Software engineering: theory and application

W. Venters

IS3 139, 2790 139

2011

Undergraduate study in Economics, Management, Finance and the Social Sciences

This is an extract from a subject guide for an undergraduate course offered as part of the University of London International Programmes in Economics, Management, Finance and the Social Sciences. Materials for these programmes are developed by academics at the London School of Economics and Political Science (LSE).

For more information, see: www.londoninternational.ac.uk
This guide was prepared for the University of London International Programmes by:
Dr Will Venters, Lecturer, Information Systems and Innovations Group, Department of
Management, London School of Economics and Political Science

This is one of a series of subject guides published by the University. We regret that due to
pressure of work the author is unable to enter into any correspondence relating to, or arising
from, the guide. If you have any comments on this subject guide, favourable or unfavourable,
please use the form at the back of this guide.

University of London International Programmes
Publications Office
Stewart House
32 Russell Square
London WC1B 5DN
United Kingdom
Website: www.londoninternational.ac.uk
Published by: University of London
© University of London 2009
Reprinted with minor revisions 2011

The University of London asserts copyright over all material in this subject guide except where
otherwise indicated. All rights reserved. No part of this work may be reproduced in any form,
or by any means, without permission in writing from the publisher.

We make every effort to contact copyright holders. If you think we have inadvertently used
your copyright material, please let us know.
<table>
<thead>
<tr>
<th>Reminder of learning outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample examination question</td>
</tr>
<tr>
<td>Part 3: The project</td>
</tr>
<tr>
<td>Chapter 16: The project</td>
</tr>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>Overview</td>
</tr>
<tr>
<td>The project as a development process</td>
</tr>
<tr>
<td>Choosing your development tool</td>
</tr>
<tr>
<td>Reuse</td>
</tr>
<tr>
<td>Suggested projects</td>
</tr>
<tr>
<td>Deliverables</td>
</tr>
<tr>
<td>Appendix 1: Sample examination paper</td>
</tr>
<tr>
<td>Appendix 2: Advice on answering examination questions</td>
</tr>
</tbody>
</table>
Notes
Introduction

Aims

This 300 course covers the methods, values, attitudes and techniques in developing complex software systems.

It is intended to provide an understanding of the need for rigour, and enable you to select and apply a relevant methodological approach to the development of well-designed and documented systems.

The main aims of this course are to:

• introduce you to the overarching concerns of software engineering practice
• provide an understanding of the various processes software engineers may employ in developing software and to critically assess their value for different types of software engineering problem
• develop an understanding of the tools and techniques employed in contemporary software engineering
• provide you with an experience of the way techniques are applied in practice (this is achieved through the individual project work)
• develop the capacity to identify relevant approaches to software engineering.

Learning outcomes

At the end of this course and having completed the Essential reading and activities, you should be able to:

• describe the expectations, pressures and problems faced in developing software and the need for processes, tools, techniques and approaches
• outline the underlying processes of software engineering and critically assess relevant approaches
• analyse, design, test and maintain software systems and document these actions correctly.

Course outline

This syllabus covers the methods, attitudes and values that underlie professional contemporary software systems development. The emphasis is on how to undertake software development through requirements specification, design and implementation, but with significant emphasis on other processes for developing software.

The guide is split into two parts. The first part focuses on the processes that are employed in developing software. The second part concentrates on the practices of undertaking software engineering within the processes introduced in Part 1.

Syllabus

If you are taking this course as part of a BSc degree you need to have studied 62 Information systems development and management and 138 Information and communication technologies: principles and perspectives before this course may be attempted.
This syllabus covers the methods, attitudes and values which underlie professional contemporary software systems development. The emphasis is on how to undertake formal software development through requirements specification, design and implementation, but within a broader understanding of software engineering practices.

Section 1: Software engineering process
- The changing pressures on software engineering practices: History of the field, definition of software, the software crisis
- The process for developing software and its importance
- The traditional software engineering process: The lifecycle model, evolutionary software development, incremental software development, spiral model. Prototyping
- Rapid software development
- Internet speed web-based application development
- End-user development
- Agile methods
- Extreme Programming
- Refactoring

Section 2: The practices of software engineering
- Introduction to structured v object-oriented paradigms
- Acquiring requirements
- Specifying requirements and design (both structured and object-oriented)
  - Structured approaches: ER diagram, data flow diagrams, data dictionary
  - OO approach (Using UML): use-case diagrams, class diagrams, object sequence diagrams, state-chart diagrams
- Features of good design
- Coding and configuration management
- Implementation and testing (both structured and object-oriented)
  - Choice of programming languages and techniques
  - Test planning
  - White-box and black-box testing
  - Testing automation
  - Implementation
- Maintenance and software evolution
  - Systems re-engineering for legacy systems
- Reuse
  - Reasons for reuse
  - Concept reuse - patterns, configurable systems products and program generators
- Component-based software engineering
- Computer-aided software engineering (CASE) tools
- Documentation and help system
• Project management in software engineering
• Managing software engineering projects
• The capability maturity model

Access to computers

It is a requirement for this course that you undertake a software engineering project. **You must send a printed copy of your coursework/project to the University of London and submit your coursework online via the virtual learning environment (VLE).** Therefore, the structure of the subject guide is built around the assumption that you have good access to a modern computer with a word processor and a suitable programming environment, as described in the project section of this guide. As of September 2009, students are required to have access to the internet to aid advanced study and to download and try various CASE tools.

