1

Introduction to the Basic Concepts

"There is no such thing as a 'finished' computer program"

Lehman [169 chapter 2]

This chapter aims to

1. Define and introduce software maintenance and software evolution.
2. Show how these topics fit within the wider context.
3. Distinguish software maintenance from software development.
4. Outline why maintenance is needed.
5. Give a flavour of the theoretical background and key skills required to implement effective software change.
6. Introduce the specific activities that comprise software maintenance.

1.1 Introduction

The discipline concerned with changes related to a software system after delivery is traditionally known as **software maintenance**. This section of the book will examine software maintenance with a view to defining it, and exploring why it is needed. An appreciation of the discipline is important because costs are extremely high. Many issues, including

safety and cost, mean there is an urgent need to find ways of reducing or eliminating maintenance problems [211].

During the past few decades, there has been a proliferation of software systems in a wide range of working environments. The sectors of society that have exploited these systems in their day-to-day operation are numerous and include manufacturing industries, financial institutions, information services, healthcare services and construction industries [176, 263]. There is an increasing reliance on software systems [69] and it is ever more important that such systems do the job they are intended to do, and do it well. In other words, it is vital that systems are useful. If they fail to be useful, they will not be accepted by users and will not be used.

In today's world, correct use and functioning of a software system can be a matter of life and death. Some of the factors that bear on the usefulness of software systems are functionality, flexibility, continuous availability and correct operation [170]. Changes will usually be required to support these factors during the lifetime of a system [256]. For example, changes may be necessary to satisfy requests for performance improvement, functional enhancement, or to deal with errors discovered in the system [176].

One of the greatest challenges facing software engineers is the management and control of these changes [131]. This is clearly demonstrated by the time spent and effort required to keep software systems operational after release. Results from studies undertaken to investigate the characteristics and costs of changes carried out on a system after delivery show estimated expenditure at 40-70% of the costs of the entire life-cycle of the software system [4, 33, 35, 176].

1.2 Definitions

Evolution – a process of continuous change from a lower, simpler, or worse to a higher, more complex, or better state.

Maintainability – the ease with which maintenance can be carried out.

Maintenance – the act of keeping an entity in an existing state of repair, efficiency, or validity; to preserve from failure or decline.

Software – the programs, documentation and operating procedures by which computers can be made useful to man [192 p.1].

Software maintenance – modification of a software product after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment [272 p.94].

1.3 The Basics

To understand software maintenance we need to be clear about what is meant by 'software'. It is a common misconception to believe that software is programs [184 p.4]. This can lead to a misunderstanding of terms that include the word 'software'. For instance, when thinking about software maintenance activities, there is a temptation to think of activities carried out exclusively on programs. This is because many software maintainers are more familiar with, or rather are more exposed to programs than other components of a software system. A more comprehensive view of software is the one given in the definitions section.

McDermid's definition [192 p.1] makes clear that software comprises not only programs - source and object code - but also documentation of any facet of the program, such as requirements analysis, specification, design, system and user manuals, and the procedures used to set up and operate the software system. Table 1.1 shows the components of a software system and some examples of each.

McDermid's is not the only definition of a software system [264] but it is a comprehensive and widely accepted one and is the one we shall use in this book.

The maintainability of a software system is something that is notoriously difficult to quantify. Certain aspects of systems can be measured. For example, there are several different ways of measuring complexity. Specific features such as interoperability or adherence to standards are also significant. However, there is no simple overall 'maintainability factor' that can be calculated. Nonetheless, recognising the features and traits that make a system easy to maintain is one of the major attributes of a good software maintenance engineer, and one of the things that makes such a person worth his/her weight in gold to a commercial enterprise. The true worth of software maintenance skills is being recognised more and more, and software maintainers are drawing level with software developers as the 'elite' of the software engineering team. The different factors that make up maintainability, and also the

issue of the 'public face' of software maintenance are issues discussed in more depth later in the book.

