6 Object Oriented Software Analysis and Design

Introduction

This chapter will teach rudimentary analysis and modelling skills through practical examples. Leading the reader to an understanding of how to get from a preliminary specification to an Object Oriented Architecture.

Objectives

By the end of this chapter you will be able to…

- Analyse a requirements description
- Identify items outside scope of system
- Identify candidate classes, attributes and methods
- Document the resulting Object Oriented Architecture

This chapter consists of twelve sections:-

1) Requirements Analysis
2) The Problem
3) Listing Nouns and Verbs
4) Identifying Things Outside The Scope of The System
5) Identifying Synonyms
6) Identifying Potential Classes
7) Identifying Potential Attributes
8) Identifying Potential Methods
9) Identifying Common Characteristics
10) Refining Our Design using CRC Cards
11) Elaborating Classes
12) Summary

6.1 Requirements Analysis

The development of any computer program starts by identifying a need:-

- An engineer who specialises in designing bridges may need some software to create three dimensional models of the designs so people can visualise the finished bridge long before it is actually built.
- A manager may need a piece of software to keep track of personal, what projects they are assigned to, what skills they have and what skills needs to be developed.

But how do we get from a 'need' for some software to an object oriented software design that will meet this need?

Some software engineers specialise in the task of Requirement Analysis which is the task of clarifying exactly what is required of the software. Often this is done by iteratively performing the following tasks:-

1) interviewing clients and potential user of the system to find out what they say about the system needed
2) documenting the results of these conversations,
3) identifying the essential features of the required system
4) producing preliminary designs (and possibly prototypes of the system)
5) evaluating these initial plans with the client and potential users
6) repeating the steps above until a finished design has evolved.

Performing requirements analysis is a specialised skill that is outside the scope of this text but here we will focus on steps three and four above ie. given a description of a system how do we convert this into a potential OO design.

While we can hope to develop preliminary design skills experience is a significant factor in this task. Producing simple and elegant designs is important if we want the software to work well and be easy to develop however identifying good designs from weaker designs is not simple and experience is a key factor.

A novice chess player may know all the rules but it takes experience to learn how to choose good moves from bad moves and experience is essential to becoming a skilled player. Similarly experience is essential to becoming skilled at performing user requirements analysis and in producing good designs.

Here we will attempt to develop rudimentary skills in the hope that you will have the opportunity to practise those skills and gain experience later.

Starting with a problem specification we will work through the following steps:-

- Listing Nouns and Verbs
- Identifying Things Outside The Scope of The System
- Identifying Synonyms
- Identifying Potential Classes
- Identifying Potential Attributes
- Identifying Potential Methods
- Identifying Common Characteristics
- Refining Our Design using CRC Cards
- Elaborating Classes

By doing this we will be able to take a general description of a problem and generate a feasible, and hopefully elegant, OO design for a system to meet these needs.

6.2 The Problem

The problem for which we will design a solution is 'To develop a small management system for an athletic club organising a marathon.'

For the purpose of this exercise we will assume preliminary requirements analysis has been performed and by interviewing the club managers, and the workers who would use the system, the following textual description has been generated.

The 'GetFit' Athletic Club are organizing their first international marathon in the spring of next year. A field comprising both world-ranking professionals and charity fund-raising amateurs (some in fancy dress!) will compete on the 26.2 mile route around an attractive coastal location. As part of the software system which will track runners and announce the results and sponsorship donations, a model is required which represents the key characteristics of the runners (this will be just part of the finished system).

Each runner in the marathon has a number. A runner is described as e.g. "Runner 42" where 42 is their number. They finish the race at a specified time recorded in hours, minutes and seconds. Their result status can be checked and will be displayed as either "Not finished" or "Finished in hh:mm:ss".

Every competitor is either a professional runner or an amateur runner.

Further to the above, a professional additionally has a world ranking and is described as e.g. "Runner 174 (Ranking 17)".

All amateurs are fundraising for a charity so each additionally has a sponsorship form. When an amateur finishes the race they print a collection list from their sponsorship form.

A sponsorship form has the number of sponsors, a list of the sponsors, and a list of amounts sponsored. A sponsor and amount can be added, and a list can be printed showing the sponsors and sponsorship amounts and the total raised.

A fancy dress runner is a kind of amateur (with sponsorship etc.) who also has a costume, and is described as e.g. "Runner 316 (Yellow Duck)".

