Topic 2 Notes:

A model is a representation of a problem domain or a proposed solution that allows us to talk, or reason, about the real thing. This allows us to increase our understanding and avoid potential pitfalls.

Types of modeling

- **Functional modeling**

Models the functional requirements of the system, that is it presents the scenario of the all use cases

- **Entity class modeling**

Determines the entity classes and their attributes, relationships and interactions between entity classes and represents the information inform of a class diagram

- **Dynamic modeling**

Determines the operations performed by or to each entity class or subclass and presents this information inform of a state chart.

Think of an architect's model of a new concert hall: it allows the architects to say 'This is what the finished concert hall will look like' and it helps them to come up with new ideas, such as 'I think we're going to need a steeper roof'. A model allows us to learn a lot without actually building anything. Much of software development involves creating and refining models, rather than cutting lines of code. Analysis is about discovering what the system is going to handle, rather than deciding how to do the handling. We need to decompose a complex set of requirements into the essential elements and relationships on which we will base our solution. Analysis is our first opportunity to get to grips with modeling the real world as objects.

An analysis model has both **static and dynamic parts**. We can depict the static analysis model using a class diagram. A class diagram shows the objects that the system will handle and how those objects are related to each other. For the dynamic analysis model, we can use communication diagrams to demonstrate that our static model is feasible. As before, rather than all the intricacies of UML notation, you'll see only the essential parts here: the parts that will suffice for most purposes.

There are two inputs to analysis:

• The **business requirements** model which contains descriptions of the manual and automated workflows of our business context, described using business-oriented versions of actors, use cases, objects, the glossary and, optionally, communication diagrams and activity diagrams.

• The **system requirements model** which contains an external view of the system, described as system-oriented versions of actors, use cases and use case diagrams, user interface sketches, an enhanced glossary and nonfunctional requirements.

UML (UNIFIED MODELING LANGUAGE)

The **Unified Modeling Language (UML**) first appeared in the **1990's** as an effort to select the best elements from the many modeling systems proposed at the time, and to combine them into a **single coherent notation**. It has since become the industry standard for **software modeling** and **design**, as well as the modeling of other processes in the scientific and business worlds.

The UML is a **tool** for specifying software systems. **Standardized diagram** types to help you describe and visually map a software system's design and structure. Using UML it is possible to model just about any kind of application, both specifically and independently of a target platform. While UML is naturally oriented towards **Object-Oriented programming,** but it is just as easy to model procedural languages such as **C, Visual Basic, Fortran** etc. The use of UML as a tool for defining the **structure of a system** is a very useful way to manage large, complex systems. Having a clearly visible structure makes it easy to introduce new people to an existing project

UML has **13 types of diagram**. The UML specification doesn't say where these diagrams should be used in any particular methodology – we're free to use whichever we think is appropriate at any stage. It gives you access to all 13 diagram types, and a **comprehensive documentation generator**, to help you communicate and share your models more effectively. The Unified Modeling Language (UML) is probably the most widely known and used notation for object-oriented analysis and design. It is the result of the merger of several early contributions to object-oriented methods.

1) **Use case diagrams** categorize the ways in which a system is used.
2) **Class diagrams** show classes and how they can be fitted together (they can also show objects).
3) **Object diagrams** show only objects and how they can be fitted together.
4) **Activity diagrams** show activity by humans or objects in a similar way to a flow chart.
5) **State machine** diagrams show the various states of any object with an interesting or complicated life cycle.
6) **Communication diagrams** show the messages sent between objects in some scenario.
7) **Sequence diagrams** show similar information to communication diagrams, but emphasizing sequences rather than connections.
8) **Package diagrams** show how related classes are grouped together, for the benefit of developers.
9) **Deployment diagrams** show machines, processes and deployed artifacts for a finished system.
10) **Component diagrams** show reusable components (objects or subsystems) and their interfaces.

11) **Interaction overview** diagrams show individual steps of an activity using sequence diagrams.
12) **Timing diagrams** show precise timing constraints for messages and object states.
13) **Composite structure** diagrams show how objects fit together in an aggregation or composition, showing interfaces and collaborating objects

Use Case Modeling

Functional Requirements and Use Cases

Uses cases specify the **functionality** that the system will offer from the **user's** perspective. Functional requirements capture the **intended behavior** of the system. This behavior may be expressed as **services, tasks or functions** the system is required to perform. Developers use uses case modeling to document the scope of the system. Use case modeling allows future users of a software system as much input as possible into its design. **Textual descriptions** (use case descriptions) provide a description of the interaction between the user of the system, the **actors** and the high-level functions within the system and the **use cases**

Use case identity the **users** of the system and the **tasks** they undertake with the system. We mean **'users'** and **'tasks'**. In UML technical terms **actors** and **use cases** are used. It uses the vocabulary of the users, not programmers. This focus on users means that the initial specification of the program can be understood both by its users and by the software engineers designing it. They are used to document the **scope of the system** from the **user's point of view**. They identify **users** of the system and the **tasks** they must undertake with the system. We mean the "tasks" and the "users". Use cases are used to document our understanding of the way a business operates – business requirements modeling – and to specify what our new software system should be able to do – system requirements modeling.