Table 1.1 Components of a software system

Software Components	Examples			
Program	1	Source code		
	2	Object code		
Documentation	1	Analysis / specification:	(a)	Formal specification
			(b)	Context diagram
			(c)	Data flow diagrams
	2	Design:	(a)	Flowcharts
			(b)	Entity-relationship charts
	3	Implementation:	(a)	Source code listings
			(b)	Cross-reference listings
	4	Testing:	(a)	Test data
			(b)	Test results
Operating procedures	1	Instructions to set up and use the software system		
	2	Instructions on how to react to system failures		

You will find many different definitions of software maintenance in the literature [209, 70, 6, 176]. Some take a focussed and specific view, and some take a more general view. The latter definitions e.g. defining maintenance as "any work that is undertaken after delivery of a software system" [70 p.233] encompass everything, but fail to indicate what maintenance entails. The former more specific definitions, whilst showing the activities of maintenance, tend to be too narrow. Typical amongst this set of definitions are

- the bug-fixing view – maintenance is the detection and correction of errors,
- the need-to-adapt view - maintenance is making changes to software when its operational environment or original requirement changes,
- the user-support view - maintenance is the provision of support to users.

The definition used in this book is from the IEEE software maintenance standard, IEEE STD 1219-1993. This (given in the previous section) draws on the different classifications, and provides a comprehensive definition.

1.4 How New Development and Maintenance Activities Differ

Although maintenance could be regarded as a continuation of new development [85, 108] (Figure 1.1), there is a fundamental difference between the two activities. New development is, within certain constraints, done on a green field site. Maintenance must work within the parameters and constraints of an existing system.

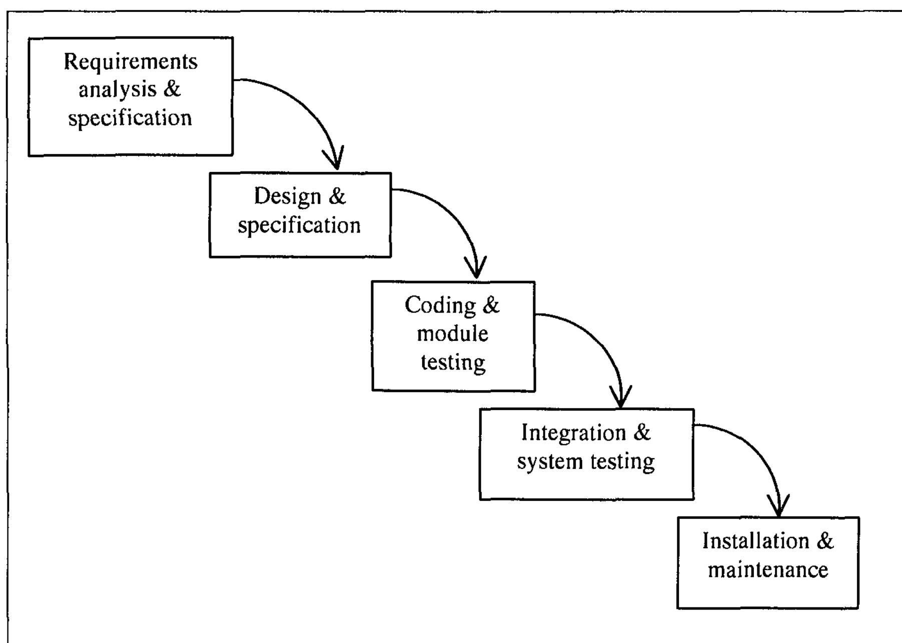

Figure 1.1 Waterfall model of a software life cycle

Before undertaking any system development or maintenance work, an *impact analysis* should be carried out to determine the ramifications of the new or modified system upon the environment into which it is to be introduced. This is discussed in more depth in Chapter 13. The impact of introducing a specific feature into a system will be very different if it is done as a maintenance activity, as opposed to a

development activity. It is the constraints that the existing system imposes on maintenance that give rise to this difference. For example, in the course of designing an enhancement, the designer needs to investigate the current system to abstract the architectural and the low-level designs. The reasons for this are explored further in Chapter 6. This information is then used to:

(i) work out how the change can be accommodated;

(ii) predict the potential ripple effect of the change, and

(iii) determine the skills and knowledge required to do the job.

