OO Analysis: Use Case Modeling
Case Study

In Islamabad for environment protection, a bike(bicycle) hire shop (Hire-n-Ride-Bike) is started Super-market. Main aim is to encourage residents and tourists to explore Islamabad and Margalla hills tracks on bikes. For that purpose, shop maintains a range of bikes (ladies, sports, children, mountain etc.).
At present all the activities of the shop are maintained through registers and paper receipts. All the information about the bikes (bike number, type, size, make, model, daily charge rate and deposit) are recorded on a register. The shop also has a small workshop which is looked after by a mechanic who is responsible to keep the bikes in running conditions and inspects the bike when it is returned to access any damages to a bike.

Shop owner is eager to have a computer-based system to manage all the activities for bike hiring and to maintain bikes inventory. To help the owner we offer to create a software-based solution for his hire-n-ride-bike shop.

First, we would create a use case model by using the following requirements finalized through information gathered from the shop manager and different scenarios observed while visiting the shop and interactions between customers and the shop manager.

Requirements for the Hire-n-Ride-Bike system

R1: The system must keep a complete list of all bikes and their details including bike number, type, size, make, model, daily charge rate, deposit.
R2: keep a record of all customers and their past hire transactions.
R3: work out automatically how much it will cost to hire a given bike for a given number of days.
R4: record the details of a hire transaction including the start date, estimated duration, customer and bike, in such a way that it is easy to find the relevant transaction details when a bike is returned.
R5: keep track of how many bikes a customer is hiring so that the customer gets one unified receipt not a separate one for each bike.
R6: cope with a customer who hires more than one bike, each for different amounts of time.
R7: work out automatically, on the return of a bike, how long it was hired for, how many days were originally paid for, how much extra is due.
R8: record the total amount due and how much has been paid.
R9: print a receipt for each customer.
R10: keep track of the state of each bike, e.g. whether it is in stock, hired out or being repaired.
R11: provide the means to record extra details about specialist bikes

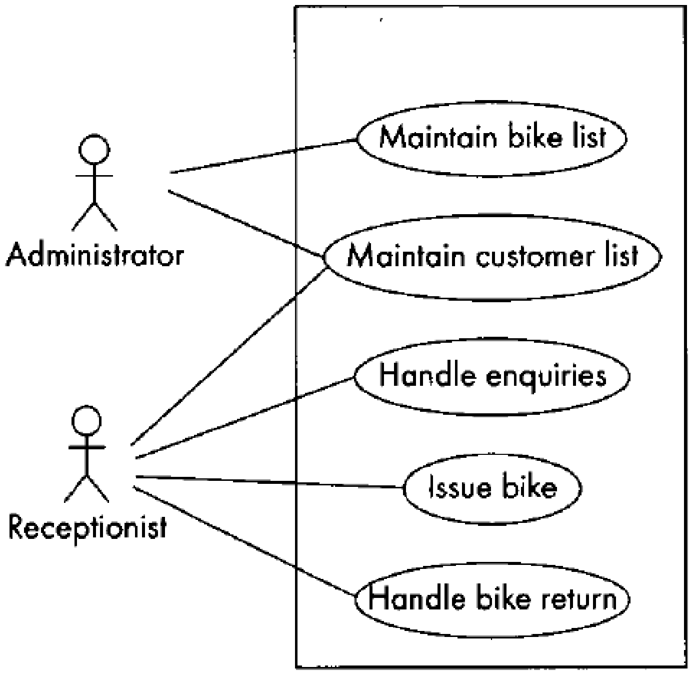
From the requirements of the system to create a use case model, we have to perform following activities:

- Identification of Actors and Use Cases
- Identification of Relationship between Actors and Use Cases
- Creating Use Case Diagram