Basic Notations and concepts

There are two main entities in the use case approach: a**ctors** and **use cases**. Jacobson invented use cases to define the way in which part of a business or a system is used. Use diagrams show the aspects of the system, actors, use cases and the system or system in the system

Actors

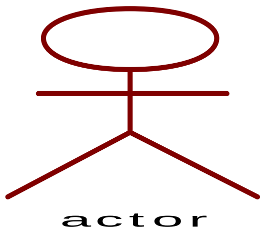
An actor is (usually) a **person/entity** who will use the system we are designing. A bank customer interacting with the software of an ATM machine is an actor. Actors represent the roles that people, other systems or devices take on when communicating with the particular cases in the system. Actually, "role" is a probably a better name than "actor." One human playing different roles may be represented by several actors. For example, in a small business, Harry Jones might be represented by an actor called "salesperson" when making a sale, but by an actor called "bookkeeper" when adding up the day's sales. Conversely, a single actor may represent several different individuals. Harry, Jose, and Elma may all be represented by the actor called "salesperson."

Actors are entities that interact with the system and initiate sequences in a use case, and can be person, another system, piece of software or by the **passage of time**. An astronomer inputting the coordinates of a star to a telescope aiming program is an actor. A **bookstore** clerk checking the computer to see if a particular book is available is an actor. Usually an actor initiates some operation, although sometimes the actor may act in other ways, such as receiving information or assisting in an operation.

Other systems connected to the one we're designing, such as a different computer system or a link to the Web, may also be actors. For example, the computer system in a particular bookstore may be linked to a remote system in the head office. This remote system can be considered an actor in the bookstore's system.

In a large project, just identifying all the actors may be difficult. The designer needs to look for people or other systems that:

• Provide information to the system
• Need information from the system
• Assist other actors

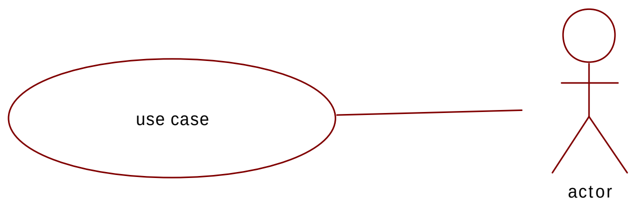


Use Cases

A use case is a specific task, usually initiated by an actor. It describes a single goal the actor wants to attain. Examples are the withdrawal of cash by the bank customer, the aiming of the telescope by the astronomer, and the investigation of a book's availability by the bookstore clerk. Actors are parties outside the system that interact with the system. In most situations the use case is initiated by the actor, but sometimes it's initiated by the system, as when the electric company's accounting program sends you a reminder that you haven't paid your bill, or your car's computer turns on a warning light when it decides the engine is too hot.

In most situations the use case is initiated by the actor, but sometimes it's initiated by the system, as when the electric company's accounting program sends you a reminder that you haven't paid your bill, or your car's computer turns on a warning light when it decides the engine is too hot.

In general, everything you want the system to do should be specified by a use case. In general, everything you want the system to do should be specified by a use case. An Actor uses a Use Case to perform some piece of work which is of value to the business. The set of Use Cases an actor has access to define their overall role in the system and the scope of their action.



Scenarios

A use case usually consists of a number of scenarios. The use case specifies a goal, while a scenario represents a particular outcome when attempting to reach that goal. A use case defines a goal-oriented set of interactions between external actors and the system under consideration.

A scenario is an instance of a use case, and represents a single path through the use case. Thus, one may construct a scenario for the main flow through the use case, and other scenarios for each possible variation of flow through the use case (e.g., triggered by options, error conditions, security breaches, etc.). Scenarios may be depicted using sequence diagrams.

Use Case Diagrams

The UML specifies how to diagram use cases. Actors are represented by stick figures; use cases by ovals. A rectangular frame surrounds the use cases, leaving the actors outside. This rectangle is the system boundary. The system inside is what the software developer is trying to design. Figure 16.3 shows a use case diagram for a bookstore computer system.

