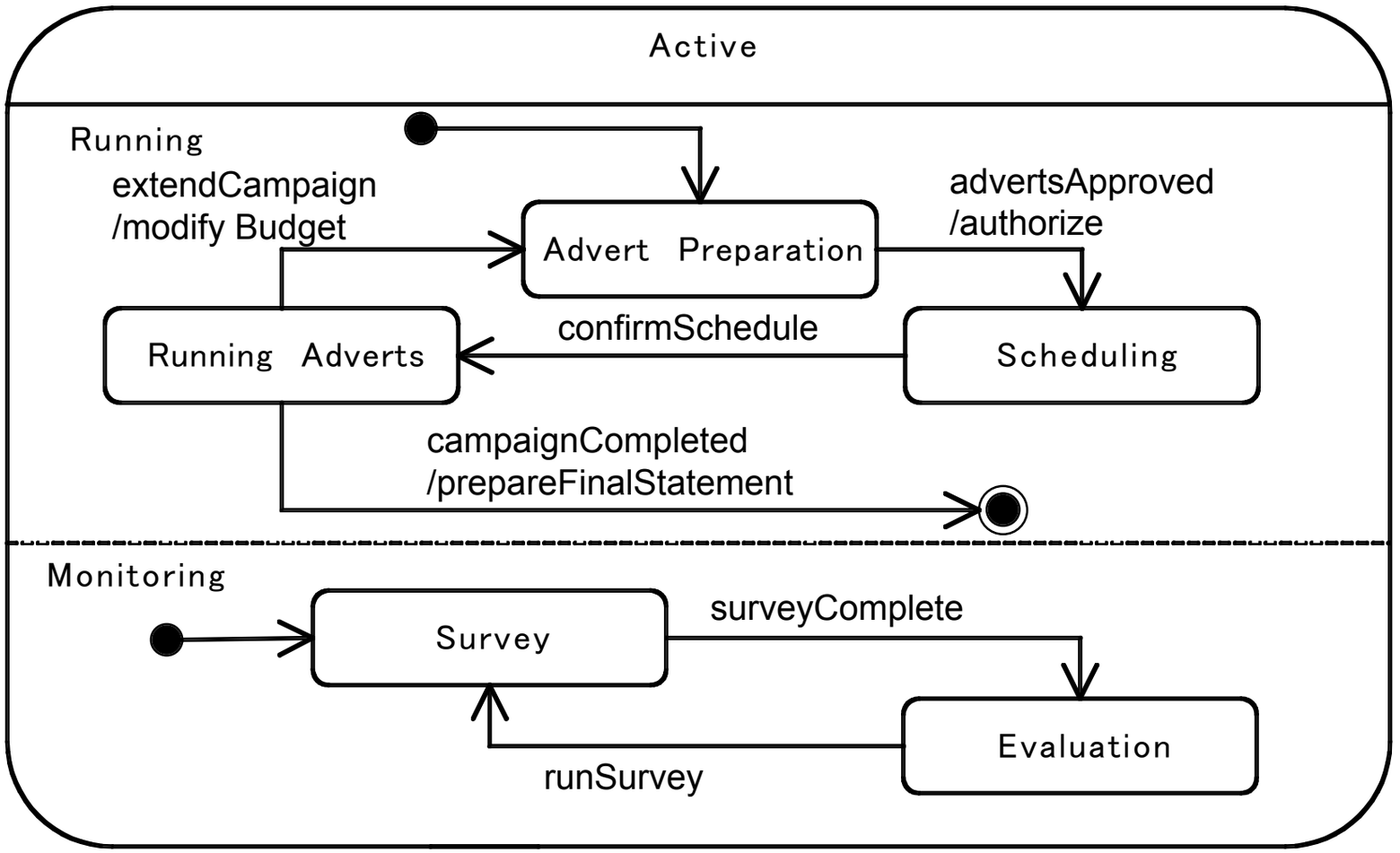


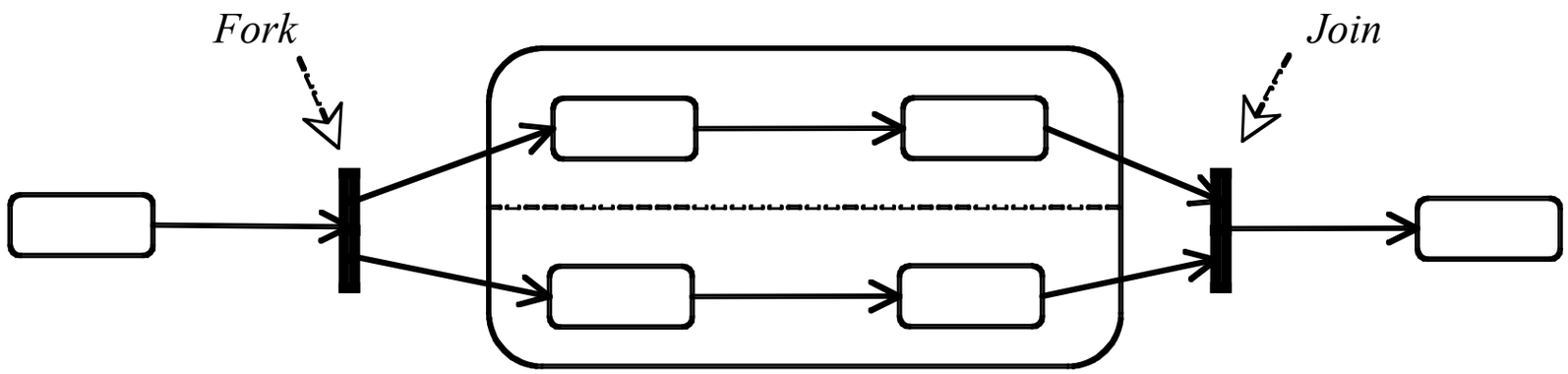

© Bennett, McRobb and Farmer 2005

# Transitions in UML SM

- *Trigger-signature [Guard] / Activity*
  - *Trigger-signature*: an event whose reception by the object in the source state makes the transition legible to fire, providing the guard condition is satisfied.
  - *Guard*: a boolean expression, evaluated when the transition is triggered by the reception of the event trigger; if the expression evaluates True, the transition is legible to fire, if the expression evaluates to False, the transition does not fire and if there is no other transition that could be triggered by the same event, the event is lost.
  - *Activity*: some behaviour executed during the transition