To explain the difference between new development and software maintenance, Jones [146] provides an interesting analogy where he likens the addition of functional requirements to a live system, to the addition of a new room to an existing building:

> *"The architect and the builders must take care not to weaken the existing structure when additions are made. Although the costs of the new room usually will be lower than the costs of constructing an entirely new building, the costs per square foot may be much higher because of the need to remove existing walls, reroute plumbing and electrical circuits and take special care to avoid disrupting the current site"*
>
> Jones (quoted in [69 p.295]).

A detailed comparison of the individual activities of software development and software maintenance can be found in [270].

Exercise 1.1 Define the term software maintenance and indicate the factors that initiate the modification of software.

Exercise 1.2 Compare and contrast software development and software maintenance.

1.5 Why Software Maintenance is Needed

Having looked at what software maintenance is, and briefly what it entails, it is important now to appreciate why it needs to be done. There are a number of factors [4, 33, 36, 35, 176] that provide the motivation for maintenance:

- *To provide continuity of service:* Systems need to keep running. For example, software controlling aeroplanes in flight or train signalling

systems cannot be allowed just to stop if an error occurs. Unexpected failure of software can be life threatening. Many facets of daily life are now managed by computer. There can be severe consequences to system failure such as serious inconvenience or significant financial implications. Maintenance activities aimed at keeping a system operational include bug-fixing, recovering from failure, and accommodating changes in the operating system and hardware.

- *To support mandatory upgrades:* This type of change would be necessary because of such things as amendments to government regulations e.g. changes in tax laws will necessitate modifications in the software used by tax offices. Additionally, the need to maintain a competitive edge over rival products will trigger this kind of change.
- *To support user requests for improvements:* On the whole, the better a system is, the more it will be used and the more the users will request enhancements in functionality. There may also be requirements for better performance and customisation to local working patterns.
- *To facilitate future maintenance work:* It does not take long to learn that shortcuts at the software development stage are very costly in the long run. It is often financially and commercially justifiable to initiate change solely to make future maintenance easier. This would involve such things as code and database restructuring, and updating of documentation.

If a system is used, it is never finished [169 ch.2] because it will always need to evolve to meet the requirements of the changing world in which it operates.

1.6 Maintaining Systems Effectively

In order to maintain systems effectively, a good grounding in the relevant theory is essential and certain skills must be learnt. Software maintenance is a key discipline, because it is the means by which systems remain operational and cope efficiently in a world ever more reliant on software systems. Maintenance activities are far-reaching. The maintenance practitioner needs to understand the past and appreciate future impact. As a maintenance engineer, you need to know whether you are maintaining a system that must operate for the next five minutes or the next five decades. These problems were illustrated with the

introduction of a new air traffic control system in the UK, where the deferring of the date of completion of the new system meant that the old system had to be supported long beyond its expected lifetime (see the case study below). The motivation to get it right comes from understanding the wider implications, which in turn comes from an understanding of the theories and complexities of the discipline of software maintenance.

- The maintenance engineer needs a far wider range of skills than just computer programming. Amongst other things, he / she needs comprehension skills and wide-ranging analytical powers

1.7 Case Study – Air Traffic Control

For many years, air traffic over England and Wales was handled by two centres run by National Air Traffic Services in West Drayton, Middlesex and in Manchester. These centres dealt with air traffic crossing British airspace and also with take-off and landing traffic at British airports.

However, with air traffic doubling every 15 years, (the 1.6 million flights a year in UK airspace in 1998, was predicted then to rise to 2 million by 2003) the centres were working at and beyond their planned capacity. Alarmingly, the number of in-flight near misses was also increasing.

It was clear that new centres were needed. Planning began in early 1990 and the decision was taken to build a brand new centre and air traffic control system to replace the control centres in London and Manchester. The aim was to increase the number of flights that controllers could handle. The new centre was to be staffed by 800 controllers. IBM was chosen to build a bespoke system. It would be based on a US system, but would require new hardware as well as new software. It was due for completion in 1996 at an estimated cost of £339 million.