6.3 Listing Nouns and Verbs

The first step in analysing the description above is to identify the nouns and verbs:-

- The nouns indicate entities, or objects, some of these will appear as classes in the final system and some will appear as attributes.
- The verbs indicate actions to be performed some of these will appear in the final system as methods.

Nouns and verbs that are plurals are listed in their singular form (e.g. 'books' becomes 'book') and noun and verb phrases are used where the nouns\verb alone are not descriptive enough e.g. the verb 'print' is not as clear as 'print receipt'.

Activity 1

Look at the description above list five nouns and five verbs (use noun and verb phrases where appropriate).

Feedback 1

The list below is a fairly comprehensive list of the nouns and verbs, not just the first five.

Nouns:- GetFit Althletic Club, field, world ranking professional, fund-raising amateur, fancy dress, 26.2 mile route, coastal location, software system, runner, result, sponsorship donation, model, key characteristic, a number, time, result status, competitor, professional runner, amateur runner, world ranking, charity, sponsorship form, collection list, sponsor, sponsorship amount, total raised, costume.

Verbs:- Organise, marathon, compete, track runners, announce results, describe (runner), finish race, specify time, check status, display status, describe (professional), print collection list, add (sponsor and amount), print list, describe (fancy dress runner)

6.4 Identifying Things Outside The Scope of The System

An important part in designing a system is to identify those aspects of the problem that are not relevant or outside the scope of the system. Parts of the description may be purely contextual i.e. for general information purposes and thus not something that will directly describe aspects of the system we are designing. Furthermore while parts of the description may refer to tasks that are performed by users of the system as they are using the system, and thus describe functions that need to be implemented within the system, other parts may describe tasks performed by users while not using the system – and thus don't describe functions within the system.

By identifying things in the description that are not relevant to the system we are developing we keep the problem as simple as possible.

Activity 2

Look at the list of nouns and verbs above and identify one of each that is outside the scope of the system.

Feedback 2

Most of the first paragraph is contextual and does not describe functionality we need to implement within the system. We also need to look at other parts of the description to identify parts that are not relevant.

Things outside the scope of the system…

Nouns:-

- GetFit Althletic Club – this is the client for whom the system is being developed. It is not an entity we need to model within the system.
- Coastal location – the location of the run is not relevant to the functionality of the system as described. Again we do not need to model this as an object within the system.
- Software system – this is the system we are developing as a whole it does not describe an entity within the system.
- Verbs:-
- Organise – this is an activity done by members of the athletic club, these may be users of the system but this is not an activity that they are using the system for.
- Marathon – this is what the runners are doing. It is not something the system needs to do.

Note: 'Finish race' is something that a runner does however when this happens their finish time must be recorded in the system. Therefore this in NOT in fact outside the scope of the system.

6.5 Identifying Synonyms

Synonyms are two words that have these same meaning. It is important to identify these in the description of the system. Failure to do so will mean that one entity will be modelled twice which will lead to duplication and confusion.

Activity 3

Look at the list of nouns and verbs and identify two synonyms, one from the list of nouns and one from the verbs.

Feedback 3

Synonyms

Nouns:-

- world ranking professional=professional runner
- fund-raising amateur=amateur runner
- runner=competitor

Note runner is not a synonym of professional runner as some runners are amateurs.

Verbs:-

- marathon=compete
- check status=display status
- print collection list = print list
- finish race = record specified time

6.6 Identifying Potential Classes

Having simplified the problem by identifying aspects that are outside the scope of the system and by identifying different parts of the description that are in reality describing the same entities and operations we can now start to identify potentials classes in the system to be implemented.

Some nouns will indicate classes to be implemented and some will indicate attributes of classes.

Good OO design suggests that data and operations should be packaged together – thus classes represent complex conceptual entities for which we can identify associated data and operations (or methods).

Simple entities, such as an address, have associated data but no operations and thus these can be stored as simple attributes within a related class.

Activity 4

Look at the list of nouns above and identify five that could become classes.

Feedback 4

Nouns that could indicate classes:-

- Runner (or Competitor)
- Amateur (or Amateur Runner)
- Professional (or similar)
- FancyDresser (or FancyDressAmateur or similar)
- Sponsorshipform

6.7 Identifying Potential Attributes

Having identified potential classes the other nouns could by used to identify attributes of those classes.

