CHAPTER 1

Introduction

1.1 Who this book is for

The word 'formal' in 'formal specification' means *mathematical.* Many practitioners and students of software engineering do not think of themselves as mathematicians and may not make use of mathematics in their work. Nonetheless, there are many ways in which the appropriate use of mathematics can aid in such work.

This book is intended for the practitioner or student who wishes to learn about formal specification in a gentle and straightforward manner. All the necessary mathematical concepts are introduced, with examples chosen to reflect everyday experience rather than technical areas. The mathematics used does *not* include: differentiation, quadratic equations, integration, trigonometry or transcendental functions; instead, it is based on discrete mathematics: the arithmetic of whole numbers, set theory and simple logic.

Even those who do not anticipate ever using formal specification in their work, or further study, will find that the concepts, specialised vocabulary and need for precision that they acquire through a study of formal specification will enrich their thinking and means of expression in all aspects of software engineering. This has been the experience of the countless students both in the United Kingdom and elsewhere who have studied this area, many by using this book.

An early review of this book described it as a 'lightweight' introduction. But that was not to be taken as a criticism, since that is precisely what the book sets out to be.

1.2 The need for formal specification

1.2.1 Problems in the creation of computer systems

There are long-standing problems in the development of computer systems: often they take too much time to produce, cost more than estimated and fail to satisfy the customer. This 'software crisis' was first identified in the 1960s and has been the subject of much research and controversy, and it is still not entirely solved.

Central to the problem is the fact that errors and inadequacies are more expensive to correct the later in the development process they are discovered. Furthermore, it is extremely difficult to clarify exactly what is required of a very complex system.

1.2.2 Software Engineering

A variety of techniques have been introduced to help deal with the difficulties of developing computer systems. These techniques are moving towards the concept of *software engineering*, where well established principles of the engineering professions are being applied to the creation of software systems. Amongst these principles is the idea that appropriate mathematical techniques should be applied in the analysis and solution of engineering problems.

1.2.3 Formal Methods

Techniques which use mathematical principles to develop computer systems are collectively known as *formal methods*. The idea of specifying what a computer system is required to do using a mathematical notation and techniques of mathematical manipulation, has led to notations for *formal specification*.

1.3 The role of mathematics

1.3.1 Background of mathematics

Mathematics has a very long history, spanning thousands of years of work by mathematicians in many different civilisations. The major principles of mathematics are now stable and proven in use in scientific and engineering applications. It is therefore appropriate to apply these to the development of computer systems.

1.3.2 Precision

Mathematical expressions have the advantage of being *precise* and *unambiguous*. There will never be any need for discussion or confusion regarding what a formal specification *means*.

1.3.3 Conciseness

Mathematical expressions are typically very concise; a great deal of meaning is concentrated in a relatively small number of symbols. This is a particular advantage when describing a very complex system.

1.3.4 Abstraction

The mathematical notion of *abstraction* plays an important role in formal methods. Abstraction involves initially considering only the essential issues of a problem and deferring consideration of all other aspects until a later stage. This *separation of concerns* is the best way of coping with the complexity of large systems.

1.3.5 Independence from natural language

The independence of mathematical forms from the context of spoken (natural) language means that mathematics can be equally well understood by all readers – irrespective of their language and culture.

1.3.6 Proofs

Deductions and conclusions expressed in a mathematical form are capable of being *proved*, by application of established mathematical laws. The ability to prove such things, rather than just to demonstrate them, or convince oneself informally of their validity, is an important aspect of the application of mathematics. This is particularly so when the consequences of an invalid conclusion could endanger life, in so-called *safety-critical* systems.

1.4 Current use of mathematics

Many practitioners involved with the development of computer systems are not mathematicians and do not make regular use of mathematics in their work. However, mathematical techniques are assuming an increasingly important role in the development of computer systems and it is appropriate that they should be known and applied by as many practitioners as possible.

1.4.1 Discrete mathematics

The mathematics used in formal specification is very simple. It is called *discrete mathematics* and is concerned more with sets and logic than with numbers. This means that even those who have studied mathematics before will probably have something new to learn. Conversely, those who have not previously been very successful at learning mathematics have a new area of mathematics to study and have the motivating factor that the benefits will be immediately applicable to their work.

In this book, all the necessary mathematical notations are introduced, with examples. Where necessary, a conventional or possible pronunciation is given. This can aid understanding and is also vital if people are to talk to each other about mathematics. However, the introductory mathematical parts are deliberately kept brief and in some places informal, since the topic of the book is the *application* of the mathematics.