Reading

Essential reading

There is no one textbook that adequately covers the entire course. The Essential readings for this course are:


Both these books are very much within the field of computer science and focus on the technical steps of producing software with less emphasis on the purpose of software with respect of complex business practices and difficult, messy and poorly understood users. While both acknowledge the important role of users (and with each new edition this acknowledgement increases) you should read these as manuals on how to do software engineering, rather than books discussing the complexity of software engineering in practice. The subject guide will help you understand this, as will the further readings.

Ian Sommerville’s work focuses heavily on safety-critical systems (for example the software controlling an aeroplane) and as such the book includes a large amount about ensuring software works correctly. This work is beyond the scope of this course (where we focus on software engineering for information systems projects). It does, however, mean that many chapters include sections on safety critical systems which, while you should perhaps read for interest, you can ignore for the examination. Sommerville is from the UK and the book has a more English and critical flavour to it than Pressman. Pressman on the other hand is American and his textbook is a little easier to follow on the details such as the modelling of software in UML.

The following are also listed as Essential reading for individual chapters:

Detailed reading references in this subject guide refer to the editions of the set textbooks listed above. New editions of one or more of these textbooks may have been published by the time you study this course. You can use a more recent edition of any of the books; use the detailed chapter and section headings and the index to identify relevant readings. Also check the VLE regularly for updated guidance on readings.

Further reading

You are also expected to read widely including specialist books on topics (which are listed in the subject guide). If you have limited resources but wish to purchase one book to help you in your Further reading then I suggest you purchase:


Unlike Sommerville and Pressman this book provides a more critical account of systems development practices and is a useful addition as it covers most of this course. Other books in the list tend to be more specific to a particular topic or interest.

Please note that as long as you read the Essential reading you are then free to read around the subject area in any text, paper or online resource. You will need to support your learning by reading as widely as possible and by thinking about how these principles apply in the real world. To help you read extensively, you have free access to the VLE and University of London Online Library (see below).

Other useful texts for this course include:

**Books**


Gamma, E., R. Helm et al. Design Patterns: Elements of Reusable Object-Oriented Software. (Reading Mass.: Addison Wesley, 1995) [ISBN 9780201633610].


Journals and internet resources


Challenging reading

Some chapters refer those of you who are interested to reading which is beyond the scope of the course. This is in no way required but purely for those who may wish to read further.


For details on refactoring see www.refactoring.com

Online study resources

In addition to the subject guide and the Essential reading, it is crucial that you take advantage of the study resources that are available online for this course, including the VLE and the Online Library.

You can access the VLE, the Online Library and your University of London email account via the Student Portal at:
http://my.londoninternational.ac.uk

You should have received your login details for the Student Portal with your official offer, which was emailed to the address that you gave on your application form. You have probably already logged in to the Student Portal in order to register! As soon as you registered, you will automatically have been granted access to the VLE, Online Library and your fully functional University of London email account.

If you forget your login details at any point, please email uolia.support@london.ac.uk quoting your student number.
The VLE

The VLE, which complements this subject guide, has been designed to enhance your learning experience, providing additional support and a sense of community. It forms an important part of your study experience with the University of London and you should access it regularly.

The VLE provides a range of resources for EMFSS courses:

- Self-testing activities: Doing these allows you to test your own understanding of subject material.
- Electronic study materials: The printed materials that you receive from the University of London are available to download, including updated reading lists and references.
- Past examination papers and Examiners’ commentaries: These provide advice on how each examination question might best be answered.
- A student discussion forum: This is an open space for you to discuss interests and experiences, seek support from your peers, work collaboratively to solve problems and discuss subject material.
- Videos: There are recorded academic introductions to the subject, interviews and debates and, for some courses, audio-visual tutorials and conclusions.
- Recorded lectures: For some courses, where appropriate, the sessions from previous years’ Study Weekends have been recorded and made available.
- Study skills: Expert advice on preparing for examinations and developing your digital literacy skills.
- Feedback forms.

Some of these resources are available for certain courses only, but we are expanding our provision all the time and you should check the VLE regularly for updates.