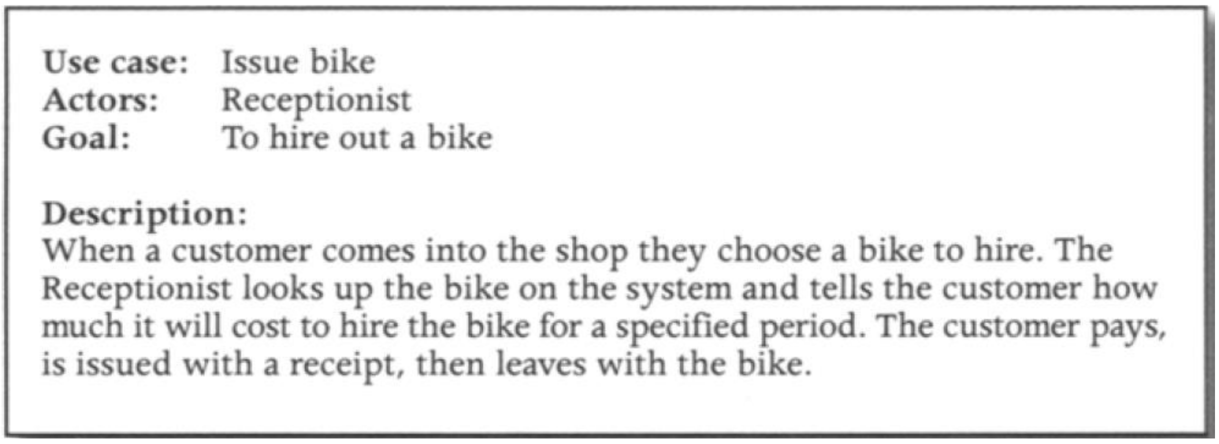
The main functionality of the system is divided into five activities being performed in the shop, that are our Use-Cases for modeling:

- Maintain bike list
- Maintain customer list
- Handle enquiries
- Issue bike
- Handle bike return

Next, we identify Actors; two actors receptionist and system Administrator initiate most of the use cases. With the help of these two Actors and their association with use cases we create our first basic use case diagram:



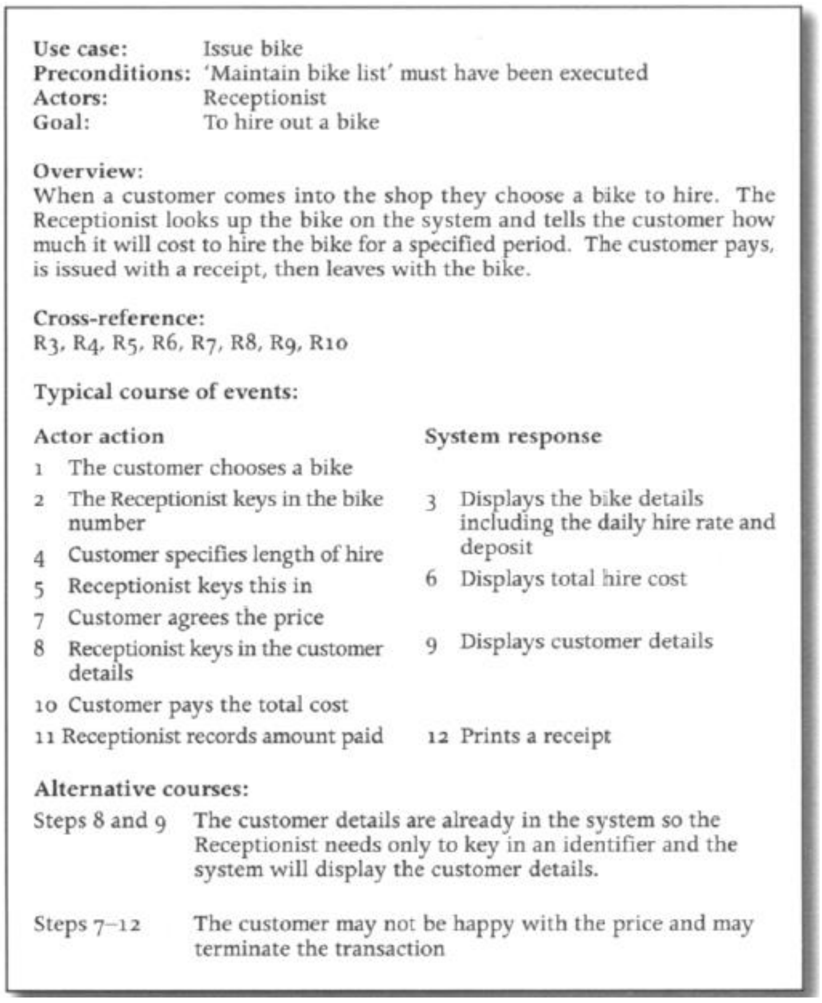
The use case description is a narrative document that describes, in general terms, the required functionality of the use case. We write High Level Description of “Issue bike” as listed below:



Use case: Issue bike
Actors: Receptionist
Goal: To hire out a bike

Description:
When a customer comes into the shop they choose a bike to hire. The Receptionist looks up the bike on the system and tells the customer how much it will cost to hire the bike for a specified period. The customer pays, is issued with a receipt, then leaves with the bike.