1.5 The specification language Z

1.5.1 Background

The specification language Z (pronounced 'zed' in Great Britain) was initiated by Jean-Raymond Abrial in France and developed by a team at the Programming Research Group (PRG) of Oxford University in England, led by Professor C. A. R. Hoare. It is foremost amongst specification notations in use at present.

Z is used to specify new computer systems and to describe the behaviour of existing complex computer systems.

It is increasingly being learned by both practitioners of software engineering and by students. The aim of this book is to help in that process.

A Z specification works by modelling the *states* that a system can take, the *operations* that causes changes in those states to take place and the *enquiries* that can discover information about those states.

1.5.2 Software tools

The Z notation uses a set of characters, many of which are not found on most typewriter or computer keyboards. Most of the extra characters are widely used mathematical symbols, but some are new and confined to Z.

In general, Z text cannot be prepared using conventional typewriting equipment and several *software tools* have been created to permit the typesetting of text written in Z. Some of these software tools also check the text to make sure that it conforms to the rules of the Z notation (its *syntax*).

Other software tools are concerned with giving automated assistance to the process of creating mathematical proofs. Some of these are called *theorem provers* and attempt the entire proof without manual intervention; others, called *proof assistants*, support the human activity of creating proofs.

1.6 Textual aspects of the Z notation

The Z notation is used in *specification documents* which consist of sections written in Z interleaved with narrative text in natural language. Z also uses a graphical device called the *schema*, which is a box containing the mathematics. As well as having a useful effect in visually separating the mathematics from the narrative, schemas have important special properties which will be explained in this book.

1.6.1 Identifiers

There is a need to invent names when creating a formal specification. The rules for constructing such *identifiers* are similar to those of computer programming languages.

- Identifiers may be any length and are constructed from the (upper- and lower-case) Roman letters without diacriticals marks (accents, umlauts, etc.) and from the numeric digits and the 'low-line' ('underscore') character ('_').
- The first character must be a letter.
- The upper-case and lower-case letters are considered to be different.
- Only a single word may be used for an identifier; when this needs to be a compound of several words the *low-line* character (_) can be used, or the convention of starting each component of the identifier with an upper-case letter can be adopted. For example:

  very_long_identifier
  VeryLongIdentifier
- By convention, identifiers of variables start with a lower-case letter, identifiers of types are written entirely in capital letters, and identifiers of *schemas* begin with a capital letter.
- The special characters Δ and Ξ on the front of an identifier, and ? and ! and ' on the end, have special meanings that will be explained later.

EXERCISES

1. An annual weekend event begins on the Friday evening and finishes on the Sunday afternoon. The date of the event is specified as: 'the last weekend in September'. What is the date of the Friday on which the event begins if the last day of September (30th) in that year is:

   (a) a Monday;
   (b) a Sunday;
   (c) a Saturday;
   (d) a Friday?

   Suggest an unambiguous specification of the date of the event.
2. On Friday the first of the month a software engineer goes away leaving an undated message on her/his desk saying: 'Software engineer on leave until next Wednesday'. A colleague from another department passes the desk on Monday 4th of the same month and

reads the message. When would the colleague expect the software engineer next to be back at work?

3. A video cassette recorder has a 'programming' facility which allows recordings to be made in the user's absence. Requests for recordings consist of: start-date, start-time, end-time and channel-number. Dates are specified by month-number and day-number. Times are given as hours and minutes using the 24-hour clock. If the end-time is earlier in the day than the start-time then the end-time is considered to be on the following day. Up to eight requests can be stored and are numbered one to eight.

   How would you expect the recorder to behave in the following circumstances?

   (a) start-date does not exist, for example, 31st April.
   (b) start-date does not exist; not a leap year, 29 February.
   (c) start- and end-times for different requests overlap (on same day).
   (d) start-date is for New Year's Eve and end-time is earlier in the day than start-time.
   (e) requests are not in chronological order. For example, request 1 is for a recording which occurs later than that of request 2.

   Look at the user handbook for a video cassette recorder similar to the one described here. Does the handbook answer these questions?

4. In the requirements specification of a computer program it is stated that: 'The records input to the program should be sorted in order of increasing key.'

   Does this mean that the program may be written on the assumption that the records will already have been sorted (by some external agent), or does it mean that the program must perform the required sorting?

5. Look at the manual(s) for any reasonably complex machine or system. Find the parts which are ambiguous or unclear and pose questions that are reasonable to ask but which are not answered by the manual.