Prediction of Software Maintenance Costs

David J Morrison

July 2001
ABSTRACT

This thesis is concerned with predicting the costs of maintaining a computer program prior to the software being developed. The ubiquitous nature of software means that software maintenance is an important activity, and evidence exists to support the contention that it is the largest and most costly area of endeavour within the software domain. Given the levels of expenditure associated with software maintenance, an ability to quantify future costs and address the determinants of these costs can assist in the planning and allocation of resources. Despite the importance of this field only a limited understanding of the factors that determine future maintenance costs exists, and maintenance estimation is more frequently applied to existing software.

A hypothesis has been postulated that suggests the inherent maintainability of the software, the scale of the activity and the degree of change that pertains will determine future software maintenance costs. The variables that contribute to the maintainability of the software have been explored through a survey of past projects, which was undertaken using a questionnaire. This was designed with assistance from three separate teams of professional software engineers. The questionnaire requires 69 numerical or ordinal responses to a series of questions pertaining to characteristics including program structure, computer architecture, software development methodology, project management processes and maintenance outcomes.

Factor analysis methods were applied and five of the most powerful predictors are identified. A linear model capable of predicting maintainability has been developed. Validation was undertaken through a series of follow-up interviews with several survey respondents, and by further statistical analysis utilising hold-out samples and structural equation modelling. The model was subsequently used to develop predictive tools intended to provide management support by both providing a categorical assessment of future maintainability, and a quantitative estimate of probable maintenance costs. The distinction between essential corrective maintenance, and other elective forms of maintenance is considered.

Conclusions are drawn regarding the efficacy and limitations of tools that can be developed to support management decision making. Subject to further work with a larger sample of projects, preferably from within a single organisation, it is concluded
that useful tools could be developed to make both categorical ('acceptable' versus 'not acceptable') and static (initial) quantitative predictions. The latter is dependent on the availability of a software development estimate. Some useful predictive methods have also been applied to dynamic (continuing) quantitative prediction in circumstances where a trend develops in successive forecasts.

Recommendations for further work are provided. These include:

- Factor analysis and linear regression has been applied to a sample of past software projects from a variety of application areas to identify important input variables for use in a maintainability prediction model. Maintainability is regarded as an important determinant of maintenance resource requirements. The performance of these variables within a single organisation should be confirmed by undertaking a further factor analysis and linear regression on projects from within the target organisation.

- The robustness of model design within this target organisation should be considered by applying a sensitivity analysis to the input variables.

- This single organisation maintainability predictor model design should be validated by confirmatory interviews with specialists and users from within the target organisation.

- Aggregate scale has been identified as another predictor of overall maintenance resource requirements, and the relationship between development and maintenance effort explored for the general case. It is desirable that development and corrective maintenance scale relationships should be explored within a single organisation. Within this environment the association between standardised effort and maintainability should be confirmed, and the value of the logistic model as a descriptor of the relationship verified.

- The approach to quantifying non-corrective maintenance that has been outlined requires further development. The relationship between annual change traffic and maintenance costs should be modelled, assuming a prior knowledge of the scale and maintainability determinants.
A sensitivity analysis should be applied to the predictive system that has been developed, recognising the potential for error in the values of the input variables that may pertain.

A goal of this further research should be the development of a suite of soft tools, designed to enable the user to develop a software maintenance estimation system.

This thesis is my own work.

David J Morrison: ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...

Date: ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
ACKNOWLEDGEMENTS

I would like to acknowledge the help and assistance of the following people:

- Dr Robert Raeside, Department of Mathematics, Napier University
- Michelle Ledgard, 600 Lathes
- Kathleen O’Connell, Ellison Machine Tool & Robotics Company
CONTENTS

Chapter 1  Introduction

1.1  Introduction  Page 1
1.2  Objectives  Page 1
1.3  Background  Page 1
1.4  Availability  Page 2
1.5  Reliability  Page 3
1.6  Software Reliability  Page 5
1.7  Maintainability  Page 7
1.8  Aims of Thesis  Page 13
1.9  Structure of Thesis  Page 17

Chapter 2  Literature Review

2.1  Software Maintainability and Maintenance Overview  Page 19
2.2  Definitions and Discussion  Page 24
2.3  The Scale of the Problem  Page 29
2.4  Aspects of Software Maintenance  Page 34
2.5  Software Maintenance Management  Page 39
2.6  Software Development and Reuse Strategies  Page 42
2.7  Software Estimation  Page 46
2.8  Software Maintainability & Maintenance Effort Prediction; Existing Programs  Page 54
2.9  Software Maintenance Effort Prediction; Future Projects  Page 60
2.10  Estimation Quality  Page 65
2.11  Summary  Page 68

Chapter 3  Survey

3.1  Introduction  Page 71
3.2  Survey Methodology  Page 71
3.3  Questionnaire Rationale  Page 73
3.4  Questionnaire Structure  Page 75
3.5 Survey Execution Page 78
3.6 Analysis of Factors Page 88
3.7 Analysis of Input Variables Page 91
3.8 Discussion of Results Page 94
3.9 Validation of Results Page 101
3.10 Further Analysis of Variables Page 106
3.11 Conclusion Page 111

Chapter 4 Static Prediction
4.1 Introduction Page 113
4.2 Categorical Prediction Page 114
4.3 Continuous Prediction Page 126
4.4 Discretionary Maintenance Page 139
4.5 Conclusions Page 142

Chapter 5 Dynamic Prediction
5.1 Introduction Page 145
5.2 Dynamic Modelling; Development Effort Updates Page 146
5.3 Dynamic Modelling; Development Effort & Maintainability Updates Page 159
5.4 Conclusions Page 164

Chapter 6 Conclusions
6.1 Introduction Page 167
6.2 Summary of Findings Page 167
6.3 Maintainability Definition Page 177
6.4 Novelty of Approach Page 181
6.5 Business Implications Page 182
6.6 Recommendations Page 183
6.7 Recommendations for Further Research Page 186

Appendix 1 Software Maintenance Page 189

Appendix 2 Analysis of Variance Page 198

Bibliography Page 199
1.1 Introduction

This thesis is concerned with predicting the costs associated with maintaining software. Software maintenance estimation is an area that has received less attention than the apparent scale of the activity should justify. In particular, the research is focussed on the design of management tools that can support the estimation of these costs early in the software development lifecycle. This is an important area because of the widespread application of software-based systems throughout modern society and because maintenance requirements are believed to be absorbing an increasing proportion of total software engineering and software management resources. There is consequent need to establish an understanding of the likely future personnel and financial commitments at the outset of a project because it will have a bearing on both operational performance and commercial viability.

1.2 Objectives

The principal objectives of this thesis are to investigate the methods that exist or could be developed to predict software maintainability and the associated maintenance costs, close to the start of software development and also during the development phase. Exploring the concepts of software maintenance, maintainability, and in particular the maintainability of software will facilitate these objectives. It is hoped that an outcome from the work will be a mathematical model developed to support such predictions.

1.3 Background

A key quality attribute of any component or system is its availability to the user. Unless it is available to the user in a predictable and stable mode of operation then other attributes such as functionality or adaptability may be rendered irrelevant.
because the component or system is either not accepted into service or rapidly loses credibility with the user. Even worse, perhaps, is the situation where a component or system does get successfully introduced into service, the user becomes dependent on the functionality being provided and then experiences an unexpected degradation or even the complete disappearance of the required functionality. Examples of both these deficiencies are widespread; from personal experience they extend from inventory management tools through airline ticketing suites to naval damage control systems. It has also been observed that recovering the confidence of the users, including the local management, can often be more problematical than resolving the issues that generated the failure. Furthermore, it has also been noted that these failures do not always have their origins in the technology, but may be related to human or organisational dimensions such as training or documentation.

1.4 Availability

This has been defined by Musa et al (1990) as being:

\[
\text{The expected fraction of time during which a system or component is functioning acceptably.}
\]

The stochastic aspect is emphasised by Fenton & Pfleeger (1997):

\[
\text{The probability that a component is operating at a given point in time.}
\]

Either of these definitions is acceptable in that they convey the idea of the availability of a system consists of the proportion of the total time during which a system might theoretically be accessed. This proportion can be anticipated and expressed in probabilistic terms.

Both sources recognise that availability is itself dependent on two other attributes; the reliability and maintainability of a system or component. This can be reconciled with practical experience of assessing availability during the design cycle, and of improving the availability of equipment which has already entered service. In the
latter case this can generally be realised by increasing the reliability of the system through redundancy or preventive maintenance, and by improving the maintainability of the system through, for example, improved documentation or training of users and maintainers.

These measures are reflected in an interesting definition of availability that is provided by Stork (1997):

\[ \text{The ability of a system to withstand faults.} \]

This ability to withstand faults could be interpreted as the capacity of the system to resist faults developing into failures and affecting the user. Clearly this capacity should be enhanced by such measures as redundant components and preventive maintenance, as discussed above. Reliability is discussed in 1.5.

1.5 Reliability

Reliability has been defined by Jardine et al (1985) as:

\[ \text{The probability of equipment operating successfully for a specified period of time under specified operating conditions.} \]

Several similar definitions of reliability also exist. It can be readily seen from this definition that the reliability of a component or system will have considerable affect on its availability.

Analysing and predicting equipment reliability is a field which expanded rapidly during and immediately after the second world war, and has continued to grow in importance ever since. This growth perhaps mirrors society's growing dependence on technology in diverse fields such as aviation, nuclear power and medicine. Other developments that have encouraged work in this area range from the hazards of manned space flight through to the competitive imperatives of capital-intensive industrial processes and 'just in time' (JIT) manufacturing.
Reliability is to a large extent built into equipment by its designer. Jardine et al (1985) support this view, recognising that the selection and configuration of components within a system determine what could be considered as being the inherent reliability of the system. This can, of course, be affected by the quality of maintenance once a system is introduced into service, although specifying the fundamental maintenance requirements is again part of the design process. Understanding and assessing reliability is therefore important if reliable and available systems are to be designed and introduced into service. Ballard (1989) recognises two distinct aspects of reliability analysis; the qualitative and quantitative dimensions. These are complementary approaches rather than being alternatives. Widely used examples of the former are ‘failure-mode and effects analysis’ (FMEA), a bottom-up approach which starts by identifying all possible component failure modes; and ‘fault tree analysis’ (FTA), an iterative method which progresses in small, logical steps from the effect to its immediate cause.

Several distinct quantitative methods are recognisable, but a prerequisite in all cases is that the boundary between acceptable (working) and unacceptable (failed) performance is explicitly defined. One commonly used quantitative approach is to analyse empirical data obtained from the performance of an existing and similar design. The underlying assumption here is that factors that were relevant in the past will also determine future performance. Another approach involves developing system reliability predictions from component reliabilities, and is most relevant to designs where components parts perform different functions but less so for equipment or structures which are subjected to highly correlated loads; this might typically be used in conjunction with the FTA method. A third approach involves modelling physical failure mechanisms, with applications ranging from the wind loading of structures through to earthquake loading. These methods, and others such as simulation, may be used individually or in combination to estimate the reliability of a system.

In summary, reliability is critical to the overall availability of a system, and reliability analysis provides a means to identify and correct potential weaknesses. The choice of
reliability modelling approach depends on the nature of the design, but all are heuristic and therefore involve uncertainty.

Reliability failures arising from the design process are a special case, and relate to software reliability. This relationship is pertinent because software fabrication is essentially a design process. In the case of hardware design a distinct production or construction phase generally follows. Software reliability is discussed below.

1.6 Software Reliability

Over the past three decades software implementations have become a ubiquitous component in virtually all major engineering projects and many new product developments. Software can process huge volumes of data accurately and reliably; it can control processes, including safety-critical ones, to a level of dependability which is sometimes impossible using manual alternatives; and it can offer reconfigurability and upgradability. The utility of any given software is closely related to the quality of the development and implementation. McConnel (1993) provides a useful summary of the relevance of these processes to software quality.

Reliability is arguably the most crucial software quality attribute. Low reliability may result in the software never being used for its intended purpose, or rapidly losing credibility after implementation.

Many features of general reliability also apply to software reliability. These are reflected in its definition which is similar to the more general definition proposed above. Musa et al (1990) define software reliability as:

\[ \text{The probability of failure-free operation of a computer program for a specified time in a specified environment.} \]

The probabilistic nature of software reliability and the relevance of associated software reliability growth prediction methods can be discerned from the definition.
Despite these similarities, software reliability has several peculiarities that must be recognised and accommodated in a reliability analysis.

Leaving aside the ‘infant mortality’ characteristic of hardware components, the failure of most hardware systems is determined by the time it takes the component parts to wear out. By contrast software does not wear out, its reliability depends on the quality of the design and implementation processes and the rigour with which modifications are incorporated.

Redundancy can be designed into hardware systems by including duplicate components; if one fails, the system switches to an identical component and successfully follows a parallel path. If a software module within a program fails when activated by a certain class of input, however, there is no recovery by switching to an identical module because it will also fail. An alternative and the nearest software analogy would be the availability of a non-identical ‘copy’, meaning a separate and independent software implementation of the same specification. This technique has been used in safety-critical situations, such as flight control systems.

Hardware failures are often permanent and frequently physically apparent. The component, and perhaps the system, stops working until a repair or replacement is effected. By contrast, many software failures are transient, with the system continuing to execute successfully in the absence of a particular class of input that causes failure. Sommerville (2001) recognised that commonly used hardware metrics such as ‘mean time to failure’ (MTBF) or ‘mean time to repair’ (MTTR) are therefore less applicable to software. Furthermore, the availability of a program does not depend purely on failure intensity, but must also reflect such factors as the degree of data corruption.

Defining and demonstrating achievement of a particular software reliability objective requires care. It is insufficient, for example, to define the objective in terms of a maximum ‘acceptable’ number of faults for a given program length; it is impossible to know deterministically whether this limit has been achieved. How do we know that the last fault has been found? Perhaps more importantly, this measure says nothing about the operational performance of the program.
The reliability requirement should be stated in terms of the behaviour of the program under operational conditions. Demonstrating achievement of this objective will usually be in probabilistic terms. Resource requirements associated with achieving the reliability criterion are likely to be a concern to the project manager.

Software reliability growth modelling involves predicting the effort required to reach a given level of failure intensity. This has implications for both resource management and cost forecasting, but obviously also provides important information on the failure intensity likely to be experienced by the user.

This is a well-researched field with Jelinski and Moranda (1972) credited with design of the first model intended specifically for software reliability growth prediction. More recently Xie (1993) produced a synopsis of much of the research in the area. Authoritative texts that explore the subject in detail have been written by Musa et al (1990) and edited by Rook (1990).

Unsurprisingly both these texts also make reference to software maintainability and the associated maintenance costs. As was observed in 1.4, the availability of a system depends on both reliability and maintainability. General maintainability and maintenance practice is reviewed in 1.7, and software maintainability and maintenance are discussed in detail in Chapter 2.

1.7 Maintainability

Having established that the availability of a system depends on both its reliability and maintainability, and reviewed reliability in 1.5, this section is concerned with understanding maintainability, and both the scale and nature of the maintenance process. Patton (1980) defines maintainability as:

*The inherent characteristic of a design or installation that determines the ease, economy, safety, and accuracy with which maintenance actions can be performed.*
A key point is the inherent nature of this attribute, which like reliability is largely embedded in the equipment by the designer. The ease with which maintenance can be undertaken will affect the duration of the maintenance activity and consequently the availability of the system. Similarly, the economy and safety dimensions can be seen to be related to the duration, while the accuracy will also affect the frequency with which maintenance is necessary. The close relationship of maintainability to the design is further emphasised in another definition provided by Moss (1985):

*That element of the product design concerned with assuring that the ability of the product to perform satisfactorily can be sustained throughout its intended useful life span with minimum expenditure of money or effort.*

This definition has the merit of introducing other dimensions including the underlying purpose of the product, and also recognises that this purpose must be expressed in the time domain as well as the functional. Again it recognises the financial aspect of the activity. These aspects are recognised where an equipment investment decision reflects the full life costs of ownership, including such items as training, insurance and maintenance, as opposed to focusing purely on the capital outlay. The investment analysis requires the equipment designer to specify the maintenance tasks in terms of content and frequency, and the prospective owners to have a maintenance policy. Taken together, an understanding of the task and a maintenance plan that considers resource and organisation enable costs to be quantified.

Patton (1980) defines maintenance as being:

*The function of keeping items or equipment in, or restoring them to, serviceable condition.*

Important points contained within the definition are that maintenance is a function in its own right and that the objective is to keep something in a state which allows it to be used. Keeping an asset serviceable, as opposed to restoring it, is a distinction that is discussed further, later in this section.
Modern maintenance is a major activity that consumes considerable resources. Wilmott (1992) estimates that by the early 1990s the EU was spending £100Bn per year on maintenance and directly employing two million people to undertake the work. These figures equate to 10% of industrial added value at the time, and were equivalent to Holland's GNP. Mobley (1990) provides evidence that maintenance represents 40% of the production costs of US heavy industries such as steel-making. More recently and locally, Dunn (1997) asserts that in the UK railway maintenance accounts for about 30% of operating costs. A key question for the manager of a manufacturing process or a service provider exists; how much should be spent on maintenance? An unscheduled outage can be very expensive if output is affected, but developing an expansive maintenance process operating at a low utilisation is also costly. Meredith (1992) demonstrates that a theoretical optimum could be calculated which considers, for increasing levels of maintenance intensity, the cost impact of breakdowns and the cost of providing maintenance. As the maintenance intensity increases, the cost of providing the service increases but the breakdown cost decreases. A point must therefore exist at which the total costs to the business are minimised. This is illustrated below in Figure 1.1.

![Figure 1.1 - Increasing Maintenance Intensity vs. Cost Impact of Breakdowns](image)

Although a useful concept, this model can be problematical to apply in many situations beyond the most simple, because the relationship between maintenance investment and performance is both complex and variable, and because of the
Intangible costs arising from breakdowns. Examples of these factors include the effect on future business and the quality of the product that was processed immediately before the breakdown. Additionally, Mobley (1990) provides evidence that perhaps one third of all maintenance activity is nugatory because it is either unnecessary or improperly performed, and identifies a shortage of factual data as the principal cause.

If a significant proportion of maintenance is being wasted, this raises another question for the management to consider; how should the maintenance budget be spent to greatest effect? The past fifty years has seen much research and considerable advances as efforts are made to answer this question. Prior to this the default strategy had essentially been 'run to failure', followed by corrective maintenance. Meredith (1992) defines corrective maintenance as:

*Repairing a machine or process when it breaks down.*

This is obviously a simple strategy because maintenance activity is determined by what happens to break down. It is also highly disruptive in most production situations, can result in damage to more valuable equipment and also will require higher levels of inventory to provide buffer stocks to accommodate downtime elsewhere in the process. It is incompatible with a 'just in time' approach to production, and is expensive both in the direct sense of asset utilisation being relatively low and inventory being relatively high, and also in the intangible sense discussed above. The logical alternative to repairing equipment when it breaks down is to undertake preventive maintenance and keep the system running. Preventive maintenance is defined by Moss (1985) as:

*Tasks performed to defer or prevent an anticipated failure occurrence.*

Similarly Meredith (1992) defines it as:

*Conducting maintenance before the machine is expected to fail or at regular intervals.*
An important distinction exists between the two forms of maintenance activity encompassed within this definition, in that maintaining the equipment at regular intervals is predominantly a 'blind' or open loop activity, whereas the alternative which involves an expectation necessitates some form of information gathering to provide feedback. The latter is a predictive approach and requires a closed loop.

This definition needs to be analysed to provide a clearer picture of the practical alternatives. Maintenance at regular intervals is called scheduled maintenance by Wilmott (1992) and is distinguishable because it is time driven and often related to a time-based variable such as MTBF. To this extent the procedure is not wholly open loop, in that account has been taken of historical failure frequencies. Mobley (1990) criticises this approach because it implies that equipment will always degrade within a typical time frame. This is seen as a source of both waste, in that maintenance may be performed unnecessarily, and also of risk, because of the experience of increased frequency of failure immediately after maintenance. Practical experience suggests that this is not the always case, with an example being the effects of ambient conditions on electronic systems. Additionally, if the schedule is purely time-based, no account is taken of the equipment utilisation since the previous maintenance service. Wilmott (1992) identifies the 1960s as the period when this management strategy prevailed, and the 1970s as the time when the predictive approaches identified above began to be introduced.

Condition-based maintenance is a predictive approach, which involves monitoring and recording the in-service loading of a component, enabling its useful life to be assessed and informed decisions to be made regarding replacement or maintenance. Difficulties with this approach can include a lack of data on component failure mechanisms, which may result in components failing unexpectedly or being replaced unnecessarily. Additionally, in its purest form this strategy takes no account of the impact of an unexpected failure. During the 1980s reliability-centred maintenance (RCM) strategies were implemented which addressed this last point by considering the consequences of failure; these may be economic, safety or service related. In the past decade total preventive maintenance (TPM) management has been developed. Wilmott (1992) describes TPM as "RCM plus the people dimension". The approach complements the rigour of reliability analysis and maintenance prediction methods.
with other techniques such as operator maintenance training and process yield analysis.

A third question for the management must relate to the measurement of availability, and by implication, both reliability and maintainability, and the impact of the various strategies described above. Well-established metrics include the MTBF and MTTR ratios described above, which reflect reliability and maintainability respectively. A commonly used measure of availability is provided by:

\[
\text{Availability} = \frac{\text{MTBF}}{(\text{MTBF} + \text{MTTR})}
\]

Equation 1.1

The limitations of this approach when applied to software were discussed in 1.6, and related to such issues as transient failures and data corruption. If this measure can be criticised in the hardware domain, it relates to its inability to recognise incremental changes in a system which progressively degrade performance and possibly output immediately before a failure, and possibly afterwards, as the equipment is brought back into service. An alternative approach described by Tucker et al (1992) in the context of a steel making process is called ‘overall equipment effectiveness’ (OEE), this addresses some of the concerns discussed above. OEE is an overall measure of how physical assets perform and may be defined by:

\[
\text{OEE} = \text{availability} \times \text{productivity} \times \text{yield}
\]

Equation 1.2

with these factors respectively addressing aspects such as:

- availability - breakdowns - frequency & duration
  - set-up times
- productivity - slow running
  - other unscheduled stoppages
yields—product defects
conversion losses

Clearly this approach is appropriate to the performance of a manufacturing process rather than an equipment component, but it does provide a means of measuring the utility of an asset. This is particularly crucial in a bottleneck situation where the output from a process constrains overall plant performance. It also provides a means of assessing the impact of improvement measures and has particular value where trade-offs exist. For example, reducing maintenance frequency might increase availability but increased product defects might completely offset any advantage. The efficacy of this measure is also recognised by Wilmott (1992) who advocates its use in conjunction with targeting current best performance as the norm, and applying external benchmarking. Other measures of performance and improvement which are proposed by Martin (1992) include planned maintenance as a proportion of total maintenance, value of lost production from equipment failures and the proportion of time spent by employees in training. These again are more process related than the narrower component or equipment orientation, and therefore tend to lay greater emphasis on the human dimension.

The difference in approach between the management of maintainability and maintenance for processes as compared with components is extruded further when these attributes are considered in the software domain.

1.8 Aims of Thesis

The objectives of the research contained in this thesis are to review previous work in the area and to continue from there to identify those input variables that are the determinants of maintenance costs. These variables will then be used to develop a predictive model designed to provide a forecast of future maintenance costs. It is intended to consider both the generation of an initial estimate, and also mechanisms that can update this estimate as development proceeds.
Given the huge range of scale, complexity and applications that software addresses, it is necessary to identify the boundaries within which this research pertains, initially in a purely subjective sense. The programs that fall within the area of interest may be characterised as being generated within a professional (commercial, industrial, government) environment, typically involving teams of personnel working on relatively large scale and probably lengthy developments, with the objective of providing a long-term solution to a customer or end-user requirement. It is also likely that some degree of risk, either arising from the application area or development approach may be present. Small scale, temporary, very simple and informal programming will tend to fall outside the area of primary research interest.

One particular constraint on the research relates to the emergent properties of a system. These properties relate to the behaviour of the system as a whole, once installed in a particular operating environment. This is often of critical importance for computer-based systems because an unexpected failure of the system to achieve a specified performance threshold may render the system unacceptable to the users. The maintainability of a system may be regarded as a non-functional emergent property. A simple example of this situation, in the context of maintainability, relates to a computer-controlled machine tool, designed and manufactured in Europe but with a Japanese control system embedded. This system is supplied, as specified, with fault diagnostic software and an associated man-machine interface (MMI) written in English. The machine tool vendor subsequently starts selling the equipment in South America, having first translated the operating MMI and related documentation into Spanish. The purchased Japanese sub-system, however, that has been embedded retains English diagnostics. The first problem in South America produces a crisis, with the machine regarded as unmaintainable by the Spanish-speaking customer. Points to note in this example include the operational aspect and the non-technical nature of the problem. Had the machine been exported to North America it is unlikely that the problem would have arisen. Alternatively, a more rigorous approach to configuration management might have prevented this shortcoming from reaching the customer. Predicting events of this kind lies beyond the scope of the research because, as Sommerville (2001) notes, "it is often difficult to predict the values of these emergent properties in advance". To the extent that maintainability is assessed during the course of the research to be a determinant of maintenance costs, these are more
likely to be attributed to software product or process issues, or to human or social characteristics of organisations from which information is obtained.

Returning to the introductory and subjective description in this section of the area of research interest, these can be specified more formally by consideration of three attributes of the activity; scale, development method and application area.

The scale is difficult and possibly meaningless to describe in terms of program size, given a likely multiplicity of languages and data structures. Development effort, however, associated with these target programs is unlikely to be less than ten man-months, extending upwards into hundreds or possibly thousands of man-months.

The development method is likely to be centred upon that software developed through some variation on the well-established waterfall process. Software reuse is a dimension that will be explored. The less structured evolutionary development approach, and the other extreme of formal methods, will not be considered. Sommerville (2001) characterises these development methods. It is anticipated that the development methods should be able to accommodate the uncertainty arising from specification changes and also conform to predefined constraints arising, for example, from interoperability requirements.

An analysis of application areas and their attributes is provided below in Table 1.1.
<table>
<thead>
<tr>
<th>Software type</th>
<th>Heavy h/w interaction</th>
<th>Multi-user</th>
<th>Multi. ext. interfaces</th>
<th>Resource sharing</th>
<th>Complex/large data</th>
<th>Data gathering</th>
<th>Data analysis</th>
<th>Control component</th>
<th>Monitor component</th>
<th>Interactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eng/ scientific</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
</tr>
<tr>
<td>Real time</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
</tr>
<tr>
<td>Artificial intel.</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
</tr>
<tr>
<td>System</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
</tr>
<tr>
<td>Business</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
</tr>
<tr>
<td>Embedded</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
</tr>
<tr>
<td>PC</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
<td>$\Box$</td>
</tr>
</tbody>
</table>

Table 1.1 - Application area attributes  
$\Box$ - Possible  
$\blacksquare$ - Probable
In terms of the attributes of the various application areas considered, those shaded in Table 1.1 are of prime interest because of their intrinsic complexity and processing intensity. This decomposition of software by system attributes is a development of an analysis provided by Pressman (1997). It is recognised that the continuing increase in personal computer power will tend to change these boundaries over time. Additionally, a distinction should be made between the PC as a ‘personal’ computer, used for office or leisure applications, and the PC as an engineering or computational workstation or component in a network. The latter falls within the area of research interest.

In summary, the research intent has been defined in terms of the scale, development method and application area. Emergent properties of the software have been explicitly excluded.

1.9 Structure of Thesis

The thesis consists of a further five chapters.

In Chapter 2 a survey of important and recent work in the field is provided. This survey starts with a review of definitions that pertain, and provides consideration of the overall scale of the software maintenance challenge. An examination of the related management processes is followed by an investigation of software estimation in general and as it applies to maintainability and maintenance modelling.

Chapter 3 begins with a description of a survey of past software projects. The factor analysis and regression methods that were applied to constructing a maintainability prediction model are described. The chapter ends with a discussion of the approaches that were taken to validating the performance of this model.

Categorical and continuous prediction are considered in Chapter 4. The former relates to classifying a software project as either ‘good’ or ‘bad’, in terms of predicted maintainability. The latter is concerned with effort prediction arising from corrective maintenance, and includes the description of a logistic model that is developed. The
provision of estimates associated with non-corrective, or discretionary maintenance is discussed at the end of this chapter.

The incorporation of new information into effort predictions is explored in Chapter 5. Various statistical approaches are considered and the deployment of some specific tools is explored. The chapter concludes with a discussion of their relevance to managing software projects.

A summary of the research is provided in Chapter 6, which also includes a discussion of the deployment of the techniques. Recommendations for further research are also outlined.
2.1 Software Maintainability and Maintenance Overview

This chapter is devoted to an examination of recent and current work in the software maintenance area, and also to a review of how contemporary technological and social change may affect future requirements. The approach taken reflects the scope of what Glass (1994) refers to as the "informational phase" of the research lifecycle, which should "gather or aggregate information via reflection, literature survey, people/organisational survey, or poll". The management of the maintenance process and, in particular, maintainability assessment and effort estimation methods, is considered in the context of technological forecasting.

The general approach taken commences with a brief overview of the management of projects, and is developed to examine software project management. An understanding of the nature of software maintenance and maintainability is pursued through a discussion of the various definitions that have been promulgated, and by exploration of the overall scale of software maintenance activity. This understanding is developed further through a decomposition of software maintenance activity into constituent elements, and by reconciling some of the terminology used within this domain. The related areas of software redevelopment and software reuse are considered and several models relating to the management of software maintenance operations are discussed. Quantification is explored through consideration of software estimation, and the particular issues that pertain to software maintenance. Software maintainability and maintenance effort prediction are explored, in the context of both current and future software. Estimation quality is also discussed. The chapter ends with a summary of conclusions that can be drawn from the review, and the implications for further research.

Forecasting, and the quality of forecasts, are the essence of this review. This pertains both to the recognition and assessment of risk, and the estimation of size, effort or cost. In the case of estimation, individual and institutional shortcomings in forecasting resource requirements accurately have probably pertained over many centuries. In the more distant past this was perhaps a lesser concern; poorer
measurement and communication, limited and less visible political accountability, and cheap labour prevailed, although sometimes offset by summary executive judgements. In the UK during more recent times prominent and nationally known examples of projects overrunning budgets include the development of the Concorde supersonic jet airliner and, a generation later, the construction of the Channel Tunnel. In the latter case the variances relate not just to the capital cost (higher) but also to the initial operating revenues (lower). Military equipment projects in the UK also provide several recent instances of the mis-assessment of technical risk and the under-estimation of resource requirements. The Nimrod airborne early warning (AEW) and Challenger tank contracts are well-publicised exemplars. A Financial Times (2000) report indicated that the 25 largest Ministry of Defence (MoD) procurement contracts were running an average of 47 months behind the original plan. The response of government in both the UK and US to these problems has been to attempt to transfer the risk from taxpayer to the contractor by introducing fixed-price competitive tendering. Unintended consequences of this policy were initially to encourage optimistic bids by competing contractors, consequential margin reductions or losses on the subsequent contracts, followed by higher and more conservative bidding on the next round of contracts. A more positive development has been the evolution and application of improved project management practice, including estimation and risk-management methods. The commercial consequence has been a move back into more of a risk-sharing culture in both the MoD and for similar reasons at the US Department of Defense, although this has also been driven by strategic and national political considerations. The former might relate to protecting a technical capability, while the latter might reflect an employment imperative.