To capture all the work being performed in a use case, normally critical use cases are described in an expanded form, and for that a template is used.
Expanded form Description of “Issue bike” Use Case is listed below:



Use case:	Issue bike
Preconditions:	‘Maintain bike list’ must have been executed
Actors:	Receptionist
Goal:	To hire out a bike

Overview:
When a customer comes into the shop they choose a bike to hire. The Receptionist looks up the bike on the system and tells the customer how much it will cost to hire the bike for a specified period. The customer pays, is issued with a receipt, then leaves with the bike.

Cross-reference:
R3, R4, R5, R6, R7, R8, R9, R10

Typical course of events:

Actor action	**System response**
1 The customer chooses a bike	
2 The Receptionist keys in the bike number	3 Displays the bike details including the daily hire rate and deposit
4 Customer specifies length of hire	
5 Receptionist keys this in	6 Displays total hire cost
7 Customer agrees the price	
8 Receptionist keys in the customer details	9 Displays customer details
10 Customer pays the total cost	
11 Receptionist records amount paid	12 Prints a receipt

Alternative courses:

Steps 8 and 9	The customer details are already in the system so the Receptionist needs only to key in an identifier and the system will display the customer details.
Steps 7–12	The customer may not be happy with the price and may terminate the transaction

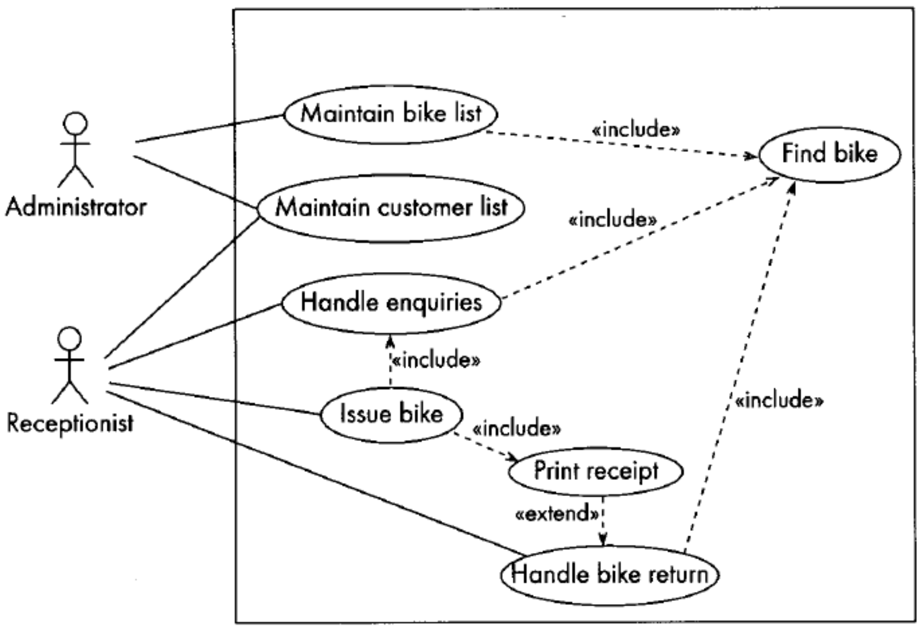
Relationships between Use Cases
An <<include>> relationship is useful when you find you have a chunk of behaviour that is common to several use cases
'Maintain bike list’, 'Handle enquiries', 'Issue bike' and 'Handle bike return' all need to find a particular bike from the list of bikes.

We create a new use case 'Find bike' and link it with an <<include>> relationship to the four use cases that need it. This tells us that each of these use cases will always use the 'Find bike' use case.
First part of the use case 'Issue bike' repeats the behaviour of the use case 'Handle enquiries’, and receptionist always tells customers the daily hire rate and deposit for a bike before going ahead with the issuing of bike. Rather than repeat a description of this behaviour in both use cases, we can remove it from 'Issue bike' and have an <<include>> relationship between 'Issue bike' and 'Handle enquiries'.