Making use of the Online Library

The Online Library contains a huge array of journal articles and other resources to help you read widely and extensively.

To access the majority of resources via the Online Library you will either need to use your University of London Student Portal login details, or you will be required to register and use an Athens login: [http://tinyurl.com/ollathens](http://tinyurl.com/ollathens)

The easiest way to locate relevant content and journal articles in the Online Library is to use the Summon search engine.

If you are having trouble finding an article listed in a reading list, try removing any punctuation from the title, such as single quotation marks, question marks and colons.

For further advice, please see the online help pages: [www.external.shl.lon.ac.uk/summon/about.php](http://www.external.shl.lon.ac.uk/summon/about.php)
Additional resources

You should also keep track of current trends in software engineering by reading relevant journals, newspapers and periodicals. Good examples include:

- Computing, www.computing.co.uk/
- Silicon.com, www.silicon.com

Wikipedia is a good place to find extensive details on software engineering topics – however be aware that it is often biased, strongly in favour of a method, or strongly against it: http://en.wikipedia.org/wiki/Main_Page

You might also like to look at some more frivolous websites that give you a flavour of professional software engineering practices:

- http://slashdot.org/
- www.dilbert.com/

Unless otherwise stated, all websites in this subject guide were accessed in April 2011. We cannot guarantee, however, that they will stay current and you may need to perform an internet search to find the relevant pages.

The examination

Important: the information and advice given here are based on the examination structure used at the time this guide was written. Please note that subject guides may be used for several years. Because of this we strongly advise you to always check both the current Regulations for relevant information about the examination, and the VLE where you should be advised of any forthcoming changes. You should also carefully check the rubric/instructions on the paper you actually sit and follow those instructions.

Format

For this subject you will have to sit an examination which counts for 40 per cent of your final mark. The examination will be three hours long and consist of seven questions of which you must answer four.

A Sample examination paper can be found at the end of this guide.

Advice

When you sit the exam it is important that you read a question very carefully before you attempt to answer it. In the past students have lost marks because they failed to address the concerns of the question or parts of it in their answers. There is a danger of picking up the ‘central theme’ of the question and presenting everything that comes to mind regarding this ‘theme’ without paying sufficient attention to the exact details that are being asked. It is worth spending some time reading and understanding the question thoroughly rather than spotting a key word and launching into a ‘pre-programmed’ response that does not answer the question.

Sometimes questions will ask you to give examples or to recall how you approached certain stages within your project. Make sure you do so. Although it is important that you have understood the theory, these kinds of questions are often intended to test whether you can also apply your knowledge in practice. Also, the exam gives you an opportunity to show the Examiners the extent to which you have read the literature. Try to show some originality, such as evidence of having read beyond one standard textbook.
Remember also that the topics in this course cannot be studied in isolation, and that exam questions may ask you about the linkages between topics (for example to compare and contrast software engineering processes).

Finally, you are expected to present your answers in the form of an essay written in correct and legible English.

Remember, it is important to check the VLE for:

- up-to-date information on examination and assessment arrangements for this course
- where available, past examination papers and Examiners’ commentaries for the course which give advice on how each question might best be answered.

**Coursework**

You will undertake a project for the remaining 60 per cent of marks. This will require you to review a software engineering approach or technique, apply this to a real problem and finally reflect on the experience. The project will require you to write a short essay describing the approach you have used and its relevance to the problem chosen. You will then present an account of how the approach was applied in practice. This should include the relevant documentary material required for the chosen approach. Finally you will need to reflect on the practice of developing the system from the approach you have chosen. This should include lessons learned and critical reflections on the process. A bibliography must also be provided demonstrating reading beyond the core textbooks.

You are not required to produce large amounts of programming code, but rather are assessed on your attempt to apply software engineering techniques and principles in practice. The focus is not on the produced system but on the quality of the process undertaken, the coherence of the documents presented and how successful the documents would be in assisting somebody else in developing the software system. Clearly, however, some of your programming will be an integral part of this exercise.

Further details of the project can be found in Part 3 of this guide and in the *Completing and submitting coursework and projects* booklet.
Chapter 1: The changing pressures on software engineering

Learning outcomes

By the end of the chapter, and the relevant readings and activities, you should be able to:

- describe the changing nature of software engineering practice
- explain the need for a software engineering discipline.

Essential reading


Further reading


Introduction

This chapter introduces software engineering as a discipline, describes its history and demonstrates why its practices remain crucial for software engineers today.

Software engineering

The increasingly widespread dependence on software applications calls for highly reliable software products. However, levels of satisfaction with software, whether in terms of its reliability, safety, quality, cost, time required for production or even delivery, are far below those found for other products such as buildings or cars. Software is a logical rather than a physical product and its intangible nature puts different demands on the software engineering process. Also, software development has to keep up with the rapid advancement of computer technology and, compared to early software applications, today's software is used by non-expert users, is more complex, larger and developed by teams.