Elsewhere in the UK public sector, several recent examples of delays to major software projects have received considerable publicity. During 1999 the Passport Agency encountered difficulties with the implementation of a computerised system, resulting in massive inconvenience to customers and considerable political comment. Larger delays to system developments relating to the national insurance system has resulted in benefit records not being updated promptly and the possibility that large compensation payments may have to be made. A third example is provided by the delays to the completion of a new London air traffic control system, which is both over-budget and four years late. These projects have involved personnel from both public and private sectors. Demands have been made for greater oversight of such high profile projects than is currently prevalent, given the limited jurisdiction of the
National Audit Office, a public agency that reports to Parliament rather than a
government minister. A recent report from the Cabinet Office entitled "Successful IT:
Modernising Government in Action" appears to demand improved project
management and more accountability. This more interventionist stance is similar to
that deployed by the US Government through their equivalent, the Government
Accounting Office.

Large-scale development projects funded by such bodies as the World Bank and the
United Nations Development Programme also provide examples of over-optimistic
forecasts, despite well-codified project evaluation procedures and substantial
expertise. Ascher (1993) describes some of these projects and the associated
forecasting failures in detail. These reflect conclusions derived from an extensive
those that are determined by the confidence limits associated with uncertainty may be
either social or methodological in nature. The former reflects some degree of
calculated manipulation of forecasts in pursuit of a particular political goal.
Methodological problems are technical and exist independent of the social dimension;
evidence for this exists in private sector projects where deliberate manipulation tends
to be detrimental to both profits and careers, but large variances still occur. This is
discussed further in the ephemeral context of software estimation in 2.7. Arguably the
technical shortcomings may have sometimes facilitated the uncertainty which enables
the political dimension to flourish. This view is supported in research by Jones et al
(1997) who provide evidence that in situations where forecasts are regularly subjected
to political influence, rigorous evaluation of forecast accuracy is less likely to be
encouraged.

Modern forecasting methods may be regarded as management tools that should be
capable of supporting 'what if' decisions. This may be considered as a form of
simulation. In the software maintenance domain, for example, what influence will the
expertise of the engineering team have on the effectiveness of maintenance
operations? Generalising, forecasting is perhaps most useful when it is used to
predict how current actions will affect future outcomes.

Methodological problems that tend to provide over-optimistic forecasts can be partly
attributed to inadequate recognition of risks and inadequate provision against the
occurrence of those risks. The dual nature of risk encompasses uncertainty, the
probability that a risk might occur, and effect, the potential impact on the project objectives if the risk materialises. Hillson (1998) defines risk as:

"any uncertain event or set of circumstances that, should it or they occur, would have an effect on achieving the objectives"

Particularly problematical is the assignment of contingency to accommodate low probability, high impact risks. Charette et al (1997) describe the approach to this issue which the US Navy has adopted for software maintenance, in which each risk is rated “as to its perceived severity and likelihood of occurrence using a simple low-medium-high scale for a risk’s consequences and an improbable-probable-certain scale for its likelihood”. Risks can then be prioritised and mitigation strategies devised. An example of the need to develop risk management methods in support of software maintenance that relates to a multitude of projects over decades, was provided by the perceived Year 2000 threat to software.

Despite the imminence of 2000, the threat went largely unrecognised until comparatively recently, and was described in the US on The National Bulletin Board (1997) as “the most expensive repair job in history”. It also predicted that the related litigation will have become a major growth industry, possibly exceeding the maintenance costs. Expert opinion is for world-wide maintenance costs in the $400-600Bn range, although some forecasts are much larger, reflecting the uncertainty and potential impact attaching to the phenomenon. A review by the Financial Times (I) (1998) reported that the likely UK costs had risen to £31.8Bn and also considered the possibility of a recession, asking “against a background of increasing computer system failures, what assets will investors want to hold?” Although with hindsight perhaps unlikely, this assertion reflects the limited aggregate application of structured and quantitative risk assessment. In any given application the risk arising from the threat may be perceived as small, but the impact could be great if it materialised. Uncertainty exists because the design limitation may be present in the assets of an organisation ranging from business computing and communication systems through to industrial and automotive control systems. The ability of an organisation to accurately forecast and budget for the maintenance of their software systems in the face of this uncertainty, possibly perceived low risk but potentially high impact, is problematical.
An analogous problem may be developing in the UK at present, arising from the advent of Economic and Monetary Union (EMU), which already involves most members of the European Union. Irrespective of eventual British participation in the project, the venture will affect systems used by both business and government. The possible denomination of British shares in Euros is an example. Again uncertainty exists, although perhaps of a lower order than in the Year 2000 case because the affected systems are more identifiable, risks may either be unrecognised or seen as small, but the impact of not being able to accommodate the changes in the environment may be great. Accurately forecasting resource requirements beyond the short term in this context is likely to be difficult. Interestingly the Financial Times (2) (1998) reported a warning from the Bank of England that the UK financial sector will be unable to prepare for an EMU entry date prior to 2005 because “banks will not even start getting their computer systems ready for the full-scale introduction of the Euro until they have dealt with the millennium bomb”. Apparently the banks have recognised that the coincidence of these two superficially unrelated events will exceed the maintenance capacity at their disposal. Whilst political pressures doubtless exist, the effects of EMU can be mitigated or delayed, whereas 2000 could not. Given limited funds, and a finite supply of the necessary skills, the software maintenance dimension may play a pivotal part in any progress toward possible British accession to EMU. The Financial Times (1999) noted that many companies are showing little interest in ‘Euro-compliance’ issues “because many have their hands full with the Y2K problem and other urgent projects”. One challenge for the software project manager, whether engaged in development or maintenance, is to provide useful estimates to the organisation, facilitating effective budgeting and planning. This chapter explores the management processes and tools which have been developed in response to this challenge.

Considerable research already exists in the fields of software development methodologies and software project management. One important element is software estimation, where many tools and methods have been produced which relate to program size and complexity. Reference is also made to the importance of maintainability and the related costs. Extensive research has been undertaken into maintainability prediction by Coleman et al (1995) for example, and related issues such as the work of Khoshgoftaar & Lanning (1995) on the identification of software modules most likely to require intensive maintenance. Also considered are Schach’s (1994) review of the effect of software reuse on maintainability, and the comparison
of software maintenance with redevelopment by Sakthivel (1994). These papers, and other related work, will be discussed in this chapter.

The maintainability of software and the associated costs possess characteristics that are related to its reliability in that they are also affected by the development method and the frequency of new releases. Over the working life of a program, which may be measured in decades, maintenance costs can be very significant. The ability to understand these issues at the initial specification stages is important to both the technical and commercial viability of the solution and the costs incurred in attaining it. The relevance of the literature to developing a better understanding of this requirement will be considered, as a basis for further research. Issues of particular interest include the definition and bounding of software maintenance endeavour, the overall scale of the activity, and the effectiveness of methods that may have been developed to characterise, quantify and predict maintenance requirements.

An initial review of definitions and their implications is provided in the next section.

2.2 Definitions and Discussion

In this section some commonly used definitions of both software maintenance and maintainability are considered. The distinction between software maintainability and complexity is also explored.

For those organisations engaged in the development and support of large software systems, software maintenance and maintainability are concerns because of increasing resource requirements and the resultant costs. Several distinctions can be drawn between software maintenance and the preceding software development phase of the lifecycle. A major difference arises from the impact of users and environmental changes on the maintainer, who may have to react to a continuing series of requests for change, creating queues and possible conflicts of priority. These changes must be reconciled with design of the system as originally developed. Banker et al (1993) observe that software maintenance is "fundamentally different from new systems development in that the software maintainer must interact with the existing system". The developer, by contrast, is responding to a requirements specification, which although not static, is likely to be comparatively stable. Alongside any technical challenges that user requests may provide is a management task, reconciling the
needs of the users with the optimum deployment of maintenance effort. A second key difference is the *a posteriori* nature of maintenance, which may require the maintainer to infer the development intent from the available documentation unless the developer remains accessible, and to address systems which may not use current methods or tools. A third essential difference lies in the operational nature of maintenance; time may be critical and defects will impact the application. In summary, the maintainer is constrained by the limitations of the existing design and operational imperatives, whereas software development may tend to allow greater freedom within cost and timescale limits.

Coleman et al (1995) define software maintenance as:

> "The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment"

This is very similar to the IEEE definition which is discussed by Bennett (1996). A slightly different perspective is offered by Hinley (1996) who defines software maintenance as:

> "The set of both managerial and technical activities that ensure that the IT function continues to meet organisational and business goals in a cost effective manner"

One interesting aspect of this definition is the explicit recognition that this is also a management activity, as opposed to being purely technical in character. Practical experience of such issues as organisation, training and configuration management can confirm the importance of this management emphasis. Potts (1993) supports this experience, noting that "not all interesting issues that arise when large, complex systems are designed and delivered are technical ones" and observing that "all the real problems around here are people problems". Use of "all" may be contentious, but people and the uncertainty attached to both individual and team performance is an issue. Similarly Glass (2000) believes that "behavioural team aspects are somewhat more important than the technical ones", while Hetzel (1995) focuses on the management dimension, observing that "projects were found to fail almost without exception because of organisational and goal problems". The Hinley (1996)
definition places the functional responsibility on IT; while this is probably correct in
that it is important to identify a single owner for the process it is perhaps worth noting
that other parties have a role. Apart from the management role recognised above,
another important group are IT's customers, the users of the system who must be able
to recognise and communicate both failures of the system and requirements for
adaptation. The alignment of the maintenance process with the business goals is a
useful addition to the definition. The weakness with this definition is the absence of
any specific description of the activity which involves making changes to software.
Additionally, the definition could be generalised to refer to software producer or
supplier, rather than "the IT function", without detriment to the meaning.

Although Vallabhaneni (1987) defines software maintenance simply as:

"The set of activities that result in changes to the originally accepted
(baseline) product set"

he recognises the existence of three categories of maintenance; corrective, perfective
and adaptive, which is consistent with the view of Coleman et al (1995). These
categories are also recognised by Arthur (1988) who relates them to the software
lifecycle and describes them as software evolution activities. He also points out that
in practice much of the work will be performed concurrently; for example, enhanced
functionality and quality improvements might be implemented and tested together.
Shooman (1988) also notes that it is often impossible to distinguish between effort
expended on redesign, which he defines as enhancement, and maintenance, which he
defines as fixing errors during the operational phase. Shooman's (1988) redesign can
be seen to be related to Vallabhaneni's (1987) perfective and adaptive maintenance,
and his maintenance, which relates to the latter's corrective maintenance. These
distinctions are explored further in 2.4.

Perhaps the most succinct definition is provided by Boehm (1981):

"The process of modifying existing operational software while
leaving its primary functions intact"
Sommerville (2001) distinguishes between maintenance and re-engineering, defining the objective of the latter as being:

“To produce a new, more maintainable system”

This is useful in that it emphasises the difference between modifying an existing system and producing a new one, although in practice the line between an extensive programme of perfective maintenance and re-engineering may be fine. Potts (1993) opines that “most development is maintenance”, and asserts that “system evolution is so common that greenfield development...is the exception”. When does a system become “new”, and does this distinction matter to the maintenance practitioner charged with keeping the system both on-line and relevant to organisational requirements? Personal experience suggests that in an environment influenced by rapidly increasing customer requirements and expectations, the adaptive dimension to maintenance can become dominant, embodying within it elements which would otherwise be regarded as development. The distinction does have value, however, both in recognising the difficulty in bounding the scope of the activity and in understanding the overall scale of software maintenance. Sommerville (2001) also provides a helpful discussion of the distinct attributes of software development and reverse engineering.

Returning to the earlier discussion of risk management and resource estimation, an important input to the size of a particular software maintenance requirement may be the inherent maintainability of the software.

Coleman et al (1995) define software maintainability as:

“*The ease with which a software system or component can be modified to correct faults or other attributes, or adapt to a changed environment*”

This can be recognised as consistent with their earlier maintenance definition and also with the more general definitions of maintainability, which were reviewed in Chapter 1. One minor difference of emphasis as compared with Patton’s (1980) general definition is the latter’s explicit reference to economy, in addition to the ease that they both identify. A system could conceivably be easy to maintain without
necessarily being economical, whereas economical maintenance is probably both easy and more desirable. Measurement of this maintainability attribute is discussed in later sections of this chapter. The inherent nature of maintainability and its close relationship to the system design was discussed in the context of the general definitions provided by both Patton (1980) and Moss (1985), and can be developed further in the software domain by introducing the concept of software delivery complexity. This is defined by Stark & Oman (1995) as:

"The degree to which characteristics that affect maintenance releases are present"

and by the IEEE as:

"The degree to which a system or component is difficult to analyse or explain"

It is important to recognise the distinction between maintainability and complexity. Whereas the latter focuses primarily on the design and related system characteristics, the former also addresses the broader range of factors that might affect the maintenance process. These factors will include characteristics relating to the management process, personnel and the environment, and might include aspects such as access to documentation, staff experience and the availability of suitable tools. Complexity, as defined, can be viewed as being more fundamental in that it addresses those features of the product which determine maintainability, whereas maintainability is more comprehensive in that the non-product factors which are relevant are considered. Confusingly, many writers in the field tend to use the terms interchangeably, when discussing this expanding field of activity.

Clarity is also required in recognising that the complexity dimension discussed above pertain to system related issues such as software, hardware and documentation. A separate source of complexity may be generated at the product level if a number of different releases, variants or options are available to different users simultaneously. The imperative in this situation is for configuration management, requiring accurate knowledge of who is operating each particular variant, and the current build state of these variants. This is a pre-requisite of effective maintenance management, and
provides a useful illustration of the interdependence of associated quality management disciplines.

In summary, this section initially considered definitions of software maintenance and noted several distinct reasons for the requirement to exist. The process, organisational and human aspects that are reflected in the definitions were also noted; technical issues appear not to be the only concerns. The concept of software maintainability was explored, and the distinction between maintainability and complexity was also discussed. It was concluded that the former is a broader concept that encompasses process attributes in addition to product characteristics. Within a software maintenance organisation this may be an important issue, in that process as well as product may influence the required maintenance effort.

The next section considers the scale and nature of software maintenance.

2.3 The Scale of the Problem

Shooman (1988) cites sources which indicate that 30 - 80% of data processing budgets are being spent in support of software maintenance, as defined. Other estimates are that maintenance consumes 40-70% of software engineering resources according to Sakthivel (1994), from Schach (1994) who states that maintenance absorbs 67% of software budgets, from Hops & Sheriff (1995) who calculate that 60-80% of total software costs are related to maintenance and Bennett (1996) who believes that 40-90% of total lifecycle costs are attributable to maintenance. Part of the uncertainty over the scale of the activity arises from the particular definition of the activity which is chosen, and more specifically because these estimates have been synthesised from a wide variety of individual inputs which themselves reflect different attitudes as to what constitutes maintenance. If, for example, the distinction drawn by Sommerville (2001) between re-engineering and maintenance is disregarded, then a larger estimate of maintenance activity might result. Although the precise figure is indeterminate, it is large and was estimated by Taylor et al (1996) to be costing £26.5Bn annually in Western Europe alone by 1990. These estimates are supported by Pressman (1997) who also predicts that maintenance will absorb an increasing proportion of budgets during the 1990s, and by Sommerville (2001) who states that "maintenance almost certainly remains the single most expensive software
"engineering activity". This increase can be ascribed to the expanding volumes of ageing software in use and the need for adaptation arising from changing requirements, and the costs and timescales associated with developing new software. This latter point is made by Sakthivel (1994) and might be extended to encompass both the risk associated with development and the absolute availability of suitable skills and other resources. An additional factor, as Schneidewind (1987) recognises, is that much of the software in current use was developed before structured programming techniques became commonplace. Coleman et al (1995) estimate that at least 75% of existing software was developed prior to structured programming. This area of engineering endeavour may be described in summary as being costly, dynamic and not fully understood; the words of Boehm (1981) still have a resonance when he states that "for an activity which consumes so much of the total software lifecycle dollar, relatively little is known about the software maintenance process and the factors which influence its cost". Although research has been pursued in the interim, the overall scale and complexity of software engineering has also grown and with it, an increasing legacy of software which is subjected to maintenance.

The nature of the problem is continuing to change as the extent and number of applications deploying software solutions increases. This is being driven both by advancing technological capability, and by social and economic change. For example, privatised utilities are offering a range of services which encompass both information technology and telecommunications, while the military undertake training on virtual battlefields supported by synthetic environment techniques. The latter reflects both a technical capability and the higher costs of conventional training methods. The power, applications and connectivity available through a desktop, or laptop, computer is increasing rapidly. Eltis (1998) asserts that "the continuing trend for microchips of a given size and cost to offer ever greater performance all the time widens the nature of the operations the new IT technologies can be designed to execute". This power has enabled complex software to be developed, distributed and maintained in dispersed circumstances far removed from the centralised data processing function of earlier decades.

An increasingly popular distribution medium and perhaps the most significant contemporary development is The World Wide Web. Although already a huge and powerful resource, the current limitations of The Web, which Bank (1997) describes as being "a grand collection of electronically linked brochures", will gradually
diminish as more bandwidth becomes available. Internet trading or e-commerce is rapidly becoming an accepted and routine mechanism for effecting a commercial transaction, whether the retailer-consumer or business-business variety. Wheatley (1999) cites predictions of huge growth, with retail transactions growing from a then current $8 billion of revenue to $108 billion by 2003. More spectacular still, intra-business transactions are expected to grow from $43 billion to $1.3 trillion over the same period. Although the figures might be challenged, little doubt exists that substantial growth in this area is probable, and with it the dependence on the quality of the software supporting these transactions. From the perspective of the potential end-user, the provenance of the software that may be downloaded could be uncertain; the originators may have developed it for their local purposes and have no commitment to maintaining the Web version. Alternatively, from the perspective of the originators, if distributed via the Internet for commercial reasons, they may have to provide maintenance support to a physically dispersed group of end-users with varying needs and capabilities.

A parallel and alternative scenario arises from the development of the application service provider or ASP who can create or provide paid access to application specific software through the Internet. Small organisations can access software that they perhaps would not otherwise have had the resources or skills to deploy, while larger organisations have the opportunity to concentrate on their core competencies rather than managing an IT function. An analogy with present-day electricity supply may be drawn, with traditional hardware and software companies performing a similar role to that of a generating company, with the ASP undertaking distribution. Extending the analogy, it may be that in the future end-users will be no more likely to manage their own computer systems than they would generate their own electricity. The maintenance issue still remains, however, but has been transferred from the end-user back along the supply chain to the ASP.

Other commercial and ethical dimensions, such as the availability of open source software (OSS) may also affect software maintenance as currently practised, in the future. Dettmer (1999) provides a useful overview of this particular aspect that he regards as “a direct challenge to the whole of the traditional software industry”. The present definitions and understanding of both software development and maintenance may become progressively more unsatisfactory in an environment where “the free distribution of source code means that users are able to modify code to make it do
what they want”. “Free” in this context means freely available, rather than necessarily being free of charge.

For the end-users, utilising software obtained from Internet sources, whether on current commercial terms or on an open source basis, is a potential source of both advantage and risk. The advantage accrues from shortened timescales and reduced costs that may be available. Risk arises from the uncertainty that may attach to the functional performance of the software, and the availability and quality of future maintenance, and the associated costs. Consequently an allowance for the maintenance of such software may be included within a larger maintenance budget if it is a component of a total system.

In summary, four principal dimensions require analysis: technology, cost, time and resource. These factors should be considered during the risk assessment phase of a project feasibility analysis. Note that the conclusion of this phase provides confirmation that the project is likely (or not) to be feasible, but does not necessarily quantify the degree of functionality required, the timescales and costs which are implicit or the quality which is likely to result. Further, more detailed planning may be needed, recognising the requirement (what is the product meant to do?), the process (how is it going to be done?) and the personnel aspect (are sufficient people with appropriate skills available?). Putnam and Myers (1997) correctly identify the repeatability of the process as being essential to accurate estimates of project durations and costs. The importance of this attribute is reflected in the Capability and Maturity Model (CMM) which identifies repeatability as an organisational characteristic of effective software organisations. Although the primary aim of CMM has been software development, similar considerations apply to software maintenance estimation.

Assessing maintenance effort during the project feasibility phase, and designing for maintainability throughout the development lifecycle, can be seen as critical to operational effectiveness and possibly commercial success. Software maintenance, with the exception of corrective maintenance that arises from earlier failures, should not be regarded as either undesirable or avoidable. Lehman (1980) recognised this fact when he formulated his laws of software evolution which state firstly:
"A program that is used in a real world environment necessarily must change or become progressively less useful in that environment"

Essentially this law says that only useful software will be subjected to maintenance, and that it is therefore a desirable attribute of a program. More recently Glass (1998) observed that "modern methods lead to more maintenance because the resulting software products are more maintainable and easier to enhance". An interesting paradox which results is that the modern methods that facilitate the more rapid development of ever more software, some of it to replace previous generations, also enables the useful life of this software to be extended because maintenance is easier. In the hardware domain, higher productivity and lower unit costs have frequently lead to shortening product lives, whereas the software parallel affords the potential for the opposite. The prospect of responsive and economical maintenance is perhaps a key discriminator which software provides. Lehman (1980) does, however, identify a caveat. It is stated in his second law:

"As an evolving program changes, its structure tends to become more complex"

Clapp (1981) who formulated three laws that reflect the dynamics of program evolution during the maintenance phase supports this; the second of these, the "Law of Increasing Entropy" describes the progressive deterioration of a program as more maintenance is undertaken. This area is the source of a second interesting paradox: despite the familiarity with a software system increasing with use and the observation that tasks generally become easier as familiarity grows, software maintenance tends to become more expensive as a system ages. Blum (1995) explores this conundrum in detail, and observes that the maintainer should have an easier task than the developer because "maintainers have a narrower focus than original designer, and there is more experience with the software", but that this advantage is frequently constrained if "knowledge is structured so that the maintainers have restricted access to it". If accepted, this observation leads to the tentative conclusion that the problem is perhaps as much related to the management process as it is to the technology, and aligns with the conclusions already attributed to Potts (1993) and Glass (2000). Estimation of maintenance resource requirements may therefore have to consider management attributes, with potential solutions also perhaps available in this domain.
Parallels between this point and the Hinley (1996) definition of maintenance can be seen.

If the evolution is managed through an effective maintenance process, then increasing complexity can be addressed by deploying resources to simplify the structure and update documentation. Possible measures of the effectiveness of a software maintenance process include the degree to which complexity is being contained, and the prevailing success in completing maintenance tasks or projects with budget. In a commercial environment this can be a source of competitive advantage. Chapin (1993) notes the potential benefits that may accrue if "existing software has been kept good enough during maintenance that management often regards enhancement as more attractive than replacement in meeting the ever-changing user needs in the organisation". Problems will arise when these processes are inadequate, and the cost of continued evolution becomes progressively more expensive and error prone. Sakthivel (1994) highlights corrective maintenance, deterioration and obsolescence as significant sources of cost and also as activities that add no value. These activities, together with other categories of software maintenance are discussed in 2.4.

In this section the assessed level of aggregate resource consumed by software maintenance and related activities has been noted, and continuing influence of social and economic change has been discussed. Four factors that should be considered during a project risk assessment were distilled from this discussion. The section then proceeded to consider the implications arising from the inherently ‘soft’ attribute of a software artefact, and the desirability of maintenance. The need to develop processes and tools to constrain complexity growth and manage resources was also observed.

The various sources of software maintenance requirements and their categorisation are considered in the next section. This is followed by a description of models of the maintenance management process that pertain.

2.4 Aspects of Software Maintenance

In this section some of the forms of software maintenance categorisation that exist are considered, and are synthesised into a single descriptive paradigm. Various kinds of maintenance were discussed within 1.6 as part of a general introduction to the field.
Maintenance activity was divided into two distinct categories, corrective and preventive, and the development of the related management strategies was reviewed. This analysis will be developed to provide an understanding of how the strategies pertain to the maintenance of software.

Software maintenance encompasses a broad range of activities, according to Hops & Sheriff (1995), including "the fixing of errors, making enhancements, adding new functionality, system conversion, training and supporting users, and improving system performance". Clearly some of these activities overlap or will be performed in parallel, as discussed in 2.2, but they can be seen to align with the definitions discussed earlier. Identifying the limits of the activity is difficult, partly because as Taylor & Wood-Harper (1996) recognise "the formalisation of the maintenance processes is less advanced than for the software development process", but also because this is a rather volatile field where changes in one direction provide opportunities in another or perhaps obviate a requirement. The advent of 'open systems', for example, makes the migration of a system from one generation of platform to the next relatively straightforward as compared with earlier difficulties of moving from one proprietary environment to another. Paradoxically, however, this flexibility means that the potential longevity of the software is increased because it is no longer limited by the lifespan of a particular platform, and an increased need therefore exists to adapt the software to changing user requirements. Structured design methods and object-oriented design are further examples of how maintenance activities may be affected by changing methods. A degree of certainty is provided in one dimension by Schach (1994) who proposes that any activity before acceptance is software development, and that any subsequent changes are maintenance; this aligns with the Vallabhaneni (1987) definition described in 2.2.

Returning to the description provided by Hops & Sheriff (1995) above, it can be argued that not all the components of it can be readily assigned to either the corrective or preventive categories identified in Chapter 1. corrective maintenance is described simply by Bennett (1996) as "the identification and removal of faults", by Schach (1994) as "the removal of residual bugs" and similarly by Coleman et al (1995) as "maintenance performed to correct faults". Hops & Sheriff (1995) provide evidence that corrective maintenance is a small part of the total effort being applied to software maintenance, and is a declining proportion; by 1990 it amounted to only 16% of the total. An estimate of 17% is provided by Schach (1994). One notable
attribute of corrective maintenance that probably pertains is the lack of discretion that it affords the maintainer. Faults, once discovered, usually have to be corrected.

A more positive view is provided by Glass (1998) who concludes that "60% of maintenance changes provide solutions, while only 17% fix errors". It intuitively seems correct that if other maintenance tasks are growing because software is being used for longer and requires more adaptation, then the corrective element will become relatively smaller. Another driver might be that with the introduction of better software engineering methods the absolute frequency of faults might be declining, although Hatton (1997) has published statistics which suggest that defect rates have remained approximately constant over the past 15 years. This is a surprising conclusion, given the investment in structured design methods, high level languages and test tools in the interim. Kemerer & Slaughter (1997), for example, conclude that code generation tools reduce the amount of corrective maintenance which would otherwise be required. If the Hatton (1997) analysis can be sustained, it may be explicable in terms of constant defect rates in increasingly complex software, but still result in declining work content because of more sophisticated tools. This is reflected in the earlier work of Tajima & Matsubara (1981) at Hitachi.

As with the more general case, various different strategies can be assigned to the preventive category. These, however, have tended to develop in parallel rather than being a progression of serial developments of the sort discussed in Chapter 1. To an extent these models are over-simplifications in that all industries have not followed the evolution of preventive methods together and, in the software domain, although differing strategies do exist some are more popular than others at any given time for reasons of technology, cost, scale or even fashion. Considering these strategies is complicated by the many writers in the field using a variety of terminology, but this also provides a useful vehicle for exploring the subject. This complication has already been encountered in 2.2. Bennett (1996) sees the purpose of preventive maintenance as being to make the software more maintainable. A difference can be discerned between this intent and the arguably more ambitious objective of general preventive maintenance which is to keep a system or process running, and running within specified limits. Reasons for the more limited software goal relate to the inevitability of further maintenance as the requirement and related software evolve and a probability that across time more faults will arise and require correction.
Bennett (1996) also recognises the existence of two further categories; adaptive maintenance which he describes as being "a consequence of environmental change", and perfective maintenance which is designed to improve system performance. Henry et al (1994) appear to combine both of these categories under the general heading of perfective maintenance which they describe as "adding functionality to a software product after it has been delivered to the customer". Both Sakthivel (1994) and Schach (1994) see the situation differently in that neither recognise preventive maintenance as a discrete category, although they concur that the corrective element must be recognised. Sakthivel (1994) argues that two further categories exist, adaptive and perfective, while Schach (1994) prefers to recognise only one other category which he calls enhancement, but this is further sub-divided into adaptive and perfective components. Adaptive maintenance is performed to respond to changes in the environment within which the software operates; an example could be a modification to a sales order processing program in response to a change in the rate at which VAT is levied. It might also occur because of a change of IT policy requiring an existing application to be replatformed for use in a new environment. Sakthivel (1994) describes adaptive maintenance as being "related to changes to data, files, hardware and software". He sees perfective maintenance as associated with enhancing software features and improving documentation, and Schach (1994) describes it as involving changes which users will see as improving the effectiveness of the product. Both these writers would presumably argue that the preventive maintenance activity can be assigned to either the adaptive or perfective category, depending on its precise nature. These apparent differences of view require some reconciliation. It is worthwhile recalling that Coleman et al (1995) defined maintainability in terms of the ease with which performance can be improved or adaptations to a changed environment can be implemented. This definition pulls together the strands of preventive, adaptive and perfective maintenance, and distinguishes the preventive dimension through its reference to the ease with which change can be implemented. One difference between preventive maintenance and the other three types which have been discussed is that it addresses a secondary feature of the software, relating to improvements to its future maintainability or reliability, whereas the others address primary features related to current functional requirements or performance. For this reason it is useful to regard preventive maintenance as a discrete category. This view is supported by Fenton & Pfleeger (1997) who assert that the "purpose of preventive maintenance is to fix problems before the user experiences
them”, and by Lano & Haughton (1993) who see the objective of preventive maintenance as “modifying a program to improve its future maintainability”.

Another maintenance dimension is proposed by Alkhatib (1992) who identifies “ongoing support” as a discrete activity and defines the purpose as being “to increase the effectiveness of communication between DP and end-user personnel”, and cites examples such as “explaining system capabilities and planning for future support”. For the purposes of this study the definition is perhaps too narrow, with the relationship being at the more general supplier/customer interface rather than explicitly involving a DP function; additionally, many of these can be subsumed into the adaptive, perfective and preventive categories without detriment subsequent analysis of the variables involved.

In summary, four distinct categories of software maintenance have been identified; corrective, preventive, adaptive and perfective. This compares with the corrective and preventive categories which exist in the general case. As the software domain is a subset of the general this situation needs to be understood. An association exists between Meredith’s (1992) corrective definition and the descriptions of the corrective process provided above. All other forms of software maintenance must therefore be accommodated within the preventive definition. Already observed above is that the objectives of the preventive maintenance are more limited when the software is compared with the general case. This leaves the question of explaining how adaptive and perfective maintenance relate to preventive maintenance. Referring back to the preventive definition in Chapter 1 provided by Moss (1985), it relates to the deferral or prevention of anticipated failures. These two remaining categories of software maintenance address the current performance of the system in terms of what the user requires and the environment demands, and both have the objective of ensuring an adequate service continues. To this extent they are deferring or preventing failures and therefore are part of prevention.

In this section the nature of software maintenance has been examined in more detail by considering the various categories of maintenance that researchers in the field have identified. An attempt has been made to reconcile some of these definitions, and also to relate them to the more general maintenance case discussed in the introductory chapter. It was noted that corrective maintenance represents the smaller part of total software maintenance activity, but that it is more likely to possess a non-elective
attribute than the other categories. Implications of this distinction for further research and management processes are the importance of understanding probable corrective maintenance costs early because of their 'fixed' nature, and also accommodating 'what if' questions arising from non-corrective maintenance where more discretion can be applied.