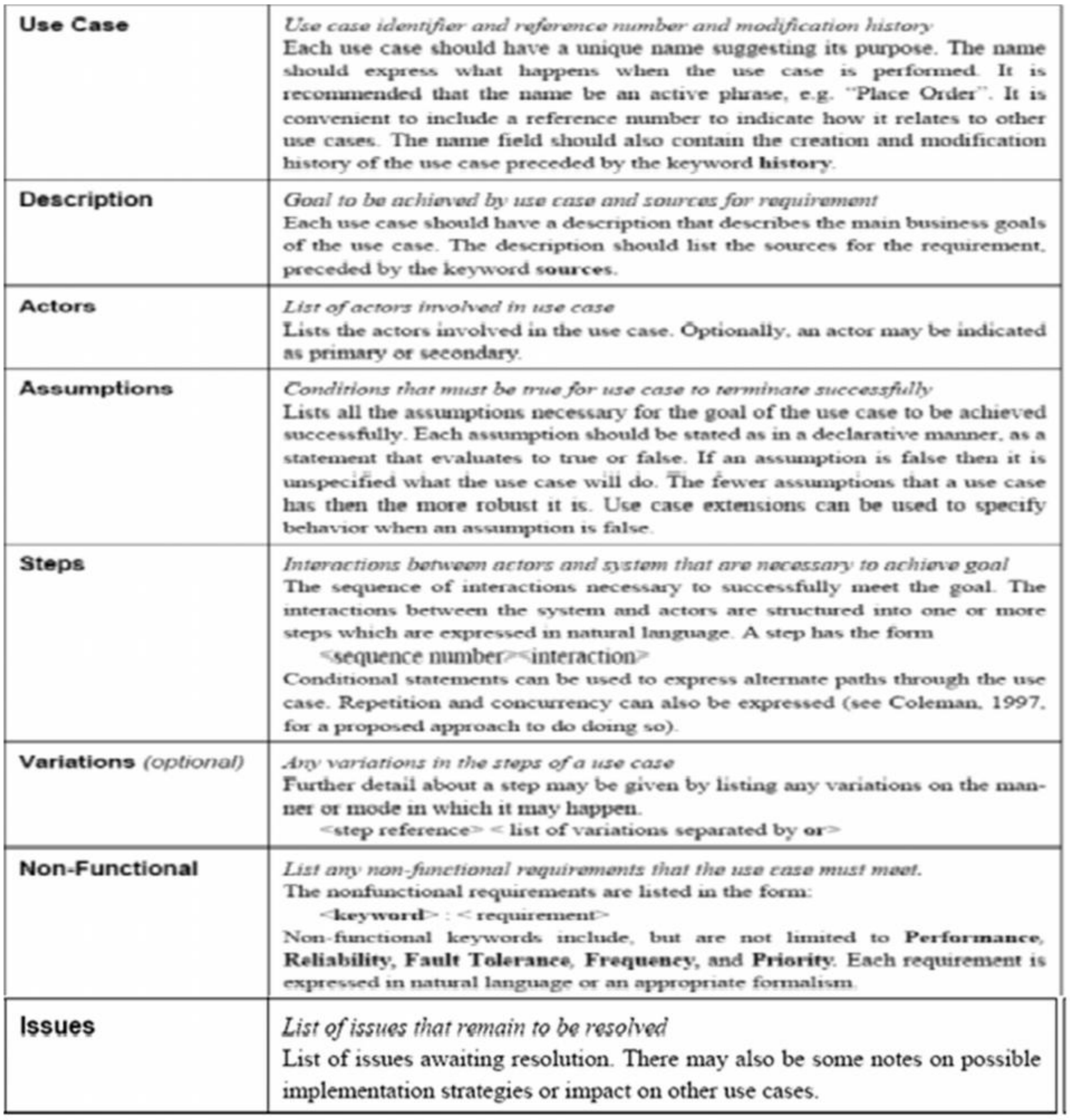
Template

Each use case is a list of scenarios and each scenario is a sequence of steps

One way of documenting use cases is to use a template which might include.

- Name of the use case
- Actor who initiates the use case
- Preconditions for the use case (things that must be true for the use case to take place)
- Post conditions (things must be true after the use case has taken place)
- Steps in the scenario
- Purpose (what the use case is intended to achieve)

Although use cases are part of UML, there is no template for writing use cases. The following is Derek Coleman's proposal for a standard use case template (Coleman, 1998), with some minor modifications.



Use Case	*Use case identifier and reference number and modification history* Each use case should have a unique name suggesting its purpose. The name should express what happens when the use case is performed. It is recommended that the name be an active phrase, e.g. "Place Order". It is convenient to include a reference number to indicate how it relates to other use cases. The name field should also contain the creation and modification history of the use case preceded by the keyword **history**.
Description	*Goal to be achieved by use case and sources for requirement* Each use case should have a description that describes the main business goals of the use case. The description should list the sources for the requirement, preceded by the keyword **sources**.
Actors	*List of actors involved in use case* Lists the actors involved in the use case. Optionally, an actor may be indicated as primary or secondary.
Assumptions	*Conditions that must be true for use case to terminate successfully* Lists all the assumptions necessary for the goal of the use case to be achieved successfully. Each assumption should be stated as in a declarative manner, as a statement that evaluates to true or false. If an assumption is false then it is unspecified what the use case will do. The fewer assumptions that a use case has then the more robust it is. Use case extensions can be used to specify behavior when an assumption is false.
Steps	*Interactions between actors and system that are necessary to achieve goal* The sequence of interactions necessary to successfully meet the goal. The interactions between the system and actors are structured into one or more steps which are expressed in natural language. A step has the form <sequence number><interaction> Conditional statements can be used to express alternate paths through the use case. Repetition and concurrency can also be expressed (see Coleman, 1997, for a proposed approach to do doing so).
Variations *(optional)*	*Any variations in the steps of a use case* Further detail about a step may be given by listing any variations on the manner or mode in which it may happen. <step reference> < list of variations separated by **or**>
Non-Functional	*List any non-functional requirements that the use case must meet.* The nonfunctional requirements are listed in the form: <**keyword**> : < requirement> Non-functional keywords include, but are not limited to **Performance**, **Reliability**, **Fault Tolerance**, **Frequency**, and **Priority**. Each requirement is expressed in natural language or an appropriate formalism.
Issues	*List of issues that remain to be resolved* List of issues awaiting resolution. There may also be some notes on possible implementation strategies or impact on other use cases.