In order to cope with these developments, better software engineering practices are being adopted. Software engineering is defined as ‘an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use’. The overall aim of software engineering is to produce software on time and to a budget that satisfies the requirements of the users of the software application in a changing environment, by providing a safe and efficient solution to the problem at hand. Although such an ideal is probably impossible to achieve in practice, software engineering helps to minimise the likelihood of common problems associated with software projects.
What is software engineering?

History and overview

The first serious software applications were developed in the 1950s. They were developed by computer specialists for use by data processing professionals. Because the computer systems had little memory or computing power, the programs they ran tended to be written by hand in machine-code or very low-level languages, and were often written from scratch. By the 1960s computers, and the applications they were expected to run, had become more complex. It was therefore necessary to think about organising and managing the process of developing software. Since the people writing software were usually engineers (often electronics engineers), they employed the methods they had learnt in their engineering disciplines to help them to write software. They also employed methods from management in order to map the way the software might work (for example flowcharts predate the development of the computer, even though we often associate them with computers today).

In the 1970s, however, computers began to be used by a wider variety of lay users; further, the reduction in the price of computers promoted their use in a wider variety of ways, and hence led to a wider variety of software. The increase in power together with the reduction of costs of computers promoted the emergence of a wide and diverse range of software applications. In the early 1970s computer specialists drew on engineering approaches further to aid them in developing software. In particular they adopted processes such as the lifecycle approach to systems development, as it is similar to other engineering approaches (e.g. designing and constructing cars, houses, or aeroplanes).

The 1980s saw the emergence of a range of methodologies specifically designed to aid systems development and software engineering. Since the 1990s, however, some authors have noted a shift away from such prescriptive methodologies towards what has been termed the post-methodology era (Avison and Fitzgerald, 2008). This has seen the reappraisal of the prescriptive methodologies role in systems development in favour of more contingent, flexible approaches. (Examples of these are rapid application development, CASE-based approaches, incremental and iterative approaches and agile methods, which we will discuss in detail in Chapters 3, 4 and 13). It is, however, worth remembering that the older approaches are not ‘out of date’ – they are still widely used for traditional software engineering projects. Rather, new forms of software engineering projects have emerged for which the older approaches prove problematic.

Alongside these various processes for developing software, there was rapid development of new practices for developing software, including ways of analysing users' problems, ways of modelling these problems, ways of writing code and ways of testing systems to make sure that they work as they are intended to.

How we will study software engineering

In the previous section the history of software engineering was briefly described, and we saw that this involved the development of a range of processes for managing the development of software, and various practices undertaken as part of these processes. In order to help us study software engineering we will therefore focus on my 4Ps of software engineering.
• **Processes**: These are the methodological processes which people will follow in order to move from an idea or problem, through to a completed piece of software in use by people. Processes involve various people undertaking a range of different software practices.

• **People**: There are many different types of people involved in a software engineering project. The common types we will use in this guide are **software engineers** (who undertake the work to develop systems, and are also sometimes called developers), **users** (who interact with the software in some way) and **customers** (who take decisions about the software's design). You might notice that users are often also customers. We might however also consider other people such as maintenance staff (looking after software in the future), lawyers (drawing up contracts for software development projects), testers, programmers, accountants, technical authors (writing manuals)...

• **Practices**: These are the specific activities a software engineer undertakes in order to develop a system. For example, interviewing users to find out their requirements is a practice.

• **Paradigms**: Paradigm is somewhat different from the other 3Ps and refers to a set of practices which are linked together around a set of beliefs about the way we should develop software. Within this guide we will discuss two paradigms – the **object-oriented approach** and the **structure systems analysis approach**. The object-oriented approach consists of practices which are linked around the idea that software should be constructed as sets of interacting objects. The structure systems analysis approach consists of practices which are linked around the idea that we should consider the word as functions operating on data-structures.

The subject guide is thus structured in three parts. The first part of the guide introduces the **process** of software development, the second part teaches the **practices** used within these processes. In doing this it employs the lifecycle model in order to structure the teaching. Part 3 of the guide introduces the **project** you must undertake as part of this course – which is your chance to try out a process and set of practices you learn.

---

**Important note on studying software engineering**

Some students with less knowledge of IT may prefer simply to read quickly through the first part of the guide, and then concentrate on learning the practices introduced in the second part of the guide, finally returning to study the first part of the guide in detail once more later on.

Others will want to follow the guide as it is written, so learning the practices of software engineering in the context of the processes in which they are used.

You are free to choose between either approach and the guide is written to support both.