Activity 5

Look at the list of nouns and identify one that could become an attribute of the class 'Runner' and one for the class 'FancyDresser'.

Feedback 5

Nouns that could become attributes...

For Runner:-

number,
resultStatus ie. finished (boolean)
time (hours, minutes, seconds)

For FancyDresser:-

costume (String)

Of course we need to identify all of the attributes for all of the classes.

6.8 Identifying Potential Methods

Having identified potential classes we can now use the verbs to identify methods of those classes.

Activity 6

Look at the list of verbs and identify one that could become a method of the class 'Runner' and one for the class 'FancyDresser'.

Feedback 6

Verbs that could become methods....

For Runner:-

describe (this will actually become an overridden version of toString())
finishRace
displayStatus

For FancyDresser:-

describe (toString() will need to be overridden again to encompass the description of the costume)

Of course we need to identify all of the methods for all of the classes.

6.9 Identifying Common Characteristics

Having identified the candidate classes with associated attributes and methods we can start structuring our classes into appropriate inheritance hierarchies by identifying those classes with common characteristics.

Activity 7

Look at the list of classes below and place four into an appropriate inheritance hierarchy. Identify the one class that would not fit into this hierarchy:-

Runner
Amateur
Professional
FancyDresser
Sponsorshipform

Feedback 7

The most general class i.e. the one at the top of the inheritance tree is 'Runner'. Amateur and Professional are subclasses of 'Runner' and FancyDresser is a specific type of Amateur hence a subclass of Amateur.

We can fit these into an inheritance hierarchy because these classes are all related by an is-a relationship. A FancyDresser is-a Amateur which in turn is-a Runner. A Professional is-a Runner as well.

A SponsorshipForm is not a type of Runner and hence does not fit into this hierarchy. This class will be related to one of the other classes by some form of an association. Looking at the description we can see that not all runners have a sponsorship form only amateurs who are running for charity. There is therefore an association between Amateur and SponsoshipFrom. Of course FancyDressers inherit the attributes defined in Amateur and hence they automatically have a SponsorshipFrom.

6.10 Refining Our Design using CRC Cards

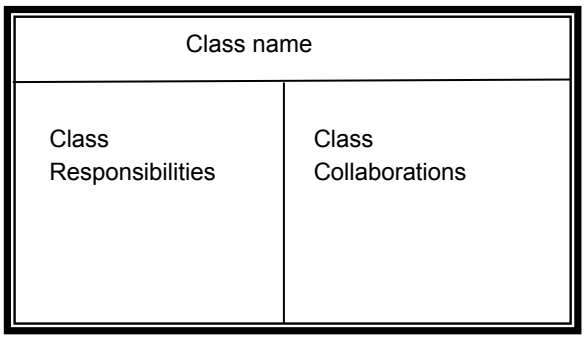
Having identified the main classes in our system, and the attributes and methods of these classes, we could now proceed to refine these designs by defining the data types and other small details, document this information on a UML diagram and program the system. However in a real world system the problem would be larger and less well-defined than the problem we are working on here and the analysis and refinement of design would therefore be a longer more complex process that we can realistically simulate.

As real world problems are more complex our initial designs are unlikely to be perfect therefore it makes sense to check our designs and to resolve any potential problems before turning these designs into a finished system.

One method of doing checking our designs is to document our designs using CRC cards and to check if these work by role-playing different scenarios.

