1 Global Overview of the Book

1.1 INTRODUCTION AND OBJECTIVES

The main goal of this book is to show how to design software systems for financial derivatives products using the object-oriented language C#. We have chosen C# for a number of reasons and we would like to explain the rationale behind this choice:

- C# is relatively easy to learn (certainly when we compare it to C++). This means that it can be learned by traders, quants and other finance professionals who do not necessarily spend all their waking hours designing and implementing software systems. For example, people with a background in VBA will find the transition to C# much easier than the transition from VBA to C++. Furthermore, in many cases developer productivity levels in C# can be four times as high as with C++.
- The .NET framework and C# offer the developer a range of functionality that he or she can use in financial applications. This is realised by the features in the language itself and by the libraries in the framework. We shall discuss these libraries and we shall also see in the rest of this book how they are used to create robust and flexible applications.
- It is possible to create *interoperable applications* that consist of a mixture of C#, C++ and VBA code and that communicate with Excel and real-time data servers. In other words, it is possible to create .NET applications that can communicate with non .NET code and it is also possible to create non .NET applications that can communicate with .NET code.
- *Usability levels* are high. Furthermore, developers do not have to worry about manual memory management as this is taken care of by *garbage collection* mechanisms resident in the .NET framework.
- C# and the .NET framework contain libraries that allow developers to create multi-threaded applications that run on shared memory multi-core processors.
- Many .NET libraries have been designed in such a way that they can be used and adapted to suit new developer needs. In particular, it is easy to use and apply *design patterns* in C# applications.

In this book we discuss each of these topics in detail.

1.2 COMPARING C# AND C++

C# is a descendant of the C programming language (K&R 1988). It is worth pausing for a moment to consider whether it is better (in some sense) to develop new applications in C# and or C++. In order to help the reader determine how to choose between C# and C++, we discuss the problem from three perspectives:

- P1: The skills and knowledge of those engineers developing applications.
- P2: The type of application and related customer wishes.
- P3: The technical and organisational risks involved in choosing a given language.

We discuss each perspective in turn. First, C++ is a huge multi-paradigm language and it supports the modular, object-oriented and generic programming models. It is based on the C language and it would seem that it is the language of choice for many pricing, hedging and risk applications. It is not an easy language to learn. There are many books that discuss C++ and its syntax but there are surprisingly few that discuss how to apply C++ to finance and even to other application domains. C#, on the other hand is a relatively new language and it supports the object-oriented and generic programming models, but not the modular programming model. This implies that everything must be an object or class in C#.

In general, C# is much easier to learn than C++. It shields the developer from many of the low-level details seen in C++, in particular the pointer mechanism, memory management and garbage collection. In short, the C# developer does not have to worry about these details because they are automatically taken care of. This is a mixed blessing because there are situations where we wish to have full control of an object's lifecycle. C++ is a vendor-neutral language (it is an ISO standard) while C# was originally developed by Microsoft for its Windows operating system.

Perspective P2 is concerned with the range of applications to which C++ or C# can be applied, how appropriate C++ and C# are for these applications and how customer wants and needs determine which language will be most suitable in a particular context. In general, customers wish to have applications that perform well, are easy to use and easy to extend. On the issue of performance, C++ tends to be more efficient than C# and tests have shown that in general C++ applications are 20% faster than the corresponding applications in C#. This difference may or may not be a critical factor in determining whether C++ is more suitable than C# in a particular context.

To compare the two languages from the perspective of developer productivity, we first need to define what we are measuring. C# has many libraries that the developer can use, thus enhancing productivity. C++, on the other hand does not have such libraries and they must be purchased as third-party software products. In particular, user-interface software for C# is particularly easy to use while in C++ the developer must use libraries such as MFC, QT or OWL, for example. In general, we conclude that productivity levels are much higher in C# than in C++.

Finally, perspective P3 is concerned with the consequences and risks to the organisation after a choice has been made for a particular language. C++ is a large and difficult language, it takes some time to master and C++ applications tend to be complex and difficult to maintain. However, C++ is an ISO standard.

1.3 USING THIS BOOK

This book represents the joint work of Andrea Germani (trader/quant) and Daniel J. Duffy (numerical analyst/software designer). The focus of this book reflects the backgrounds of the authors and the objectives that they had when they first had the idea in a Milan *ristorante* all those years ago (or so it seems) to write a practical book that would appeal to traders and to quants who work in the financial markets. The outcome is what you have in your hands. It contains 26 chapters that are logically grouped into major categories dealing with C# syntax, .NET libraries, Excel integration, multi-threading and parallel programming and applications to fixed-income products such as caps, floors, swaps and swaptions that we price and for which we calculate rate sensitivities. We also discuss the pricing of equities using lattice and

PDE/finite difference methods. It is clear that this book is not just about C# syntax alone but it is a complete introduction to the design and implementation of C# applications for financial markets. We employ object-oriented, generic and functional programming models to create flexible and efficient software systems for finance professionals. The book has a dedicated forum at www.datasimfinancial.com.

We have provided source code at the above site for all examples and applications that are discussed in the book. We recommend that you review the code, run it and extend it to suit your particular circumstances.