The success of an organisation in realising software maintenance objectives will depend on the particular circumstances and approach, and crucially on the rigour and robustness of management processes. As noted in 2.2, both Potts (1993) and Glass (2000) regard the problems associated with software maintenance as including organisational and human issues, rather than being only technical in nature. A discussion of the software maintenance process is pursued in the next section.

2.5 Software Maintenance Management

In this section the software maintenance management process is considered in more detail. Objectives include the review and classification of descriptive models that may exist, recognising commonalties of view where they exist, and discussing differences.

Hinley (1996) recognises two essential objectives in the management of the software maintenance process as being to satisfy customer requirements and adhere to quality standards. Although appropriate, these objectives apply to most fabrication and maintenance activities. Bennett (1996) identifies the imperatives associated with the maintenance activity as requiring quick and cost effective change, and no degradation to either the resultant reliability or maintainability of the system. Decomposition of these imperatives may identify attributes including the ability to estimate how easily software can be changed, the immediate availability of the new functionality upon release, and no degradation to the retained functionality. A stable and risk-sensitive maintenance process could also be recognised as both a prerequisite, and a discrete and desirable attribute.

Hinley's (1996) definition of software maintenance recognises the management dimension of the discipline. Researchers in the field have produced various descriptions of the management process. West (1996) recognises five stages to the maintenance process; change analysis, identification, implementation, test and
distribution. This process description provides an overview of the stages from a high level, suitable perhaps for an introduction to the field but perhaps requiring a second level that provides further detail. A similarly high level overview of the process, adapted from the work of Sommerville (2001), is shown in below.

![Maintenance process overview](image)

Figure 2.1 - Maintenance process overview

The development of a Context Model which considers the generic software maintenance process and, in addition to addressing the strategies discussed in 2.4, also recognises seven management elements within the scope of the maintenance process is described by Hinley (1996). These range from strategic systems thinking which considers organisational goals and customer objectives, through to software product release control which extends to the user interface and associated acceptance issues. This is a useful approach because it places the software maintenance process within a broader organisational context. A greater degree of granularity is provided by Henry et al (1994) who describe a similar process in eight stages. These are narrower perspectives than that provided by Hinley (1996), and relate to maintenance operations within one division of a large equipment manufacturer. The major benefit to be derived from the model is that it describes one instantiation of what other models treat as a more generic process, but probably only applies in totality to other similar organisations. Calibration of both descriptive and statistical models to the attributes of particular organisations is an issue that consideration of this particular model prompts.

The change analysis stage present in most of these models is related to impact analysis, which is discussed in detail by Turver & Munro (1994), who describe it as "the assessment of the effect of a change on the overall system". They provide a 15-element model of the stage, which provides an idealised description of the activities involved. This generic process could readily be expanded into a procedure relevant to
the peculiarities of a specific organisation. Again the emphasis is placed on the organisational and managerial dimensions, rather than the purely technical. This is a crucial stage in that it is the opportunity to assess the resource, cost and risk aspects of the proposed change. Turver & Munro (1994) emphasise that this assessment should not be confined to considering possible effects on existing source code, but must also address documentation. Another key area is user training, which can range in importance from major to irrelevant, but should be considered.

Middleton (1995) argues that conventional approaches to maintenance have failed to satisfy the user, and that higher quality and productivity could be attained by adopting a management strategy based on 'just in time' (JIT) principles. He argues that in some software maintenance organisations the objectives have degenerated through the imposition of onerous documentation requirements, hindering innovation and alienating engineers. Since software maintenance is a systematic process, JIT principles can be applied to facilitate a more responsive organisation whilst reducing work in progress and providing a more creative working environment. This approach has merit and is consistent with other work relating to the application of JIT in service organisations. Meredith (1992), for example, describes a JIT environment that has been established in a direct marketing operation. It is also consistent with processes derived from the management models described earlier in this section, and can be regarded as a philosophy that can be usefully embedded in any given instantiation of the software maintenance process.

In this section several different methods of describing the maintenance process have been considered. A recognition that certain of these models are complementary has been acquired. They could perhaps be categorised as overview models that provide an introduction to the process, detail models provide more information on the process or a part of the process, and instantive models that address an implementation in a particular class of environment. The issue of calibrating models, whether wholly descriptive or statistical, to a particular environment is an important observation derived from consideration of these models.

Irrespective of the particular management approach that is adopted, it remains essential that the risks, costs and general viability of a maintenance project can be understood at the outset. Options such as replacing software with a wholly new development, or reusing existing software from another implementation might be
explored to confirm whether a proposed maintenance strategy is optimal. These alternative approaches are considered in the next section.

2.6 Software Redevelopment and Reuse Strategies

Given a need to ensure the continuing availability of a particular software utility over an extended period, it is likely that changes of user requirement, and therefore continuing software evolution, will occur. In this section one particular means of evaluating the merit of continuing maintenance through the processes discussed above is considered, as opposed to starting afresh with a new software development. The benefits of reusing existing software are also explored.

In the absence of management intervention, one effect of maintaining software is to degrade its future maintainability because the software will tend to deteriorate with consequences such as data inaccuracy and inconsistency. This degradation is consistent with the laws proposed by Lehman (1980) and by Clapp (1981), and has been considered in 2.3. This management intervention can be regarded as preventive maintenance of the sort described by Bennett (1996) and discussed in 2.4.

Sakthivel (1994) has explored these effects and introduces the concept of deterioration costs which he defines as:

"The increase in the maintenance cost which would otherwise not have been incurred had the software been new"

Taken together with the related concept of obsolescence costs which he defines as:

"The opportunity cost of not using the latest technical developments that reduce maintenance costs"

he has developed an economic model that recognises the combined effect of deterioration and obsolescence will have on maintenance costs compared to new software. This model is used to calculate what is termed the total equivalent annual lifespan costs. These costs are associated with the continued maintenance of an existing software asset to identify the theoretical economic lifespan of the software,
and an *equivalent annual cost comparison* term to compare continuing software maintenance with the alternative of procuring a solution based around current technology. Based on straightforward replacement theory, this approach has the advantage of being easily understood by both engineers and non-technical personnel alike. Sakthivel (1994) recognises that the model cannot address certain non-quantifiable factors such as regulatory requirements. Although, as described, this is correct the model could be extended; for example, safety critical software might require independent testing before it can be qualified and released for operational use. This testing has a cost that could be incorporated into the model. Other limitations, which are identified, relate to competitive pressure and organisational preference. Further practical difficulties may also pertain. For example, estimates of deterioration and obsolescence rates must be made; the conclusions offered by the model depend directly on the values selected, and assessing these values may prove problematical. Additionally, in a budget sensitive environment it may be necessary to continue with a sub-optimal maintenance policy if the capital funding for replacement is unavailable. In practice, and particularly within larger enterprises, there may also be an organisational dimension to replacement decisions. Once a maintenance team of given size has been established to support the software asset, social or political pressures within the organisation may create an inertia in support of continued maintenance despite a replacement analysis recommending an alternative strategy. This issue relates to what Boehm (1981) describes as the "*level of effort*" approach to software maintenance management, and serves as a reminder that modelling and management tools are but one input into a more complex decision-making process.

One possible management strategy that could reduce the impact of deterioration and obsolescence, and delay the point at which continued maintenance becomes uneconomical is to reuse software. Much analysis of the effect of reuse on development has been undertaken, but comparatively little on its relationship with software maintenance. For example, Fenton & Pfleeger (1997) describe the application of reuse to software development by NASA and Hewlett-Packard. Despite this, evidence exists which suggests that the economic impact of reuse on software maintenance is greater than that on software development.

Schach (1994) conducted a detailed review of the relationship between reuse and maintenance, and concluded that the potential cost savings during software maintenance as a consequence of reuse could exceed those available from the
development phase when more than about half of the total budget is devoted to maintenance. This conclusion is qualified by a number of assumptions, relating primarily to the comparable effort required to engineer new and reused components, but the general result still has merit; reused software may be more easily understood by a software maintainer, providing an opportunity for higher productivity and lower costs. The risk management issue discussed in 2.1 is also impacted by software reuse. Although not explored in detail by Schach (1994) it is likely that reusing proven software from a prior application will tend to reduce the level of uncertainty associated with a particular project.

One obvious weakness with reuse is that it can most readily be applied to future projects, whereas its relevance to the owner of a legacy system is confined to that which can be salvaged from the software. Schach (1994) describes this process as accidental reuse, as opposed to the planned alternative. The implementation of a planned reuse policy is a significant management task requiring the definition and application of software engineering standards within the organisation, the creation of an effective software engineering archive to facilitate future access, training and meaningful metrics. The latter is needed to enable objective 'design for reuse' by imposing complexity constraints during the software fabrication process. Wei Li (1997) describes a study that indicates that the McCabe (1976) cyclometric complexity metric (CC) may relate to the reusability of a given function. Intuition suggests that the greater the comparative complexity of a software module, the harder it will be for an engineer to reuse it and obtain the potential cost advantages discussed above. The notion of comparative complexity is introduced because the potential for reuse should reflect organisational norms; reuse will become harder if the software and supporting information appears more complex than is usual within the organisation. Given the discussion in the previous section of the probable requirement for calibration, it is likely that a threshold value of CC should probably be peculiar to a given organisation, if the approach is to be applied. However, CC is not universally accepted as an adequate measure of software complexity with Fenton & Pfleeger (1997), for example, arguing that it only provides a partial view. Stark & Oman (1995) rejected the measure because they believe other factors such as environment play a significant part. Hops & Sheriff (1995), however, concluded that CC related well to other measures of complexity, including lines of code (LoC). This final point is interesting, both because Wei Li (1997) also observed this phenomenon but also because it is not intuitively obvious; CC is a logic-control complexity measure.
whereas LoC is a function size metric. This is important because LoC is easy to
measure and, along with CC, could provide a valuable 'design for reuse' tool if this
observation could be validated for the general case.

There are some possible analogies that can be drawn between software
redevelopment and reuse issues and the 'cost of quality' (CoQ) concept. Oakland
(1993) provides a comprehensive description of the latter, which sub-divides general
quality costs into the three principal categories of prevention, appraisal and failure.
One theoretical possibility is that a minimum CoQ can be attained by deriving an
optimum mix of costs; up to some limit marginal investment in prevention will
produce marginal reductions in appraisal and failure costs, and similarly more
appraisal investment might result in reduced failure costs. Implicit in this hypothesis
is that some level of failure is acceptable. In practice, and depending on the
application, this may be true, although objective analysis leading to some 'right'
answer may be problematical. Analogies pertain because of common features
including limited resources and time, multiple resource inputs and a customer
satisfaction dimension. In both cases it may be difficult to draw unambiguous
conclusions because of their multivariate and dynamic natures.

In this section variables associated with continuing maintenance and the alternative of
redevelopment have been discussed. It is noted that the redevelopment model may
have potential as a decision support tool capable of providing recommendations
designed to constrain the total cost of ownership of a software asset within a specified
financial limit. Advantages accrue from the comparative simplicity of the method,
although difficulties may arise in assigning values to variables within the model.
Accounting conventions may, for example, be a source of complications. Calibrating
the model to reflect practice within an organisation would be essential. Additionally it
remains necessary to possess a means of assessing future maintenance costs to
facilitate use of the model. As noted earlier in Chapter 2, designing for and therefore
being able to quantify maintainability is relevant to this objective. Reuse of existing
software may tend both to reduce effort and mitigate risk.

Irrespective of the level of reuse, whether accidental or planned and managed through
a 'design for reuse' policy, and also irrespective of the availability of a realistic
redevelopment alternative, questions remain which relate to the effect of proposed
development or maintenance actions on future maintainability. These may have consequences for maintenance costs.

In the next section the general approaches taken to software estimation are explored as a precursor to considering the specifics of maintenance estimation.

2.7 Software Estimation

In this section the generality of the software estimation challenge is considered, with particular attention being paid to the alternative methods that exist and their respective attributes. The section should be regarded as an introduction to the field prior to considering the more specific case of maintenance estimation.

Before attempting to predict the probable effort associated with any particular class of activity on a possible future software project, it is useful to recognise the characteristics and limitations of an estimate. Although a judgement, a calculative attribute reflects a structured approach and therefore a process or method. This is one distinction between an estimate and a guess, the latter being based upon supposition rather than definitive knowledge and calculation. Londeix (1990) extends this distinction and argues that an estimate "pretends to be true within certain limits". This concept is an important one, in that the interval within which some level of confidence in an estimate can exist reflects assumptions which must be understood by those using the estimate, and is a product of a method rather than a guess. In summary, an estimate should be regarded as a probabilistic assessment of an outcome, and therefore lies at the centre of a distribution of possible outcomes. Fenton & Pfleeger (1997) quote a definition credited to Demarco (1982) which supports this view:

"A prediction which is equally likely to be above or below the actual result"

and provide their own definition:

"The median of the distribution"
Fenton & Pfleeger (1997) use this view to assert that an estimate is not, and should not be used, as a target and cite evidence that projects are more successful where no targets are set at all. This appears to run counter to intuition, evidence from the everyday world and personal experience of software projects. Commercial reality usually requires a project to be completed within an agreed timescale utilising a defined level of resource. Whether described as a target, a budget or a deadline that must be met, it would seem essential that the project manager and preferably the whole project team are aware of the boundaries that exist. Even in a non-commercial environment, neither time nor resource are unlimited, so similar objectives remain relevant. No particular reason exists why the median (the estimate) should not be used as a target. Caveats include the need for the project manager to create a contingency plan to accommodate the inability of the team to meet the target, and to possess the resources such that the target is a credible objective for the project team. This standpoint is consistent with the tenth of Deming’s (1982) fourteen points for management which calls for managers to “eliminate numerical goals.....without providing methods”. Target setting is, however, a peripheral concern; estimates remain heuristic, reflecting the best available information based on previous performance and a judgement as to the closeness of the resemblance between the future and the past.

This last assumption of repeatability is critical and closely related to the generally poor reputation that software engineering projects have for achieving, or even getting close to, estimate. Fenton & Pfleeger (1997) describe the performance in this area as notorious and McConnell (1993) describes estimation as “one of the most challenging aspects of software project management”, supported by evidence from the USA during the 1980s which suggests that most large software projects were substantially late and over-budget. An investigation by Van Genuchten (1991) describes a survey of 72 software development projects, which recorded an average effort overrun of 36%, and schedule overrun of 22%. Similarly, Tinharn (2000) reports the results of a UK survey that indicates that a third of all software projects miss their completion dates by 10%, and that one in ten projects overrun by 100%. Reasons cited were “unrealistic deadlines, unfocused objectives and poor communication”. In the software maintenance domain, Henry et al (1996) believe that “project managers struggle to estimate maintenance effort, schedule and cost” and ascribe this difficulty to “the very different types of activities taking place during maintenance”. These problems will be explored in detail in Chapter 3.
The two principal sources of these variances are the quality of the estimates used to set budgets and timescales, and the methodologies and tools used to control the software fabrication and evolution processes, and the resulting project. The second of these is beyond the scope of this thesis but has been the subject of considerable development and is reflected in various standards including ISO9000 and CMM. Repeatability is relevant because the attributes of a new project will probably vary from those experienced on the previous project, or even from those aggregated from a collection of recent projects, if available. More problematical still is the need to discern which of these attributes are important and how they will affect the estimate.

Chatfield (1997) identifies three forecasting methods: judgmental, univariate and multivariate. This classification can be usefully extended to recognise several distinct but related approaches to software estimation, which are categorised in the literature. McConnell (1993), for example, cites nine different methods. However, most of these methods can be allocated to one of four discrete classes consisting of expert opinion, analogy, decomposition and mathematical modelling. The first three of these would be regarded by Chatfield (1997) as judgmental. These classes can each be applied either from the 'bottom up', with estimates for individual sub-systems being aggregated, or 'top down', with an overall system estimate being calculated and apportioned to sub-systems. The choice may reflect the degree to which a system design has been developed at the time of making an estimate. Estimation is usually an iterative process which may well require 'top down' and 'bottom up' estimates during the feasibility and budgeting phases of the project, and continuing re-estimates as the project proceeds. An advantage of the 'bottom up' approach is that the accumulation of multiple small estimates may tend to enable errors in one direction to be compensated by errors in the opposite direction.

Expert opinion is a common approach which is often used in combination with other methods. The approach essentially relies on the accumulated knowledge of an experienced practitioner, or group of practitioners, within a particular environment who act as consultants during the initial estimation phase. They, themselves, may use other methods including models, analogy or other 'rules of thumb', but to the enquirer they remain the provider of an expert view. Analogy is regarded by Fenton & Pfleeger (1997) as a formalised extension to the expert opinion approach, in that it requires structured and documented comparisons to be created. To the extent that the analogies are likely to be made by experts, this may be true; the key difference is
perhaps the application of a visible and auditable estimation process which provides the opportunity to assess the comparative effect various factors may have on the resulting estimate, rather than just providing an answer. Decomposition, it could be argued, takes the process further and requires the project to be broken down into the smallest readily identifiable entities and estimated; these estimates may themselves employ one of the other techniques which have been classified. Allowance must also be made for the connectivity between the entities. Mathematical models may also be used in support of other approaches; a system might be decomposed and a model applied to estimating some components while an expert opinion may be obtained for other elements, for example. These models are valuable tools in estimating software effort, particularly for large systems which may be beyond individual experience or knowledge, and should have the potential to make an increasing contribution to software estimation, whether for initial development or continuing evolution.

Modelling differs from the other three classifications in that they tend to provide a single product in the form of an estimate, whereas a mathematical model has the potential to provide two deliverables. The primary output remains the estimate, but a secondary deliverable is the increasing understanding of the relationship between the input variables and the output, which will accumulate as the comparison of actual results with estimates grows. This provides an organisation with opportunity to learn, and appreciate the factors that are cost critical in their products and processes. Campbell (1998) shares this view, stating that “mathematical modelling should be viewed as an essential learning tool with which we can understand engineering problems”. The ultimate results of this enhanced understanding should be a less wasteful and therefore more effective operation, supported by a refined estimation process. None of the other classifications can capture this knowledge in such a formal and objective structure, although a well-defined decomposition procedure is the closest. Expert opinion and analogy tend to be people dependent with knowledge retention more tenuous, whereas decomposition and particularly mathematical modelling require greater formality.

Despite their potential value, experience with mathematical models for software estimation has been disappointing, whether applied at a high level to the overall project lifecycle, or to the detail of maintaining a specific software module. An example of the latter is the work of Jorgensen (1995). Various reasons are have been suggested for this disparity between expectation and performance, and are widely
discussed in the literature; a synopsis of recent conclusions is provided by Fenton & Pfleeger (1997), which includes model structure and complexity, and particularly the general application of parameters which may not be valid universally. An additional and fundamental criterion they identify is size estimation; size is perhaps the most commonly used input variable in effort estimation models, and effort estimates will therefore depend on the quality of the size estimate. Typically size is quantified using a lines of code (LoC) count, but care must be taken to define LoC for a particular environment as no single definition is universally applied. Other issues include the relationship between LoC and language, assessing the probable LoC in a future project and establishing a relation between coding activity and other effort requirements. Coding is usually the smaller part of the total effort, perhaps below 20% of the aggregate. Other potential sources of difficulty, drawn from personal experience include the rigour with which process data is captured for subsequent calibration of models, and also the training of those engineers using the models. Local calibration and validation of the model to reflect local standards and practice is essential. Sommerville (2001) regards this as important, stating that “cost modelling can only be effective if the model is calibrated to an organisation’s own software development practices”.

Most mathematical models used in this field are resource models which typically predict effort requirements and project durations. Three principal categories of model are generally recognised and have been named by Boehm (1981) as Static Single-Variable, Static Multi-Variable and Dynamic Multi-Variable. Some of the literature refers to these categories by alternative terms provided by DeMarco (1982): respectively Corrected Single-Factor, Corrected Multi-Factor and Time Sensitive. This categorisation may be viewed as an extension to the univariate and multivariate classes previously attributed to Chatfield (1997). Attempts to produce models started with the regression analysis of data from past projects, but other approaches to model building do exist. Alternatives include case based reasoning, learning by induction and neural networks. Case based reasoning uses analogy by utilising cluster analysis techniques to identify past projects with similar attributes, while learning by induction employs decomposition through decision tree tools to identify projects, or sub-projects, with common attributes. In both methods a priori decisions are required about those attributes that determine effort; the effort drivers.
Neural network techniques use a network of cells which receive inputs, for example effort drivers, which are then multiplied by weightings, summed and processed through an activation function to generate an output, in this context an estimate. The power of the technique lies in enabling the system to 'learn' which value of weightings produces results closest to known past outcomes. A difficulty which has been experienced by researchers is in understanding why particular results arise. For example, Hughes et al (1998) found that "there appeared to be no way of investigating the reasoning behind the estimate provided". Although this difficulty may partly arise from limited understanding of either the technique or a particular tool, it nevertheless is an empirical finding which may occur if deployed within an estimation suite. As with the other approaches, effort drivers need to be identified in advance. An interesting development of this approach is described by Lee et al (1998) who used cluster analysis methods to group similar objects in multi-dimensional space, and use these clusters to drive the network. Again, however, the attributes have to be assumed at the outset. Neural networks are discussed further in the context of maintenance estimation in 2.8.

A well-established and conceptually simpler method is that of regression modelling. This technique involves defining the best fitting line or plane in multi-dimensional space such that the sum of the squares of the deviations of the observed values from those predicted is minimised. It is widely used in many different fields including software estimation where models are typically used to relate resource (R) to project size (S) and producing linear equations with the general form:

$$\log R = \log x + y \log S$$

Equation 2.1

In the real domain these take the form:

$$R = xS^y$$

Equation 2.2
An example of this form is provided by Shooman (1988), and credited to Walston & Felix (1977):

\[ E = 5.2L^{0.91} \]

where \( E \) represents effort and \( L \) code length

\textbf{Equation 2.3}

An equation of similar form is provided for the project duration. This is an example of Boehm's (1981) static single-variable classification. In practice, all the equations of this kind have a variety of correction coefficients applied to them in an attempt to minimise the residual error that arises from the regression calculation. The equation consequently takes the general form:

\[ R = xS^yC \]

where \( C \) is the correction coefficient

\textbf{Equation 2.4}

In the Walston & Felix (1977) example, a total of 29 variables were recognised in the correction coefficient. Perhaps the best known model of this kind is COCOMO which was originally designed by Boehm (1981) utilising data from 63 projects and in three modes (organic, semidetached and embedded) reflecting the software product type, and in three versions (basic, intermediate and detailed) relating to the maturity of the design and precision required. It has subsequently been upgraded to COCOMO2.0 in order to reflect advancing methods and technologies, and is available in a 'soft' format. COCOMO takes account of 15 variables to correct the base estimate in the intermediate version. Boehm (1981) also considered maintenance cost estimation within the model; this is discussed further in \textbf{2.9}.

A static multi-variable model takes the general form:

\[ R = aB^c + dE^f + gH^i + \ldots \]

\textbf{Equation 2.5}

Practical experience indicates that this class of model is hard to calibrate, verify and use; it was, for example, disregarded by GEC-Marconi (1993) as being impractical.
The best known example of a dynamic multi-variable model was created by Putnam (1978) using observations obtained from IBM for use by the US Army. It takes the general form:

\[
R = \frac{S^3}{(C_k^3 \times t_d^4)}
\]

Equation 2.6

\(C_k\) is a technology constant and \(t_d\) reflects the development time. Fenton & Pfleeger (1997) describe the difficulties experienced in establishing a value for \(C_k\) and criticisms of the fundamental design. For these reasons it is not considered further.

Most of these models possess an exponential component that accommodates the past experience of a non-linear relationship between costs and project size. For example, as the size of the team increases so the associated management overhead cost may increase. Whether or not this is an inviolable rule in the future is a matter of conjecture. As code generation techniques and configuration management systems become more sophisticated, and project management tools and documentation tools become more closely coupled with the primary engineering tasks, might a contrary ‘economy of scale’ influence pertain, with larger projects absorbing no more management effort than smaller ones? Whether or not this will remove the non-linearity is uncertain, but it is probable the values of the coefficients will be affected. The implications for the model designer are to facilitate flexibility. These observations must also apply to software maintenance estimation modelling, and arguably to a greater extent because more potential variables exist.

Chatfield (1997) warns that it is not necessarily the case that more complex multi-variable models will produce better results and argues for “choosing a method which is appropriate to the situation”. Similarly, Ascher (1978) concludes that “the presumed advantages of sophisticated methodologies simply have not materialised”. In practice, an important dimension must be the level of expertise that is available; in many organisations software engineers or project managers rather than specialist estimators must produce estimates. Simplicity and robustness are perhaps the keywords. Additionally, Chatfield (1997) refers to theoretical evidence that simpler single-variable models are more resistant to the effects of model misspecification.
The prime point noted in this section is the generally poor performance of software estimation in predicting probable resource requirements that is reported by researchers. Reasons cited extend from unrealistic expectations through to the broad range of activities that are involved. Personal experience is also of incomplete specification and poor definition, through inconsistent application of processes and methods, to simple over-optimism. Four distinct categories of estimation approach were identified, including mathematical modelling. The value of this last category lies not just in the generation of estimates, but in two other areas. One is as a learning tool, perhaps capable of more explicitly relating input variables to outcomes and of answering 'what if' questions. The other relates to the lesser degree of dependency on individuals within an organisation. By extension, greater potential for creating an organisational 'knowledge store' may lie in this direction. Several different approaches to mathematical modelling were reviewed, but none appears to generate consistently satisfactory results. With the more complex mathematical methods, comprehension by the user may also be an issue. Simplicity was identified both to assist application and also because evidence exists that the resulting estimates may be more robust.

The application of these methods to predicting maintainability and maintenance effort at both the software module and project levels is discussed in the subsequent sections of Chapter 2.

2.8 Software Maintainability & Maintenance Effort Prediction; Existing Programs

In this section the general discussion of estimation that has been pursued is developed further with regard to the software maintenance domain. The distinct but related areas of maintainability and maintenance effort estimation are both considered in the context of existing software. This is a field where considerable research has been undertaken.

This is an area in which a number of methods have been applied at various levels of abstraction. These can relate to both the maintenance process and the product. Since maintainability depends not only on the structure of the software but also on other factors including the skills of the engineer and the availability of supporting tools and
documentation, the direct approach is to measure the process. Examples of process metrics that might be captured include frequency of corrective maintenance requests, effort required to undertake impact analysis and change request lead times. The alternative indirect method is to identify and measure product attributes and demonstrate a relationship with maintainability or maintenance effort, recognising that a correlation with a particular attribute is not a measure of the characteristic. Either approach has to be calibrated to reflect the characteristics of a particular environment. Henry et al (1994) observe that "the statistical strength of the relationships will vary from organisation to organisation". Fenton & Pfleeger (1997) support this view, distinguishing between what they describe as the external (process) and internal (product) methods. They draw a parallel with measuring usability and argue that it cannot be considered solely in terms of product structure and complexity.

The degree of abstraction is related to the reasons for making the prediction and extends from identifying error-prone modules in existing code through assessing the maintainability, or change in maintainability, at a software module level, quantifying maintenance costs for an existing program to estimating the maintenance costs arising from a future development project. The latter aspect is discussed in 2.9. At any given level of abstraction, the prediction method must take account of the management process and other environmental factors which pertain.

At the lower end of the abstraction scale, various techniques have been applied to assessing changes in maintainability arising from modifications to an existing program. The initial requirement is to identify those metrics which will be incorporated within the model; Coleman et al (1995) describe the use of principal components analysis and factor analysis to reduce or eliminate metric collinearity. The objective is to reduce a set of variables to a hypothetical and smaller number of components or factors that generate variances. Coleman et al (1995) concluded that this method of creating a regression model could produce credible predictions but was too time-consuming for practical application in a commercial environment. This conclusion, however, is confined to the limited objective of a maintainability index, rather than effort estimation. Uncertainty must also pertain as to the genericity of the components or factors; how accurate would predictions be in another application area, or after an extended period of time? Another method of identifying the variables for a regression model is to use existing software metrics. An example of this approach is provided, again by Coleman et al (1995), who created more than fifty simple models
using various metrics and ordered them according to their correlation with an expert's subjective evaluation of maintainability. The conclusion was that the Halstead (1977) volume and effort metrics facilitated the models with the greatest predictive power.

Interestingly, this result was obtained despite the doubts which have been expressed over the Halstead metrics. Both Fenton & Pfleeger (1997), who describe them as "confused and inadequate", and Sommerville (2001) make reference to earlier papers which question their validity. Shooman (1988), however, found his metrics to be sufficiently convincing to justify application within his own work. Coleman et al (1995) obtained results which they found sufficiently promising to justify further research. They generated a three-term polynomial based on a Halstead (1977) effort metric (E), CC and LoC, and applied it in an industrial setting. Their conclusion was that they could be used in the form of a maintainability index to monitor the effects of changes as the software is modified through a series of maintenance cycles. Any part of a system whose index fell below a pre-determined value would be subjected to preventive maintenance. Specifying this threshold value would require calibration of the model, probably using the subjective judgement of a maintenance expert. They also concluded that the method could be applied to code at both the module and system levels. It is doubtful that the particular model which they developed could be usefully applied in a different environment, and it is also uncertain whether E, CC and LoC would remain the most appropriate metrics in that environment.

At a higher level of abstraction, a basic approach to effort estimation is:

\[
\text{Maintenance effort} = \frac{\text{size}}{\text{mean maintenance productivity}}
\]

\textbf{Equation 2.7}

which Jørgensen (1995) used as a baseline to assess the efficacy of more sophisticated approaches to estimation. This is a process-oriented method, which takes no account of code attributes but reflects past performance, as represented by mean productivity.

\[
\text{Mean productivity} = \frac{(\text{LoC inserted} + \text{LoC updated} + \text{LoC deleted})}{\text{effort used}}
\]

\textbf{Equation 2.8}
Other limitations of the approach include defining size purely in terms of code, with no direct consideration given to the effects of the change on the associated documentation; and from a statistical standpoint the use of a mean needs to be qualified with a confidence level. Chatfield (1997), who criticises the concentration on "point forecasts even though an interval forecast is often what is required", supports this latter point. As a baseline, however, the method provides a useful reference.

An approach to identifying variables relating to existing code was pursued by Jorgensen (1995) who selected only those which correlated significantly \((p<0.05)\) with maintenance productivity. Using this method four variables were selected for inclusion within effort prediction models from a set of eleven variables for which information was collected. It is possible, however, that other significant variables exist beyond the eleven which were considered, and will be the source of error in the model predictions. Jorgensen's (1995) experience with six regression models created using these variables was that most of them did not produce consistently better predictions than the simple baseline model and that they did not provide an adequate alternative to the judgement of an expert maintainer. This conclusion, however, relates to the application of regression to resource estimation, unlike Coleman et al (1995) who were trying to create an index rather than arrive at an estimate. Another disadvantage of regression methods which Jorgensen (1995) identified is that they cannot be easily used by engineers to develop their understanding of the maintenance process and perhaps improve the quality of expert predictions. This is a surprising observation and perhaps highlights a training need within the target organisation; one of the advantages of creating any model, mathematical or otherwise, should be the ability to develop a greater appreciation of the interaction between inputs, operators and outputs. In an earlier study Henry & Wake (1991) describe the application of regression methods to data obtained from a single program. This was used "to predict .... the total number of changes during the maintenance phase". Two interesting aspects of this study are that the number of changes likely to be required is seen more as an indicator of maintainability than an effort predictor, and that the metrics used all primarily reflect product attributes. The study concludes that the regression approach has merit, but that significant amounts of data must be analysed. Henry & Wake (1991) envisage a separate regression equation being developed for each different working environment. The addition of a process dimension to their analysis might enable their model to be genericised for wider application.
In summary, whilst regression models have the merit of comparative simplicity, the initial challenge lies in identifying those factors which contribute to generating variances in the maintainability of a particular program.

