INTRODUCTION AND CASE STUDIES

Reader’s Guide
This section introduces the book and the case study material that is used throughout it. Read the first chapter to get an overview of what the book is about. The second chapter gives an introduction to modelling and the Unified Modeling Language; some readers might like to skip that, but if you want to get a quick understanding of modelling notations it will be useful to read it. The case study material is very extensive. You may not want to read the third chapter in detail at the beginning, but you will need to refer to it repeatedly. A quick skim of the chapter is advisable.

Chapter 1: Introduction
This introduces the text, and the main themes of the book.

Chapter 2: Modelling and Notation – The Unified Modeling Language
This introduces the notion of modeling and notation are introduced for those readers unfamiliar with such concepts. The main notation used in this book, the Unified Modeling Language, is briefly introduced.

Chapter 3: Case Studies – ICANDO Oil
Five substantial case studies based on real applications in an oil company are introduced. These will be used extensively throughout the book.
CHAPTER 1

Introduction

IN THIS CHAPTER YOU WILL LEARN:
1. What the book is about, and the approach it takes to introducing software development
2. What the software development process is
3. The notions of modelling and notation
4. The concept of architecture in software development
5. The role and importance of organization in software development

1.1 Aims and Objectives – the Adventure Begins

You, the reader of this book, are likely to be an undergraduate at the outset of your study, or a software developer at the beginning of your career, or an IT manager or developer who wants to gain an overview of modern software development. In the preface, I introduced IT development as a great modern adventure, akin to the great engineering adventures of the past. This book is very much a ‘Cook’s Tour’, in that we shall visit all aspects of computer system development. However, we shall linger in each of the areas long enough, and give you sufficient pointers, for you to have a real feel of what a computer system development project is about.

The motivation is to provide you with a holistic view of computer development early on in your studies. Many books focus on a limited aspect, or provide over-emphasis on some aspect, or are just a hodge-podge of badly linked material. The aim here is to give you a sense of modern system development using methods, tools and techniques that fit together. It is, to a large extent, a selective view, but a comprehensive view of different methods and approaches would be far too extensive and overwhelming; this is a springboard for deeper and more thorough study later on.

If you are reading this out of general interest, you may follow the Hatter’s advice to Alice, and begin at the beginning, go through to the end and then stop. If you are reading this to support a course, you might want to delve into the later chapters, skipping some of the earlier ones. As far as possible, I have tried to make the chapters stand alone, though it gets harder as we move further into subjects such as analysis and design.

This book will introduce four major concepts. Figure 1.1 shows these concepts arranged as the structure of a house. The plot in which the development will take place is the organization that carries out the development. The foundation of a good development is the process by which software is developed. Architecture
defines the structure of the product that is being constructed. Notation is the means of recording communication and provides the roof under which the development takes place. The success of any development depends on all of these working in harmony. Though notation is one of the key concepts introduced by this book, notably the Unified Modeling Language (UML), this on its own is not sufficient. It must be used appropriately.

Along the way we will meet some important concepts. One of these is the notion of ‘object orientation’. At its simplest, this is very straightforward to understand, and it has had a huge impact on the way software is developed. Like many ideas though, it has gathered a collection of myths and folk tales. This is perhaps the most important base concept to understand, and it will underpin a lot of the understanding of the notation and process. However, object orientation is only one aspect of developing software, and it is possible to develop software without that concept. This book is not truly an ‘object-oriented’ book, though the basic concepts are founded in the notion of an object. Thus, if you are not expecting to develop in Java or some other object-oriented language, you should still find value in the book, and even in the UML notation, though that is based on object-oriented concepts.

The aim of this book is to make software development accessible and understandable as a complete, holistic endeavour. Along the way, we shall explore some of the best contemporary notations and guidelines. Whatever role you adopt in your career (and it is likely that you will adopt many), it is important to know how this fits into the process of constructing a software system. Understanding the breadth of a software development project makes it more meaningful and more exciting. If this book succeeds for you, you will gain a sense of that excitement and want to go on to your own adventures.