Scenario 1

Books and journals

The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow upto 12 items at any one time. Only members of staff may borrow journals

1. **Identify actors**

1. Book borrower
2. Catalog browser
3. Journal borrower
4. Librarian

2. Identify roles that each actor plays

Book borrower

1. Borrow book copy of book
2. Return copy of book
3. Extend loan
4. Reserve book

Journal borrower

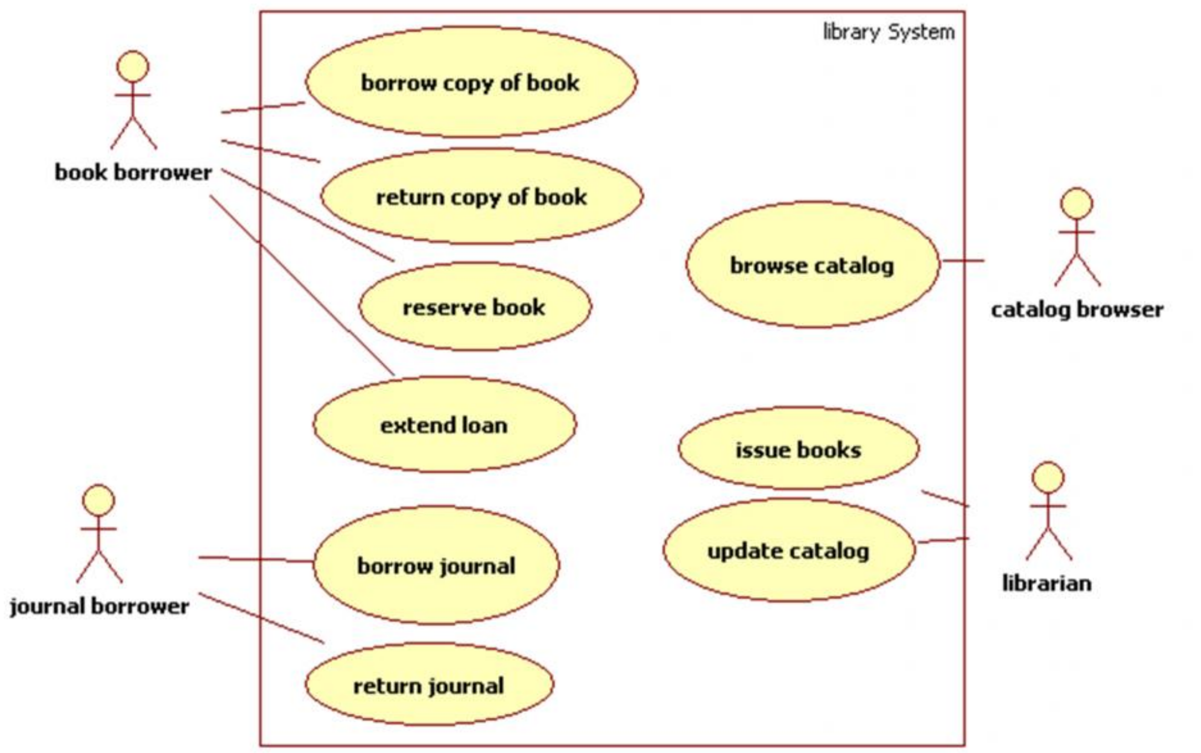
1. Borrow journal
2. Return journal

Catalog Browser

1. Browse catalog

Librarian

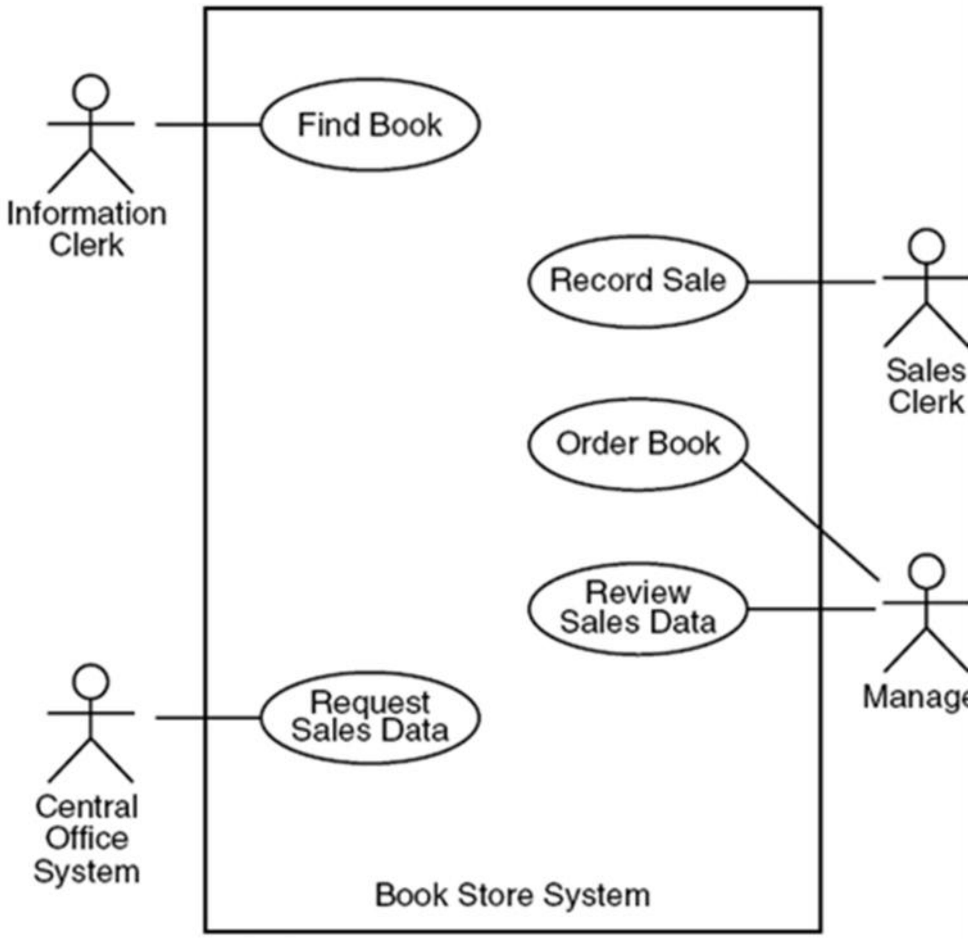
1. Issue books
2. Update catalog



Scenarion2

Let's consider a use case consisting of a bookstore clerk querying the store's computer system for the location of a particular book. There are several possible outcomes or scenarios:

- The book is in the store and the computer displays its shelf location.
- The book is out of stock, but the system gives the customer the opportunity to order it from the publisher.
- The book is not only out of stock, it's out of print; so the system informs the customer that she or he is out of luck. In a formal development process, each scenario would have its own documentation, describing in detail all the events in the scenario



Review Questions

Consider the following flight check-in subsystem requirements. In order to successfully check-in passengers, the attendant must weigh the passengers' luggage and assign them a seat. There are two ways to assign a seat: assigning a window seat and assigning an aisle seat, but only one need be completed in the process of assigning the passenger a seat. Create a simple Use Case diagram for the above problem. (10 marks)

Suppose we want to develop software for an alarm clock. The clock shows the time of day. Using buttons, the user can set the hours and minutes fields individually, and choose between 12 and 24-hour display. It is possible to set one or two alarms. When an alarm fires, it will sound some noise. The user can turn it off, or choose to 'snooze'. If the user does not respond at all, the alarm will turn off itself after 2 minutes. 'Snoozing' means to turn off the sound, but the alarm will fire again after some minutes of delay. (10 marks)