# Concurrent States



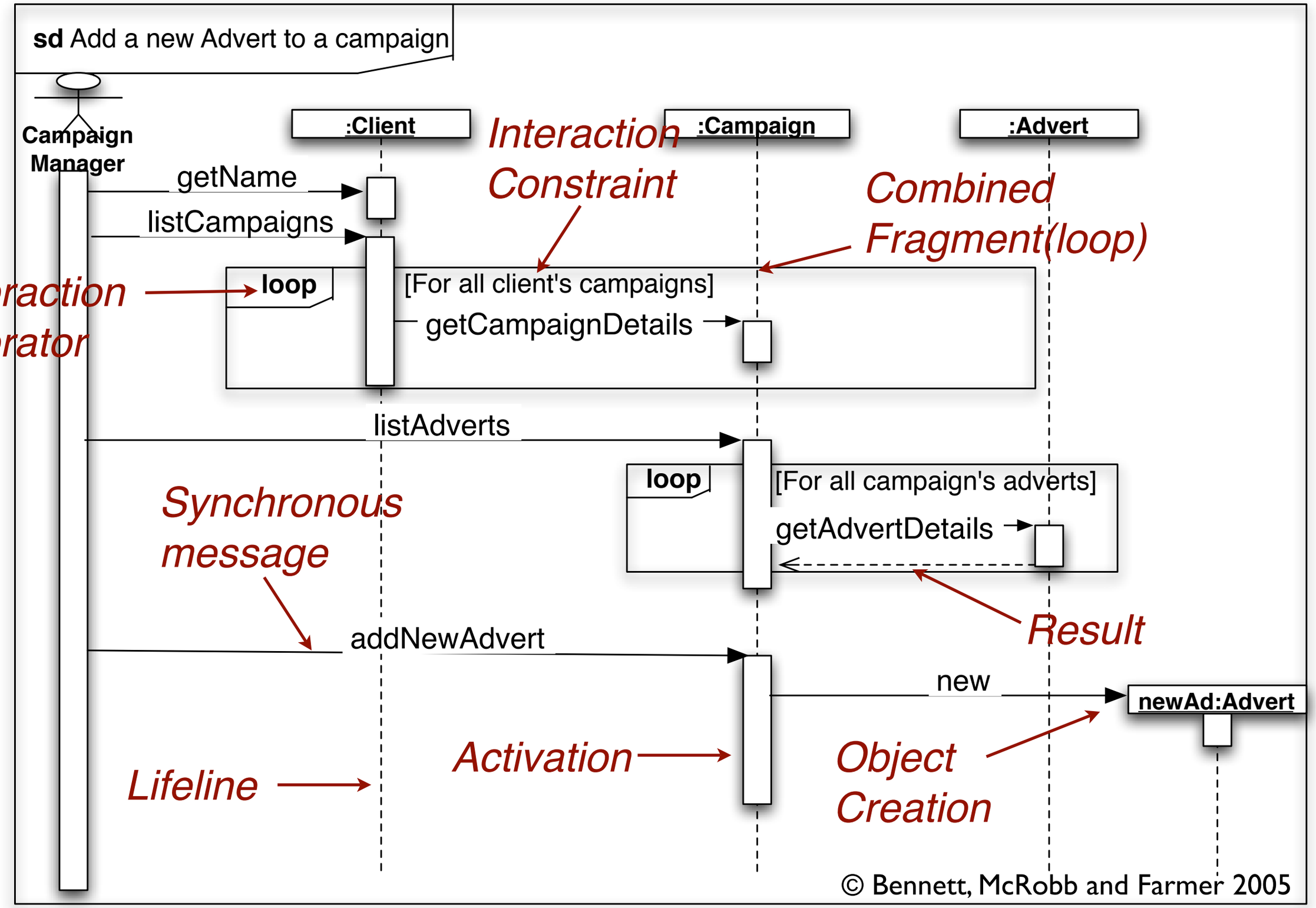




© Bennett, McRobb and Farmer 2005

# Sequence Diagram

- Shows an interaction between lifelines (e.g. objects) arranged in a time sequence.
- Can be drawn at different levels of detail and to meet different purposes at several stages in the development life cycle.
- Typically used to represent the detailed object interaction that occurs for one use case or for one operation.