---

**Activity**

You have been called in by a car manufacturing company to develop a computer-aided manufacturing (CAM) system for their particular needs. The manager of the company does not understand why the development of such a system is a complex undertaking. Explain to him, using the 4Ps above (Process, Paradigm, People and Practices), why the development of such software is different from the manufacturing of cars.
A reminder of your learning outcomes

By the end of the chapter, and the relevant readings and activities, you should be able to:

• describe the changing nature of software engineering practice
• explain the need for a software engineering discipline.
Part 1: Software engineering processes
Chapter 2: The process for developing software

Learning outcomes
By the end of the chapter, and the relevant readings and activities, you should be able to:
• outline the variety of software engineering processes
• discuss the role of process within software engineering.

Essential reading

Further reading

Introduction
The previous chapter introduced the changing nature of the software engineering process. This chapter will describe the need for a process for developing software systems, in particular contrasting prescriptive, tightly managed approaches with newer, more flexible and contingent approaches.

Why is a process important?
Within the 62 Information systems development and management course you will have encountered information systems development and management concerns within the larger context of the organisation in which the system is implemented. Within this subject guide, however, we are not concerned with the wider organisational issues, such as problem identification and feasibility studies. Rather the central concern of software engineering is the process of software development once the need for the software system has been identified.

The software engineering process involves the use of a number of different approaches, methods and techniques which support the manageability and the success of software development. There are a large number of different software development approaches (see Table 2.1 for a small sample of those available). The most common ones include the classic life cycle, evolutionary development approaches such as prototyping, and reuse-oriented approaches (Sommerville, 2007).
In the following chapters we are going to discuss a range of approaches to software engineering. It is important however to realise that all of these are idealised views of software engineering – it is unlikely (perhaps impossible) that a project will follow one of these methods precisely. Indeed, you will probably adopt one of these approaches to guide your own software engineering project, and yet you will quickly discover that you have to break away from the formal rules, that you will do some things earlier than expected, others later. This is not unusual in professional software engineering either. You will notice that some of the approaches you are presented with expect the software engineer to follow their methodology quite precisely (for example the life cycle and spiral model) and are founded in the belief that it is possible to construct rationally a model of the problematic situation (as a documented specification), then follow a controlled process through which the software solution emerges. In contrast, other approaches are much softer, believing that the world is constantly changing and that in software engineering creativity, imagination and improvisation are crucial, and that the best a methodology can do to support this is to provide a set of good practices and nice ideas which might help the developer respond to this changing world better (agile methods like Extreme Programming are examples of this).

It is also worth remembering that some software development projects (especially larger ones) may use a range of different development approaches throughout their development. Also, these projects might require the iteration of particular development phases, as the system requirements may change over time.

The development approaches are introduced in this part of the guide, with the next part of the guide introducing the various methods that are employed within these approaches (e.g. data modelling, or object-oriented modelling). A method, in turn, may use a number of techniques. For example, entity-relationship models are used within data modelling. You will be familiar with some of these techniques from 136 Information systems and organisations, while others will be very new. The main difference, however, will be that you will apply these tools as part of the overall process of software development rather than in isolation.
A reminder of your learning outcomes

By the end of the chapter, and the relevant readings and activities, you should be able to:

• outline the variety of software engineering processes
• discuss the role of process within software engineering.
Chapter 3: Software engineering processes

Learning outcomes

By the end of the chapter, and the relevant readings and activities, you should be able to describe the following software engineering processes and their advantages and disadvantages:

• the classic waterfall model
• evolutionary software development
• prototyping
• Boehm’s spiral model
• incremental development
• rapid application development
• internet speed development.

Essential reading


Sommerville, I. Software Engineering. (Wokingham: Addison Wesley, 2007) Chapters 2, 4 and 17

Further reading


Beynon-Davis, H. Mackay and D. Tudhope “‘It’s lots of bits of paper and ticks and post-it notes and things...’: a case study of a rapid application development project’, Information Systems Journal 10(3) 2000, pp.195-216. This article outlines how RAD works in practice.


Brooks, F.P. The Mythical Man-Month. (Reading, Mass.: Addison Wesley, 1995) Read the early parts of this excellent book – but also make sure you read Chapter 19 – reflecting on 20 years of this book.


Challenging reading

The following may be of interest but is beyond the scope of the course.

**Introduction**

In the previous chapter we discussed the role of process within software engineering. Today a wide range of systems development processes exist, each with different approaches and philosophies. Within this chapter, and the next, you will be introduced to a range of different development approaches which are currently in use.