- **Relationship among use cases**

1) Include and extends relationships

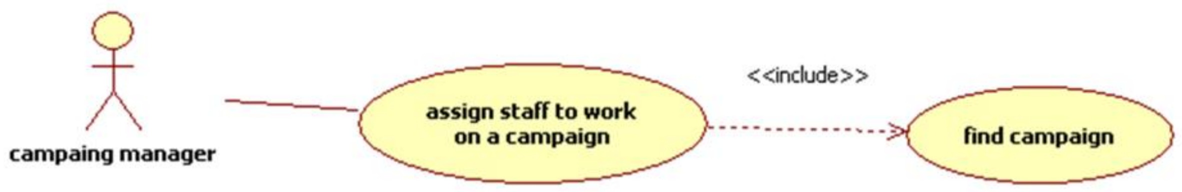
2) Generalization

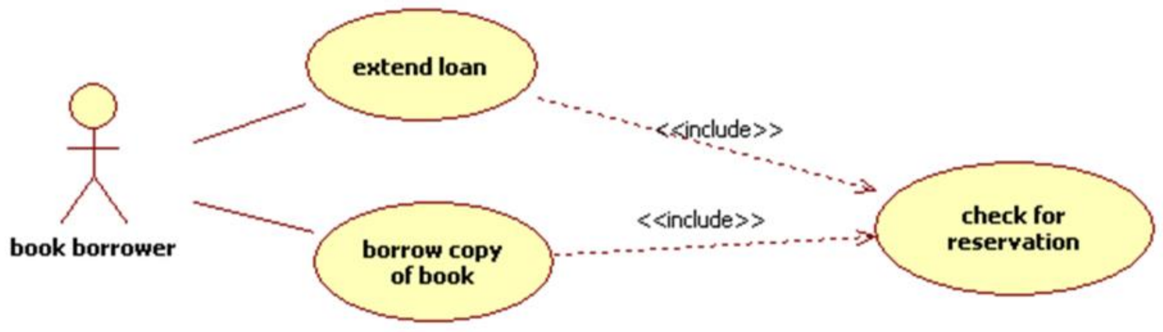
- **Inclusion (includes)**

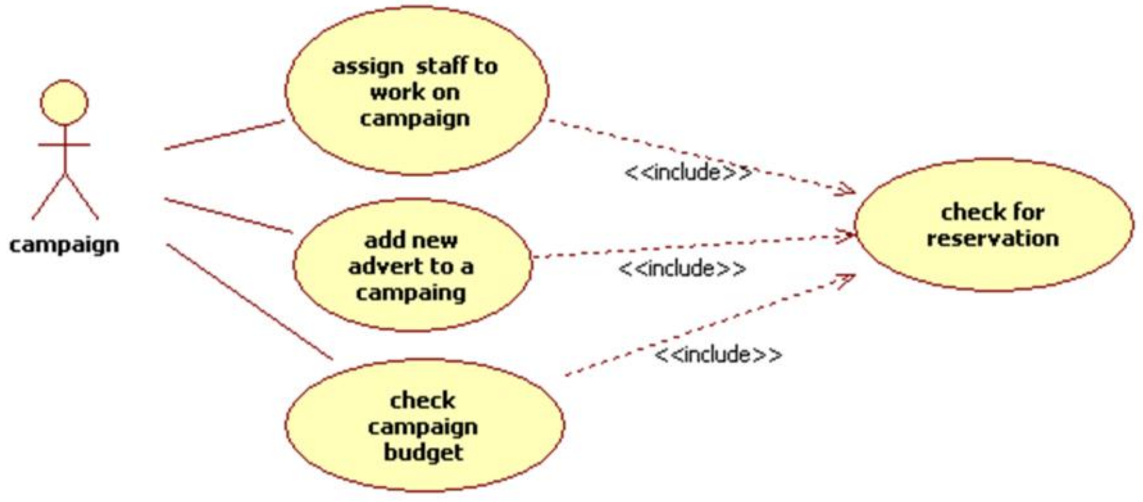
Enable you to reuse one use cases steps inside another use case.

Includes applies when there is a sequence of behavior that is used frequently in a number of use cases and you want to avoid copying the same descriptions of it into use case which it is used for.

The most vital case is when we are able to factor out common behavior from two or more of our original use cases by using a component.







- **Extends/extension (separating behavior)**

Used when you want to wish to show that a use case provides additional functionality that may be required in another use case.

Extensions allow you to create a new case by adding steps to an existing use case.

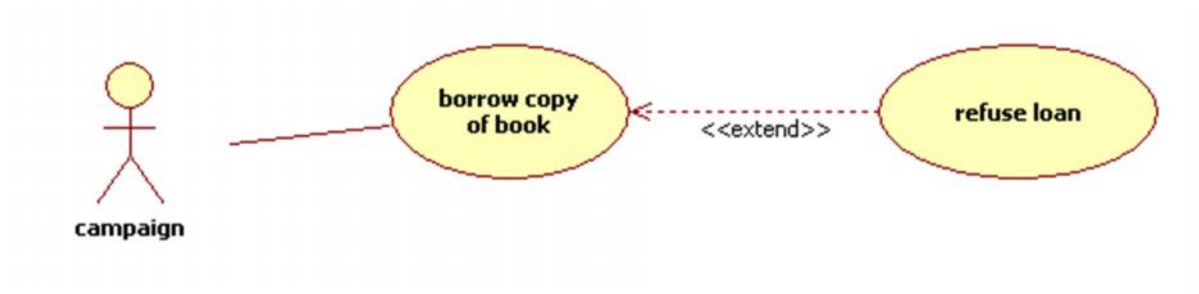
If a use case has two significant scenarios you may decide to be clearer to show these as a main case and one as or more subsidiary cases.

There may be more than one way of extending a particular use case, and these possibilities may present significant variations on the way the user uses the system

Rather trying to capture all these variations in one use case, you would document the core functionality in one and extend it to the other.

Example

Separate scenario 'borrow copy of book' into the normal case which the user is allowed to borrow the book and the unusual case in which the user is not allowed to borrow the book, because he/she has already borrowed the maximum number of books.(items)



Extension Point

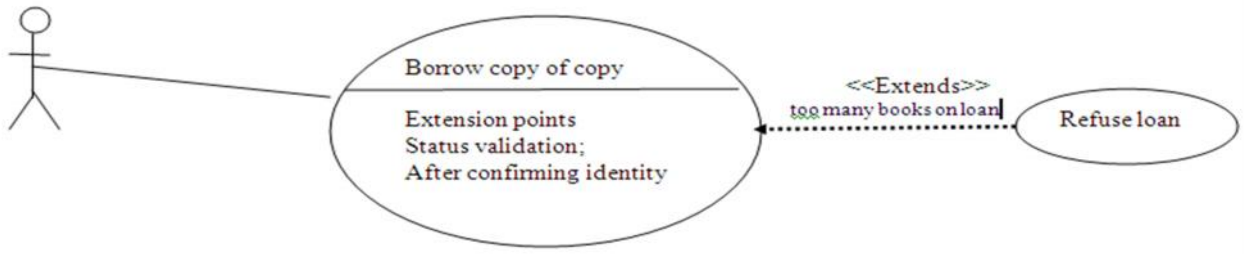
In the new version of the decomposed use case we must show the condition under which the exceptional case applies. The point at which the condition is tested and the behavior may diverge is the extension point.