1.2 Organization

The basis for any development is the organization undertaking that development. Organizations vary in size, type and maturity. The larger the organization, the more formal the structure tends to be. Organizations are groupings of people, who mostly specialize in certain aspects of development. Large teams split up into roles, and these roles focus on different parts of the development process. Sometimes people adopt several roles in the development, but it is rare for an individual
to undertake all the tasks in software development. People usually have different skills and orientations, and are more suited for some roles than others. I am lucky to have adopted many roles, but certainly not all of them.

Organization is the key to success on major ventures. Edmund Hillary may have been the first to the top of Everest, but there were dozens of people setting up base camps and preparing the path for the final assault, and more people finding funding and supplying goods. Of course, that level of preparation is unnecessary for a fine Sunday afternoon jaunt up Helvellyn in northern England. The bigger the task, the greater the need for good organization. Organizations drive software development processes just as car manufacturers run factories. Mature organizations are able to reflect on their processes and improve them. Immature organizations often invent processes on the fly, and discard them with little reflection.

Development of substantial computer systems is a collective activity that requires a vast number of people with different skills. There are the obvious people, like the analysts and programmers that focus on the design and construction of the system. In addition, there are the senior managers who monitor the overall progress, the project managers who deal with day-to-day progress, the team leaders who organize the craftspeople, planners, testers, trainers, operators, technicians and help desk staff, and the end users themselves. They all have roles in the development of computer systems.

Good organization is something that requires experience and skill. Organizations develop through a range of maturity. Young organizations have to invent processes as they develop. They can be somewhat anarchic, but can be the most stimulating to work for. They are fleet of foot, but often short on memory and frequently reinvent processes. As an organization grows and matures, it begins to reflect on its processes, and change them more methodically. Organizations can also become geriatric, and unable to revise the way they operate.

Organization has a dramatic impact on the effectiveness of individuals. This is something that is difficult for novices to understand. Often there is an intense feeling of frustration in an organization when it seems to be behaving counter to the way an individual thinks is most effective. Again, experience is the great educator. However, some understanding of the typical organization and management of software development is invaluable for a developer. We shall therefore devote Chapter 6 to the way software developments are organized.

1.3 The Software Development Process

The construction of software is a process. It takes time, energy and money. Even the writing of a small program that takes one person a week to complete follows some process. Larger projects with tens or hundreds of people involved for many months or years follow a much more obvious and explicit process, or they would not succeed. One of the great problems of teaching IT is getting the learner to have some sense of the importance of process. Usually we teach only some part of the process, often as if it were the whole or at least the most important part. The greatest problem of education, in terms of giving realistic experiences to students, is that it focuses on individual performance, while great enterprises focus on teams and collective performance.
This book is structured along the lines of a process of construction of software. Sometimes this is known by the grand title of the ‘Software Development Life Cycle’. This is an unusual way of introducing some of the elements – most books focus on some part of the process, or start with some later part of the process. We shall explore software development in roughly the order it takes place. The most prominent process for software development is the Unified Process, and we shall discuss that a little along the way. There are, however, many more processes, and every organization that develops software has its own distinct way of carrying things out. There is no single way of constructing software, any more than there is one single way of manufacturing cars.

Unlike the manufacture of goods, such as electronic goods or cars, the process of constructing software is not easy to see or visualize. If you visited a car factory, you could follow the process from beginning to end, and see how each part of the process contributes to the finished product. You would see how metal sheets are pressed into body parts. The wiring harness could be followed through from the point where the strands of wire are pulled off reels to the point where it is bound together and connected to switches and devices in the car. After a while it would be possible for most people to see how the raw materials are transformed into the finished product, and have some idea of how each stage of the process works.

Walk into a software development project, and you see a large number of people peering at computer screens, organizing meetings and engaging in jargon-filled conversations. For the uninitiated, how computer systems are constructed is a complete mystery, just as the first visit to a jungle reveals nothing but greenery, insects and strange, frightening noises. It is a key purpose of this book to unfold this mystery. When you really examine what is going on, it is not that different from any other construction process, except that the tools and artefacts are not that visible.