**The classic life cycle**

The oldest and most widely used approach in software engineering is the classic life cycle (sometimes referred to as the waterfall model). Within the classic life cycle the software engineering process follows a number of sequential phases. These generally include the analysis of the requirements, the design, the coding, the testing, the implementation and the maintenance of the software system. Each phase has to be completed before the subsequent phase can commence, and each stage involves the production of documents detailing what is to be undertaken (and meaning a different group of people can undertake each stage). This approach to systems development is very similar to that found in other engineering disciplines. For example in the construction industry, customers’ requirements for a building are usually established before the architect begins designing. The design blueprints must be completed before any construction begins, and the building must be fully built and finished before it starts to be occupied (used). Although the strictly sequential approach has disadvantages, it still represents the most popular approach to software development. One of the advantages of this approach to software engineers is that it attempts to document and agree the specification in advance of development. This allows contracts to be drawn up for the work being undertaken.

**Activity**

Discuss how the production of computer software differs from the construction of a building. From this discussion, identify benefits and weaknesses of the classic life cycle for developing computer software.

**Contemporary software engineering**

While we can criticise the life cycle approach in many ways, it remains an important and widely used approach. It is particularly relevant to problems where the requirements are easy to specify precisely, and where those requirements are unlikely to change (for example writing software to control an elevator mechanism).

Today’s organisations change rapidly, and they usually use ICT as a fundamental part of the way they work. This means that requirements are usually very difficult to specify as they involve lots of people, systems and components, and they are interconnected with other systems. Add to this the fact that the organisation will change while the system is being developed and you quickly realise why the life cycle model is increasingly problematic.

The rest of this chapter, and the next, discuss approaches to systems development which respond to these pressures in contemporary organisations and to the overheads associated with ‘heavyweight’ software development methods.

In reading this chapter we must remain aware that, in Fred Brooks’s words, ‘there is no silver bullet’ for software development. By that he means that there does not (and cannot) exist a single approach which is
perfect for developing all software. We need to make difficult decisions about which approach is most appropriate for which type of software. This chapter presents some alternatives to the waterfall which are useful when a software engineering project faces a high degree of uncertainty, either regarding the requirements for the software (so called business uncertainty), or uncertainty about the process of developing the software (so called development uncertainty). The following fictional story provides an account of an organisation facing a high degree of development and business uncertainty. It provides an example of why a company might need to adopt different software development practices. As you read through this example and the rest of the chapter think about the advantages and disadvantages of the lifecycle in comparison with new approaches. In particular think about the advantages and disadvantages of each approach for the following people:

- customers of the software
- project managers managing the software development project
- software engineers
- lawyers drawing up contracts for those involved in the development
- future maintenance staff maintaining software.

Example story: problems at Alpha Corporation

Alpha Corporation is based in Chicago and is responsible for producing trays of meals for the airline industry. With the largest airport in the USA to serve, this is big business and they realised early on that computer technology would be central to their success. Their in-house development team began developing a simple scheduling system in the 1970s to ensure that the trays were ready and waiting for the aeroplanes to arrive. While quite complex, this system was essentially operated by one person (the manager) who entered the details of flights coming in to the airport, and the number of food trays required. The system would then print out scheduling lists to be handed to the workers on the production line, and data-tapes for the finance department to bill the airline. In developing the system they had followed the life cycle, using structured systems analysis (entity relationship diagrams and data flow diagrams) to model the data flows, and constructing data-base tables from their ER diagram. The system had taken two years to develop and had been well received when launched.

Today Alpha’s operation has changed and it is trying to keep up by developing a new computer system to handle its orders. The airlines are highly automated and expect Alpha to plug into their computer systems. Messages for orders are sent electronically in a variety of differing formats (depending on the individual airline’s system) which need translating into Alpha’s order formats. The food trays are no longer standard for all airlines, but must be tailored to specific requirements (different coloured trays, cutlery etc). For business class the larger airlines are now demanding trays produced for each individual customer, so that the customer can select their choice of food from a menu. This requires that each tray be barcoded, and that the production line be equipped with screens telling workers what food is required on each tray. This also means that the ingredients need to be ordered on a per-tray basis and that the computer system interacts with the food wholesalers in order that the ingredients are delivered ‘just in time’ so that they remain fresh.

In addition Alpha is under pressure from their competitor (Beta Inc) and needs to cut costs to a minimum. In achieving this, the Board of Directors
of Alpha are demanding that the new computer system provide them with a wide range of statistics on costs and profit. Unfortunately the Board have little idea what statistics they require, simply saying 'we will know what we need when you show us what you can give us'. They are also demanding that the new system be ready in three months since some of their contracts expire then and the system will be important in winning new contracts.

In developing the computer system the in-house development team must understand this complex situation; they need to interact with the airlines, airport and wholesaler in identifying requirements (all of whom may at any time decide to change their own systems), respond to demands from the board of directors, and ensure that the system is easy to use. The users of the system will include almost all the company, from the managing directors down to the people working on the production line.

Clearly this is not going to be an easy task for the in-house development team to achieve using the life cycle.