An alternative approach to effort prediction which Jorgensen (1995) also explored involves the use of neural networks to facilitate the design of non-linear regression models which incorporate larger numbers of variables. Despite this additional complexity, the results obtained from his trials were no better than with most of his other regression models, and were inferior to the best results. Related work has been undertaken by Khoshgoftaar & Lanning (1995) who have applied neural network methods to software quality and complexity metrics, to classify software according to the risk it will represent during the maintenance phase. Unlike Jorgensen (1995), they were not trying to produce an estimate, nor even a continuous variable such as the Coleman et al (1995) maintainability index, but a discrete classification consisting of either low or high risk. Their approach is product-based and avoids some of the uncertainties and inconsistencies associated with collecting process-based metrics. Their conclusion was that the approach has promise but that larger scale studies are required. This view is supported by Chatfield (1997) who states that "assessment of the current state of neural nets suggests that more empirical evidence is needed to establish when they are worthwhile to use". One general disadvantage with neural networks is the probable limited awareness of the principles amongst the potential users; if the objective is to provide simple tools that can be applied by maintenance practitioners, then some training investment may be essential in this case. No discussion is offered by Khoshgoftaar & Lanning (1995) as to the potential for applying neural networks at a higher level of abstraction.

From the above it is apparent that making accurate predictions of the maintainability or maintenance effort associated with an existing program is complicated. There are many potential variables relating to both product and process; the variables may not be independent and their relevance may depend upon the environment or application. The selection of a suitable model and assignment of model parameters is essential to making useful forecasts. Chatfield (1996) identifies three principal sources of uncertainty in a typical forecasting situation: model structure, model parameter estimation and unexplained data variation. He attributes the last of these factors to random variation, measurement error and recording errors. In the software engineering domain consistent data definition can also be a source of error;
differences in terminology and engineering practice can generate large variations. Chatfield (1996) argues that the choice of model structure is fundamental and that biased forecasts will arise when insufficient attention is paid to this process. In practice, the problem for the software estimator, as opposed to a statistician, may be in discerning whether a problem estimate arises from the use of an inappropriate model or parameters, or doubtful data. For example, in the earlier review of the work of Jorgensen (1995), it is unclear where the source of the difficulty lies; in practice it may be a combination of the factors described above.

No explicit discussion of the need for a given degree of accuracy is evident in much of the published work, although measuring the quality of estimates is discussed further in 2.10. There is perhaps an implicit recognition that the provision of useful management tools is of higher priority to maintenance practitioners than the development of maintenance estimation techniques. Readily identifying error-prone modules in existing code, and quantifying relative maintainability, may be more critical than estimating work content. Estimation can be undertaken by a maintenance expert within the target organisation who has the advantages of familiarity with both the software product and the management process. Jorgensen (1995) found that his models could not consistently approach the accuracy of an expert. Problem identification enables management to reduce risk and is the intended deliverable of the work of both Coleman et al (1995) and Khoshgoftaar & Lanning (1995) amongst others, who concentrate on measurement and classification rather than estimation. Several other exemplars of the recent priority given to risk reduction rather than quantification exist. Hops & Sherif (1995), for example, review various approaches to identifying error-prone code, all of which relate to complexity metrics. They conclude that factor analysis and regression modelling can be useful in identifying error-proneness, and provide results which suggest that modules which are complex will require the most maintenance. Kemerer & Slaughter (1997) hypothesise that five factors determine the degree of maintenance that a software module is likely to require: functionality, development practices, complexity, age and size. Using multivariate regression they conclude that complexity and age are key determinants which can be used to identify maintenance-prone modules. West (1996) adopts a two-dimensional approach which measures both the maintainability of the software and the capabilities of the maintaining organisation. An unusual aspect of this method is that it formally addresses both the product and process attributes of a maintenance environment.
In summary, managing risk by predicting error-proneness or the comparative maintainability of existing software has seen progress through various methods including regression modelling and neural networks. Internal attribute structures of the product can be used to create these models. These may be regarded as tools designed to support the management of the process by identifying areas of risk. Maintenance effort estimation is more challenging, requiring rigorous measurement of the maintenance process. Experience suggests that estimation models have not generally provided predictions that are as consistently accurate as the subjective judgement of a maintenance expert. In both the risk identification and effort estimation cases, the selection of variables for inclusion within a mathematical model is critical to the quality of the results. Identifying areas of maintenance risk is probably more important to the management than improving the accuracy of effort estimates when addressing an existing program. This may also be the view of the users of the system who will probably be more concerned to ensure the availability of their system than a marginal improvement that primarily benefits the system provider. Additionally, understanding the internal product factors that affect maintainability may also be regarded as a prerequisite to further development of effort estimation methods. These two imperatives are reflected in the prevalence of work addressing risk rather than effort estimation.

Moving to a higher level of abstraction and considering the feasibility of a future software project, the priorities tend to be reversed. Identifying error-prone modules or assessing comparative maintainability has less meaning, although these methods can be used to establish design rules for the software development phase, whereas the financial viability of the project may depend upon the magnitude of future maintenance costs arising from both software and documentation. These issues are discussed in the next section.

2.9 Software Maintenance Effort Prediction; Future Projects

In this section the variables that may be determinants of the effort required to maintain a software entity are considered in the context of a program that has yet to be developed. The importance of considering maintenance costs at an early stage in the life of a project arises from their substantial contribution to total lifecycle costs. Additionally, it is possible and perhaps likely that those decisions taken during the
development phase will influence future maintenance costs. Less research has been identified that explicitly addresses the estimation of maintenance effort associated with future software, as compared to characterising an existing program using the methods discussed in 2.8. This may reflect the higher priority assigned to risk management that was postulated above.

In considering a future project, the priority should move from purely being concerned with the management of risk arising from poor maintainability to also include the estimation of the effort likely to be absorbed in the maintenance of the resulting system. Many of the factors discussed in 2.8, such as size and complexity, may remain relevant, as do some of the methods that have been applied. More fundamental, however, is the large body of work that has been performed in the general area of software effort estimation and discussed in 2.7; maintenance estimation for future projects may be regarded as a subset of this substantial area. For this reason the COCOMO model has been selected for closer review; it is a widely used tool which has been updated to align with current practice, and has features which specifically address the maintenance phase.

COCOMO provides a set of empirical formulas for estimating software costs, including a family of correction coefficients that can be calibrated to address a particular situation. The model relates the expected level of maintenance effort to the magnitude of the development phase through the introduction of another coefficient, the annual change traffic (ACT) which reflects the level of modification applied to the original software in a year. This is expressed as:

\[
\text{ACT} = \frac{\text{NNL} + \text{NML}}{\text{NOL}}
\]

Equation 2.9

where: 
- NNL is the number of new lines
- NML is the number of modified lines
- NOL is the number of original lines
The maintenance cost is given by:

\[ R_{\text{MAIN}} = \text{ACT} \times R_{\text{DEV}} \]

Equation 2.10

where:
- \( R_{\text{MAIN}} \) is the maintenance resource
- \( R_{\text{DEV}} \) is the development resource

The logic supporting this approach is that the maintenance resource which will be required is related to the overall scale of the project and therefore to the size of the development phase, and by the anticipated level of activity which is reflected in the change traffic variable. Estimating the degree of change for a new project may itself be problematical, with comparison with earlier projects providing a potential source of information, but uncertainty must still exist. Boehm (1981) also recommends modifying two of the 15 correction coefficients or "cost drivers" used in the intermediate and advanced versions, to reflect differences in the nature of the maintenance phase. The RELY and MODP coefficients, which respectively reflect the required software reliability and the prevalence of modern programming practices are given different productivity multipliers. In effect, an effort adjustment factor is being applied to the model:

\[ R_{\text{MAIN, ADJ}} = \text{ACT} \times R_{\text{DEV}} \times \text{EAF} \]

Equation 2.11

where:
- \( R_{\text{MAIN, ADJ}} \) is the adjusted maintenance resource
- \( \text{EAF} \) is the effort adjustment factor

One weakness with this approach is that it takes no direct account of maintainability or the capability of the maintenance process. As observed above, for example in work undertaken by Jorgensen (1995), maintenance costs seem to depend on many factors which need to be accommodated in a metric which reflects maintainability. Sommerville (2001) also recognises this problem and states that "maintenance costs are related to a number of product, process and organisational factors", and proceeds to identify eleven such factors which are shown in Table 2.1 below. These factors are high level in nature and might themselves require several variables to characterise them.
Further dimensions, which do not appear to be considered by Sommerville (2001) in his discussion of these factors, may include the quality of the project management process and the degree of involvement of the customer or user. These issues will be discussed further in Chapter 3.

<table>
<thead>
<tr>
<th>Non-technical factors</th>
<th>Technical factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application domain</td>
<td>Module independence</td>
</tr>
<tr>
<td>Staff stability</td>
<td>Programming language</td>
</tr>
<tr>
<td>Program age</td>
<td>Programming style</td>
</tr>
<tr>
<td>External environment</td>
<td>Program validation</td>
</tr>
<tr>
<td>Hardware stability</td>
<td>Documentation</td>
</tr>
<tr>
<td></td>
<td>Configuration management</td>
</tr>
</tbody>
</table>

Table 2.1 Maintenance Cost Factors (Sommerville 2001)

The nearest COCOMO comes to recognising these factors is through the correction coefficients relating to product complexity and personnel attributes. The differences between complexity and maintainability have already been explored in 2.2. None of these measures are recommended for adjustment to reflect the peculiarities of the maintenance phase. The importance of these variables is recognised in the work of Coleman et al (1995) and West (1996), both of whom attempt to provide methods for creating an index of maintainability. Within a single organisation with a limited range of similar products, and over a short time span, maintainability may not vary sufficiently to justify another correction factor, but beyond this limited scenario it may become a significant phenomenon. It is not obvious that the other thirteen COCOMO correction coefficients will remain the same for the maintenance phase as for development. These concerns are partially recognised in the work of Granja-Alvarez & Barranco-Garcia (1997) who provide evidence that maintainability is very influential in determining software maintenance costs, and then propose incorporating an additional maintainability index ($I_{MAIN}$) term into the COCOMO expression provided by Boehm (1981), giving:

$$R_{MAIN} = ACT \times R_{DEV} \times I_{MAIN}$$

Equation 2.12
This expression can be seen to correct for the maintainability of the software product. Granja-Alvarez & Barranco-Garcia (1997) provide a detailed discussion of an approach to calculating $I_{\text{MAIN}}$, which involves decomposition into three components: understanding, modifying and testing.

This results in:

$$R_{\text{MAIN}} = A \times R_{\text{DEV}} \times (I_U + I_M + I_T)$$

*Equation 2.13*

where: $I_U$, $I_M$ & $I_T$ are understandability, modifiability and testability indices respectively.

They proceed to demonstrate how these indices can be calculated and therefore what the required maintenance resource requirement is likely to be in their example. The method is obviously critically dependent on the quality of the analysis supporting the calculation of the indices. Concerns with their particular approach include the absence of any process related metrics, and the intricate and potentially error-prone computations which are required. Could a simpler method of calculating $I_{\text{MAIN}}$ be applied without reducing the overall sensitivity of the estimation model?

Boehm (1981) appears to have recognised the need for more work, stating at the end of his chapter on maintenance estimation that topics for further research could “develop simplified or more accurate relationships which apply in limited contexts”.

In this section evidence has been identified in support of a hypothesis that maintenance effort may be related to the scale of the software development and the degree of change that can be anticipated. Recognition exists that the factors likely to affect maintenance effort may differ from those associated with software development. In particular maintainability has been identified as a potentially important influence on maintenance costs. Quantification of maintainability is therefore important. Process and product related metrics are both considered as being possible influences. This will be explored further in Chapter 3.
2.10 Estimation Quality

Monitoring the performance of an estimation process is important for many of the same reasons that any process monitoring application exists: primarily to ensure that the process output, in this case an estimate, conforms to an expected standard. Secondary objectives may relate to process improvement. The primary objective is important in the software estimation process because it can provide an early indication that changes among the factors that may impinge on the software engineering lifecycle are affecting productivity. Without re-calibration or other adjustment to the estimation method in this circumstance, it is probable that the estimates and actual results will tend to diverge. Methods of evaluating the accuracy of software engineering estimates typically involve comparison of estimates with actual outcomes, either within a project as it progresses or at the conclusion. The method needs to be easily understood by a user, and provide as much information as possible. In particular, it is desirable and important to discern the effect of outliers that may be the result of special causes. Although these methods tend to address the development lifecycle, the principles are readily transferable to the maintenance phase.

A series of measures have been proposed by Conte et al (1986), starting with the relative error (RE) which compares the estimate (E) with the actual outcome (A):

\[ RE = \frac{(A-E)}{E} \]

Equation 2.14

A weakness with this method is that equal errors above the actual value result in a greater magnitude of relative error than those below the actual value. For example, for an actual of 150 and an estimate of 100 the result is:

\[ RE = \frac{150-100}{100} = 50\% \]

Equation 2.15
If the actual is 100 and the estimate is 150 then:

\[
\text{RE} = \frac{100 - 150}{150} = 33.3\%
\]

Equation 2.16

As the magnitude of the error is the primary concern, this weakness limits the ability of the method to provide meaningful comparisons of estimation accuracy over time between estimates. Trends in estimation performance may be erroneous, and the capacity of the organisation to monitor the process will be constrained. A simple solution to the problem, which is supported by Makridakis (1993), is to divide the error by the average of actual and estimate:

\[
\text{RE} = \frac{A - E}{(A + E)/2}
\]

Equation 2.17

In the above example this method provides an error magnitude of 40%, irrespective of the direction of the error.

If the relative error can be calculated for a series of estimates, then a mean relative error (\(\overline{\text{RE}}\)) can be calculated:

\[
\overline{\text{RE}} = \frac{1}{n} \sum_{i=1}^{n} \text{RE}_i
\]

Equation 2.18

Another measure, the mean magnitude of relative error (\(\overline{|\text{RE}|}\)), can be calculated to obviate the possibility of smaller relative errors on some tasks off-setting larger errors on others:

\[
\overline{|\text{RE}|} = \frac{1}{n} \sum_{i=1}^{n} |\text{RE}_i|
\]

Equation 2.19

It can be seen that the smaller the value of \(\overline{|\text{RE}|}\), then the less the divergence of the estimate from the actual result. Conte et al (1986) suggest that an \(\overline{|\text{RE}|}\) value of less than 0.25 is acceptable, and introduce the concept of prediction quality; \(k\) is the
number of projects or tasks from a total of $n$ whose mean magnitude of relative error
does not exceed $e$. This can be expressed as:

$$\text{PRED}(e) = \frac{k}{n}$$

Equation 2.20

For example:

$$\text{PRED}(0.25) = 0.55$$

Equation 2.21

means that 55% of the estimates are within 25% of the actual outcome. Conte et al
(1986) believe that an estimation method which provides a $\text{PRED}(0.25)$ of 0.75 is
acceptable.

An interesting and fundamental assumption implicit in all these metrics which have
been designed to address the performance of the estimation process is that the actual
result is in some sense standardised, and can therefore provide a meaningful datum.
Consideration of this issue leads full circle to the recognition that it relates to the
predictive power of the estimation model and the degree to which those variables that
affect performance are embedded within the model. Haworth (1996) develops this
thinking further by proposing a maintenance estimation model for existing software
with a coefficient of determination ($R^2$) of 85%, which he argues is sufficiently high
to facilitate control of the process from the estimate, rather than worrying about the
quality of the estimate.

Despite being designed to address effort estimation, it would also be possible to adopt
some of the measures described above to address the maintainability risk assessment
models discussed in 2.8. In this circumstance the objective comparison of estimate
with actual effort would have to be replaced by a comparison of the $a\ priori$
maintainability index with an $a\ posteriori$ maintainability score provided by the
maintainer on completion. Of necessity, there is a subjective element to this possible
application.
2.11 Summary

It is apparent from a review of the literature that the challenges of managing projects to timescale and budget are not new. The twin attributes of estimating resource requirements and controlling risk must be addressed. Various tools, models and methods have been developed, and applied with varying degrees of success. In recent decades there has been a growing requirement to manage software projects, but this activity is regarded as having a particularly poor record of achieving objectives. Estimating software size and effort requirements have been the subject of considerable research. Estimation of software maintenance effort requirements may be regarded as a special case because additional factors may exert an influence. A related area is that of predicting the maintainability of software.

Several definitions of software maintenance have been considered, and some common themes were identified. The activity is generally regarded as occurring after the initial release of the software, and involves improvements that leave the primary intent of the original design unaltered. Many of the problems with achieving software maintenance objectives are believed to relate to processes, organisation and people rather than arising from technical issues. The distinction between complexity and maintainability has been discussed, and the potential impact of the process aspects of maintainability on maintenance costs noted.

The importance of the field arises primarily from the substantial scale and associated costs that can be attributed to it. Several researchers regard this area as the predominant software activity. Furthermore, the inherent desirability of maintenance has been explored, with the majority of maintenance effort being directed to improve the performance of a program rather than merely to correcting faults. Several different categories of maintenance have been identified. Corrective maintenance can be distinguished from the other categories because of the essentially non-elective nature of the requirement. Beyond managing resource and cost though effective processes, the need to ensure the continuing maintainability of the software by constraining complexity growth is recognised as being essential. An ability to measure maintainability is perceived as being relevant both to containing risk and estimating resource requirements.
Several distinct approaches to the measurement of maintainability and estimation of resource requirements have been reviewed. Many researchers regard software estimation as problematical, and to the extent that effective tools have been developed it is important that they are calibrated to accommodate the characteristics of a particular environment. Mathematical modelling is one such approach, and should offer particular organisational benefits if successfully implemented. Experience suggests that the performance of these models is rarely superior to expert judgement. The merits of a simple modelling method have been considered, and relate both to ease of application and robustness. Retention of organisational knowledge may be another benefit.

In the maintenance domain most attention has been given to modelling the maintainability of existing software. This is believed to relate to risk management issues arising from increasing complexity. Less research appears to have been directed at estimating the maintenance effort arising from future software development, but it is believed that an understanding of the impact of maintainability on future costs is a prerequisite. Other determinants of these costs are likely to be the overall scale of the target program and the level of change that is envisaged.

In summary, software maintainability quantification and maintenance estimation are substantial areas of research in which considerable work has been undertaken in the last three decades. Related areas include general reliability and maintenance theory, software estimation and replacement analysis. Some progress has been made in identifying risk prone software and quantifying maintainability, although there is no consensus regarding the variables that exert the most influence. Less success has been realised in estimating maintenance resource requirements and therefore costs, either at the level of existing code or when considering the feasibility of a future project.

Implications for further research in the field must include an examination of the determinants of maintainability. Mathematical modelling is a valuable tool in this area, but the performance of the models that can be developed depends heavily on the identification of the key variables. Maintainability is believed to be crucial to the design of enhanced effort estimation models, which must also reflect scale and degree of change.
The quantification of maintainability is explored in Chapter 3 using data captured from a population of past software projects. Maintenance effort estimation methods are then considered further in Chapter 4.
3.1 Introduction

The methodology described in this chapter begins with the rationale behind a survey of a sample of software projects that have entered the maintenance phase of the lifecycle. This element of the research may be regarded as a continuation of the "informational phase" described by Glass (1994), which was begun in the previous chapter, but leading on into the "propositional phase" in which a "model, theory or solution" is proposed, and subsequently into the "analytical phase" which enables "a demonstration or formulation of a principle or theory". The approach taken throughout falls within the "quasi-experimental" classification provided by Haworth et al (1992), reflecting a relatively low level of control exerted "on the independent variables and on the exogenous variables".

The survey was questionnaire-based and had as its objective the gathering of data that can characterise a particular software project, and specifically relate the many potential input variables to the maintenance outcomes. The chapter then advances into what Glass (1994) describes as the "propositional phase" in which a hypothesis is constructed for deployment in subsequent chapters, and research proceeds to "propose and/or build a hypothesis, method or algorithm, model, theory or solution". The information derived from survey proceeds to be used in selecting variables for inclusion within mathematical models designed to predict maintainability and subsequently for some follow-up confirmatory case studies.

By the end of this chapter a maintainability prediction model will be developed, reflecting the data captured in the survey of past projects. This predictor will then be used in Chapter 4 as an input to a maintenance estimation model.

3.2 Survey Methodology

The survey approach was adopted as being the optimum. Case studies were rejected as an a priori vehicle to gather data because of the need to understand how the many variables interact with different applications and environments. This is consistent
with the Potts (1993) plea is for an "industry as laboratory" approach, with research
driven by "empirical observation of real projects", as distinct from the "research then
transfer" methodology which he describes as "more solution-driven than problem-
focused". The research responds to this plea by being wholly oriented toward
practical experience, with industrial software projects being used to identify
determinants of maintenance performance and individual case studies used a
posterior for confirmatory purposes.

Although the literature gives some indicators as to what the key variables might be,
they are to some extent contradictory and require to be verified. Stark & Oman (1995)
open their paper with the observation that "software maintenance is a multi-faceted,
multi-dimensional effort that, upon inspection, is larger and more complex than it
first seems". Past studies of the application of algorithmic techniques to software
estimating have been described by Lederer & Prasad (1992) as "ineffective due to the
unsophisticated experimental techniques and a shallow view of the nature of
programming", while Sommerville (2001) recognises that "it is difficult to devise
systematic approaches to maintenance cost estimation". Although these are
generalisations they do lend weight to the view that a fresh look at the variables and
outcomes would be useful. For these reasons a survey was designed and
implemented. Prior to proceeding with the survey a definition was sought, and a brief
review of good practice in the field undertaken. Fink (1995) defines a survey as
being:

"a system for collecting information to describe, compare, or explain
knowledge, attitudes, and practices or behaviour"

Description, comparison and attitudes are all aspects of this particular survey. The
mechanism which was used to prosecute the survey was a questionnaire designed for
completion by the recipient, independent of the originator. The majority of the
questionnaires were delivered by either post or email, and returned by the same route.
Recognising the likelihood that "the way you ask questions prescribes the answers",
as suggested by Fink (1995), all the questions are 'closed' and require answers in a
standard format, albeit with the opportunity to provide additional comments. They
seek either some numerical responses to objective questions, or ordinal responses to
more subjective issues.
3.3 Questionnaire Rationale

The questionnaire takes the respondent through a structured series of questions which relate both to input variables and to the maintenance outcomes. The goal-question-metric (GQM) paradigm suggested by Basili & Rombach (1998) was adopted to provide a structure for this research. This choice is supported by Potts (1993) who asserts that "until you know what questions to ask and what to measure, emphasis on quantitative data may be misguided" and describes GQM as "a goal-oriented methodology for determining which metrics to obtain". Although developed primarily to support process improvement, this particular model was readily adapted to assist with questionnaire development. An illustration of this approach is provided below in Figure 3.1. The full extent of the metrics which were explored is discussed in 3.4. Pfleeger (1995) describes the use of GQM in building a metrics plan to facilitate process improvement and observes that "the GQM approach must be supplemented by one or more models which express the relationships among the metrics". In a sense the questionnaire is taking the opposite approach in that GQM is being applied to select metrics which can be used to investigate relationships and then construct a model. Iteration can then proceed and the model is then available to be used, as Pfleeger (1995) describes, as an improvement vehicle.

**Figure 3.1 - G-Q-M Method**
GQM was applied to develop a systematic approach to researching the area, and to provide a range of possible questions. Initial work on developing this structure drew primarily on personal experience in the field, supplemented through reference to earlier work on designing surveys intended to investigate the software maintenance domain. Most notable amongst this work is that of Lientz & Swanson (1980) and Lientz & Swanson (1981). More specifically, the process developed for data gathering by Swanson & Beath (1989) was studied in detail. Key points noted included the importance of utilising a common and standard questionnaire, and the desirability of soliciting objective and quantitative data. This has the dual benefits of minimising the degree of subjectivity during the early stages of data analysis, while also facilitating statistical analysis. One difference of approach compared to Swanson & Beath (1989) was the scale of the survey; their target was limited to twelve organisations that participated in discussions with the researchers prior to undertaking the survey. By contrast this research is intended to address a larger number of target organisations, but without the pre-selection "on the basis of mutual interest" as practised by Swanson & Beath (1989). However, given that the survey results and conclusions were offered to the candidate respondents, by implication some level of interest in the outcomes is likely to pertain in the case of the actual respondents.

Returning to the content of the survey, the ideas of Somerville (2001) on significant factors in determining maintainability were embedded in the questionnaire at this stage. Thereafter the draft questionnaire was developed further with assistance from senior software practitioners. Teams of professional software engineers and software project managers from three separate locations contributed to the design, commenting on the questions, terminology and structure. It is divided into discrete sections pertaining to inputs including the complexity of the software deliverables, design process, project characteristics and maintenance policies; and to outcomes relating to both development and maintenance results. A total of 69 responses are sought.

This design was then tested by applying it to an established and well-understood project, to confirm that the characterisation aligned with subjective knowledge of the project. Finally, the questionnaire was applied to another project by a third party to confirm comprehensibility. The structure is described in detail in 3.4.

The questionnaire was also designed to provide for follow-up interviews with a subset of the respondents. It was envisaged that the interviews would enable any generic
ambiguities in the questionnaire to be addressed, and also allow qualitative observations and causal relationships to be explored. One hour was the anticipated approximate length of each interview. This is again at variance with the Swanson & Beath (1989) approach, which afforded one or two days for each interview. A discussion of the interviews that were undertaken during the course of this research, together with the additional insights provided, can be found in 3.9.

3.4 Questionnaire Structure

The questionnaire, together with supporting notes, is located in Appendix 1. It is divided into seven sections, which are described below.

The first section (1.1) within the questionnaire is designed to address the structure of the code and attempts to capture information relating to size (1.1.1), structure (1.1.2-3) and provenance (1.1.4-5). The relationship between these attributes and maintainability is an interesting and controversial one, which was introduced in 2.8 in the context of analysing the maintainability of an existing program. For example, McCabe’s cyclometric complexity measure is described by McConnel (1993) as “most influential”, but is regarded by Rook (1990) as “very superficial” because all program structures are regarded as equivalent. Similarly, Sommerville (2001) regards the McCabe method as “not really adequate” because it takes no account of the impact of data on complexity. Fenton & Pfleeger (1997) provide a detailed discussion of the McCabe method. Shooman (1988) adopts a simpler approach and demonstrates a relationship between program length and complexity; this is supported by Musa et al (1990) who state that “no one has been able to develop another metric that is consistently superior”. Sommerville (2001) provides a variation on this theme in which the length metric is modulated by a ‘fan factor’ which reflects data flows, but concedes that it has not been independently validated. The questionnaire requires numerical responses, and is supported by a glossary. Size is simply quantified in terms of number of lines of delivered source instruction (KDSI). Modularity is quantified in terms of computer software configuration items (CSCI), and the provenance aspect requires classification of the total module count between new development, reused and third-party components.
The second section (1.2) is related to the environment in which the application has been written, and requires either descriptive statements or numerical responses. Stark & Oman (1995) identify "the code and documentation being produced" and "the maintenance and target computer system environment" as being components which "contribute to the complexity of the software maintenance effort". The documentation aspect is explored in the third section of the questionnaire, which is described below. The environment is defined through identification of the principal language (1.2.1) and hardware platform (1.2.2). The proportion of the available memory typically used is quantified (1.2.3) and architecture is also considered (1.2.4).

The third section (1.3) of the questionnaire has been designed to facilitate investigation of several aspects of the design process which has been followed, focusing initially on the procedures and their suitability. A combination of descriptive and ordinal responses is sought. Responses reflect subjective judgements regarding the quality of specifications (1.3.2-3), design reviews (1.3.4), planning (1.3.6) and user documentation (1.3.5). The section then proceeds to explore the experiential aspect. This reflects the observation provided by Rook (1990) that "the development of each particular software product is a complex intellectual and social process". In this case the responses are wholly ordinal in nature, reflecting subjective judgements being made by the respondent as to the experience of the team. Issues considered include the experience of the design authority (1.3.7-8) and various experiential attributes of the development team relating to domain, methodology and system (1.3.9-14). The final aspect of the design process that is examined relates to the quality assurance disciplines that applied to the project. The relevance of this dimension is described in detail by Hall & Wilson (1997) who provide evidence that "that producing software was easier when quality structures are in place". Again numerical or ordinal responses are sought. Aspects considered include code review coverage (1.3.15), test performance (1.3.17 & 19), defect logging (1.3.20) and configuration management (1.3.21).

The fourth section (1.4) of the questionnaire addresses the project, and attempts to characterise its magnitude, the management philosophy and certain external factors. Putnam & Myers (1997) believe that many software projects are "not very well thought through", and identify several threats including poor risk management, and a subsequent "failure to plan the work in the construction phase and then control it according to the plan". The first attribute considered relates to the age and duration of
the project (1.4.1-5). All the questions require numerical answers. The section continues, with the emphasis moving to address the scale and management of the project through a series of ordinal and numerical questions. Team size, total effort and sub-contracting are explored (1.4.6-10). Project management experience (1.4.11) and project review disciplines (1.4.12) are also considered. The section ends by seeking some information about external factors relating to the project, including location (1.4.13), customer (1.4.15) and application (1.4.16). Responses required in this instance are either descriptive or numerical.

The questionnaire advances in the fifth section from investigating the system attributes, development characteristics and management practices to explore features of the maintenance process. The first part of this section considers the maintenance team itself, and quantifies their knowledge and availability. Their experience, and familiarity with the system design and language are explored (2.1-2.5). This section continues with consideration of the assets that the maintenance team has at their disposal. Aspects such as access to documentation and senior personnel associated with the development project are considered (2.6-2.9). The final group of input variables which are considered in this section relate to the maintenance methodology. Issues such as who undertakes the maintenance, and where, are considered together with the level of proceduralisation which supports the maintenance process (2.10-2.15).

The content of the five sections discussed above relates to input variables that may affect the probability of the project meeting its objectives. The remainder of the questionnaire examines the outcomes from the projects, relating to both the development and maintenance phases of the lifecycle. In the sixth section (3.1) of the questionnaire the result of the development phase of the project is quantified in terms of both time and resource, compared to plan.

