As an example of structured English, we will specify the calcDaysOverdue() operation, following the informal description as shown below.

calcDaysOverdue()
This operation uses today's date from the system clock and the attributes Hire.startDate and Hire.numberDays to calculate whether the bike has been returned late and if so by how many days. It calculates the overdue amount (Bike.dailyHireRate multiplied by the number of days late) and records it by executing latenessDeduction(amt).

In structured English this operation could be specified as follows:

```
Add numberDays to startDate to give return date
number of days late = today's date – return date
IF return date > today's date
    THEN (*bike is overdue*)
        latenessDeduction = Bike.dailyHireRate * number of days late
    ELSE (*bike is not overdue*)
Display latenessDeduction
```

When an operation involves a number of decisions, it is often helpful to specify these using a decision table or decision tree. For example, in a more sophisticated bike hire system, the hire charges could depend on the number of bikes a customer has hired in the past year and the number of bikes in the current hire. Let us imagine that Wheels introduce discount hire rates as follows:

> If a customer has already hired at least five bikes during the past year they get a 15% discount, unless the current hire is for three bikes or more, in which case they get a 25% discount.

Figure 6.17 shows how this could be represented in a decision table.

In the decision table all the possible conditions are listed in the top left-hand quarter of the table and all the actions in the bottom left-hand quarter. The top right-hand corner of the table tabulates, in separate columns, all possible combinations of conditions (a dash indicates that the condition is currently irrelevant). This is the Rules section. The appropriate actions are shown below each rule, in the bottom right-hand corner of the table.

Figure 6.18 shows the same information as a decision tree.

Other methods of specifying complex operations include the Object Constraint Language (OCL), which is UML's formal language for specifying constraints on an object model. The Wheels system is not nearly large or complicated enough to justify using OCL, but you can find details about the language in Bennett *et al.*, (2002).