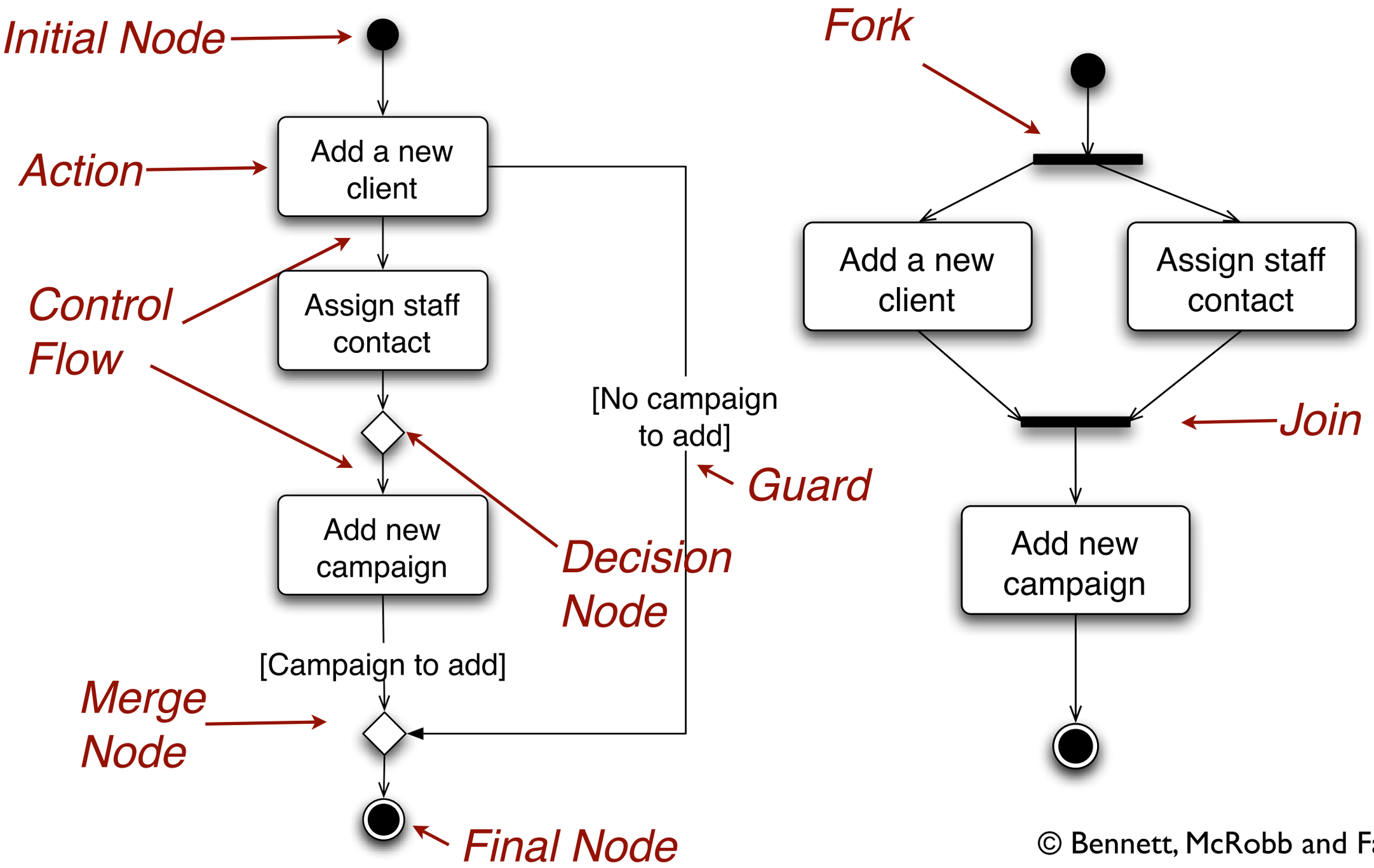
# Sequence Diagrams



# UML Activity Diagrams

- are used for the following purposes
  - to model business activities
  - to model a process
  - to model a function represented by a use case (e.g., during requirements elaboration)