The seventh and final section (3.2) has been designed to capture certain of the outcomes from the maintenance phase. Of necessity, the maintenance phase is usually on-going, as opposed to the initial development phase of the project which may be regarded as complete in terms of the original definition of the scope of the project. The dimensions considered relate to the frequency (3.2.1) and degree of difficulty (3.2.2) associated with the continuing maintenance task.
3.5 Survey Execution

The survey included 34 software projects from ten discrete organisations, and encompassed a wide range of application areas extending from large commercial database management systems to real time defence and simulation programs. Initially a total of around 200 questionnaires were sent to about 170 organisations including general manufacturing and service businesses, professional software houses and computer OEMs, government agencies and academic institutions. Responses were sought from both UK and foreign sources. Despite this variety one common theme was the considerable reluctance encountered in persuading potential participants to assist by completing a questionnaire. Various reasons were offered ranging from other competing demands on available time, through commercial confidentiality to lack of information. One pattern which did emerge suggested that software engineers were often more interested in the ‘coal face’ technical challenges rather than in an issue which they perceive as a management problem, while many project managers engaged in development projects had limited recollection and little knowledge of records relating to past projects. This accords with personal experience of software development projects and has implications for both the training of staff and the implementation of management systems to gather data useful for future analysis and application to future estimates. An impression which was created by reluctant or unwilling respondents in several organisations was one of embarrassment. A recognition, perhaps, that the information being requested was unavailable despite being of a reasonably basic nature and susceptible in many cases to subjective judgement. The content reflects the expected outputs from good (but not necessarily ‘best’) practice software engineering. An inference might be that relatively few organisations are deploying such good practice, or more charitably, were not at the time that any potential target projects were being developed. Of necessity, given the maintenance theme, the questionnaire had to explore past development rather than the present. The most positive responses to the questionnaires came from senior staff engineers who perhaps had both the experience and time to address the issues raised. The survey was completed over approximately twelve months.

A wide spectrum of project types was included within the survey and included in the subsequent analysis. The scale extended from very large database systems used in banking and insurance applications through medium or large real-time programs
developed for defence or simulation use, to relatively small and bespoke business and logistical support tools. Development project durations, team sizes, hardware platforms and computing languages varied accordingly. The distribution of program sizes ranged from 8 KDSI to in excess of 1000 KDSI, with a median of 53 KDSI, and the preponderance below 250 KDSI.

![Program Size Distribution (KDSI)](image)

**Figure 3.2 – Sample Size Characteristics**

This distribution is depicted in Figure 3.2. Typical module size was approximately 1.5 KDSI, with 66% modules consisting of new software, 32% reuse of existing code and the balance third-party software.

Four distinct but generic architecture types were considered, and all are represented within the sample. These categories relate to open and proprietary systems, and to centralised and distributed designs.
Eight possible customer types were identified. These categorised the customer as being either military or civil, distinguished between end-users and prime contractors, and also between UK and overseas customers. The two predominant categories are UK civil end-users (such as power companies) and UK military end-users (such as the Royal Air Force); together they represent 70% of the sample. No projects for overseas military prime contractors were included in the sample. The distribution is reflected in Figure 3.4.

Three distinct project types were identified. These relate to new software development, reuse of existing code and extension to an existing programme. In practice these categories could overlap, so respondents were asked to identify the predominant characteristic. No projects were identified in which the extension theme was the strongest. This is summarised in below.
Applications were also categorised. Two primary groups were identified, one relating to scientific, modelling or real-time operation and another to transactional or database operation. A third category, which is a combination or intermediate, was also defined. The distribution is shown in Figure 3.6.

Aspects of the technical management competencies associated with the sample projects are reflected in the plots in Figure 3.7. Each plot records the range of scores for the particular attribute extending from the best possible result (giving a score of 1) to the least good (giving a score of 9), together with the mean value obtained from the sample population for every technical management attribute. The lowest means, reflecting higher competencies, were recorded for the expertise of the system and
software design authorities, with the poorest mean scores relating to the design team expertise in the areas of requirements analysis, design methodology and code review.

Figure 3.7 – Sample Technical Management Characteristics

Where

SRS is Quality of System Requirement Specification
SDS is Quality of System Design Specification
SysDAEXP is System Design Authority experience
SDAEXP is Software Design Authority experience
DOMEXP is Domain experience
LANGEXP is Language experience
REQANEXP is Requirements Analysis experience
DESMEXP is Design Methodology experience
DEVSYEXP is Development System experience
TARSYEXP is Target System experience
CREVEXP is Code Review experience
The significance of the variation between these eleven distributions has been tested using Analysis of Variance method. The SPSS one-way ANOVA tool was used. Most of the differences between the variables are not significant. An exception at the 95% level is in the variations between DESMEXP on the one hand, and SRS, SysDAEXP and SDAEXP on the other. DESMEXP receives a significantly inferior score. This could be rationalised by concluding that the relative weakness in the experience of the software teams with their respective development methodologies has been offset by the presence of a stronger design authority. This might also be reflected in the better SRS score. Note, however, that SDS is not significantly better than DESMEXP, perhaps indicating a tendency on the part of the design authority to underplay the importance of this stage. This may be reflected ultimately in the structure of the software, and could have implications for the ease with which maintenance may be performed. This analysis might therefore provide an early indication that SDS may exert significant influence over the maintenance phase.

A similar approach has been adopted in examining four input variables relating to project management. The results obtained are similar for each variable, the project review aspect providing the best mean result. One-way ANOVA suggests that no significant difference between these distributions exist. This is shown below in Figure 3.8.

The distribution of project size, as measured in man-months of effort, is shown in Figure 3.9, with projects of less than 500 man-months dominant.

Another project management output variable, which is summarised in Figure 3.10, relates to the overall project lead-time. In particular, the analysis compares the actual duration with the initial plan, and indicates that most of the sample projects overrun, typically by anything up to 50%. The mean sample slippage is 29% of the original plan. Only one project took less than the plan.
Figure 3.8 – Sample Project Management Characteristics

Where

PROJPLAN is Project Planning expertise
PMEXP is Project Management experience
PROJREV is Project Review expertise
STTURN is Degree of Staff Turnover

Figure 3.9 – Sample Size Distribution
Quality management attributes were also explored. Seven variables have been considered, with the best mean results obtained from the defect logging and configuration management attributes. This may reflect these attributes being processes whereas the remainder require active design of test strategies or documentation. Less scope for individual discretion and error may typically exist in the former case.
Where

ATS is Quality of Acceptance Test Specification
MODTEST is Quality and Extent of Module Testing
INTTEST is Quality and Extent of Integration Testing
DEFLOG is Quality of Defect Logging
CONFIGMT is Quality of Configuration Management
ARMDES is Quality of Design for Availability, Reliability & Maintainability
USERDOC is Quality of User Documentation

In this case one-way ANOVA indicates that a significant difference exists between two sub-groups consisting of DEFLOG and CONFIGMT that both score significantly better than the other sub-group, consisting of ARMDES and USERDOC. This difference could be interpreted as suggesting that the quality management practices in the sample are more focused on internal process than on deliverables to the customer. Implications for the maintenance phase may include the importance of gaining access to users if documentation shortcomings pertain. Additionally, the relative weakness of ARMDES aspect may imply comparatively more maintenance activity than might otherwise have been required. Both of these observations suggest that maintenance location may be important.

Maintenance management attributes have been explored as either personnel- or resource-related.

Figure 3.12 – Sample Maintenance Personnel Characteristics
Where \( \text{STAFFEXP} \) is Staff Experience

\( \text{SYSEXP} \) is System Experience

\( \text{LANGEXP} \) is Language Experience

\( \text{STAVAIL} \) is Availability of Staff

\( \text{TEAMWORK} \) is Team Member Familiarity

Note that these attributes relate purely to the maintenance team, as opposed to the earlier software development team. Application of one-way ANOVA identifies a difference between \( \text{STAVAIL} \), which scores significantly worse than \( \text{LANGEXP} \) and \( \text{TEAMWORK} \). Assuming maintenance personnel to be drawn from the development team then it is understandable that familiarity with both the language and other team members exists. An inferior score for staff availability is perhaps also consistent with personnel being assigned to future development work at the expense of continuing maintenance.

![Diagram showing maintenance resource characteristics](Figure 3.13 - Sample Maintenance Resource Characteristics)
Where

SYDAACC is Readiness of Access to the System Design Authority
SDAACC is Readiness of Access to the Software Design Authority
HDAACC is Readiness of Access to the Hardware Design Authority
DOCACCC is Readiness of Access to the Development Documentation
REPROD is Ease of System Re-installation
RPLCAACC is Readiness of Access to a Replica System
ISO9000 is Maintenance Process Compliance with ISO9000

One-way ANOVA identifies only one significant difference in the maintenance resource distributions; REPROD scores better than the rest. This may again be consistent with management processes that are internally and development focused, rather than possessing an external and maintenance orientation.

3.6 Analysis of Factors

The data generated through the survey was initially accumulated in an Excel spreadsheet. At the end of the data-gathering phase the information was converted into SPSS 8.0 (Statistics Package for Social Sciences) format for analysis. This tool was selected primarily because of the extensive range of features it offers and also because of the good reputation it enjoys amongst users. The approach taken was firstly to explore the data with the intention of understanding relationships that might exist between the variables, and subsequently to move forward to identify those input variables that have the most predictive power in determining the probable maintenance outcomes. The latter is discussed in 3.7.

Factor analysis was used to explore the relationships within the data, and also to reduce the dimensionality of the data set to more manageable proportions. The technique facilitates identification of underlying variables, or factors, that may explain a pattern of correlations within a set or sub-set of the observed variables. This can be achieved by identifying a smaller number of factors that explain most of the variance observed in the larger number of initial variables. An advantage of this method is that provides a means of minimising the effects of collinearity during a subsequent regression analysis. It can also be used to create hypotheses relating to causal relationships. Sharma (1996) provides a good description of the method. Using the Data Reduction option within SPSS, factor analysis was applied through a 'top
down’ approach, in which all the appropriate variables were initially considered as a single group, with the result that fourteen factors were identified by SPSS. Examination of the output data indicated that many of the strongest correlations seem to exist within individual sections of the questionnaire, rather than between variables from throughout the questionnaire. This seems to be a reasonable observation when translated into the operational domain, and can be readily illustrated. For example, the quality of both the SRS (System Requirement Specification) and the SDS (System Design Specification) is likely to reflect the fastidiousness with which the specification process is defined and observed within a given organisation; a stronger correlation is likely to exist between the two than, say, between either one and maintenance staff availability.

The factor analysis process was repeated, but in this second iteration it was applied at the section level from the questionnaire. The result was the generation of sixteen factors, two more than with the earlier iteration. The creation of the additional factors can be explained by the ‘sub-optimisation’ generated by considering multiple smaller groups of data; where strong correlations do exist between variables from different groups, this effect is no longer able to minimise the total number of factors. The relationship between the observed variables and the resultant factors is shown in Table 3.1. The column headed “effect” is a reflection of the drivers behind a particular factor. The explanation was derived by considering the weightings through which individual variables exert influence on the various factors. For example, the ‘project characteristics’ attribute is described by three factors described as ‘scale’, ‘team stability’ and ‘project management expertise’. The ‘scale’ factor relates primarily to duration, effort and team size; ‘team stability’ to staff turnover; and ‘project management expertise’ to experience, reviews and sub-contracting. The descriptors assigned in the “effect” column are a means of characterising the factor that reflects the predominant variable or variables derived from the factor analysis.
<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>VARIABLES</th>
<th>FACTOR</th>
<th>EFFECT</th>
<th>% VARIANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code complexity</td>
<td>5</td>
<td>cc1</td>
<td>'size/structure'</td>
<td>76.3</td>
</tr>
<tr>
<td>Environment</td>
<td>3</td>
<td>env1</td>
<td>'computer sys.'</td>
<td>45.9</td>
</tr>
<tr>
<td>Project Characteristics</td>
<td>11</td>
<td>pc1, pc2, pc3</td>
<td>'scale', 'team stability', 'proj. mngt. expertise'</td>
<td>79.8</td>
</tr>
<tr>
<td>Design Process</td>
<td>21</td>
<td>despro1, despro2, despro3, despro4, despro5</td>
<td>'domain/specification', 'req. mngt.', 'code review', 'config. mngt.', 'test'</td>
<td>76.3</td>
</tr>
<tr>
<td>Maintenance Policies</td>
<td>15</td>
<td>mp1, mp2, mp3, mp4</td>
<td>'resource access', 'staff location', 'sub-con. skills', 'sub-con. location'</td>
<td>70.7</td>
</tr>
<tr>
<td>Development Performance</td>
<td>3</td>
<td>dperf1</td>
<td>'meeting plan'</td>
<td>77.0</td>
</tr>
<tr>
<td>Maintenance Outcome</td>
<td>2</td>
<td>mo1</td>
<td>'cost'</td>
<td>74.5</td>
</tr>
</tbody>
</table>

Table 3.1 - Factor analysis

A linear regression analysis was performed with the maintenance outcome factor being treated as the dependent variable, and the other 15 considered as the independent variables. The best result that was found to be significant at the p<0.05 level produced an adjusted Coefficient of Determination ($R^2_{adj}$) of 40%, and utilised four of the factors. The $R^2_{adj}$ coefficient is the ratio of the reduction in the sum of squares of deviations obtained by using the model to the total sum of squares of deviations around the sample mean. The adjusted element reflects the multiple terms in the model. Mendenhall & Beaver (1991) provide an introduction to the approach. This result was obtained by SPSS from twelve models and was generated using the elimination selection form of linear regression.
The result of the regression can be expressed in the form:

\[
PMO = (0.73 \text{ DPERF1}) + (0.73 \text{ PC1}) + (0.30 \text{ DESPRO2}) + (0.65 \text{ CC1})
\]

\[
(0.15) \quad (0.24) \quad (0.14) \quad (0.24)
\]

Equation 3.1 - Regression Model - Ease of Maintenance - Factor Basis

where PMO is the predicted maintenance outcome

DPERF1 relates to development performance – 'meeting plan'

PC1 relates to project characteristics – 'scale'

DESPRO2 relates to design process – 'requirements management'

CC1 relates to code complexity – 'size/structure'

PMO attempts to predict MO1 prior to the commencement of the development project. The figures in parentheses below each of the coefficients contained in the equation are the values of the standard errors. Conclusions that can be drawn from consideration of the model are that an unsatisfactory development performance and project management approach will tend to result in more problematical maintenance. More complex code will also tend toward a similar result, although with slightly less influence. The design process produces a similar effect, but the degree of influence is substantially less. These factors would seem to be plausible determinants of the maintenance outcome. The absence of any of the four factors relating to maintenance policy is perhaps surprising. The factors that are most powerful all relate to earlier in the lifecycle or to more fundamental attributes of the software, suggesting that maintenance policies have a marginal effect once the essential features of the software have been established.

3.7 Analysis of Input Variables

Having developed some understanding of the data, the analysis proceeded to identify those input variables that possess the power to make useful predictions of the likely maintenance outcomes. To provide predictions in an operational environment, any modelling tool should preferably relate to readily obtainable and understandable variables, rather than to factors that are themselves composed of variables. This view is consistent with the reservation expressed by Fenton & Pfleeger (1997) regarding factor analysis: that it "produces metrics that cannot be easily nor directly interpretable in terms of program features". Conceptually, the approach is similar to
that taken by Alkhatib (1992) who identified ten "meta-factors" which are analogous to the factors discussed above, and then proceeded to consider which affected maintenance to a significant degree. The factor analysis work described in 3.6 provides guidance as to which factors, and therefore a subset of variables, are most likely to make the best predictions.

Utilising the output from the factor analysis, together with linear regression that was this time applied to individual variables contained within the most predictive factors, enabled a more powerful model to be developed. Once again the maintenance outcome factor was treated as the dependent variable, with the individual variables being regarded as the independent variables. This model possesses an $R^2$ value of 0.706 and, allowing for the degrees of freedom involved, an $R^2_{adj.}$ of 65%, includes five of the input variables and is again significant. It is noted that Haworth (1996) identifies research that indicates that "values of $R^2$ near 0.66 are 'gratifying' in behavioural research".

The SPSS output includes coefficient and standard error values for each of the input variables, together with confirmation of their individual significance.

This result can be expressed in the form:

$$ PMO = -2.5 - (0.67 A1) + (0.0005 D4) + (0.75 ONBUDTC) + (0.11 LOCATION) + (0.18 SDS) $$

$$\text{Equation 3.2 - Regression Model - Ease of Maintenance - Full Sample Basis}$$

The figures in parentheses below each of the coefficients contained in the equation are the values of the standard errors.

where:  
PMO is the predicted maintenance outcome  
A1 relates to possessing an open architecture  
D4 reflects the domain knowledge of the development engineers  
ONBUDTC reflects how well the development project met budget  
LOCATION reflects the proportion of maintenance performed at site  
SDS reflects the quality of the design specification
A detailed discussion of the results is provided in 3.8. Prior to this, however, it is important to develop an appreciation of the structure of the model, and the relevance of the variables. Initially it should be noted that the level of collinearity between the variables was established as being low, although a degree of collinearity does exist. This is discussed further in 3.10. As structured, 'ease of maintenance' increases as the value of PMO diminishes. The first independent variable in the equation, A1, relates to the presence (A1=1) or absence (A1=0) of an open and distributed architecture. Given the negative sign which precedes A1, the presence of an open and distributed architecture will tend to reduce the value of PMO; the implication is that the presence of this attribute tends to facilitate 'ease of maintenance'. The D4 term relates to the domain experience of the development team, multiplied to a fourth power. This seems to suggest that domain experience is a very important variable, to which 'ease of maintenance' is highly sensitive. Although the reasons for this are explored below in 3.8, it can be seen that a low value of D4, meaning high domain experience, will tend to reduce the magnitude of PMO, providing a better maintenance outcome. Similarly, a low value of ONBUDTC, which indicates that the development project was completed within budget, will also tend to reduce the value of PMO, indicating a better maintenance result than would have otherwise been the case. A low value for the LOCATION variable indicates an increasing proportion of the maintenance activity being performed at the customer site, and again the lower value of LOCATION tends to reduce PMO. Finally, low value for SDS is generated by a good design specification, and will tend to reduce PMO, suggesting that maintenance performance will be improved by a structured approach to the initial system design.

The quality of the model prediction is illustrated below, with actual maintenance outcome plotted against that predicted by the model.
3.8 Discussion of Results

The equation indicates that:

*Ease of maintenance is likely (65%) to depend on the development project having started with a good design specification and been completed within budget on an open/distributed architecture by engineers with high domain knowledge, with the subsequent maintenance performed at the site of the operational system.*

The content aligns with the view articulated by Sommerville (2001) that "maintenance costs are related to a number of product, process and organisational factors". It is also consistent with research undertaken by Haworth et al (1992) which suggests that software maintenance is influenced by four variables: environment, maintenance task, programmer skill and maintainability of source code. It also broadly agrees with the conclusion reached by Alkhatib (1992) who identifies several facets of a software system with the "most significant influence over maintenance: real-time processing, database processing, end-user on-going support, module size, number of runs and run-time per module, and number of reports and number of
copies per function”. It is possible to identify mappings between certain of these facets and the variables identified during the regression analysis. For example, end-user on-going support can be related to maintenance performed at the site of the operational system, and module size to having a good design specification. Differences with Alkatib (1992) perhaps reflect his method, involving interviews with a range of organisations and subsequent pursuit of a “detailed empirical analysis” of product and process considerations within a single organisation. He has decomposed operating practices within a specific organisation, identifying specific issues relevant to an existing system, whereas the regression analysis has retained a generic approach, reflecting the objective of developing a method of general applicability and relevant to the feasibility phase of a new project. This latter point relates to a time that may be long before discrete operating practices are defined. Additionally, the questionnaire and subsequent factor analysis has been designed to enable consideration of organisational and personnel issues in multiple organisations.

One issue arising from the conclusions drawn from the equation is that completing the development project on budget is important to maintainability. Several reasons can be rationalised for this association. Perhaps the prime reason is simply that a project that has been managed and engineered to a budget is also more likely to have been designed and produced with future product evolution in mind. As stated, this is partially a reflection of the competence of those undertaking the project. A second but related reason is both organisational and cultural, in that an organisation that has the processes in place to ensure that development projects can be delivered to budget may also be one in which design practice encompasses future maintenance. The organisation may, for example, promote reuse of proven and documented software or require that design reviews address maintenance considerations. A third observation, to an extent the corollary of the previous two and drawn from personal experience, is that the weaknesses in the management processes which may have contributed to the variances to plan might also be reflected in those processes which should ensure adequate testing and documentation. A project which is running above budget and possibly also behind schedule is one in which shortcuts may be taken with testing and documentation, to the detriment of the future maintenance of the system. Kan (1995) supports the hypothesis that maintenance is affected by the quality of the prior development, observing that “the number of defect or problem arrivals is largely determined by the development process before the maintenance phase”. Given the hypothesis that the success or otherwise of the development project will affect
subsequent maintenance, one difficulty that arises is that the overall objective is to identify input variables which enable useful predictions to be made regarding future maintenance before development commences; completion of development within budget is obviously an unknown at this stage. In response to this problem some further analysis of the data was undertaken to identify factors which affect the probability of the development project meeting budget. This was deliberately confined to the minimum number of variables, primarily to facilitate simplicity. Linear regression was again the tool that was deployed, with the outcome a regression model.

The result can be expressed in the form:

\[
\text{ONBUDTC} = 1.68 + (0.01 \text{ PMEX2}) - (0.33 \text{ PROJTYPE})
\]

\[
\text{Equation 3.3 - Regression Model - Meet Development Budget}
\]

The figures in parentheses below each of the coefficients contained in the equation are the values of the standard errors.

where: \( \text{ONBUDTC} \) reflects whether the development project met budget

\( \text{PMEX2} \) relates to the experience of the project manager

\( \text{PROJTYPE} \) reflects the level of reuse of existing code

The equation indicates that:

An on-budget development project is likely (34%) to be determined by employing an experienced project manager and reusing existing software.

Although perhaps not the only factors which influence the likelihood of successfully completing development within budget, these two variables alone have the merit of being quantifiable before the start of development, and explaining one third of the variation in meeting the budget.
Four other variables were identified as being particularly useful in predicting the probable maintenance outcome. Of these, the domain knowledge of the engineers undertaking the development seems to be the most powerful determinant. This attribute relates to the knowledge possessed by the engineers of the subject matter they are developing a system to support, as opposed to awareness of software or systems engineering methods which are more generic in application. For example, a team developing an air traffic control system requires some knowledge of air traffic control principles beyond responding to the content of a requirements specification. The model suggests that the greater this knowledge, the more maintainable will be the delivered system. Similarly, development of an air traffic control training simulator will benefit from a team with an appreciation of both air traffic control and teaching methods. Several reasons can be adduced to explain this phenomenon; they can collectively described as 'understanding the needs of the customer'. At a high level of abstraction, the system will have been constructed to reflect how the users operate, or intend to operate; this achievement alone will tend to reduce the frequency of user-driven requests for change. It is also likely that test strategies will be more robust if they exercise the potentially large number of different, but desirable, combinations of features that a complex system may offer to the users. Documentation will tend to be structured and written from the perspective of the application domain, rather than that of the software engineer; fewer queries, mistakes and mis-understandings are the likely result. Future adaptation may also be facilitated by a recognition of how requirements may change, and embedding this appreciation in the design. In summary, understanding the problem the user is trying to solve lays the foundation for the whole project, from formal requirements capture through to future maintenance. Boehm (1981) cites evidence from several large projects that an error in appreciating the customer requirement which is not detected until the maintenance phase is a hundred times more expensive to correct than at the requirements definition stage, while McConnell (1993) suggests that even on a smaller project a multiplier of twenty is likely. Although the costs are inherently important, they are also a surrogate for the level of disruption and dissatisfaction that may be experienced by the customer.

A second variable reflects the quality of the design specification produced early in the development lifecycle. The importance of an adequately defined design approach is widely recognised, and indeed is important in most areas of engineering endeavour. This is reflected in the wide adoption of principles embodied in standards such as
ISO9001, and in the software domain, in the ideas contained in the Capability and Maturity Model (CMM). These principles are arguably of particular importance to software development, given the ethereal peculiarity of the software discipline and the inherent ‘softness’ of the medium. As Glass (1998) observes, “the software product is ‘soft’ and thus easily changed compared to other ‘harder’ disciplines”. As already discussed in 2.3, this is a powerful attribute of the software artefact and of fundamental importance to the maintenance phase, but during development requires discipline which needs to be reflected in a structured process and associated documentation conventions. As the scale of the project increases, so the significance of this imperative mounts. Symptoms of inadequate attention to these processes might include code written or structured inconsistently, hardware constraints, inadequate testing or delinquent documentation. Any of these deficiencies, and others, is likely to have a detrimental impact on the later maintenance activity. They will probably have also caused delay during the development phase. One interesting issue that merits rationalisation is the greater predictive power of the design specification, when compared to the requirement specification. This is believed to pertain because of the mappings that exist between these specifications and the development and maintenance phases respectively. An inadequate requirement specification is likely to have a first-order effect on the success of the development phase because the system will be less likely to meet the needs of the customer, and may require modification to gain acceptance. Its effect on the maintenance phase is likely to tend towards the second-order, with additional and possibly more problematical requests for adaptive maintenance than might otherwise have been the case. The impact of the design specification is more likely to be first-order in both cases, for the reasons already outlined above. These mappings are summarised below in Table 3.6.

<table>
<thead>
<tr>
<th>Requirements Specification</th>
<th>Development Phase</th>
<th>Maintenance Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>System does not meet the needs of the customer - early impact</td>
<td>Additional requests for adaptive maintenance - later impact</td>
<td></td>
</tr>
<tr>
<td>Design Specification</td>
<td>System is hard to integrate - early impact</td>
<td>Corrective maintenance necessary from entry into service - early impact</td>
</tr>
</tbody>
</table>

Table 3.2 - Effect of deficient specifications on lifecycle phases
Another variable that was identified as possessing some predictive power is that of the location at which the maintenance is undertaken. The model suggests that the more maintenance is undertaken at the site of the operational system, the more satisfactory will be the maintenance outcome. There are some parallels with the issue of domain knowledge, in that one of the implications of maintaining at the user location is a better appreciation of day to day concerns and problems; once again, 'understanding the needs of the customer'. This is consistent with, although not the same as, the conclusion reached by Alkhatib (1992) that "increased end-user ongoing support would reduce average time per repair maintenance". Although there is some intuitive merit with this view, there is perhaps a caveat. Taken to the extreme, sub-optimisation could be an issue; one argument for doing the opposite and centralising maintenance might be that it is therefore possible to support a critical mass of specialists with much greater experience of the domain than would be possible at a single customer location. To an extent, this policy will be determined by the particular application. Maintaining a large database for a bank or insurer could justify a full-time team at the customer site, whereas the designer of a word processing package cannot assign a maintainer to sit next to each user. Although these examples are arguably taken from two extremes, it is still possible to identify established strategies that represent good practice and support the use of this variable. For example, the word processing vendor may establish a virtual location through the use of a web site which enables problems and enquiries to be logged and support to be provided; beta test sites may also be used, both to identify faults and facilitate fast feedback from representatives of the user community. Additionally, involvement may be required of the package distributors, both to ensure they are close to the user and to provide further feedback.

The final variable contained in the model relates to the presence, or otherwise, of an open and distributed architecture. Three other possible architectures were considered, involving various combinations that include proprietary and centralised alternatives. The benefits derived from the development and evolution of open systems are well documented and understood, and can be seen as advantageous to maintenance. Apart from their relative ease of adaptation and portability, they have the advantage of widespread application of a range of conventions that pertain, largely irrespective of the platform. Benefits include a greater understanding of the maintenance issues from an informed user through to more flexibility in the recruitment and deployment of maintenance staff. Domain knowledge, rather than pure systems expertise, can then
perhaps become a larger consideration. One issue which deserved specific investigation was whether the open attribute was the real variable, or whether open systems, which will tend to be the newer ones was a surrogate for another attribute which may have arisen in parallel. Structured design methods, for example, have continued to evolve over a similar period and could perhaps have been the underlying source of variation. Further work using both factor analysis and regression confirmed the original observation that the open attribute appears to be the determinant. The distributed attribute potentially enables parts of systems to be subjected to maintenance without necessarily affecting the overall operational mission of the system, and without a full replica system being retained on a dedicated platform.

Responsiveness and economy are amongst the advantages to be derived from the open and distributed combination.

Besides the variables already considered because of their inclusion within the model, there are others whose absence is also worthy of note. Those relating to size, modularity and general complexity are particularly interesting, given the research into the relationship between complexity and maintainability which has already been documented. Although a factor ccl was developed during the factor analysis, which reflected program size, module density and degree of software reuse, and was one of the sub-set of factors which generated the best $R^2_{adj}$. result from the complete population of factors, none of the five variables contained within ccl is itself a direct contributor to the model which was developed. Neither size, complexity (as measured by size divided by number of modules) nor ccl itself exhibited any significant power in determining the maintenance outcome. The reason for this apparent diffusion of influence on the maintenance result is believed to be one of context; within the individual organisations surveyed a norm exists as to the typical size and structure of the systems and software developed. Within limits, complexity is not recognised by individual respondents as a particularly strong variable and is therefore not identified during the analysis as a notable determinant of the maintenance outcome. If the complexity of a project extended beyond this limit for any given organisation, it may be that complexity would immediately become an important variable. It is also noted that Alkhatib (1992) did not discern any strong correlation between complexity and maintainability when analysing variation within the context of a single organisation. Fenton & Pfleeger (1997) also observe "our intuition about module size is wrong"; smaller modules are not necessarily of higher quality and there is no simple
determinant of an optimal module size if, for example, fault density minimisation is the objective. This idea will be pursued further during the model development in Chapter 4. It is noted, however, that the code complexity factor does make an indirect contribution to determining the maintenance outcome. One of the variables contained within the factor, reuse, is a determinant of development performance, which in turn influences the maintenance result. This probably explains why the factor contributed to the Coefficient of Determination that was achieved using factors rather than individual variables.

Program age is another variable that might be expected to exert a strong influence on the maintenance outcome, but again was not discerned during the analysis. Reasons for this may primarily reflect that no great range existed in the software surveyed, and that its relevance may be indirectly reflected through the use of open and distributed architectures, as already discussed.

Prior to moving on to develop a predictive model, validation of the observations and hypotheses discussed above was pursued.

3.9 Validation of Results

Two separate strategies were adopted in attempting to validate the results outlined in 3.8. They can be described as extrinsic, involving follow up interviews with contributors to the survey, and intrinsic, requiring some further limited analysis of the data.

The extrinsic validation consisted of interviews with six contributors from different projects in separate locations who were invited to give a subjective view as to what was important to successful maintenance. The contributors were either senior staff engineers or project managers. The interviews were undertaken on a 'one to one' basis, using the completed questionnaire as a guide. A typical interview duration was about one hour. The interviewees were invited to identify the key issues from the perspective of a participant in the project, and where possible, to identify linkages between the input variables and the resulting maintenance outcome. These judgements, by their nature, could only be subjective; the purpose of the interviews, however, was to provide some qualitative validation of conclusions drawn from the objective content of the questionnaire. Several distinct themes emerged which are
tabulated below in an order that reflects an aggregation of the opinions of the interviewees. The comments attached reflect remarks made during the interviews.