In 1996, stability problems were found with the new software and the centre was not opened. Because of continued delays, an independent study was carried out in 1998. The option of scrapping the project altogether was considered and experts warned that the system might

never work properly. Completion was rescheduled to between late 1999 and 2001 at a revised cost of £623 million.

An independent audit of the new software in 1999 found 1400 bugs in the two million lines of code. By August 2000, the number of bugs was reported to be down to 500, an estimated 200 of which were deemed to be serious. At this stage, 400 computer technicians were working on the system and managers were warning of possible failure to meet the new deadline.

Over a year was spent clearing the 1400 bugs found by the 1999 audit. Tony Collins of Computer Weekly reported "enormous success" at this stage i.e. in the latter months of 2000. He warned however that confidence in the plan to sort out the remaining serious bugs might be thrown into disarray if one of them turned out to be more serious than realised.

In the event, the new centre opened on Sunday, January 27th 2002, six years behind schedule and about £300 million over budget.

What went wrong with this project? The reasons were varied, and numerous, but stemmed from the initial decision to start from scratch. Mr Butterworth-Hayes, author of Jane's Air Traffic Control Special Report said this decision was "probably wrong" and that off-the-shelf components should have been used. He pointed out that the system needed to last 25 to 30 years, but software advances would be likely to outdate a new system in 18 months.

There were other problems. In 1994, IBM sold the project to another company, which was later taken over. This inevitably caused disruption to schedules and work plans.

Interestingly, in 1995, the US system that was the template, was scrapped.

In failing to meet both its deadlines and revised deadlines, the project was showing a classic early warning sign of serious problems.

The new system was brought on-line, not without further teething problems, but to date has provided a safer, more efficient environment and has a lot of much-needed spare capacity.

Was safety compromised by the delays? Almost certainly, yes. The controllers at the West Drayton centre had to cope with far more work than the old systems had been designed to handle. This compromised both the safety of air traffic and the health of the controllers themselves who were forced to work in high-stress conditions. It was during the most stressful period, towards the end of the new project, that staff had to be taken off live duties to be trained on the new system. This was a cause of further delay.

It is interesting to compare this project with the European mainland, where less ambitious upgrades resulted in much earlier working versions. These may not have been the ideal solutions conceptually, but they worked in practice because they took better account of what could actually be delivered with the available technology. The new French system for example, used off-the-shelf software to be updated every year. It was introduced with far fewer problems and far more cheaply.

The last word goes to Mr Butterworth-Hayes, speaking in 2002. "There are four major air traffic improvement programmes in Europe at the moment: France, Britain, Italy and Germany. Who has been the most successful? You have to say the UK is not in the top three."

1.8 Categorising Software Change

Software change may be needed, and initiated, for a number of reasons and can be categorised accordingly. In brief, software change may be classified under the following categories:

- Modification initiated by defects in the software.
- Change driven by the need to accommodate modifications in the environment of the software system.
- Change undertaken to expand the existing requirements of a system.
- Change undertaken to prevent malfunctions.

The categorising of software change is not simply an academic exercise. It is vital in understanding when and how to make changes, how to assign resources and how to prioritise requests for change. These areas are expanded and explained in more detail in the following chapters.

1.9 Summary

The key points covered in this chapter:

- The cost of maintaining systems in many organisations has been observed to range from 40% to 70% of resources allocated to the entire software life-cycle.
- It is important to understand what is meant by the basic terms that underlie the field, *software, maintenance, evolution* and *maintainability,* in order to understand the importance of software maintenance and how it fits into context in the modern world.
- Software maintenance and software development, although closely linked, are different. It is important for those involved in maintenance fully to appreciate the differences.
- In a world ever more reliant on software systems, the discipline of software maintenance is becoming more and more important.
- As with other fields, software maintenance is underpinned by a theoretical base. It is important to be aware of this in order to be an effective software maintainer.

Having introduced the basic concepts and given a broad overview of maintenance activities, the next chapter looks at the overall framework within which maintenance is carried out.