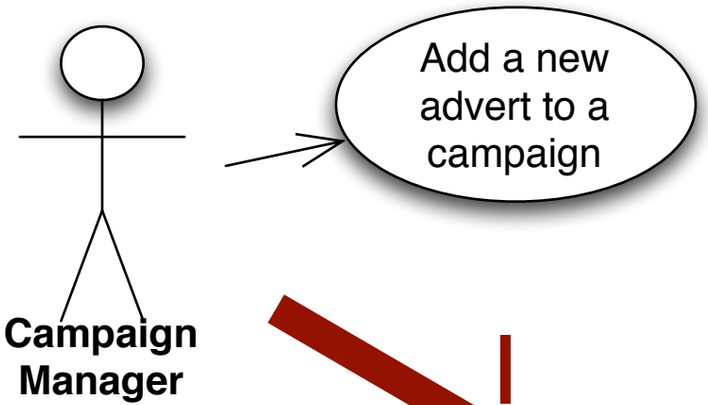
# Activity Diagram Examples

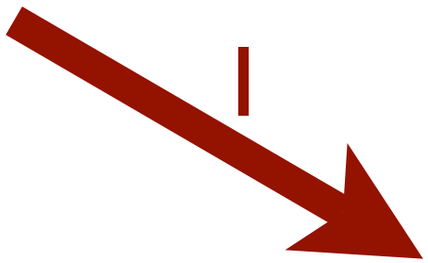


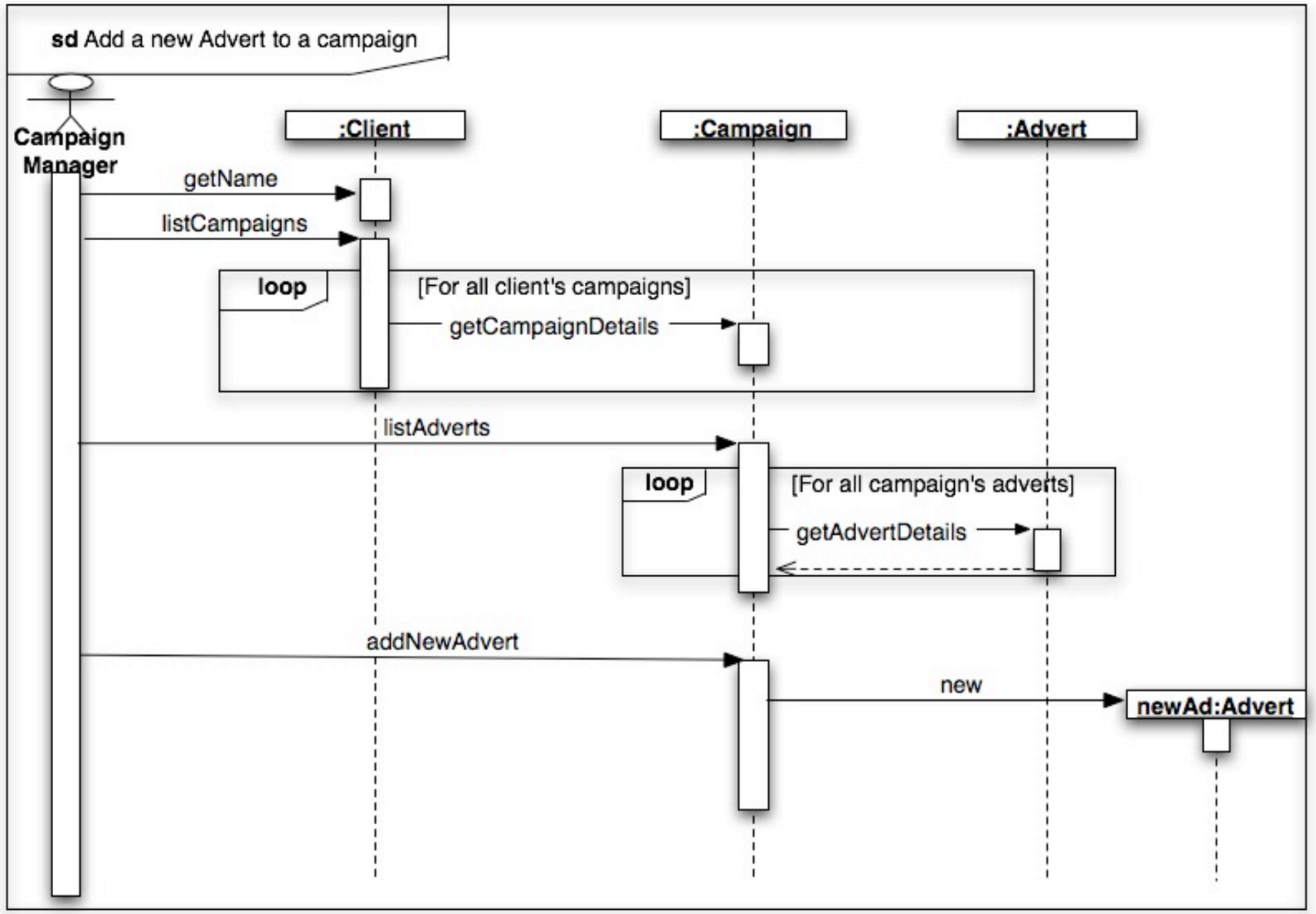

© Bennett, McRobb and Farmer 2005

# From Requirements to Models

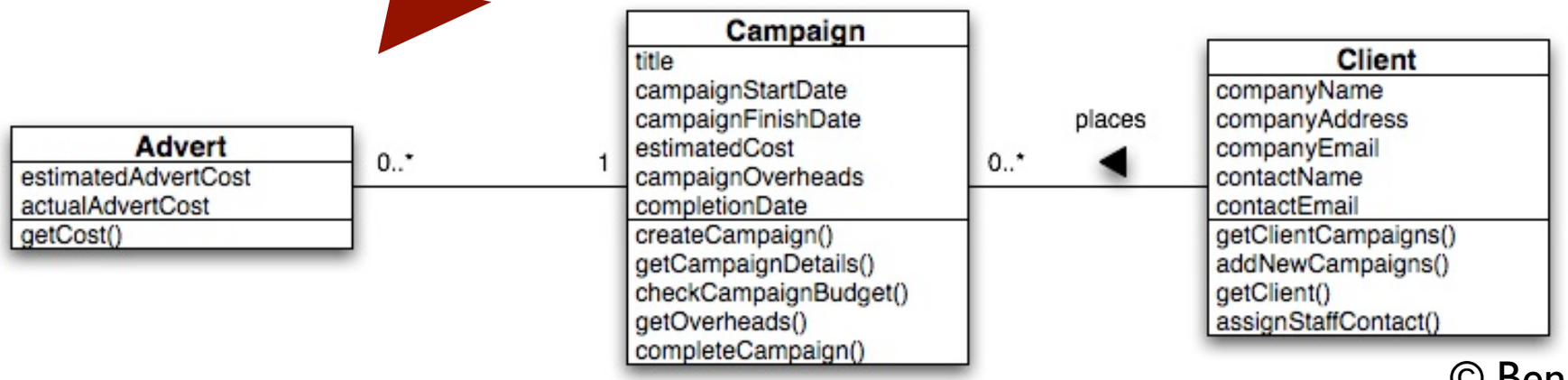
Ragnhild Van Der Straeten - ULB - Génie logiciel et gestion de projets - 2008/2009








© Bennett, McRobb and Farmer 2005

# CRC cards

- Class - Responsibility - Collaboration
- focus on behaviour, the idea is that you should be able to take any class and summarize it with a handful of responsibilities.
- *Responsibility*: a short sentence that summarizes something that an object should do:
  - an action the object performs,
  - some knowledge the object maintains, or
  - some important decisions the object makes.
- *Collaboration*: the other classes that this class needs to work with. This gives you some idea of the links between classes—still at a high level.

© Bennett, McRobb and Farmer 2005

| Class Name | Client |
|---|---|
| Responsibilities | Collaborations |
| Provide client information.<br><br>Provide list of campaigns. | <br><br>Campaign |

| Class Name | Campaign |
|---|---|
| Responsibilities | Collaborations |
| Provide campaign information.<br><br>Provide list of adverts.<br>Add a new advert. | <br><br>Advert<br>Advert |

| Class Name | Advert |
|---|---|
| Responsibilities | Collaborations |
| Provide advert details.<br><br>Construct adverts. | |

# References

- The previous slides highly depend on:
  - [Sommerville] Ian Sommerville. Software Engineering. Eighth Edition. 2007. ISBN: 9780321313799
  - [Benett et al.] Simon Benett, Steve McRobb and Ray Farmer. Object-Oriented Systems Analysis and Design. (using UML) Third Edition.

# References


- UML: http://www.uml.org/
- Pierre-Alain Muller, Nathalie Gaertner. Modélisation objet avec UML. Groupe Eyerolles. ISBN 2212113978