<table>
<thead>
<tr>
<th>THEMES</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintenance reflects prior development.</td>
<td>“software maintenance can only be as good as the software allows”</td>
</tr>
<tr>
<td></td>
<td>“software must be designed for maintenance”</td>
</tr>
<tr>
<td></td>
<td>“documentation as well as code”</td>
</tr>
<tr>
<td>Good people are essential.</td>
<td>“trained and motivated engineers make the difference”</td>
</tr>
<tr>
<td></td>
<td>“managers as well as engineers”</td>
</tr>
<tr>
<td>Must meet needs of customer.</td>
<td>“success can only be judged by the customer”</td>
</tr>
<tr>
<td></td>
<td>“must understand the customer”</td>
</tr>
<tr>
<td>Modern platforms help.</td>
<td>“old architectures mean people problems”</td>
</tr>
</tbody>
</table>

Table 3.3 - Maintenance themes

The most striking feature of this output is the emphasis placed on the management dimension, and particularly the people-related themes of organisation, education and relationships. Even the architecture aspect, which can be seen as a technical issue, was raised in the context of generating difficulties with engineers; specifically that it becomes hard to recruit people with the relevant knowledge to maintain an aged-architecture based system. Those working on it may want to move before their skills become obsolete and those who remain may require mechanisms to keep their skills current. This emphasis on the human dimension reflects the observations made by Potts (1993) which were discussed in 2.2.

Similarly Van Genuchten (1991), when considering the reasons for software projects to run behind schedule, observes that “investigation of the subtle reasons for delay indicate that the reasons were not technical in nature, but were related to organisational, managerial and human aspects”. Neither the complexity of the software nor its age was raised unprompted by the interviewees as major variables in their experiences of maintenance. One other aspect that received attention was the integrity of documentation, but this was seen as closely aligned with the quality of the development phase of the project. Blum (1995) asserts that “often, the fact that the maintainer is tasked to work with imperfect understanding leads to sub-optimal modifications” and attributes this result to “the absence of effective high-level documentation about the software features to be changed”.

102
It was concluded that some agreement exists between the sample of respondents interviewed and the variables identified as important by analysis of the questionnaires. A key aspect of this particular study is that it does support a degree of causation between those variables seen as important, and the maintenance outcome; as opposed to there merely being an association. Additionally, the range of interviews and organisations undertaken suggests that the conclusions reached regarding those factors that are the greatest determinants can be generalised. This provides support to the use of the factors in further model development.

The intrinsic validation of the conclusions reached involved using a model derived from a sub-set of the 34 questionnaires to predict the maintenance outcome for the remainder of the sample. In SPSS a random set of 22 results from the population were selected, and then used to develop a predictive model through regression analysis using the process already described in 3.7. This result of this analysis can be expressed in the form:

\[ PMO = -2.5 - (0.74 \text{ A1}) + (0.0005 \text{ D4}) + (0.64 \text{ ONBUDTC}) + (0.14 \text{ LOCATION}) + (0.18 \text{ SDS}) \]

\[
(0.58)(0.31) \quad (0.00) \quad (0.26) \quad (0.04) \quad (0.07)
\]

Equation 3.4 - Regression Model - Ease of Maintenance - Random Sample Basis

Comparison with Equation 3.2 provides confirmation that similar coefficients and standard errors have been generated. An \( R^2_{adj} \) of 64% pertains. This model was then used to predict the likely maintenance outcomes for the balance of 12 projects not used to develop the model. The results of these predictions are plotted in Figure 3.15.
From the survey, 22 of the questionnaires came from three sites within a single organisation, with the balance from a further nine organisations. A third regression model was developed in the same way as the general model in 3.7 and the randomly selected modelling approach described above, but using only the 22 results from the single organisation. This model was then used to make a prediction relating to maintenance of the 12 projects in the other organisations. This result of this analysis can be expressed in the form:

\[
PMO = -2.8 - (0.78 \text{ A1}) + (0.0005 \text{ D4}) + (0.79 \text{ ONBUDTC}) \\
\text{(0.60)} \quad \text{(0.30)} \quad \text{(0.00)} \quad \text{(0.35)} \\
+ (0.14 \text{ LOCATION}) + (0.21 \text{ SDS}) \\
\text{(0.05)} \quad \text{(0.07)}
\]

Equation 3.5 - Regression Model - Ease of Maintenance - Selected Sample Basis

Once again, similar values have been generated to those found in Equation 3.2. Again an $R^2_{adj.}$ of approximately 64% is achieved. The predicted maintenance outcomes for the other organisations are plotted in Figure 3.16.
The predictive power of these two models was then compared using several established statistical methods. The results are tabulated below.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Random Sample Basis</th>
<th>Selected Sample Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAE</td>
<td>0.20</td>
<td>-0.03</td>
</tr>
<tr>
<td>$</td>
<td>MAE</td>
<td>$</td>
</tr>
<tr>
<td>MAPE</td>
<td>0.22</td>
<td>0.98</td>
</tr>
<tr>
<td>$</td>
<td>MAPE</td>
<td>$ (Makridakis 1993)</td>
</tr>
<tr>
<td>MAPE(Makridakis 1993)</td>
<td>-0.83</td>
<td>-0.30</td>
</tr>
<tr>
<td>$</td>
<td>MAPE</td>
<td>$ (Makridakis 1993)</td>
</tr>
</tbody>
</table>

Table 3.4 – Model comparison

Several points can be noted from the results obtained. Those measures that utilise the modulus of the value obtained, and therefore disregard the direction of the error, seem to produce results that are closer for the two samples. The results obtained using the various methods do not favour one or other of the samples taken. The inference is therefore drawn that the two samples are similar and that no particular properties relate to the selected sample when compared to the random one. This inference was
tested more formally through an Analysis of Variance methodology. Mendenhall & Beaver (1991) provide a description of the approach. The null hypothesis:

\[ \text{no systemic difference exists between predictions made by the random- and selected-sample models} \]

was tested and accepted. The test statistic, $F$, gives a value of 2.96 compared to an $F_\alpha$ value of approximately 4.00 ($\alpha=0.05$). The supporting calculations are provided in Appendix 2.

This test supports the argument that the analysis undertaken and results obtained possess sufficient robustness to enable the conclusions drawn to have some generic application. It is therefore reasonable to use the factors identified as important for further analysis and modelling.

3.10 Further Analysis of Input Variables

The conclusions reached in 3.9 reflect the application of two powerful multivariate techniques, factor analysis and multiple linear regression analysis. These methods impose one major constraint in that they only allow exploration of a single relationship at a time, between dependent and independent variables. Reality, however, may be more complicated than this, with interdependencies between variables resulting in consequences that may not be wholly apparent when investigated using these methods. For this reason another methodology, structural equation modelling (SEM) has also been pursued, either to confirm the earlier conclusions or develop an alternative paradigm capable of explaining which inputs are most significant in determining future maintenance costs. SEM may be regarded as a development of, or extension to, other well-established regression methods, rather than a wholly distinct alternative. The method involves the specification and solution of a series of separate but interdependent regression equations. This is achieved through a well-defined process that requires the creation of path diagrams reflecting the hypothesis, their conversion into structural equations that are then solved against a data set using commercially available software. A detailed description of the method is provided in a concise format by Spirtes et al (1998). SEM enables several dependence relationships to be examined simultaneously, and
has particular value when a dependent variable becomes an independent variable in a subsequent relationship. Collinearities that are hypothesised can therefore be readily accommodated. Hair et al (1998) summarise the benefits of the method as being a “method of dealing with multiple relationships simultaneously while providing statistical efficiency” and whilst also providing “a transition from exploratory to confirmatory analysis”. The importance of the latter point is that SEM can facilitate the evaluation of several relationships that together constitute a larger-scale model.

The method has been applied to several models that have been hypothesised to describe possible relationships between the various factors and the maintenance outcome. The first of these, designated Model 1 reflects the content of the initial eight-factor model already developed in 3.6. Note that for completeness and to illustrate the method the non-significant terms have been retained. This is illustrated in the path diagram shown below in Figure 3.17.
Note that the insignificant linkages between the terms are represented by dotted lines. **Model 2** is an enhanced version of **Model 1**, reflecting several collinearities that have been postulated. This is shown below in Figure 3.18

The proposed collinearities reflect inter-dependencies between the input variables and are identified by the curved and arrowed lines connecting them. For example, the model indicates that the computing environment (env1) influences code complexity (ccl). The SEM method requires the conversion of the path diagram into a series of linear equations. For the model delineated in Figure 3.18 these equations are:
\[ m_1 = a_2 \text{despro}_5 + a_3 \text{env}_1 + a_4 \text{despro}_1 + a_6 \text{dperf}_1 + a_7 \text{mp}_2 + a_8 \text{pc}_1 + a_1 \text{despro}_2 + a_5 \text{pc}_2 + e_y \]
\[ \text{dperf}_1 = b_1 \text{pc}_1 + b_3 \text{cc}_1 + e_x \]
\[ \text{cc}_1 = g_1 \text{env}_1 + g_2 \text{despro}_5 + e_w \]
\[ \text{mp}_2 = k_1 \text{cc}_1 + e_t \]

The equations describe the relationships between the factors. These are consistent with the nomenclature developed in Table 3.1, with coefficient terms included. Also added is an error term on the end of each equation. The first of the equations defines the relationship between the maintenance outcome and factors that are believed to be important. Equation 2. has been included to describe the relationship between development performance and the variables discussed in 3.8. Similarly, Equation 3. has been included to reflect the postulation that code complexity is influenced by the computing environment and the design specification, and that a collinearity therefore exists between the factors. Finally, Equation 4. reflects the influence that code complexity may have on the location of maintenance staff. These equations are solved simultaneously against the same data that was collected and utilised during the previous linear regression analysis, and results in a series of maximum likelihood estimates of the coefficients in the equations. The objective of the analysis is to test whether a more plausible description of the relationships can be developed by recognising the collinearities that may exist. Given that a very large number of potential models exist and could be reflected in path diagrams, the approach taken has been to take the maintainability predictor developed in 3.7 and modify it to reflect possible collinearities. Comparisons are then made between this model and with the original developed through linear regression to assess whether the predictions it provides make objectively better predictive performance than the original.

Model 3 describes the maintainability predictor and is shown below in Figure 3.19.
Model 4 is a development of the maintainability predictor, reflecting the existence of collinearities between the variables.

Figure 3.20 - Model 4 - Path Diagram - Maintainability Predictor Model Including Postulated Collinearities
The SEM analysis of Model 4 indicates an $R^2$ value of 0.754, compared to a 0.706 for the maintainability predictor before collinearities were considered. The performance of the model has been improved by less than 10% by considering the effects of interdependencies between the input variables. This confirms the observation made in 3.7 that only a low level of collinearity between the variables exists. Given that the robustness of a predictive model is related to the simplicity of the model design, it is therefore concluded that Model 3, the maintainability predictor developed in 3.7 provides the best basis for proceeding into the predictive element of the research.

3.11 Conclusion

Chapter 3 has involved the structured analysis of 69 variables associated with 34 software projects obtained from ten organisations, with a variety of different customers. Size, architectures and application areas also varied widely. Information was collected in a standardised format through a questionnaire. Factor analysis was initially used to consolidate the input variables into a smaller number of groups, and those factors that appeared to exert the most influence on the maintenance outcome were identified. A predictive model was constructed by applying linear regression methods to the factors. These factors subsequently provided a basis for selecting those individual variables that possessed the greatest predictive power. Linear regression was again deployed in the development of a more powerful model which includes five independent variables, and appears capable of explaining approximately 65% of the variation in maintenance outcomes within the sample projects. These variables related to the quality of the initial design specification, the presence or absence of an open architecture, the domain knowledge of the engineers on the development team, the success the development team had in meeting budget, and the location at which the subsequent maintenance phase was undertaken. The validity and generic applicability of this conclusion was verified through a combination of interviews with contributors to the questionnaires, and further regression analysis. Structural equation modelling was also applied to determine the effects of collinearities, and explore whether a simpler and more powerful predictive model could be developed. This approach indicated that a marginally more powerful model could be developed, but only at the expense of greater complexity, and arguably by diminishing the general applicability of the model.
It is concluded that the five variables identified, together with the general form of the model, are a valid basis from which to proceed to predicting probable future maintenance outcomes and costs. This is explored further in Chapter 4.
4.1 Introduction

In Chapter 3 conclusions were drawn from a sample of user information regarding those variables which possess the greatest predictive power in determining the maintenance outcomes which are likely to be encountered subsequent to a proposed software development. Five variables were identified as being most significant, and this observation was substantiated through several strategies including case interviews and the application of structural equation modelling.

These variables are used to make predictions of maintenance outcomes in this chapter. The attributes that were identified in 3.4 to characterise a maintenance outcome relate to the frequency of continuing maintenance events, and the associated degree of difficulty. Factor analysis was used to aggregate these inputs into a single variable. The objective is to design models capable of making useful predictions of the probable value of this variable, and then to establish a prognosis regarding a future maintenance requirement. These predictions are of two types; categorical and continuous. In the former, methods of distinguishing maintenance outcomes that are likely to be acceptable from those that may not are explored, whereas the latter addresses the question of quantifying probable maintenance effort. In both cases these predictions are obtained from a single accumulation of data early in the life of the project; no attempt is made to update these predictions, as further information becomes available. To that extent the content of this chapter has been termed “static”.

This chapter may be regarded as the beginning of what Glass (1994) regards as being the “evaluative phase” in which a proposal or analytic finding is evaluated “by means of experimentation or observation, perhaps leading to a substantiated model, principle or theory”. In this case statistical methods are applied to develop useful tools, based upon the conclusions reached during the propositional and analytical phases explored in the preceding chapter.
In Chapter 3 the analysis of data gathered from 34 software projects was used to develop a predictive model that provided an indication of the maintainability of the software. This analysis is summarised in Figure 4.1.

![Figure 4.1 - Maintainability Predictor Development](image)

### 4.2 Categorical Prediction

This form of prediction is useful in situations where it is desirable to predict an outcome based on some defined input variables. The predictions are categorical because, typically, they tend to provide outputs such 'good' or 'bad'. Examples can be found in areas such as healthcare where, for example, lifestyle data is used to make predictions about susceptibility to heart disease or cancer. Similar techniques are applied to credit rating by major lenders. This latter area is one of substantial current interest, although work in the field has continued over many years. Wigington (1980) provides a good description of the principles which remains valid today.

A similar approach can be applied to making a prediction of whether a future maintenance outcome is likely to be good or bad. The particular statistical technique that has been adopted is logistic regression. This is a well-established method which, although less computationally efficient than the alternative discriminant analysis approach, is regarded as more robust because it makes no distributional assumptions regarding the data. Sharma (1996) provides a description of this technique. The approach taken has been to consider the maintenance outcome factor and categorise the values obtained into high ('bad') and low ('good') scores. The model generated in Chapter 3 and reflected in Equation 3.2 was used to provide a predicted result and a comparison made using logistic regression.
The method adopted was to consider the data analysed in Chapter 3, and in particular the maintenance outcome factor. The analysis undertaken in 4.2 is summarised below in Figure 4.2.

The distribution of maintenance outcome factor (MO1) scores obtained was examined and is plotted below in Figure 4.3.

Figure 4.2 – Categorical Model Development Process

Figure 4.3 – Maintenance Outcome Factor Score Distribution
These scores are standardised around a mid-point of zero, with a score below this value reflecting a more favourable outcome. This distribution can be shown to approximate to the normal distribution.

Given this distribution, the outcomes (MO1) were categorised (MOZ) according to whether the score is less than zero ('good') or greater than zero ('bad').

Several regressions were performed to test the robustness of the model. If the variables used in the model are applied directly to the logistic regression as the covariates in the analysis then the results are as indicated below in Table 4.1.

**Classification Table for MOZ**

The Cut Value is .50

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Observed</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>131</td>
</tr>
<tr>
<td>Bad</td>
<td>2</td>
<td>15</td>
</tr>
</tbody>
</table>

**Overall 85.29%**

**Table 4.1 - Performance based directly on variables used in basic model.**

Equation 4.1 reflects this prediction.

\[
\text{logit (MOZ)} = -6.044 - (3.188 \ A1) + (0.001 \ D4) + (1.985 \ ONBUDTC) \\
(2.654) \quad (1.342) \quad (0.000) \quad (1.199) \\
+ (0.398 \ LOCATION) + (0.288 \ SDS) \\
(0.197) \quad (0.287) 
\]

**Equation 4.1 – Logistic Regression Model – Maintenance Outcome – Direct Variable Basis**
This result arises from the partial elimination of error embedded within the model. During the model development in Chapter 3 it was established that the model explained 65% of the variation in maintenance outcomes. One source of error is removed by applying the five variables as covariates.

One difficulty is that the variables include ONBUDTC, which reflects the success experienced in completing the development project within budget. This was discussed at length in 3.8, where two independent variables that contribute to the development outcome were identified. A further regression was undertaken with the two variables, PMEXP and PROJTYPE, substituted for ONBUDTC. The result is shown below in Table 4.2.

**Classification Table for MOZ**

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Good</th>
<th>Bad</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>0</td>
<td>I</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>Bad</td>
<td>1</td>
<td>3 I</td>
<td>14 I</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>14</td>
<td>I</td>
</tr>
</tbody>
</table>

**Overall 82.35%**

Table 4.2 - Performance based on variables (with ONBUDTC replaced by PMEXP and PROJTYPE) used in basic model.

This prediction is reflected below in Equation 4.2. No PROJTYPE term is present because it is not significant in this case.

\[
\text{logit (MOZ)} = -4.168 - (3.898 \text{ A1}) + (0.001 \text{ D4}) + (0.521 \text{ PMEXP}) \\
(3.073) (1.668) (0.000) (0.446) + (0.432 \text{ LOCATION}) \\
(0.211)
\]

**Equation 4.2 - Logistic Regression Model - Maintenance Outcome - Direct Variable (including PMEXP and PROJTYPE) Basis**

117
The result is consistent with that obtained previously, giving a performance around the 80% level.

Two further logistic regressions were undertaken. In the first case ONBUDTC was again replaced, but this time with the predictive model PONBUDTC derived in Equation 3.3.

**Classification Table for MOZ**

The Cut Value is .50

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Good</th>
<th>Bad</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed</td>
<td>0</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>Good</td>
<td>0</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>Bad</td>
<td>1</td>
<td>3</td>
<td>14</td>
</tr>
</tbody>
</table>

Overall 82.35%

Table 4.3 - Performance based on variables (with ONBUDTC replaced by PONBUDTC) used in basic model.

This result is reflected below in Equation 4.3.

\[
\text{logit (MOZ)} = -5.103 - (3.274 A1) + (0.002 D4) + (3.154 \text{ PONBUDTC}) + (0.315 \text{ LOCATION})
\]

\[
(4.114) \quad (1.310) \quad (0.001) \quad (2.645) \quad (0.161)
\]

**Equation 4.3 – Logistic Regression Model – Maintenance Outcome – Direct Variable (including PONBUDTC) Basis**

The second regression reverted back to the use of a PMO model developed in Equation 3.2, but this time the ONBUDTC term in PMO was substituted by the PONBUDTC model, with the result labelled as \( V_{pmo} \), the predicted maintenance outcome variable. The output from the regression is shown in Table 4.4.
Classification Table for MOZ

The Cut Value is .50

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Good</th>
<th>Bad</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 I</td>
<td>1 I</td>
<td>88.24%</td>
</tr>
<tr>
<td>Observed</td>
<td>+----</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>Good</td>
<td>0 I</td>
<td>15 I</td>
<td>88.24%</td>
</tr>
<tr>
<td>Bad</td>
<td>1 I</td>
<td>4 I</td>
<td>76.47%</td>
</tr>
</tbody>
</table>

Overall 82.35%

Table 4.4 - Performance based on $V_{pmo}$ developed from the basic model.

This prediction is modelled in Equation 4.4 below.

$$\text{logit (MOZ)} = 0.368 + (2.722 V_{pmo})$$

$$\text{(0.489) (0.901)}$$

Equation 4.4 – Logistic Regression Model – Maintenance Outcome – Basic Model Development

The coefficients of the various significant variables in the five equations are summarised in Table 4.5, together with the associated error terms.

<table>
<thead>
<tr>
<th>Eq</th>
<th>PMO</th>
<th>A1</th>
<th>D4</th>
<th>ONBTC</th>
<th>LOCN.</th>
<th>SDS</th>
<th>PMEXP</th>
<th>PONBTC</th>
<th>$V_{pmo}$</th>
<th>% GOOD</th>
<th>% BAD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>-3.2</td>
<td>0.001</td>
<td>1.985</td>
<td>0.398</td>
<td>0.28</td>
<td></td>
<td>82.4</td>
<td>88.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>0.000</td>
<td>1.199</td>
<td>0.197</td>
<td>0.28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>-3.9</td>
<td>0.001</td>
<td>0.432</td>
<td>0.521</td>
<td>0.446</td>
<td></td>
<td>82.4</td>
<td>82.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.7</td>
<td>0.000</td>
<td>0.211</td>
<td>0.446</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>-3.3</td>
<td>0.002</td>
<td>0.315</td>
<td>3.154</td>
<td>2.645</td>
<td></td>
<td>82.4</td>
<td>82.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>0.001</td>
<td>0.161</td>
<td>2.645</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td></td>
<td></td>
<td>2.722</td>
<td>88.2</td>
<td>76.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.901</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5 – Logistic Model Summary
All the models that have been explored provide an overall performance of approximately 80%. This, however, is not necessarily synonymous with achieving a predictive performance at this level. What has been demonstrated is an association between the covariates and the outcomes. The best performance is obtained from Equation 4.1, which contains all of variables from the maintainability predictor as discrete covariates. However, relatively little of this performance is lost in moving from the situation of a known outcome for the software development phase, as reflected in Equation 4.1, to the more realistic situation in which a prediction has to be made, as represented by the last three models. To assess the predictive power of the method a 'hold-out' sample of five projects was withdrawn from the 34 in the survey and Equation 4.4 was effectively recalculate using the balance of 29 projects. This resulted in the model shown below.

$$\text{logit (MOZ)} = 0.736 + 2.900*V_{pmo}$$

$$\text{(0.590) (1.090)}$$

Equation 4.5 – Logistic Regression Model – Maintenance Outcome – Maintainability Predictor Development – Hold-out Sample

Equation 4.5 can be seen to be similar in form to Equation 4.4, with the difference that the constant term has become significant.

Applying Equation 4.5 to the balance of five projects produced a high level of agreement between the calculated result and observed result; for this small sample all five of the calculations gave the same categorisation as had been observed.

This analysis suggests that a simple and easy to use management tool can be developed that can provide a rough assessment of the likely success which will be experienced in maintaining software, prior to the commitment of development resource. There are, however, certain prerequisites. For any given project it will be necessary to have a minimum level of knowledge. The nature of the proposed computer architecture must be understood. A scoring regime must be implemented which facilitates a consistent approach to rating the more subjective aspects of the model such as the likely quality of the design specification and the domain experience of the engineering team. Two possible approaches to addressing the consistency question, perhaps used in tandem, are envisaged. A ‘bottom up’ methodology would
engender greater objectivity through the identification of attributes against which the particular variable would be rated. For example, attributes relating to domain experience might include average team member years of experience in the field, academic and professional qualifications, and outputs from recent performance appraisals. These would aggregate into a synthetic score and map across to the ‘1-9’ range required by the questionnaire. A ‘top down’ approach would require several independent ratings to be produced that would then be processed through a Delphi-type method to converge on an agreed input value. The degree of rigour applied to these approaches is likely to be related to the size and sophistication of the organisation, but may impact on the quality of the predictions subsequently made. It is also possible that the model performance may be improved by tuning to reflect the dynamics of a particular organisation. Although the five covariates that are present in the model developed in Equation 3.2 probably remain relevant their relative importance as influencers of the maintenance outcome may vary, depending upon the nature of the business and the way in which software development and maintenance is organised.

Several other aspects of the method require exploration. One particular issue is the effect on the modelling approach of the definition of ‘good’ and ‘bad’ outcomes. In all the examples cited above the boundary was defined by a maintenance outcome value of zero, with outcomes below this value being regarded as good. Scale may also be an issue. For example, below a certain size it may be that some of the variables become irrelevant. Given a prediction of the likely maintenance outcome for a proposed software project, another important dimension is the sensitivity of the prediction to the values of the input variables. The ability of the management to identify options that will affect the predicted outcome is essential if the tool is to provide practical value. These issues are explored in the remainder of 4.2.

This sensitivity of the approach to these considerations is explored below.

In the first case obtaining a ‘good’ outcome required a more stringent criterion to be achieved, with a value of less than –0.05 required. The maintenance outcomes (MO1) were categorised on this basis into a new variable (MOX). The logistic regression used the prediction PMO as the covariate and MOX as the dependent variable.
The opposite situation can be considered by setting the acceptance criterion at the less demanding 0.05 level, and categorising on this basis into another variable (MOY). The results of this analysis are summarised below in Table 4.6.

<table>
<thead>
<tr>
<th>Acceptance Criterion</th>
<th>Dependent Variable</th>
<th>Constant</th>
<th>PMO Coefficient</th>
<th>Performance (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>Logit(MOZ)</td>
<td>-0.240*</td>
<td>2.651</td>
<td>76.5</td>
</tr>
<tr>
<td>-0.05</td>
<td>Logit(MOX)</td>
<td>0.511*</td>
<td>3.041</td>
<td>82.4</td>
</tr>
<tr>
<td>0.05</td>
<td>Logit(MOY)</td>
<td>-0.240*</td>
<td>2.651</td>
<td>76.5</td>
</tr>
</tbody>
</table>

* - Not significant

Table 4.6 - Performance Based on PMO Calculated using the Maintainability Predictor with Different Acceptance Criteria.

In both cases of revised acceptance criteria the results again provide a performance around the 80% level, reflecting the robustness of the method. It is possible that calibrating the model to reflect the characteristics of a particular organisation could enhance this. Not only are scoring criteria required for the variables embedded in the model, but a common understanding of what characterises a ‘good’ or ‘bad’ outcome must also be established by the organisation. This may vary between organisations. Differences of culture, methods and technologies between organisations may need to be reflected in the coefficients embedded in the model, attaching to the variables already identified as possessing some general predictive applicability. This is explored by applying Equation 3.4, the model developed using data from within just one organisation, to two logistic regressions. In the first it was applied to the maintenance outcomes from that single organisation, and in the other it was applied to a randomly drawn sub-set of the sample projects. Since Equation 3.4 was developed utilising data from a single organisation, it may be more closely calibrated to the norms of that organisation rather than to those of a random sample. This calibration effect should be reflected to some extent in a better predictive performance for the single organisation than for the random sample. The result of the regressions was a model performance of around 86% in the first case, compared to 80% in the second. This result supports the intuitive view that calibrating the model within an organisation may have some benefits, but that they may be small. Calibration would be achieved within an organisation by adopting an approach similar to some of the research in Chapter 3. This process is reflected in Figure 4.4.

122
Figure 4.4 – Categorical model implementation

A statistically significant population of past projects with known maintenance outcomes would have to be rated, recognising the prerequisite of developing a consistent scoring regime, and a regression calculation performed. The resultant model should then be tested against data from some other past projects from within the organisation.

The applicability of the logistic regression approach to projects of different sizes is an important consideration. The peculiarities of any given organisation may affect the weightings given to the five factors identified in Chapter 3 as being consistently important across a range of projects, but are there general conclusions that can be drawn about the relationship between the scale of the project and the performance of the model? Segregating the sample projects on the basis of program size around a median value of 53 KDSI indicates a model performance of approaching 90% for the larger programs compared to 70% for the smaller ones. A randomly selected set of
17 programs drawn from the sample projects indicated a model performance around the 80% level. Extending the analysis further to consider progressively smaller populations at either end of the size distribution, constrained by the statistical significance of the sample sizes, confirmed this divergence. Although the model that has been generated reflects an aggregation of the properties of the sample population, other effects may begin to influence the maintenance outcome to an increasing extent where smaller programs are concerned. Several reasons for this divergence can be postulated, relating both to the intrinsic properties of the program and also to the environment in which it has been produced. One contributor to this divergence may be that the smaller programs in the sample tend to have a greater preponderance of proprietary architectures, compared to the tendency toward open architectures used for the larger projects. This may result in constraints to the maintenance process, either because the proprietary system is inherently harder to maintain because personnel with the requisite knowledge, or hardware, are less readily available. Maintenance staff availability and access to replica equipment both score worse for the smaller projects, while in most other respects the results are similar. Uncertainty exists as to whether these shortcomings arise from the constraints associated with a proprietary system, or that the smaller size of the project made it less important to the organisations in the sample, resulting in low priority access to available resources. The conclusion that can be drawn is that limits do exist as to the applicability of a single model across all program sizes because other factors begin to exert influence.

The sensitivity of the predicted maintenance outcome to changes in the value of the input variables was explored. This was achieved by examining the effect of changing the values on the average maintenance outcome score for the whole sample population. For example, if 10% more of the maintenance activity is performed at the customer site, what is the impact on the predicted maintenance outcome? Table 4.7 indicates the sensitivity of the maintenance outcome to various strategies. Commencing with a baseline of zero the impact of improved scores for each of the five input variables was considered by hypothesising alternative improvement strategies. A cumulative improvement in all five of the variables has been assigned a value of 100%, and improvements to individual variables have been expressed as a percentage of this aggregate. The analysis indicates that ensuring that an open and distributed architecture is adopted can engender the largest and most beneficial change. If this is already planned, or is impossible to adopt, then the next biggest impact results from ensuring that the development project meets budget. Examination
of the influence exerted by the two variables identified as being important to meeting the development budget indicates that a high proportion of software reuse has much greater significance than the experience of the project manager. However, even if reuse predominates there is still no certainty that the budget will be met; other variables are also at work. The other three variables are perhaps easier to influence but have less impact on the overall result. If all three of these variables are assumed to have been improved by the stated amounts then their combined effect equates to that of the architecture change.

<table>
<thead>
<tr>
<th>Management Strategy</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Baseline – Reflects average input variable scores contained within the sample.</td>
<td>0%</td>
</tr>
<tr>
<td>2. A 10% increase in the proportion of maintenance activity undertaken at the customer site is assumed.</td>
<td>7%</td>
</tr>
<tr>
<td>3. The SDS is assumed to be a qualitative 10% better.</td>
<td>14%</td>
</tr>
<tr>
<td>4. The domain experience of the development team is assumed to be improved by a qualitative 10%.</td>
<td>16%</td>
</tr>
<tr>
<td>5. The development budget is assumed to be met.</td>
<td>29%</td>
</tr>
<tr>
<td>6. An open and distributed architecture is assumed.</td>
<td>36%</td>
</tr>
<tr>
<td>7. All the above are assumed.</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 4.7 – Sensitivity of Maintenance Outcome

General conclusions that can be identified regarding the approach explored in 4.2 are that a tool can be developed to enable an organisation to access a 'rough cut' assessment as to whether a planned software project possesses characteristics likely to result in an acceptable maintenance outcome. It is probable that a model whose coefficients have been developed to take account of the particular organisation will tend to generate more powerful results. Within the organisation it is important to recognise that the model has applicability across a certain size range, and beyond this limit the predictions may become progressively less accurate. Advantages of the method include ease and economy of development, given access to information relating to past organisational maintenance performance, and simplicity of application. Beyond that it is also possible to infer from the model and resultant prediction which changes to expected project characteristics are likely to engender an improved maintenance outcome.
4.3 Continuous Prediction

In 4.2 methods of exploring the likely acceptability of a maintenance outcome were investigated. The output was categorical, with a discrete output consisting of a 'good' or 'bad' result. In 4.3 the analysis proceeds to generate a continuous result, providing a numerical prediction of the probable level of corrective maintenance effort. The objective is again to demonstrate the construction of a useful management tool relevant to quantifying future maintenance effort at the project feasibility phase. To be useful the method needs to generate meaningful results and preferably be sufficiently simple to apply that specialist statistical expertise is not required, other than in customising the tool to accommodate particular organisational characteristics and maintaining it in the same context.