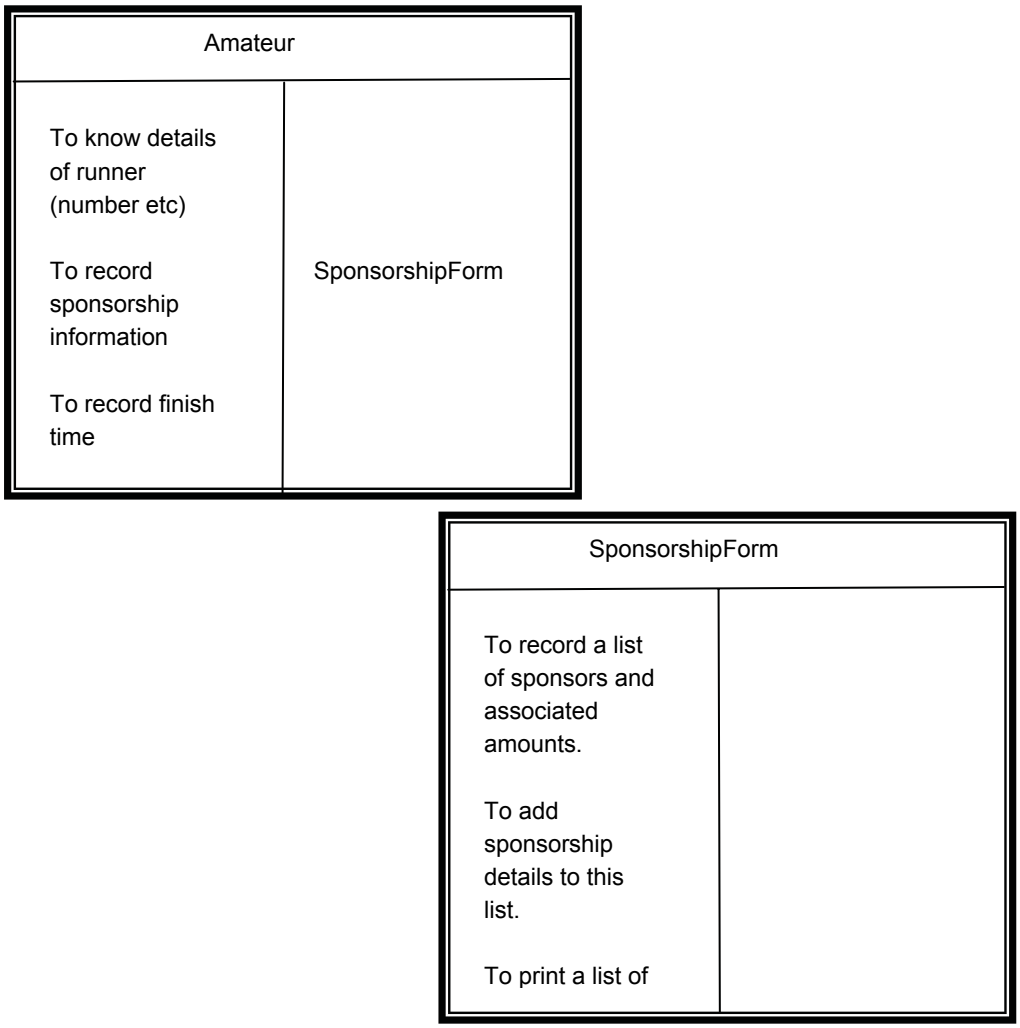
CRC cards are not the only way of doing this and are not part of the UML specification.

CRC stands for Class, Responsibilities and Collaborations. A CRC card is set out below and is made up of three panes with the class responsibilities shown on the left and the collaborations shown on the right.



Responsibilities are the things the class needs to know about (ie the attributes) and the things it should do (ie. the methods) though on a CRC card these are not as precisely defined as on a UML diagram. The collaborations are other classes that this class must work with in order to fulfil its responsibilities.

The diagram below shows CRC cards developed for two classes in the system.



Having developed CRC cards we can roleplay a range of scenarios in order to check the system works 'on paper'. If not we can amend before getting into the time consuming process of programming a flawed plan.