**Activity**

Consider how you might advise Alpha in developing its system. What problems might they face if they used the life cycle approach? Consider how this company's business might change in the future, and how the developers could respond. Revisit your answers once you have studied this chapter.

**Evolutionary systems development**

Evolutionary software development approaches allow software engineers to develop increasingly complete versions of a system over time (Pressman, 2005).

Within prototyping the software developer builds a first system from a rough specification of the system requirements and presents it to the user for comments. The user experiments with the prototype, makes suggestions for refinements and improvements, and the prototype is altered accordingly. It is particularly appropriate for systems of a highly interactive nature where requirements are not precisely defined, as users can try using the system early in the process and explore their requirements from the system by trying it.

We can distinguish between two types of prototyping: evolutionary prototyping and throwaway prototyping.

**Evolutionary prototyping**

Within evolutionary prototyping the evolved system - once the user is satisfied with the result - is used as the first working system. However, the rapid nature of the prototyping process often means that the prototype tends not to be 'well designed' and many implementation decisions will be undocumented.

**Throwaway prototyping**

Due to the often poor quality of the prototype and its documentation, the prototyping is often used 'merely' to determine the system requirements. In this case the prototype is discarded once the user requirements have been determined, and the system is then developed with special attention to software quality and maintainability.

Prototyping has many advantages. It tends to lead to a more static requirement specification where users do not ask for additions when they
first see the final working system. Users may also be more inclined to use
the system because they have been involved in its development.

However, there are management problems associated with prototyping.
There is no clear indication as to when the prototyping process should
stop. It is always possible to go for a further iteration. Also, the developer
(and system user) is generally keen to get a prototype working quickly.
As a consequence prototyping generally does not pay sufficient attention
to software quality, maintainability and documentation. In the case of
throwaway prototyping, there may be some user dissatisfaction with
the delay in receiving the working version of the system they had
apparently just seen fully operational. Some of these problems can be
avoided through a careful management choice between throwaway and
evolutionary prototyping.

**Incremental development**

The main idea behind incremental development is to give customers of
software systems the opportunity to delay decisions about their system
requirements to a point at which they have had some experience with the
actual system. For this purpose the customer (often the user) is asked to
determine and prioritise the functions the system is supposed to deliver.
The system is then developed in increments, starting with the function
with the highest priority and integrating further increments as they are
completed. With the completion of each increment the growing system
is delivered to the system user who is allowed to gain some experience
with the implemented increments. Although the use (the validation) of
the completed increments may encourage the refinement of the remaining
functions (requirements), the user is not permitted to demand changes for
the completed increments.

Each increment is developed using the development approach that
seems most appropriate. Functions that can be clearly specified may be
developed using the life cycle model, although less clear specifications may
be developed following an evolutionary approach.

Although the identification of increments suitable for incremental
development may be a difficult task, the incremental approach to software
development has a number of advantages. First, the system user can make
use of the most important component of the system at an early stage and
does not have to wait until the entire system is completed. Secondly, the
early use of part of the system supports the requirements determination
for the remainder of the system. Thirdly, the incremental development
may reduce the risk of overall project failure. Finally, the most important
system components are exposed to the most rigorous testing by the user
since they are implemented and delivered first. This reduces the likelihood
of encountering problems with these components once the system is
completed and in full use.

**Rapid application development (RAD)**

Rapid application development is an example of an incremental software
development process. The approach is used when the overall requirements
are well known (but the detailed requirements are poorly specified)
and where the project’s scope is constrained. The aim is that a small
development team (4–10 people) can develop a working system in a period
of about 60–90 days. In doing this it sidelines many traditional systems
development practices and therefore can produce somewhat error-prone
and difficult to maintain systems; however, this also means it can produce
working systems quickly. The aim is to undertake development as a series of ‘timeboxes’ – short periods of development in which analysis, design and implementation are undertaken in a fixed time without slippage.

Central to the method are a set of assumptions:

- that it is impossible to specify complex requirements in advance
- that the application (i.e. the software) acts as its own model – and therefore that the code includes comments which make it clear how it works, and for what purpose (hence avoiding lots of diagrams and documents)
- that design and testing is iterative
- that we should keep the application software in a working state at all times
- that users form an integral part of the development team, working alongside the developers. Unfortunately this is expensive (in users’ time) and may be difficult to arrange.

In order to achieve these rapid speeds it aims to accelerate the development of software through **incrementally developing software** (See Pressman, pp.80–83) and by **employing tools** (such as databases, user interface design software, CASE tools).

Since the aim is not to miss deadlines the approach focuses on making hard decisions about requirements - it is better to drop requirements than to miss deadlines. Rather than simply asking what users want, it suggests developers and users should debate requirements in order to rate them in the following way (known by the acronym MoSCoW)

- **M**ust-have
- **S**hould have
- **C**ould have
- **W**on’t have.