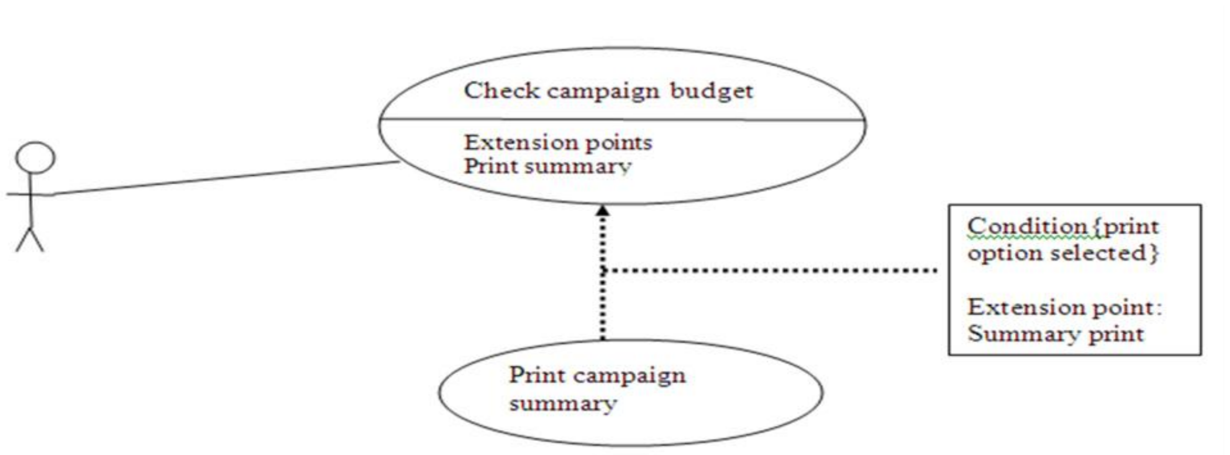
The Extension Points of a Use Case show exactly where an extending use case is allowed to add functionality.

An extension point describes a point in a use case where an extending use case may provide additional behavior. Examples for a travel agent sales system might be the use case for paying for a ticket, which has an extension point in the specification of the payment. Extending use cases may then extend at this point to pay by cash, credit card etc.

In use cases modeling the extension points are shown in the diagram as separate compartments in the use case ellipse (central use case) headed Extension Points and extensions points if it exists it must have a name. The condition must be true for the extension to take place in a particular instance of the use case.

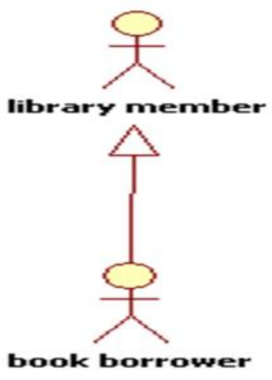
Extends with extension points



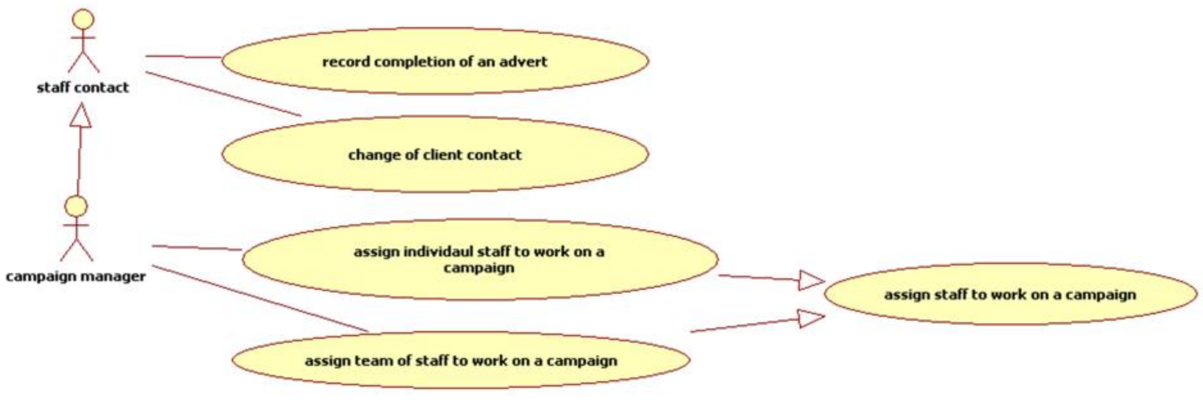


Generalization

Uses cases and actors can inherit from one another. In use case inheritance, the child use case inherits behavior and meaning from parent and adds it to its own behavior. Modeled with a solid line with an open triangle pointing to the parent.