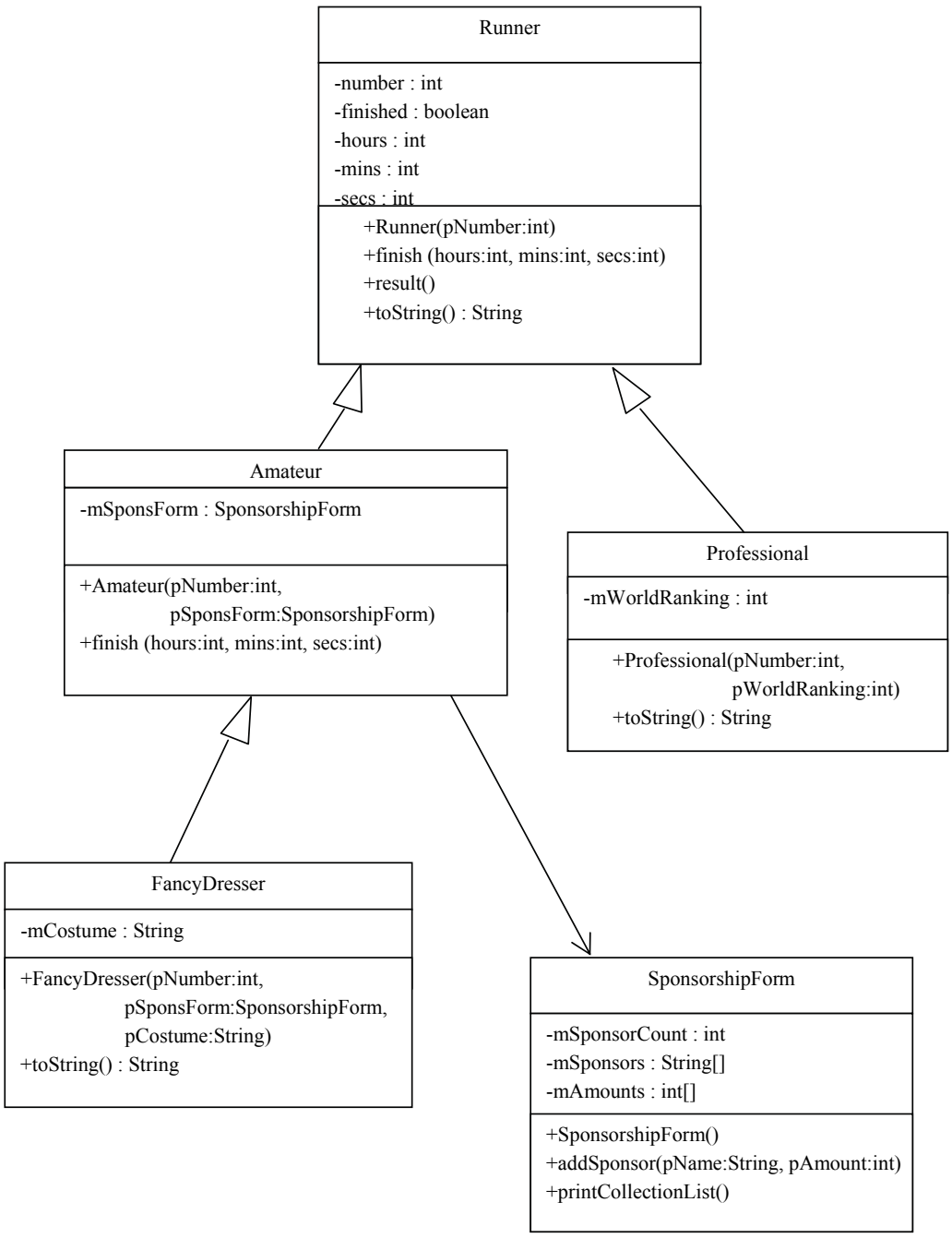
One sample scenario would be when a runner gets an additional sponsor. In this case by looking at the CRC cards above a Runner is able to record sponsorship information in collaboration with the SponsorshipForm class. The SponsorshipForm class records a list of sponsors and can add additional sponsor to this list.

Testing out a range of scenarios may highlight flaws in our system designs that we can then fix – long before any time has been wasted by programming weak designs.

6.11 Elaborating Classes

Having identified the classes in our system design, and documented and tested these using CRC cards, we can now elaborate our CRC cards and document our classes using a UML class diagram. To do this we need to take our general specification documented via CRC cards and our resolve any ambiguities e.g. exact data types.

Having elaborated our CRC cards we can now draw a class diagram for proposed design (see below):-



6.12 Summary

Gathering User Requirements is an essential stage of the software engineering process (and outside the scope of this text).

Turning a complex requirements specification into an elegant simple object oriented architectural design is a skilled task that requires experience. However a good starting point is to follow a simple process set out in this chapter.

Through a sequence of tasks we have seen how to analyse a textual description of a problem. We have:-

- Looked for aspects of the description that are outside the scope of the system,
- Identified where the description refers synonymous items using different words
- Used the nouns and verbs to identify potential classes, attributes and methods
- Looked at the classes to identify potential inheritance hierarchies and to identify other relationships between classes (e.g. associations).
- Document the resulting classes using CRC cards and tested the validity of our design by role-playing a range of scenarios and amending our designs as appropriate
- Finally we can elaborate these details and document the results using a class diagram.

The design process set out in this chapter will be demonstrated again in detail using the case study described in chapter 11.