Figure 1.2 shows a flow of work in the development of any software product. Although this is drawn as a staged process, each of the stages is likely to be running in parallel. For example, a business case may be updated many times during the development of software, as new factors come to light. Systems requirements for part of the system might still be under investigation as a part of the system already constructed is deployed. We shall be considering later how this development process can be organized.

The first stage in any software development is determining what people want and whether it is worth doing. All software development starts with some idea that a new system or a change to an existing system will make someone’s life better, or make some money. How those ideas arise depends very much on the organization or individual that desires the development. It might be the bright idea of a senior manager, or a suggestion from an office worker, or some clever sales operation. We will look at some typical examples of how projects arise.

First and foremost in software development is the construction of a business case or some other justification for the development. This can be very formal, and large systems development needs a very rigorous process that determines costs and benefits up front and tracks them carefully throughout the whole of the development process. Smaller systems can be handled much more informally, and the smallest systems are often developed with little formal justification. Whether explicit or not, some decision is made to start the development, and some benefit in the project anticipated.
Arguably most of the major tragedies in software development are because this process of justification, monitoring and review is weak. Too many projects overestimate the benefits and underestimate the costs and risks involved. In Chapter 4 we will look at project inception, and how this can be organized to minimize the risks and maximize the chances of success. The continual monitoring of a project throughout its activity is vitally important.

Once the principle of developing some software has been agreed, it is important to understand the context in which the system is to operate. In this book, we will make this an explicit step. We will introduce some UML notation to allow us to describe the services provided by a business, and how those services are provided. Impatient software developers miss this stage at their peril. Software always implies some change in a business operation, or even the creation of a new business operation. The notation we will introduce will permit explicit modelling of the business environment in which the system will fit. These notations will be common to some of those used in the description of the system itself.

With a purpose for the system and a clear understanding of the business environment in which it will fit, the development team is ready to define clearly how the system will meet the goals and fit into the business environment. The core concept we will introduce is known as a ‘use case’. A use case is a meaningful unit of work carried out by a computer system. Through the identification and definition of use cases, we determine what the proposed system will do and how it will be used. Use cases have become central to object-oriented development methods, as they represent the core unit of delivery – use cases are discrete pieces of functionality.
The modelling of a business and the definition of system behaviour is often termed ‘requirements analysis’. This is often thought of as a discrete phase. In practice, it is something that usually takes place continually, as expectations and understanding grow and change, and the business environment into which the system will be introduced evolves.

Once a clear understanding of the desired behaviour of a system is established, the definition of the operation of the system begins in earnest. Now the detail of the system behaviour needs to be filled in. We begin to introduce notations that are much more computer-specific and that begin to be more precise. This is known as ‘systems analysis’. Considerable detail is gathered on the way the system operates and interacts with the environment. We will meet the notion of an object and a number of diagrams that allow us to model the behaviour of the system and its interactions. Here the Unified Modeling Language comes into play in earnest.

Once the behaviour of the system is defined fully, this needs to be transformed into a design that can ultimately be implemented. The design stage is where the physical implementation of the system in terms of computers, networks and interfaces is first considered. Considerable skill goes into the design of good systems. Good design is supported by a good architecture. Just as in the construction of large buildings, there is the need to have a good overall plan and structure to a system, and this is known as the architecture of the system.

From the design, a system is constructed. This is the phase that most people think of when they consider software development. It involves writing computer programs and defining interfaces such as computer screens or Web pages. Although it is a considerable and complex activity that needs substantial skills, it is not the whole process. In fact, in terms of overall cost, the actual programming of a system can be relatively small.

After construction comes testing. Once it is built, does it do what it is supposed to do? Is it reliable and usable? Like hostile invaders, a testing team will push the system until it breaks. Testing needs to be linked back to the requirements capture, and we will look at how to transform the use case definitions and business models into test plans. Testing is something of a Cinderella subject. Yet it is as vital to success as good analysis, design and programming.