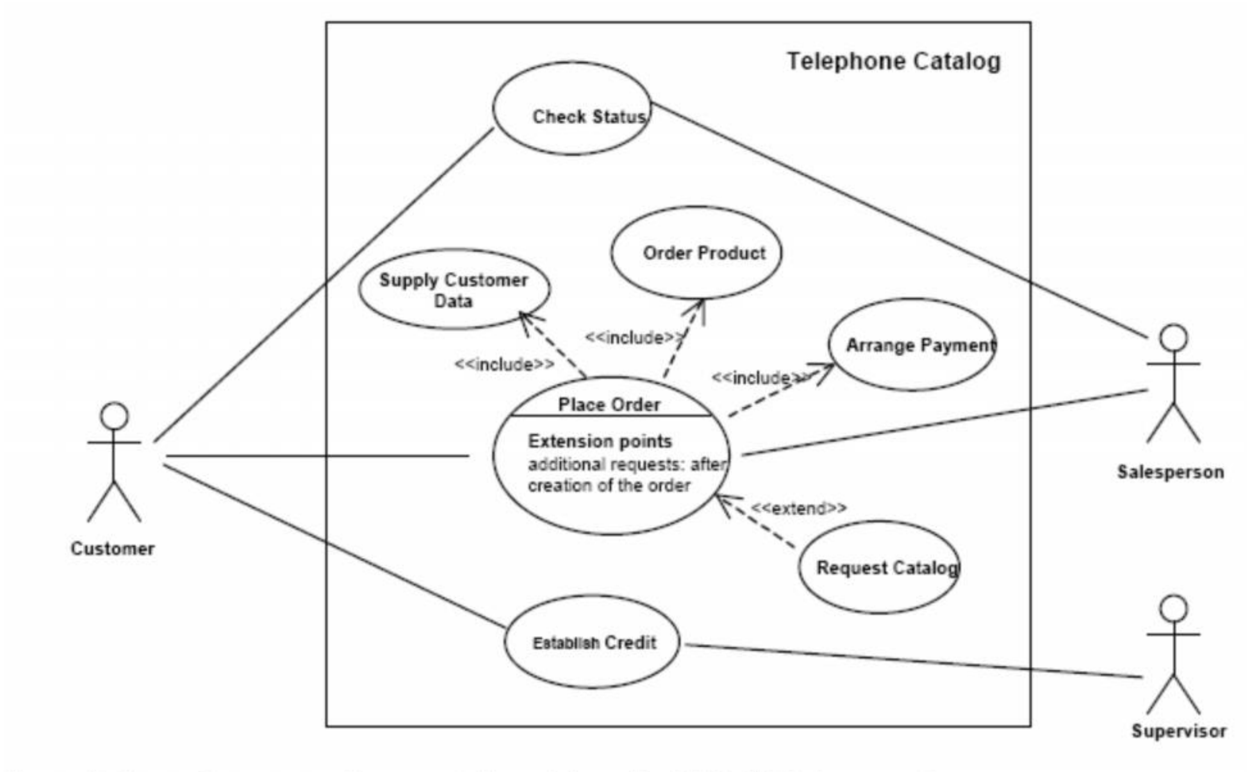
The common functionality is best represented by generalizing out that functionality into a "super case" and showing it separately

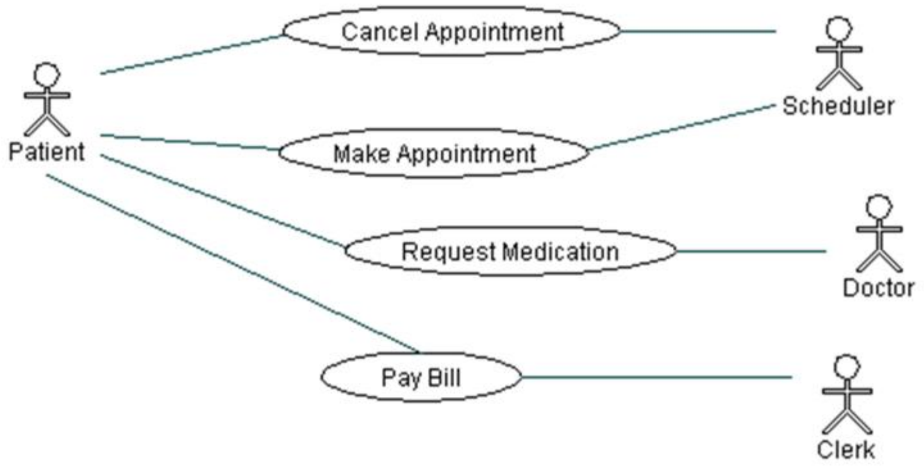

staff contact
record completion of an advert
change of client contact
campaign manager
assign individaul staff to work on a campaign
assign team of staff to work on a campaign
assign staff to work on a campaign

generalisation of actors and use cases

Revision Questions

Give a descriptive analysis of the use case below





Revision Questions

3. A golf club wants to develop software to support a number of its activities. The club secretary will use the system to manage membership details, which includes adding and removing members. To become a member an application has to be recommended by two current members. The secretary can add tournaments as well as printing tournament results. Members of the club can enter tournaments if they wish and can view results. Draw a use case diagram for the following scenario

4. When a customer goes to shop in a supermarket, he/she starts by filling all the items they need on the trolley. The customer then pushes the trolley to the trolley to the cashier who calculates the total cost of all the items and presents the same o the customer. Thereafter, the customer may decide to pay in cash or present a card. The credit card is then swiped by the cashier through a card reader that debits the same amount from the customer's account. The cashier then returns the card to the customer. Capture the system requirements of the above scenario using use case diagrams (10 marks)

5. A golf club wants to develop software to support a number of its activities. The club secretary will use the system to manage membership details, which includes adding and removing members. To become a member an application has to be recommended by two current members. The secretary can add tournaments as well as printing. Draw a use case diagram for the following scenario (10 marks)

6. As part of the college's it strategy, the electronic prospectus is intended to provide means of viewing course information for staff, students and managers. The system should provide lists of lecturers, courses and access to more detailed information about which units are taught on which course, individual lecturer records may be queried for information such as their office room number and email address as well as reporting the units a lecturer teaches. Each course in the prospectus can be identified by unique course named, consists of a number of units at each level. Every course has one lecturer acting as the course leader who is responsible for administering the course and updating includes the prospectus which includes functions such as allowing lecturers and courses to be removed from the system. Draw a use case diagram to model the scenario above (15 marks)

7. The applicant enters the license application number, the system retrieves the information related to it, and the system displays this information. Using the scenario given, identify the actors and the use cases depicted in the system. i) Using suitable notation, describe the two major components of a use case diagram. ii) Providing suitable examples, elaborate on the following use case relationships (i.e Extends, Includes