In pursuit of this continuous objective some new data is introduced to complement the information from 34 projects (Survey 1) utilised in Chapter 3 and 4.2. This consists of material from a further six projects (Survey 2) from within a single organisation. These projects all relate to the development of training simulators. In addition to the survey content further numerical data relating to levels of corrective maintenance has also been acquired. The approach adopted in 4.3 is outlined below in Figure 4.5.

![Figure 4.5 - Continuous Model Development Process](image-url)
As outlined in Figure 4.5, the analysis to be pursued in 4.3 involves several stages. Initially the predicted maintenance outcome is calculated for each of the projects in Survey 2 using Equation 3.2, the maintainability predictor developed using the data from the 34 projects in Survey 1. This model is restated below in Equation 4.6.

\[
PMO = -2.5 - (0.67 \, A1) + (0.0005 \, D4) + (0.75 \, ONBUDTC) \\
(0.44) \quad (0.22) \quad (0.00) \quad (0.22) \\
+ (0.11 \, LOCATION) + (0.18 \, SDS) \\
(0.03) \quad (0.06)
\]

**Equation 4.6 - Maintainability Predictor**

The figures in parentheses below each of the coefficients contained in the equation are the values of the standard errors.

where: 
- **PMO** is the predicted maintenance outcome
- **A1** relates to possessing an open architecture
- **D4** reflects the domain knowledge of the development engineers
- **ONBUDTC** reflects how well the development project met budget
- **LOCATION** reflects the proportion of maintenance performed at site
- **SDS** reflects the quality of the design specification

These six predictions arising from Survey 2 are then compared with the actual result achieved on each of these projects, as calculated using the factor analysis method previously applied in Chapter 3. Confirmation is sought that some reasonable association can be identified between the prediction and the factor analysis outcome. This initial confirmatory stage can be regarded as an example of the Deming (1982) PDCA cycle. Also undertaken is an analysis of the relationship between development size and maintenance resource requirements on each of the six projects. Given an understanding of this relationship and confirmation of the maintainability predictions, a plot is generated of standardised (in relation to development effort) maintenance effort against predicted maintainability. A model is then developed to fit this plot. This model then enables a prediction of future maintenance effort to be hypothesised, given an estimate of likely development effort and a maintainability prediction. This analysis is described in detail in the remainder of 4.3.
A summary of the comparative attributes of the two samples is shown in Table 4.8.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Survey 1</th>
<th>Survey 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median program size (KDSI)</td>
<td>53</td>
<td>205</td>
</tr>
<tr>
<td>Average module size (KDSI)</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>% modules of new software</td>
<td>66</td>
<td>74</td>
</tr>
<tr>
<td>% open/dist. architecture</td>
<td>40</td>
<td>33</td>
</tr>
<tr>
<td>Predominant project type</td>
<td>New development</td>
<td>New development</td>
</tr>
<tr>
<td>Predominant application type</td>
<td>Scientific</td>
<td>Scientific</td>
</tr>
<tr>
<td>Average SDS score</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Average domain exp. score</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Ave. project. mngt. exp. score</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Ave. ‘maint. at site’ score</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>% dev. projects on budget</td>
<td>53</td>
<td>83</td>
</tr>
<tr>
<td>Ave. dev. effort (man months)</td>
<td>338</td>
<td>395</td>
</tr>
</tbody>
</table>

Table 4.8 – Comparison of Project Attributes – Survey 1 vs. Survey 2

The information gathered from the two surveys has many similarities and a few differences. The median size of program is larger for Survey 2, although if the mean is considered Survey 1 is the greater; the spread of sizes contained in Survey 2 is much smaller. Although comparable amounts of average development effort are associated with the two surveys, development productivity is higher for Survey 1 because of the greater mean program size. Although Survey 1 has a larger proportion of projects with open and distributed architectures, Survey 2 has the greater proportion with open architectures, whether centralised or distributed. The projects from Survey 2 have a greater propensity to meet their development budgets, despite the higher proportion of new software involved. The predominant application areas and project types are the same, and the qualitative scores are all similar.

The approach taken to quantifying likely levels of corrective maintenance effort reflects some of the work of Boehm (1981) and Granja-Alvarez & Barranco-Garcia (1997), as described in 2.9. Two principles particularly underlie the predictive methods that they have developed. The first relates to the likely existence of a relationship between development effort and the subsequent maintenance effort. This
seems to be a rational starting point in that the amount of effort required to develop the software product is likely, to some degree, to be a reflection of both the scale and perhaps also the complexity of the project. These attributes may also affect the levels of maintenance effort that are subsequently needed. Evidence for this is contained in the results obtained by these researchers. The second principle involves an acknowledgement that the inherent maintainability of the software is likely to influence the required maintenance effort. The less maintainable the software, the more effort may be consumed in achieving a particular maintenance objective. This principle is recognised implicitly, although perhaps incompletely, by Boehm (1981) in the application of an “effort adjustment factor” in his maintenance resource model. The method was reviewed in 2.9. Granja-Alvarez & Barranco-Garcia (1997) demonstrate a more explicit recognition of the principle in their deployment of a “maintainability index” designed to reflect maintainability. The difference of approach adopted in the analysis developed in this section is in the quantification of maintainability through the application of the predicted maintenance outcome variable, \( V_{pno} \). This variable is the output from the maintainability predictor, Equation 4.6, moderated by the development performance as assessed in Equation 3.3, and reflects the values of the input variables that seem to exert the greatest influence on the probable maintenance outcome. These were identified as being the presence (or absence) of an open and distributed architecture, the domain experience of the development team, the quality of the system design specification, the location at which the maintenance was undertaken, and whether or not the development project was completed within budget. Given that the objective is to assess likely maintenance effort using \( V_{pno} \) prior to the commencement of the development, the model also utilises the assessed level of software re-use and the project management expertise to predict whether or not the development budget will be met.

The hypothesis that has been developed, consistent with the work of both Boehm (1981) and Granja-Alvarez & Barranco-Garcia (1997), and the earlier identification of the principal influencers of maintainability embedded in the calculation of \( V_{pno} \) is:

\[
R_{\text{maint.}} = f(V_{pno}, R_{\text{dev}})
\]

**Equation 4.7 – Maintenance resource influencers**

where:

- \( R_{\text{maint.}} \) is the maintenance resource required
- \( V_{pno} \) is the predicted maintenance outcome variable
- \( R_{\text{dev}} \) is the expected development resource
The nature of this relationship is explored using the data collected in Survey 2. Apart from the variables captured through the questionnaire described in 3.4 and located in Appendix 1, this survey also contains information relating to levels of corrective maintenance experienced during the first two years of operational use subsequent to acceptance of the developed system. Development and maintenance effort expended on each of the six projects is summarised in Table 4.9.

<table>
<thead>
<tr>
<th>Project</th>
<th>Development effort (man months)</th>
<th>Corrective maint. effort (man months)</th>
<th>Application area</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>191</td>
<td>29</td>
<td>Train driving simulation</td>
</tr>
<tr>
<td>M2</td>
<td>705</td>
<td>166</td>
<td>Power station control simulation</td>
</tr>
<tr>
<td>M3</td>
<td>176</td>
<td>34</td>
<td>Train driving simulation</td>
</tr>
<tr>
<td>M4</td>
<td>227</td>
<td>28</td>
<td>Train driving simulation</td>
</tr>
<tr>
<td>M5</td>
<td>621</td>
<td>188</td>
<td>Power station control simulation</td>
</tr>
<tr>
<td>M6</td>
<td>448</td>
<td>101</td>
<td>Medical procedures simulation</td>
</tr>
</tbody>
</table>

Table 4.9 – Development and corrective maintenance effort

The average ratio of corrective maintenance to development is 21%, but with a spread of 12% to 30%. The results, based on a small sample, suggest that there may be a relatively linear relationship between development effort and maintenance. Applying the Pearson Coefficient of Determination generates a value of 0.972, indicating a high and statistically significant level of correlation between development effort and corrective maintenance effort. This relationship can be illustrated on a scatter diagram, as shown in Figure 4.6.
The other source of variation, $V_{pno}$, identified in Equation 4.7 arises from the maintainability of the software. This is explored below in Table 4.10. The table includes both $V_{pno}$, and also the maintenance outcome factor (MO1), which is included for reference. The value of MO1 is determined by factor analysis of the maintenance frequency and difficulty outcomes reflected in the scores for each project. This methodology was discussed in 3.6. The variable, $V_{pno}$, is an attempt to predict MO1 prior to the commencement of the development project, using Equation 4.6, the maintainability predictor developed in Chapter 3, and Equation 3.3, the development predictor.

<table>
<thead>
<tr>
<th>Project</th>
<th>Development effort (man months)</th>
<th>Corrective maint. effort (man months)</th>
<th>Maint. outcome factor</th>
<th>$V_{pno}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>191</td>
<td>29</td>
<td>-0.62</td>
<td>-0.64</td>
</tr>
<tr>
<td>M2</td>
<td>705</td>
<td>166</td>
<td>0.55</td>
<td>0.23</td>
</tr>
<tr>
<td>M3</td>
<td>176</td>
<td>34</td>
<td>-0.15</td>
<td>-0.40</td>
</tr>
<tr>
<td>M4</td>
<td>227</td>
<td>28</td>
<td>-1.56</td>
<td>-0.62</td>
</tr>
<tr>
<td>M5</td>
<td>621</td>
<td>188</td>
<td>1.24</td>
<td>0.26</td>
</tr>
<tr>
<td>M6</td>
<td>448</td>
<td>101</td>
<td>0.55</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 4.10 – Effort and associated maintainability factors
In summary, Table 4.10 is an extension of Table 4.9, including both the effort statistics but also the maintenance outcome factor (MO1) derived for each project in Survey 2 using factor analysis, and the predicted maintenance outcome variable ($V_{p\text{mo}}$). A test of the strength of the correlation between MO1 and $V_{p\text{mo}}$ confirms that they are significantly correlated.

The association between the factor analysis and the predicted ratings of comparative maintainability is noted below in Table 4.11. Both the outcome and prediction should be interpreted as regarding M1 and M4 as the two most maintainable projects and M5 as the least maintainable. The Spearman Coefficient of Correlation is 0.928, confirming a high level of correlation between the outcome and prediction.

<table>
<thead>
<tr>
<th>Project</th>
<th>Maint. outcome factor</th>
<th>Outcome rating</th>
<th>$V_{p\text{mo}}$</th>
<th>Predicted rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>-0.62</td>
<td>2</td>
<td>-0.64</td>
<td>1</td>
</tr>
<tr>
<td>M2</td>
<td>0.55</td>
<td>4</td>
<td>0.23</td>
<td>5</td>
</tr>
<tr>
<td>M3</td>
<td>-0.15</td>
<td>3</td>
<td>-0.40</td>
<td>3</td>
</tr>
<tr>
<td>M4</td>
<td>-1.56</td>
<td>1</td>
<td>-0.62</td>
<td>2</td>
</tr>
<tr>
<td>M5</td>
<td>1.24</td>
<td>6</td>
<td>0.26</td>
<td>6</td>
</tr>
<tr>
<td>M6</td>
<td>0.55</td>
<td>4</td>
<td>0.07</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 4.11 – Maintainability prediction**

Returning to continuous estimation, if a broadly linear relationship is attributed to the influence of development effort on corrective maintenance, with other variation attributable to the inherent maintainability of the software, then the associations can be simplified by standardising the maintenance effort in relation to the development effort. A new variable, the standardised effort statistic, $R_{\text{std}}$, can be defined that reflects this ratio. Given that an estimate already exists for the likely development effort associated with the project, probably determined by one of the methods described in 2.7, and that maintainability can be predicted through $V_{p\text{mo}}$, then any general relationship that can be established between them should enable corrective maintenance effort to be estimated.
Equation 4.7 can be simply replaced by:

\[ R_{m/d} = f(V_{p_{mo}}) \]

Equation 4.8 – Maintenance variation function

Table 4.10 can be simplified into terms, \( R_{m/d} \) and \( V_{p_{mo}} \). The former is the ratio of maintenance effort to development, and the latter is the maintainability predictor. This is shown below in Table 4.12.

<table>
<thead>
<tr>
<th>Project</th>
<th>( R_{m/d} )</th>
<th>( V_{p_{mo}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.15</td>
<td>-0.64</td>
</tr>
<tr>
<td>M2</td>
<td>0.24</td>
<td>0.23</td>
</tr>
<tr>
<td>M3</td>
<td>0.19</td>
<td>-0.40</td>
</tr>
<tr>
<td>M4</td>
<td>0.12</td>
<td>-0.62</td>
</tr>
<tr>
<td>M5</td>
<td>0.30</td>
<td>0.26</td>
</tr>
<tr>
<td>M6</td>
<td>0.23</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 4.12 – Standardised effort statistic and associated maintainability predictor

The nature of the relationship between these variables can be illustrated with a scatter diagram. This is shown below in Figure 4.7, with achieved standardised effort (\( R_{m/d} \)) plotted as a function of a predicted maintainability (\( V_{p_{mo}} \)). A Pearson Coefficient of Correlation of 0.938 pertains between these variables, indicating high correlation.

Corrective Maintenance Effort vs.

Predicted Maintainability

Figure 4.7 – Variation in maintenance effort with predicted maintainability
The scatter plot does provide an indication that the resource required to maintain these programs, as represented by the standardised effort statistic, seems to vary with the predicted maintainability of the software. This relationship can be explored further through the application of a mathematical model. Assuming that such a model can be identified the opportunity then exists to make predictions relating to probable corrective maintenance effort, given an estimate of likely development effort and an algorithm to quantify maintainability. The weakness of the analysis reflected in Figure 4.7 is the limited amount of data available. However, one observation that may pertain is that the relationship between maintenance effort and the predicted maintainability factor may be non-linear. Plausible reasons for this hypothesis exist and are reflected in existing software estimation models. For example, both economies and diseconomies of scale may pertain. Fenton & Pfleeger (1997) note that "most models suggest that effort is approximately proportional to size, but they include an adjustment for diseconomy of scale". Both Fenton & Pfleeger (1997) and Shooman (1988) explore this in detail. Examples include project management effort and system integration costs. It is likely, however, that an upper limit applies to the level of maintenance effort that can be usefully applied.

A commonly used non-linear model of the general form shown in Equation 4.9 was selected to explore this relationship further.

$$y = \frac{A}{1 + \left(\frac{A}{B} - 1\right)e^{-C\alpha}}$$

Equation 4.9 – Logistic Model

In this model A is the saturation value, B is initial value and C is the initial rate of growth. In the context of maintenance estimation the independent variable (x) is the maintainability predictor (V_{pmo}), and the predicted standardised effort statistic ($\hat{R}_{m/d}$) would be substituted for the dependent variable (y). Consideration was given to the maximum possible value of A, and particularly the option setting assigning it a value of one. This would have the effect of providing a normalised value for the ratio, because it would always lie between zero and one. However, as no reason exists for future maintenance resource requirements not to exceed estimated development effort, it is possible for $\hat{R}_{m/d}$ to be greater than unity. Therefore A remains a variable, with its value determined by non-linear regression.
Fitting Equation 4.9 for the data contained in Table 4.12 using the SPSS non-linear regression tool generates a predictive model of the form:

\[
\hat{R}_{m/d} = \frac{1.175}{1 + (\frac{1.175}{0.226} - 1)e^{-0.83SV_{pmo}}}
\]

\[= \frac{1.175}{1 + 4.2e^{-0.83SV_{pmo}}} \]

Equation 4.10- Predicted standardised effort

Comparison of the value of the predicted standardised effort statistic (\(\hat{R}_{m/d}\)) calculated using Equation 4.10 with the achieved standardised effort (\(R_{m/d}\)) is made below in Table 4.13.

<table>
<thead>
<tr>
<th>Project</th>
<th>(R_{m/d})</th>
<th>(\hat{R}_{m/d})</th>
<th>(V_{pmo})</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.15</td>
<td>0.14</td>
<td>-0.64</td>
</tr>
<tr>
<td>M2</td>
<td>0.24</td>
<td>0.26</td>
<td>0.23</td>
</tr>
<tr>
<td>M3</td>
<td>0.19</td>
<td>0.17</td>
<td>-0.40</td>
</tr>
<tr>
<td>M4</td>
<td>0.12</td>
<td>0.14</td>
<td>-0.62</td>
</tr>
<tr>
<td>M5</td>
<td>0.30</td>
<td>0.27</td>
<td>0.26</td>
</tr>
<tr>
<td>M6</td>
<td>0.23</td>
<td>0.24</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 4.13 – Achieved & predicted effort ratios; and maintainability predictor

The performance of the model can be seen in Figure 4.8, in which both the actual and predicted effort ratios are plotted against the predicted maintainability. A high degree of correlation between \(\hat{R}_{m/d}\) and \(V_{pmo}\) is confirmed by the Coefficient of Correlation test.
Corrective Maintenance Effort

vs. Predicted Maintainability

(Actual and Predicted Effort)

Figure 4.8 – Variation in maintenance effort with predicted maintainability

Within the constraints of the available data and the range of predicted maintainability ratios considered, a good fit is obtained between predicted and achieved standardised effort ratios. This methodology has the potential to be developed into a management tool capable of making useful predictions of probable maintenance effort.

This approach has the merit of building on the same foundation as that used in the categorical prediction method discussed in 4.2. It requires the construction of a maintainability predictor utilising the method developed in Chapter 3, resulting in the model summarised in Equation 4.6. This process is summarised below in Figure 4.9.
Establish scoring regime:
- dev. domain experience
- quality of SDS
- project manager experience
- level of re-use
- maintenance location

Establish architecture definition:
open & distributed

Survey past projects

Develop & verify maintainability predictor (\( V_{pm} \))

Develop & verify standardised effort predictor (\( \hat{R}_{std} \))

Implement & maintain predictor (\( \hat{R}_{std} \))

Access development estimation model

Figure 4.9 – Continuous model implementation

Additionally, the assumption is made that a method has already been developed by the organisation to assess development effort. From the limited data set contained in Survey 2 the inference has been made that a reasonably linear relationship exists between the scale of the development effort and the likely magnitude of the subsequent corrective maintenance. It is recognised that this inference is an approximation that can be affected by several factors. For example, time taken to capture and document requirements that are reflected in a comprehensive and formal design specification, and ultimately in the delivered system may tend to increase the size of the development task, but reduce the amount of corrective maintenance required. Some of the larger deviations from the linear can be recognised by using the maintainability predictor as a correction factor. This general relationship is reflected in Equation 4.8, while Equation 4.10 is the specific form calculated by a non-linear regression method as being appropriate to the particular organisation associated with Survey 2.
In adopting the modelling approach described above it is important to recognise that what is being produced is an estimate, which possesses the characteristics described in 2.7. The notion that it lies at the centre of a distribution of possible outcomes is particularly significant, and implies that the model should be used to provide a range rather than a single point estimate. This enables the uncertainty associated with a prediction, and the attendant risk, to be better understood. The concept of the confidence interval is therefore introduced. Interval estimates are commonly used in times series analysis; the confidence interval is used to assign limits to the probable values of parameters within a model. Table 4.13 can be extended to include upper and lower confidence intervals. These are included in Table 4.14 and also in Figure 4.10.

<table>
<thead>
<tr>
<th>Project</th>
<th>$R_{m/d}$</th>
<th>$\hat{R}_{m/d}$</th>
<th>UCI</th>
<th>LCI</th>
<th>$V_{pme}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.15</td>
<td>0.14</td>
<td>0.18</td>
<td>0.10</td>
<td>-0.64</td>
</tr>
<tr>
<td>M2</td>
<td>0.24</td>
<td>0.26</td>
<td>0.30</td>
<td>0.22</td>
<td>0.23</td>
</tr>
<tr>
<td>M3</td>
<td>0.19</td>
<td>0.17</td>
<td>0.20</td>
<td>0.14</td>
<td>-0.40</td>
</tr>
<tr>
<td>M4</td>
<td>0.12</td>
<td>0.14</td>
<td>0.18</td>
<td>0.10</td>
<td>-0.62</td>
</tr>
<tr>
<td>M5</td>
<td>0.30</td>
<td>0.27</td>
<td>0.31</td>
<td>0.23</td>
<td>0.26</td>
</tr>
<tr>
<td>M6</td>
<td>0.23</td>
<td>0.24</td>
<td>0.27</td>
<td>0.21</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 4.14 – Achieved & predicted effort ratios; & confidence intervals

![Figure 4.10](image-url)

Figure 4.10 – Variation in maintenance effort with predicted maintainability (Interval analysis)
The intervals that have been calculated are at the 95% level. Note that in practice it would be desirable to undertake these calculations using data from a larger population of projects, which would tend to reduce the width of the confidence interval.

4.4 Discretionary Maintenance

The discussion developed above in 4.3 relates to the corrective maintenance task that may be regarded as essential and frequently urgent, once the fault is identified. The imperative exists because the software is unable to perform to specification and consequently does not fully satisfy the requirement. Urgency may pertain because of operational demands. This deficiency may be purely objective in nature, or may also possess a qualitative dimension with the software not addressing a requirement satisfactorily from the customer perspective. In either case the outcome is likely to involve the commitment of maintenance resource until the software performance satisfies either customer expectations or commercial obligations, or preferably both.

The associated level of effort required within any given organisation is likely to be a function of the scale of the software project and the inherent maintainability of the software.

Various forms of non-corrective maintenance were identified and discussed in 2.4. These may be regarded as discretionary to the extent that either the scope or timing of the maintenance activity can be proactively planned and scheduled, rather than a limited resource reacting to unscheduled events. Managing these maintenance objectives again requires an understanding of the likely levels of resource involved. As was noted in Chapter 2 the majority of maintenance activity tends to be of a non-corrective nature. It is desirable that tools are available to quantify the cost and resource implications of this activity during the early feasibility stages of the project. From a commercial perspective this may be of particular interest because of the income it may provide.

The methodology described in 4.3 can be adapted to support this objective in the non-corrective context. As with corrective maintenance, scale and maintainability are likely to be determinants of the effort involved. A third dimension, which reflects the partially elective nature of these other maintenance categories, is simply the amount of maintenance that is planned and the attendant resource demand. An established
approach to this issue is the concept of annual change traffic (ACT); this was identified and discussed in 2.9. The method, used by both Boehm (1981) and Granja-Alvarez & Barranco-Garcia (1997), utilises lines of code and their rate of change as a measure of the degree of discretionary change being planned or undertaken. This parameter is calculated using Equation 4.11 below.

\[
\text{ACT} = \frac{\text{NNL} + \text{NML}}{\text{NOL}}
\]

Equation 4.11 – Annual Change Traffic

where:

- NNL is the number of new lines
- NML is the number of modified lines
- NOL is the number of original lines

New and modified lines of code, expressed as a proportion of the original program size, are a surrogate for the maintenance effort as a proportion of the original development effort. An implied assumption is that a relationship exists between resource requirements and lines of code. To the extent that, for example, an adaptive maintenance project can be regarded as a small development project this is probably valid. Many of the tasks, ranging from requirements capture through to system test and acceptance may be seen to have parallels. Areas of potential difference may relate to the individual situation. If the project is to be undertaken by the original development team, with access to the target system, end users and user documentation, then a particular resource level will pertain. Alternatively, if a new team undertakes the project, with limited system access and poor quality documentation, then a different resource requirement may well pertain. Issues of scale and complexity may also arise. A comparatively small number of line changes may result in disproportionate documentation changes and training if the user interface is undergoing maintenance. Changes to a modelling algorithm within a real time system may require only minor code changes but substantial mathematical development and extensive testing. These concerns may exist at the limit and for the general case, but within the norms of a single organisation, annual change traffic may well provide an adequate means of measurement.

Within the limitations of the ACT metric discussed above, it is possible to develop the paradigm, again building on the work of Boehm (1981) and Granja-Alvarez & Barranco-Garcia (1997), and described by Equation 4.12.
\[ R_{\text{maint}} = f(ACT, V_{\text{pmo}}, R_{\text{dev}}) \]

**Equation 4.12 – Non-corrective maintenance influencers**

Pursuing the approach taken in 4.3 and summarised by Equation 4.8 facilitates further development.

\[ R_{\text{m/d}} = f(V_{\text{pmo}}) \times f(ACT) \]

**Equation 4.13 – Non-corrective maintenance variation function**

**Equation 4.13** is aligned with the concept of a linear scale relationship between development and maintenance resource requirements, as discussed in 4.3. The idea of using the ACT metric as a direct multiplier is also contained in the model. The closeness of this function to that attaching to corrective maintenance, as illustrated in 4.3, depends on the similarities between the corrective and non-corrective maintenance processes within a given organisation. The precise form of the mathematical function relating to \( V_{\text{pmo}} \) requires to be explored in the context of a population of past non-corrective projects, preferably from within a particular organisation. Although intuitively there are similarities, this would require verification. This investigation will require access to data from the target organisation. Despite being beyond the immediate scope of this thesis, and in the absence of a suitable data set, this area is noted as an important field for future research because of the predominance of non-corrective maintenance.

The objective of this non-corrective analysis is to facilitate generation of a management tool capable of supporting planning and scheduling of resources. The format of the developed tool is illustrated in Figure 4.11 below. It is not necessarily the case that a linear relationship as illustrated would pertain; given the experience with the corrective relationship described by **Equation 4.10** it is more probable that a non-linear association prevails. Nevertheless, the utility of the tool is demonstrable. For a given degree of change, and an anticipated level of maintainability, predictions of the level of maintenance effort expressed as a proportion of the estimated development effort can be made.
Once again, the prediction interval approach should also be embedded within the design of the model, reflecting the notion of a range rather than a point estimate.

4.5 Conclusions

Both corrective and non-corrective maintenance prediction have been considered in Chapter 4. In the former case, two complementary approaches have been explored: categorical and continuous prediction. For the latter, ideas relating to the application of the continuous approach to discretionary maintenance have been outlined.

Categorical prediction should enable a judgement to be made early in the life of a planned software project as to the probable acceptability of the subsequent maintenance costs which may be incurred. The analysis, which was undertaken using data drawn from a range of organisations, suggests that useful predictions may be achievable. Although it is necessary to possess some minimum degree of knowledge regarding the proposed design, manning and project execution, no requirement exists to create continuous estimates of either development or maintenance activity. What is essential, however, is the design and implementation of a structured scoring regime capable of generating individual project attribute ratings that are consistent. Additionally, it is believed that limits probably exist regarding the program size range
against which a single predictive model can be applied. An important aspect of this predictive process is the ability to model the effects of changes in the values of input variables on the probable outcome. Inferences can be drawn about the impact of resource changes on the likely future maintenance experience. This latter point is of particular importance, given that an objective is to develop tools capable of providing support to management decision-making.

The objective of the development of an approach to continuous prediction is to enable probable future maintenance effort to be estimated at the project feasibility stage, prior to the commitment of significant development resource. Simplicity of application has been regarded as an important factor. The approach adopted reflects the hypothesis that maintenance effort requirements are likely to be largely determined by the overall scale of the development activity and the inherent maintainability of the software. This hypothesis has been explored using a second survey of several past software projects. It concludes that an approximately linear relationship exists between development and maintenance effort, and that standardised (in relation to development) maintenance effort possesses a non-linear relationship with predicted maintainability when quantified through the predicted maintenance outcome variable. These relationships can be readily modelled and have the potential to be developed into a useful management tool.

In both the categorical and continuous cases it is believed that calibrating the models using data extracted from past projects within the organisation can enhance the predictive power of the models.

Discretionary, or non-corrective, maintenance has also been considered in the context of the continuous approach. The application of the annual change traffic approach was considered as appropriate and a method of developing a management tool capable of accommodating varying levels of change was outlined. This area requires further research using additional data from past projects.

The analysis undertaken in Chapter 4 is all dependent on information gathered during the initial stages of the project, and certainly prior to the commitment of significant development effort. Indeed, the objective of the research undertaken has been to enable useful observations to be made regarding software maintenance costs at the earliest possible stage. However, assuming that the project has passed this first
phase, and software development has begun, can further useful information relating to likely maintenance resource requirements be obtained? The value of making a series of predictions as the project proceeds is examined in Chapter 5.
5.1 Introduction

The approach developed in Chapter 4 built on earlier work that identified those variables that appear to be the most important determinants of software maintainability. The hypothesis that maintenance effort is likely to depend on both the scale of the software development activity and the maintainability of developed software was explored using data relating to six past projects in one organisation. Some success was achieved in demonstrating this association, and a predictive model was constructed. However, the outputs may be regarded as being static in that they reflect only information available early in the life of the project, probably during the feasibility analysis phase or during the course of preparing a tender. In Chapter 5 these ideas are taken further, with consideration given to mechanisms that could be developed to enable forecasts of future maintenance requirements to be updated as more information becomes available. The benefits of making such updates within an organisation relate to the containment of risk. Either the updated predictions of maintenance requirements, and therefore costs, will confirm the initial expectations or will serve to highlight a deviation from plan, enabling corrective action to be taken if required, or financial provision to be made. The content of this chapter may be regarded as being a continuation of what Glass (1994) describes as being the "evaluative phase", with further work proceeding with the intent of designing useful management tools.

The content of Chapter 5 utilises additional data relating to the six software projects considered in the previous chapter. In particular consideration is given to regular monthly reviews of forecast development effort requirements that were undertaken during the software development phase. Given the hypothesis that maintenance costs are partially dependent on the scale of the development activity, can useful revisions be made to the initial maintenance forecast in the light of this new information? Methods of generating updates reflecting revised development forecasts are explored. Consideration is also given to the inclusion of new information relating to the expected maintainability of the software. This may be summarised by recalling the content of Equation 4.7, which is reiterated below.
As quantitative updates to $R_{dev.}$ become available, and possibly also qualitative revisions to $V_{pmo}$, it may be useful for $R_{maint.}$ to be revised. The approach adopted in this chapter is illustrated below in Figure 5.1.

*Figure 5.1 – Dynamic Model Development Process*

5.2 Dynamic Modelling; Development Effort Updates

The regular review of effort expended and progress achieved during the life of a software development project, and the application of concepts such as 'estimate to complete' and 'earned value' are common in many professional software project
environments, and in the more general project management domain. One output is the 'forecast cost at completion' (FCAC), which is simply a statement of the expected total effort or costs likely to have been incurred by the time that the development project has been completed. As the project proceeds and tasks prove more or less onerous than anticipated, and consequently effort consumed or required is at variance with the original budget, so the FCAC will tend to fluctuate. Given that the FCAC in this case is the best available estimate of the total software development effort at any given time, and the hypothesis that the scale of the development effort is a determinant of maintenance costs, consideration of the FCAC provides a means of revising the maintenance estimate. For each of the six projects already discussed in Chapter 4 a series of FCAC's has been captured, reflecting the output from monthly project reviews within the organisation. This information will be used in exploring mechanisms designed to produce updated forecasts of likely maintenance effort. The six projects are not considered in numerical order, but after M1 tend to be grouped according to their characteristics.