Then, when the system is all working sufficiently well to be used, the system needs to be deployed. This involves installing the software, training people in its use, and supporting them in day-to-day use. A huge organization is often required to undertake this. Keeping hundreds of users happy, fixing problems and making sure that they use the system safely and securely is an endless task.

Once the system is operational, issues will arise. There will inevitably be problems with the software. It may stop working, and some urgent action will be needed to correct the problem. It may do something wrong, and part of the system may need to be changed. Someone may want the system to do more or do something differently, and these suggestions need to be collected and considered. The majority of the cost of a system is keeping it live and operational. This is known as software maintenance, or more fashionably today ‘software evolution’. Unlike mechanical devices, software does not wear out, but it does present errors caused by faults (known as bugs), and it needs to keep pace with changes in business needs.

As you can see, the journey from a bright idea to something that works and is useful is a long and complicated process. For small systems, the process is
shortened, and can be very informal. For big systems, it has to be explicit and formal. Different organizations will provide a different emphasis on each part of the process. However, the nature of the product means that the stages from conception through to support and enhancement must be gone through.

This looks like a staged process. In fact, the phases usually overlap. A large project needs to keep on track and adjust to changes in the environment, so the justification stage is revisited regularly. The analysis and design of part of the system can take place alongside the requirements definition for another part. More often than not, systems are developed incrementally, and are often deployed incrementally (that is, a bit at a time).

The most prominent definition of a software development process is the Unified Software Development Process (Jacobson et al., 1999). This is a comprehensive approach to the development of large-scale software, and advanced students are advised to study this in detail. This book is loosely aligned to this process, and we will refer to it from time to time. There will not, however, be scope to consider it in detail.

Other software development processes exist. A development process that has gained ascendancy in recent years is Extreme Programming (Beck, 1999). This seeks to reduce the amount of intermediate modelling. It folds systems analysis, design and construction together, and has explicit processes for managing requirements and testing. Other methods, such as Rapid Applications Development (RAD) and prototyping approaches, have similar goals. Each of these has different emphases. There is no absolute right or wrong, and approaches will vary in success, depending on the environment and scope of a project and the skill of the practitioners.

However software is developed, it follows some process. Mature organizations have well-developed and finely tuned processes. Novice organizations and those that are immature will have processes that are obscure and are often invented as a system is developed. At the end of the book, you will have assimilated the outline of a process and be better placed to judge or define a process yourself.

1.4 Architecture and Design

Architecture defines the broad structure of the system to be developed. For small systems, this is often informal. The best analogy is building architecture. A garden shed or tree house does not need much architectural planning. A house needs some planning before it is constructed. A tower block needs careful consideration of the architecture, or the craftsmen will get nowhere as they try to construct the building. There are some classical architectural forms that are emerging for software development, and we shall look at these. Architecture is a skill learnt over time. In this book we can only give a flavour of what architecture is about.

Architecture is an increasingly important concept in the development of systems. It is a hard subject to grasp, and a good architect has many years of experience, ideally in different roles. It will be hard to give much more than a fleeting glimpse of architecture in this book. Fortunately for most developers, early in their careers, architectures are already defined for them. However, they must understand and apply the local architectural models as they construct their parts for the whole system and integrate them.
Architecture at the gross level considers the computers and networks that are used to construct the overall system. There is also a definition of the software tools that are used to construct the system, such as compilers, modelling tools, databases and packages. These decisions need to be made early on in a project. Often they are determined by the environment that an organization already has. This is known as the technical architecture.

Within the technical architecture, the software can be broken down into major components that cover items such as the user interface (computer screens and the like), detailed processing, and storage. These need to be distributed over the technical architecture. This is known as the application architecture. Major technologies, such as the Internet, have a distinct impact on the application architecture.

Good architecture is a craft that is learnt slowly. Building architects undertake many years of training, and require long experience before they become masters. The same is true of software architects. The only way to learn is to do. This book cannot hope to do much more than introduce the importance of architecture and indicate how it fits into the overall software development process.