The reason for doing this is that it is noted that 80 per cent of a system's functionality can be delivered with around 20 per cent of the effort needed to complete 100 per cent of the requirements (this is called the Pareto Principle or 80/20 rule).

Finally, it is worth remembering that the cost of an application should not simply be measured in terms of development costs (for which RAD may be cheaper than traditional approaches), but for the whole life of an application (where the difficulty of maintenance may make it more expensive).

---

**Activity**

Consider how RAD might be used within Alpha Corporation. What advantages and disadvantages might it provide?

---

**The spiral model**

Boehm’s spiral model combines the iterative nature of prototyping with the controlled linear approach of the classic life cycle in the form of a spiral in which each loop represents a phase of the development process, such as requirements determination, design etc. (Pressman, 2005; Sommerville, 2007). Particular emphasis is put on the consideration of risks, e.g. the risk of using a new programming language for the coding of the system. Each loop of the spiral model is divided into four major tasks:
1. The objectives of the commencing phase (loop) are set out, a management plan is created and possible project risks are identified.

2. The identified risks are analysed, and action is taken to reduce these risks.

3. Depending on the main risks that have been identified, a suitable development approach is chosen.

4. The project is reviewed in order to decide whether the next development loop should be entered or whether the project should be abandoned.

**Activity**

Consider how the use of the spiral model might aid the Alpha Corporation. Compare and contrast its use with the other methods introduced in this chapter.

**‘Internet speed’ development**

While not leading to a specific software engineering method, it is worth considering the impact of developing computer software for websites. The explosive growth of organisations trying to capitalise on the internet and develop applications running on the internet has drastically affected the way in which software developed. In a study in 2003 (Baskerville et al., 2003) of companies developing software for the internet it was suggested the extreme **time pressure** and the very vague requirements (as these are often created through imagination and innovation) led to a very different approach to systems development. The study suggested that organisations adopt the following approaches when developing internet speed systems (they are therefore particularly applicable to web-based companies such as Ebay, Yahoo or Amazon):

- develop in parallel
- release more often
- depend on tools
- establish stable architectures
- assemble and reuse components
- implant customers in the development environment
- ignore maintenance
- tailor the methodology daily.

It is worth noting that this appears to suggest that we should not worry about developing software precisely as speed-to-market is more important. Clearly some elements of this approach are abhorrent to those schooled in traditional approaches (‘ignoring maintenance’ might not go down well with traditionalists), while others appear risky and difficult.

This approach to systems development is focused only on projects where business conditions are changing exceptionally rapidly and where the time to get a piece of software running is therefore crucial to the company's performance. Developing in parallel thus ensures that we can have new features being developed continuously, for as one feature is implemented and released, another parallel development team can be analysing the next feature. Releasing often is possible because the only version of the software is the one running on the company's web servers; it is also desirable to ensure that the company can capitalise on features (perhaps before the competition catches up). Depending on tools, establishing stable architectures and taking advantage of reuse are clearly aimed
at speeding up the development of the software. Likewise, implanting customers is aimed at ensuring the developing software features reflect customers' needs. Ignoring maintenance is rational because we expect to continue to employ the original developers for some time (so they can fix problems), because we can make changes to the software and introduce them very quickly, and because the risk of launching late is higher than the risk of facing maintenance issues.

This final point is perhaps the most crucial to our discussion of contemporary software engineering. It is the realisation that we must balance cost, time and risk against each other in selecting our approach to software engineering. The next chapter will consider this balancing act in detail.

### End user development

Finally it is worth reflecting on the fact that a large amount of important systems development occurs without any formal software engineering process at all. Within offices, businesses and factories, a wide range of people (end-users of software) regularly tailor their software, or construct new software, to support their work without any reference to software engineering. So called end user development is the practice of users writing small scripts in this way. Spreadsheets, simple access databases, word-processing macros and even simple scripting languages are often used. These end-user developed systems can, however, become crucial to the work practices of businesses, and software engineers may end up needing to look at them to discover the requirements for a new system.

### Activity

For each of the software development processes introduced in this chapter give a typical example of a software development project that is likely to benefit from the approach.

### A reminder of your learning outcomes

By the end of the chapter, and the relevant readings and activities, you should be able to describe the following software engineering processes and their advantages and disadvantages:

- the classic waterfall model
- evolutionary software development
- prototyping
- Boehm’s spiral model
- incremental development
- rapid application development
- internet speed development.

### Sample examination question

Describe the key features of rapid application development (RAD). Compare and contrast RAD with the spiral model of systems development, discussing their suitability for the following types of software engineering projects:

- an order processing system for a construction company
- an online auction site
- an interactive system to support staff in a call centre which provides help on using mobile phones
- the software to control a telephone exchange.