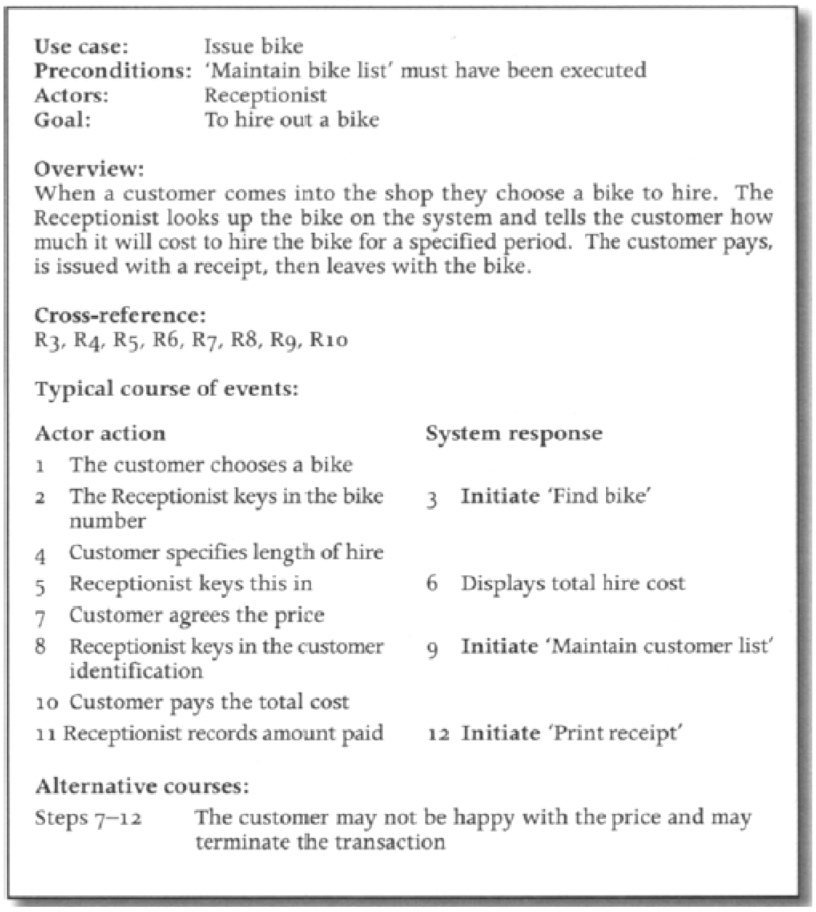
The <<extend>> relationship is used as a way of specifying significant alternative behaviour in a use case. Minor variations can be covered in the extended use case description.
We would use an <<extend>> to describe:

- Extra functionality that is available if required, e.g. printing a list rather than just viewing it on the screen.
- Behaviour done only under certain conditions, e.g. printing an extra receipt if the whole deposit is not returned.

<<include>> and <<extend>> relationships are depicted in the following extended Use Case Diagram:



If we have added <<include>> or <<extend>> relationships to a use case diagram, we must document them in the use case description. This is done using the keyword 'initiate' to invoke the called use case.
The “Issue bike” use case description with included and extended use cases is listed below:



Use case:	Issue bike
Preconditions:	‘Maintain bike list’ must have been executed
Actors:	Receptionist
Goal:	To hire out a bike

Overview:
When a customer comes into the shop they choose a bike to hire. The Receptionist looks up the bike on the system and tells the customer how much it will cost to hire the bike for a specified period. The customer pays, is issued with a receipt, then leaves with the bike.

Cross-reference:
R3, R4, R5, R6, R7, R8, R9, R10

Typical course of events:

Actor action	**System response**
1 The customer chooses a bike	
2 The Receptionist keys in the bike number	3 **Initiate** ‘Find bike’
4 Customer specifies length of hire	
5 Receptionist keys this in	6 Displays total hire cost
7 Customer agrees the price	
8 Receptionist keys in the customer identification	9 **Initiate** ‘Maintain customer list’
10 Customer pays the total cost	
11 Receptionist records amount paid	12 **Initiate** ‘Print receipt’

Alternative courses:

Steps 7–12	The customer may not be happy with the price and may terminate the transaction