Design involves the gross details of architecture and the fine detail of how individual components of a system are constructed. We will be following an object-oriented approach to design and using the different notations in the Unified Modeling Language to produce detailed designs. As with architecture, good design comes with practice and experience. There are many guiding principles to design, and much to study beyond this book. By the end of the section on architecture and design, you should have a systematic means of approaching design that can reliably lead to implementations that are effective and maintainable.

1.5 Notation and the Unified Modeling Language

Large projects involve a great deal of communication. The more people that are involved, the more complex that communication becomes. Notation is nothing more than a convention for the way we write down ideas, descriptions and designs. It is important, as far as possible, that the members of a project team share a common notation wherever possible. A good notation clarifies communication and helps to structure the thinking in a group.

The notation that we will use in this book is the Unified Modeling Language, usually known as UML. This has become the most widespread standard language in the IT industry for recording analysis and design. This book introduces the language at the point of use, and covers the major features of the language. For those wanting a more detailed and comprehensive description of the language, a book such as Bennett et al. (2001) is recommended.

The software development process involves the construction of a series of models, starting with simple descriptions of need and the way the physical environment of the system works, through to the construction of models that can actually carry out work (known as computer programs). Instead of introducing notation in isolation, we will meet the notation at the point in the software development process where it is needed.

Probably the greatest problem in terms of notation is knowing where and when to use it, not how to use it. On the whole, notations are easy to understand, but their application is often puzzling. Notation is about communication, and that
communication needs to be concise, accurate and appropriate – knowing all the words in a dictionary will not make you a great novelist. Notations such as the UML are the primary means of recording and communicating in an IT project. Of course, everything could be recorded in a natural language such as English, but most disciplines, such as architecture or engineering, have special languages. Notation allows for more precise exchange of information.

The UML is a diagrammatic notation with some textual additions. This helps in the visualization of the structure of a system. Most modern methods use some visual notation – either the UML or something similar. The UML is based on object-oriented concepts, which are generally accepted as best practice in the development of IT systems. Given the importance of this notation, there is a brief introductory chapter following this one on object orientation and the UML.

1.6 Case Studies

Guides to adventures are best supported by realistic tales. We shall therefore be looking at five case studies of oil industry applications. These are based on my experiences. They have been chosen to represent different sizes and orientations of project. One will look at a large-scale sales order processing system, which, on the face of it, is not very exciting, but in fact is the type of project that can give the most benefit to large organizations. The second is a research and development project that is trying to apply the latest techniques to solving a difficult engineering safety problem. The third looks at an Internet development for retail promotions, which is very much in tune with many contemporary developments. The fourth considers a system for handling retail outlet transactions that involves linking existing systems. The fifth is an oil trading application, akin to many financial trading applications such as commodities or shares.

Each chapter illustrates the concepts with examples drawn from the case studies. There are also exercises for you to undertake. It is largely through doing rather than abstract theorizing that the fundamentals of software development are learnt. There is also a fully worked example in the final chapter that I have used extensively with my students. Experience shows that a variety of case study material best illustrates a complex set of ideas.

1.7 Conclusion

Software Development is a complex and time-consuming activity that involves organization, process, architecture and notation. Software developers need an appreciation of all these aspects in order to contribute to large-scale IT development projects, and to develop expertise in some aspect of IT development.

The aim of this introduction has been to quickly give you a comprehensive overview of the book and the approach to software development that it describes. The rest of the book is opening up the detail and introducing the principles and practice of software development.

At the outset, software development was described as a great adventure. If this introduction has given you some sense of the sweep of that adventure, and a kindergarten picture of some of the creatures and landscape that lie ahead, then it
has been successful. Now we shall delve deeper into the forest, and see some of the creatures close up.

**REVIEW QUESTIONS**

1. What are the four major concepts that this book is aiming to introduce?
2. What does UML stand for?
3. What are the stages in the development of software?