Taking project M1 as an example, a series of twenty monthly reviews of forecast development costs have been recorded. This series reflects the best view at any given review of the probable total cost of development. A relationship between development and corrective maintenance costs was reflected in Equation 4.10. This relationship is:

\[ \hat{R}_{m/d} = \frac{1.175}{1 + 4.2e^{-0.835V_{pmo}}} \]

Equation 5.2 - Predicted standardised effort ratio

For the specific case of project M1 it was established in Chapter 4 that this ratio is 0.14. The values for all six of the projects were recorded in Table 4.14. The relationship pertaining in the case of M1 may therefore be summarised as being:

\[ R_{maint} = 0.14 \times R_{dev}. \]

Equation 5.3 – Project M1 Maintenance / Development Relationship

A time series that indicates forecast maintenance costs, based on the above ratio and the development forecasts can be constructed. Additionally, the concept of the prediction interval could be utilised. Montgomery & Johnson (1976) define the prediction interval as "an interval that has a stated probability of containing the
actual future value". Mendenhall & Beaver (1991) provide a useful description of the computational aspects. These intervals tend to narrow as information on past fluctuations accumulates, but diverge as the current value moves away from the mean of the sample of results to date. However, the limitation of this approach is that it makes little use of the prior knowledge that has been accumulated and is reflected in the preceding results. It is merely a calculation of an interval around the current value. A better approach may be to condition the current prediction with recent information.

Rather than merely taking the development forecast, as reflected in the FCAC, as the best available prediction, it may be possible to embody an improved forecast that reflects the attributes of a particular development time series into a maintenance prediction.

An established method of generating a useful forecast in a situation where there is continuing volatility in the time series is to apply discounted regression. This approach assumes that current results may provide a more accurate description of the existing performance of the system than those from the more distant past, and therefore allocates a greater weight to the more recent observations. Montgomery & Johnson (1976) provide detailed description of this method, noting that "observations that are close to the current time period may be of more importance...as the recent data may be more indicative of the true behaviour of the process" and propose seeking an estimator that "minimises the discounted, or weighted, sum of squares of the errors". This concept has been considered in more detail through analysis of M5 using the SPSS autocorrelation function, which analyses the relationship between a current value in a time series and the preceding values. This analysis confirms that any given forecast is heavily influenced by the preceding value and is only affected by about the previous six predictions. This is quite likely in the case of a software development project, with the understanding of the scale of the task improving as the project proceeds and new facts come to light. It is probable that a method that places emphasis on the most recent results may produce superior forecasts.

An alternative approach to making serial predictions of future maintenance resource requirements relates to the application of conditional probability, using Bayesian principles. Mendenhall & Beaver (1991) introduce these ideas through discussion and worked examples, describing it as "a method of incorporating the information from sample observations to adjust the probability of some event". Pole et al (1994)
provide a comprehensive description of the approach. An established resource that supports this methodology is provided by the BUGS Project (2000), a suite of Web accessible software designed to perform “Bayesian inference on complex statistical problems for which there is no exact analytical solution, and for which even standard approximation techniques have difficulties”. The approach could have relevance to the dynamic prediction of future maintenance requirements because conditionality is pertinent and, as observed above, the prediction interval method has limitations. A Bayesian approach might provide a superior alternative. A review of BUGS concluded that it has the benefit of providing a graphical means of developing and describing a model, but that it is complicated to apply. In particular, initial construction of a model would require specialist knowledge, and the relationship between input variables and the output is not transparent for the non-specialist. The comparative benefits of the method would therefore need to be significant to compensate for the overhead costs arising from this complexity.

A better alternative may be a method that facilitates a prediction and an associated confidence interval utilising prior and recent knowledge, but through a conceptually simple and computationally easy process. An approach that was explored and adopted relates to the work of Pole et al (1994) in the development of their Steady State Model (SSM). They observe that “not only is the structure a useful way to introduce the basic procedures of Bayesian forecasting, but it is itself valuable and widely used”. Two equations, respectively the observation equation and the system equation define the method. The former assumes that a true underlying state exists, which is surrounded by random disturbances or noise. The latter accommodates variation or drift in the underlying state over time. A general constraint with the model is that this drift is comparatively slow. The method is straightforward to apply. A forecast mean, \( q_t \), reflects the posterior parameter estimate, \( m_t \). This is stated more formally in Equation 5.4.

\[
q_t = m_{t-1}
\]

**Equation 5.4 – Forecast mean**

A forecast variance, \( Q_t \), is calculated using the posterior variance, \( C_t \), and two constants, \( W \) and \( V \), that are initially assigned to reflect noise and rate of drift. This is described below.

\[
Q_t = C_{t-1} + W + V
\]

**Equation 5.5 – Forecast variance**
An update coefficient, $A_t$, is also required.

$$ A_t = \frac{C_{t-1} + W}{Q_t} $$

*Equation 5.6 – Update coefficient*

An error term, $e_t$, compares the difference between the actual observation, $y_t$, and the forecast, $q_t$.

$$ e_t = y_t - q_t $$

*Equation 5.7 – Error term*

Two recurrence relationships must also be calculated.

$$ m_t = m_{t-1} + A_t \times e_t $$

*Equation 5.8 – Posterior mean*

$$ C_t = A_t \times V $$

*Equation 5.9 – Posterior variance*

Application of the method to project M1 commenced by noting that the predicted corrective maintenance effort equates to 14% of the development effort, reflecting the calculated maintainability of the software. This value is one of the required start conditions, $m_0$. An assessment of the initial variance value reflects an observed average movement in development resource requirements of 11% over the six sample projects. Regarding this as a 95% level means that 11% can be treated as two standard deviations, and an initial variance, $C_0$, can be derived. Values of $V$ and $W$ need to reflect organisational characteristics. Pole et al (1994) provide a method for calculating these parameters, but this requires access to a population of past data. A simpler approach involves setting initial values and adjusting them to optimise the result, and applying them to subsequent projects. In the cases discussed below the value of $W$, which relates to the underlying trend, was set at 5% of $m_0$. $V$, which relates to noise, was given a value 20 times as large as $W$, reflecting the experiences of Pole et al (1994). Given these starting conditions and the relationships described above, a series of predictions can be readily generated using a spreadsheet.
This is summarised below in **Table 5.1**. The key comparison is between the forecast mean value (the observed value), \( q_t \), and the next prediction, \( y_t \).

<table>
<thead>
<tr>
<th>Time</th>
<th>( q_t )</th>
<th>( Q_t )</th>
<th>( \lambda_t )</th>
<th>( y_t )</th>
<th>( e_t )</th>
<th>( m_t )</th>
<th>( C_t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>26.60</td>
<td>48.80</td>
<td>0.18</td>
<td>26.60</td>
<td>0.00</td>
<td>26.60</td>
<td>6.8</td>
</tr>
<tr>
<td>R2</td>
<td>26.60</td>
<td>49.21</td>
<td>0.19</td>
<td>26.60</td>
<td>0.00</td>
<td>26.60</td>
<td>7.21</td>
</tr>
<tr>
<td>R3</td>
<td>26.60</td>
<td>49.49</td>
<td>0.19</td>
<td>26.60</td>
<td>0.00</td>
<td>26.60</td>
<td>7.49</td>
</tr>
<tr>
<td>R4</td>
<td>26.60</td>
<td>49.67</td>
<td>0.19</td>
<td>26.60</td>
<td>0.00</td>
<td>26.60</td>
<td>7.79</td>
</tr>
<tr>
<td>R5</td>
<td>26.60</td>
<td>49.79</td>
<td>0.20</td>
<td>26.60</td>
<td>0.00</td>
<td>26.60</td>
<td>7.86</td>
</tr>
<tr>
<td>R6</td>
<td>26.60</td>
<td>49.86</td>
<td>0.20</td>
<td>26.60</td>
<td>0.00</td>
<td>26.60</td>
<td>7.91</td>
</tr>
<tr>
<td>R7</td>
<td>26.60</td>
<td>49.91</td>
<td>0.20</td>
<td>26.60</td>
<td>0.00</td>
<td>26.60</td>
<td>7.94</td>
</tr>
<tr>
<td>R8</td>
<td>26.60</td>
<td>49.94</td>
<td>0.20</td>
<td>26.60</td>
<td>0.00</td>
<td>26.60</td>
<td>7.96</td>
</tr>
<tr>
<td>R9</td>
<td>26.60</td>
<td>49.96</td>
<td>0.20</td>
<td>26.60</td>
<td>0.00</td>
<td>26.60</td>
<td>7.98</td>
</tr>
<tr>
<td>R10</td>
<td>26.60</td>
<td>49.98</td>
<td>0.20</td>
<td>26.74</td>
<td>0.14</td>
<td>26.63</td>
<td>7.99</td>
</tr>
<tr>
<td>R11</td>
<td>26.63</td>
<td>49.99</td>
<td>0.20</td>
<td>26.74</td>
<td>0.11</td>
<td>26.65</td>
<td>7.99</td>
</tr>
<tr>
<td>R12</td>
<td>26.65</td>
<td>49.99</td>
<td>0.20</td>
<td>27.02</td>
<td>0.37</td>
<td>26.72</td>
<td>7.99</td>
</tr>
<tr>
<td>R13</td>
<td>26.72</td>
<td>49.99</td>
<td>0.20</td>
<td>27.02</td>
<td>0.30</td>
<td>26.78</td>
<td>8.00</td>
</tr>
<tr>
<td>R14</td>
<td>26.78</td>
<td>50.00</td>
<td>0.20</td>
<td>27.02</td>
<td>0.24</td>
<td>26.83</td>
<td>8.00</td>
</tr>
<tr>
<td>R15</td>
<td>26.83</td>
<td>50.00</td>
<td>0.20</td>
<td>26.88</td>
<td>0.05</td>
<td>26.84</td>
<td>8.00</td>
</tr>
<tr>
<td>R16</td>
<td>26.84</td>
<td>50.00</td>
<td>0.20</td>
<td>26.88</td>
<td>0.04</td>
<td>26.85</td>
<td>8.00</td>
</tr>
<tr>
<td>R17</td>
<td>26.85</td>
<td>50.00</td>
<td>0.20</td>
<td>26.88</td>
<td>0.03</td>
<td>26.85</td>
<td>8.00</td>
</tr>
<tr>
<td>R18</td>
<td>26.85</td>
<td>50.00</td>
<td>0.20</td>
<td>26.74</td>
<td>-0.11</td>
<td>26.83</td>
<td>8.00</td>
</tr>
<tr>
<td>R19</td>
<td>26.85</td>
<td>50.00</td>
<td>0.20</td>
<td>26.74</td>
<td>-0.09</td>
<td>26.81</td>
<td>8.00</td>
</tr>
<tr>
<td>R20</td>
<td>26.83</td>
<td>50.00</td>
<td>0.20</td>
<td>26.74</td>
<td>-0.09</td>
<td>26.81</td>
<td>8.00</td>
</tr>
</tbody>
</table>

**Table 5.1 – Project M1 Maintenance Forecasts**

This analysis is also displayed graphically in **Figure 5.1**, with a trend line superimposed on the prediction series.
Figure 5.1 delineates man-months of forecast corrective maintenance effort, as predicted utilising information relating to forecast development effort presented at each of the twenty development project reviews that were undertaken. The analysis may be described as ‘scale only’ because no consideration is given to changes in the anticipated maintainability of the developed software. It is noted that the actual development outcome varies from the original budget by 0.53%, which is arguably a small margin given the earlier discussion of project performance in Chapter 2.

An upper confidence interval (UCL) can be readily calculated. See Figure 5.2 below.

![Figure 5.2 - Project M1 Maintenance Forecasts; and 95% Confidence Interval](image)

It is noted that the confidence interval stays consistently above the forecast, reflecting the stability of the development review outputs, and therefore means that even the initial UCL tends to overstate the eventual final forecast. The variance, $Q_v$, is influenced by the value assigned to $V$, which appears large for M1. However, across the six projects the variation and arguably therefore the noise is typically greater than in the case of M1. An organisation would tend to select values for both $V$ and $W$ that reflected the average performance, with M1 appearing to a more stable project than some others.

Should the development FCAC prove to be more volatile for a particular project, with a consequent impact on the forecast maintenance costs, then the integrity of the 95% confidence interval as a prudent predictor of maintenance effort is likely to come
under greater threat. An example of this sort is provided by project M5; over the duration of the development phase the FCAC increased by 24%. The predicted standardised effort ratio for this project, recorded in Table 4.14, is almost twice that calculated for M1, indicating that the resultant software is also likely to be more costly to maintain. Interestingly, it may be that the input variables that generated this prediction may also reflect deficiencies in the execution of the development project. A multiplier effect can be seen to pertain, with some of the critical variables that determine maintainability and maintenance outcomes also influencing development scale and costs, which is postulated to affect future maintenance costs. In this example M5 scores relatively poorly for both the quality of the system design specification (SDS) and the domain experience of the development team. The relationship summarised below demonstrates this point; a high (compared to M1) effort ratio is coupled with increasing forecasts of the magnitude of $R_{\text{dev}}$, leading to higher expected values of $R_{\text{maint}}$ than would otherwise be the case. The impact of revised maintainability forecasts on projected maintenance costs is discussed further in 5.3.

$$R_{\text{maint}} = 0.27 \times R_{\text{dev}}.$$  

Equation 5.10 – Project M5 Maintenance / Development Relationship

The maintenance forecast for this project, as updated over the life of the development phase and reflecting purely the scale aspect, is as plotted below.
Note that the 95% upper confidence level (UCL) equates to the final forecast when the development project is approximately 30% complete. In the case of Project M2, which is plotted below, the UCL exceeds the final forecast throughout the development phase.

Figure 5.4 – Project M2 Maintenance Forecasts; and 95% Confidence Interval

In this case it appears that the underlying trend is of a continuing upward drift, but with a relatively low level of volatility. The 95% confidence level continues to provide a prudent assessment of likely maintenance costs in this situation.

Consideration of Project M3 provides a further example of the same effect, in which a continually rising development forecast is reflected into the scale-based maintenance prediction. The plot for M3 is shown below.
A common feature of the last three projects (M5, M2 and M3) to be analysed is that the central forecast increased by in excess of 15% during the course of software development. In the case of M1 the increase was less than 5%.

A further example of a more stable series of forecasts is provided by Project M6.

Figure 5.5 – Project M3 Maintenance Forecasts; and 95% Confidence Interval

Figure 5.6 – Project M6 Maintenance Forecasts; and 95% Confidence Interval
The quality of these resource projections within a particular organisation may be considered more formally by adapting the ideas of Conte et al (1986) that were explored in 2.10. Prediction quality was quantified through the relationship:

\[ \text{PRED}(e) = \frac{k}{n} \]

Equation 5.11 – Prediction Quality

in which \( k \) is the number of projects or tasks from a total of \( n \) whose mean magnitude of relative error does not exceed \( e \). The value of \( e \) can be seen to be the divergence between the forecast and the outcome, expressed as a percentage of the outcome. Within the context of dynamic prediction the quality attribute that is of most interest relates to the probability that a predicted level of maintenance resource at a specified point in the software development phase will not be exceeded by the prediction at the end of the phase. Ideally the objective would be to ensure that the initial estimate is not exceeded but this may result in an unduly pessimistic view, given that a reasonable level of confidence is being sought. As the development phase proceeds and knowledge accumulates about the stability of the development forecast and expected maintainability, the quality of the maintenance cost projections should improve. This can be quantified. An example is provided below in Equation 5.12. In this case the equation states that at the 95% level, the forecast of maintenance costs at the end of the development phase will not exceed the upper confidence level as calculated after the first quarter of the planned software development in 90% of cases.

\[ \text{PRED}_{95\%}(0.25\text{UCL}) = 0.90 \]

Equation 5.12 – Maintenance Prediction Quality

The precise values chosen may vary between organisations and are dependent on factors including the effectiveness of the development estimation process and rigour applied to project managing the development phase. In the case of the organisation from which the six projects under consideration have been drawn, this quality attribute has been considered and is summarised below in Table 5.2.
For this small sample of projects it can be seen that the quality of the maintenance resource predictions is as described by Equation 5.13.

\[
PRED_{95\%(0.25\text{UCL})} = 0.83
\]

**Equation 5.13 – Maintenance Prediction Quality (@ 25% of development)**

This result reflects five out of six predictions made using the upper confidence level after one quarter of the development phase generating an estimate that did not understate the maintenance resource requirement anticipated by the end of the development phase.

The performance of the model after 50% of the development phase was also calculated, and can be seen to produce a better performance.

\[
PRED_{95\%(0.50\text{UCL})} = 1.00
\]

**Equation 5.14 – Maintenance Prediction Quality (@ 50% of development)**

An organisation might conclude that the 83% predictive performance afforded at the 95% level is acceptable, or may decide that a better performance is necessary. Improvement is attainable by enhancing the quality of the initial development estimation processes. In particular, methods adopted to quantify effort and manage risk will tend to reduce both the trend and noise effects observed in the time series. If achieved, the values of W and V in the model can be lowered, which will bring the UCL closer to the steady state values. This will reduce the tendency seen in the
sample for the UCL to be apparently high in relation to one of the more stable projects, while understating the outcome of the more volatile projects.

One shortcoming of the modelling approach outlined above relates to those projects in which linear growth in forecast costs develops. The nature of the SSM that has been deployed does not readily accommodate such a trending attribute. However, an alternative approach is provided by the Dynamic Linear Model (DLM), which is again described by Pole et al (1994). This model differs in that in addition to assuming a current level, it also incorporates a locally constant growth factor. The DLM is more complex than the previous method. This additional complexity is derived primarily from the variance calculation, which must address variances in both the level and growth, and covariances. The DLM has been applied to Project M2, which exhibited approximately linear growth in forecast over the development phase, and a comparison made with the performance of the SSM. This is plotted below in Figure 5.7.

![Figure 5.7 – Project M2 Maintenance Forecasts; DLM vs. SSM](image-url)
Compared to the lagging SSM trend, the DLM remains close to the monthly forecasts and could provide a more useful tool for extrapolating the final forecast. A similar effect can be seen below with Project M5.

![Graph showing Project M5 Maintenance Forecasts: DLM vs. SSM](image)

**Figure 5.8 – Project M5 Maintenance Forecasts; DLM vs. SSM**

The conclusions that can be drawn from these observations are that the usefulness of the method is closely related to model selection, to the quality of the information entered into the model, and also to the ability of the users to interpret the output from the model. This implies an understanding of the underlying principles; the method depends on the amount and volatility of past data. These constraints, however, can be accommodated through well-defined management practices. For example, a minimum number of data points from the time series could be specified, although admittedly limiting the availability of early predictions. Improved risk management disciplines leading to early identification of increased development estimates would obviously also improve performance.

### 5.3 Dynamic Modelling; Development Effort and Maintainability Updates

**Equation 5.2** has been developed to describe the relationship between development and maintenance effort requirements that has been developed, and includes the effect of maintainability as quantified through \( V_{pmo} \), the predicted maintenance outcome variable. From earlier work in **Chapter 3** the hypothesis was established that an
association existed between certain measurable inputs to software development phase and the eventual maintenance outcome. It was considered that these input variables are amongst the major determinants of the eventual maintainability of the software. This paradigm was developed further in Chapter 4 where the notion of scale was also introduced to enable quantitative predictions to be made, and subsequently to provide dynamic predictions in 5.2. These particular dynamic predictions resulted from updated predictions of the scale of the development effort, with the maintainability dimension as represented by \( V_{pmo} \) assumed to be constant.

However, variations over time in the value of \( V_{pmo} \) are also a potential source of volatility in the projection of future maintenance costs during the software development phase. Determinants of \( V_{pmo} \) identified Chapter 3 related to the computer architecture, domain knowledge of the team, quality of the design specification and the proposed location of future software maintenance. Additionally the success of project in meeting the development project was regarded as important, with software reuse and project management skills identified as useful \textit{a priori} predictors. Once the development project is underway and cost forecasts start to be generated then these two surrogates become redundant. Given that the above variables can be quantified and used to calculate \( V_{pmo} \), then any change in the assumptions about the variables may change the value of \( V_{pmo} \) and therefore the predicted maintenance costs.

Considering project M1 and the related architecture assumption can illustrate this. The assumptions at the outset of the project did not include an open/distributed architecture and generated a \( V_{pmo} \) value of \(-0.64\). If this assumption is revised and an open/distributed architecture is incorporated then the value becomes \(-1.31\), indicating the improved maintainability that this architecture should facilitate. The relationship previously described in Equation 5.3 can be modified to accommodate this revised assumption.

\[
R_{\text{maint}} = 0.09 \times R_{\text{dev}}.
\]

\textbf{Equation 5.15 – Revised Project M1 Maintenance / Development Relationship (Open/Distributed Architecture Assumption)}
Given that the multiplier in Equation 5.3 was 0.14, this revision can be seen to have a significant impact, reducing anticipated maintenance costs by 35% if nothing else changed as a result. By way of further illustration, suppose that the decision to move to an open/distributed architecture is accompanied by a recognition that the development engineering team composition needs to be modified. This change involves acquiring the appropriate systems skills but losing 25% of the domain experience possessed by the earlier composition as a result. Anticipated maintenance costs begin to increase again.

\[ R_{\text{maint}} = 0.10 \times R_{\text{dev}}. \]

**Equation 5.16 – Revised Project M1 Maintenance / Development Relationship (Open/Distributed Architecture & Reduced Domain Knowledge Assumptions)**

The multiplier effect identified in 5.2 is particularly interesting in the context of an increasing development forecast cost at completion. The relationship between the scale of the development phase and future maintenance costs has already been explored, and the relationship between containing the software development to budget and future maintainability is reflected in \( V_{\text{pmo}} \). This is illustrated through extension of the situation reflected in **Equation 5.16**. Given the supposed move to an open/distributed architecture but reduced domain knowledge within the development team, another conclusion might be that the development FCAC would no longer be contained within the budget. This has the effect of increasing \( R_{\text{maint}} \) both because of the effect on \( V_{\text{pmo}} \) and because of the growth in \( R_{\text{dev}} \). **Equation 5.17** illustrates the example.

\[ R_{\text{maint}} = 0.17 \times R_{\text{dev}}. \]

**Equation 5.17 – Revised Project M1 Maintenance / Development Relationship (Open/Distributed Architecture, Reduced Domain Knowledge Assumptions & Increased Development FCAC)**

Clearly the application of management processes that reviewed earlier assumptions regarding these variables may be regarded as being as relevant to updating maintenance forecasts as are the development cost reviews. For example, an organisation might implement a technical review process to complement the monthly cost reviews described above. This technical review would consider whether the
initial assumptions pertinent to maintainability remained appropriate. If a deviation from the opening position arose then the approach illustrated above would enable the probable impact on maintenance costs to be assessed. Where necessary, options for corrective action can be explored and the potential impact of alternatives explored.

An illustrative example is provided in Table 5.3.

<table>
<thead>
<tr>
<th>Review</th>
<th>$R_{dev.}$</th>
<th>Maint. factor</th>
<th>$R_{maint.}$ forecast</th>
<th>Project Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>190</td>
<td>0.14</td>
<td>26.60</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>190</td>
<td>0.14</td>
<td>26.60</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>190</td>
<td>0.14</td>
<td>26.60</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>190</td>
<td>0.09</td>
<td>17.10</td>
<td>Design changes to open/dist. architecture</td>
</tr>
<tr>
<td>R5</td>
<td>190</td>
<td>0.10</td>
<td>19.00</td>
<td>Team changes reduce domain knowledge</td>
</tr>
<tr>
<td>R6</td>
<td>190</td>
<td>0.10</td>
<td>19.00</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td>190</td>
<td>0.10</td>
<td>19.00</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>190</td>
<td>0.10</td>
<td>19.00</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>190</td>
<td>0.10</td>
<td>19.00</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td>190</td>
<td>0.10</td>
<td>19.00</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>191</td>
<td>0.17</td>
<td>32.47</td>
<td>$R_{dev.}$ forecast to exceed budget</td>
</tr>
<tr>
<td>R12</td>
<td>191</td>
<td>0.17</td>
<td>32.27</td>
<td></td>
</tr>
<tr>
<td>R13</td>
<td>193</td>
<td>0.17</td>
<td>32.81</td>
<td></td>
</tr>
<tr>
<td>R14</td>
<td>193</td>
<td>0.17</td>
<td>32.81</td>
<td></td>
</tr>
<tr>
<td>R15</td>
<td>193</td>
<td>0.17</td>
<td>32.81</td>
<td></td>
</tr>
<tr>
<td>R16</td>
<td>192</td>
<td>0.17</td>
<td>32.64</td>
<td></td>
</tr>
<tr>
<td>R17</td>
<td>192</td>
<td>0.15</td>
<td>28.80</td>
<td>Increase in planned on-site maintenance</td>
</tr>
<tr>
<td>R18</td>
<td>192</td>
<td>0.15</td>
<td>28.80</td>
<td></td>
</tr>
<tr>
<td>R19</td>
<td>191</td>
<td>0.15</td>
<td>28.65</td>
<td></td>
</tr>
<tr>
<td>R20</td>
<td>191</td>
<td>0.15</td>
<td>28.65</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3 - Project M1 $R_{maint.}$ Updates (Both scale & maintainability revisions)

It should be noted that the events inserted into M1 are hypothetical, and are merely used to demonstrate the principle of a maintenance forecast being updated as the scale and maintainability of the software both vary over time. These effects are shown below in Figure 5.9, with both the level and the UCL calculated using SSM.
The objective of this illustration is to highlight the impact that changes in the predicted maintainability of the software may have on future maintenance costs. Although this will vary to some extent between organisations, and is dependent on the influence of the particular input variable, it is probably insufficient to consider only the scale dimension; maintainability can also be important. One difference between the impacts of scale and maintainability on resource forecasts is the tendency of the latter to insert step changes into the forecast. A useful property of Bayesian methods is the ability to introduce judgmental information into the model, as it becomes apparent. Pole et al (1994) provide a modified version of the SSM capable of accepting additional information. This is achieved by enabling the modeller to insert changes to the level over the course of the time series, and also to vary the values of the noise and drift constants, V and W. The tool might be used in the context of predicting maintenance effort to enable the project manager to investigate what future forecasts might be, given a sudden change to the development project. For example, for Project M1 a sudden change in likely maintenance costs is predicted at Review 11 when it is recognised that the development project is unlikely to be completed within budget. The impact of this event can be modelled and is plotted below in Figure 5.10.
Figure 5.10 – Project M1 Maintenance Forecast, Utilising SSM with Intervention (Scale & Maintainability based)

As may be discerned from Figure 5.10 a step change can be inserted at R11. This enables an early prediction to be made. Confirmation can be sought through a comprehensive reforecast of the likely impact on maintenance costs, given the expectation of the development phase not being completed to budget.

5.4 Conclusions

In Chapter 5 consideration has been given to the opportunities and methods available to update an initial forecast of maintenance costs. This research builds upon the earlier work in Chapter 4 that developed an approach to generating an initial estimate based upon the projected size and the predicted maintainability of the software. A Bayesian approach to updating has been explored, with mechanisms investigated that will enable both regular and random inputs of information to be accommodated.

In particular the use of output from software development project reviews was considered, initially on a ‘scale only’ basis in which a revised size estimate is used to update the maintenance prediction. The application of the Steady State Model was explored as a tool to predict what the next forecast was likely to be, given the recent
past history. It had already been established using the SPSS autocorrelation function that the recent past is the most powerful determinant of future forecasts. Observations noted from the use of SSM were that it was most effective where no growth trend or significant random perturbations pertained. In a sense it is best at making predictions in the steady state, as the name implies. However, given that the immediate prior forecast, perhaps undertaken more formally by the project manager analysing progress on individual work packages to determine the development FCAC already exists, SSM is of only limited value as a predictor of future maintenance costs. SSM may provide more value as a tool to be used by the project manager in conducting the work package reviews.

The use of the UCL as a prudent predictor of maintenance costs was considered, but in the steady state condition it may be regarded as overly pessimistic. One possible option might be to adopt the more demanding 50% level, rather than 95% as a predictor. A difficulty perhaps lies in recognising that the steady state pertains; some decision rules might be established, defining a number of reviews without appreciable change in forecast that might be construed as representing the steady state.

The converse of this situation, that of linear growth, normally positive and representing an increasing magnitude would also be defined by default. Leaving aside perturbations, anywhere that the steady state does not apply could be regarded as linear growth. In this circumstance the Dynamic Linear Model may usefully be applied. Experience with projects M2 and M5 suggest that tools could readily be developed using the DLM to give valuable projections of future maintenance cost forecasts, providing some advance warning of cost and resource commitments. In the linear growth situation the UCL is also more useful as a prudent predictor of the eventual forecast.

The development of the concepts described by Conte et al (1986) provides a basis for the deployment of an organisation-wide monitor of prediction quality in the dynamic situation. Local calibration would probably be important, defining the point in the development phase and the confidence level that pertain.

In the 'scale & maintainability' maintenance prediction situation, the 'SSM with intervention' method has particular merit as a simple mechanism to enable hypothetical developments to be explored. This perhaps has less relevance to
calculating the immediate effect on maintenance costs, which can be directly calculated through Equation 5.2 by modifying either or both of $V_{pno}$ and $R_{dev}$, but rather to assist in the interpretation of the time series as it unfolds subsequent to a step change.

In summary, of the methods explored the DLM has utility in the situation where a linear growth trend in forecasts is apparent, and the UCL is most useful here as a prudent predictor of the eventual forecast. ‘SSM with intervention’ may be useful in exploring the impact of sudden changes on future predictions. Although these methods have been considered primarily in the context of maintenance cost prediction, they also have relevance to software development cost forecasting.

All of the methods are conceptually straightforward and computationally simple to apply, with little more than spreadsheet tools required. Careful thought needs to be given to their optimum use within any given organisation. They might, as noted above, be applied initially to the management of the software development process. As explored and discussed in Chapter 5 the tools have been considered as a rapid means of obtaining predictions, or perhaps answering 'what if' questions, with existing project management processes within the organisation subsequently providing a more rigorous analysis. An alternative might be to consider embedding these tools within the management processes, perhaps generating more responsive management routines as a consequence.
6.1 Introduction

In Chapter 6 the results of the research described in the preceding chapters are summarised. The implications of this work for the management of software projects are considered and recommendations for the application of these results are made. Recommendations are also made for further research.

6.2 Summary of Findings

- **Software Project Management**: Considerable evidence has been identified to support the contention that successful project management in the areas of software development and maintenance is difficult. In particular, beyond effort estimation, the identification and containment of risk is critical. Many instantiations of a general approach to the management of risk exist, involving assessments of likelihood of occurrence and severity of impact. An example relating to practice in the US Navy is provided by Charette et al (1997). The efficacy of the method is limited to the degree that risks and their consequences can be foreseen. The objective, however, should extend beyond attempting to identify future eventualities to consider the likely success of mitigation strategies that might be implemented.

- **Software Maintenance Scale**: The characteristics of projects involving the maintenance of software have been explored. This area is important because of the scale of the activity, which has been quantified in several different ways. Attaining an understanding of the expected magnitude of future software maintenance costs, and the underlying determinants, is important to the commercial viability and operational performance of a program, given the resources that may be required. Most researchers agree with Hops & Sherif (1995) that maintenance absorbs the majority of software expenditure, and also with the view expressed by Pressman (1997) that this proportion may be growing because of the increasing volumes of ageing software and the need to adapt it to changing requirements. This trend is likely to continue. Reasons for the uncertainty associated with maintenance costs include the limited understanding of the processes and variables, interacting in what may be regarded as
an unfashionable area of software endeavour, and also because of the variety of definitions of software maintenance that still pertain.

- **Software Maintenance Attributes:** Several important themes can be distilled from the various software maintenance definitions. Most fundamentally, software maintenance can be regarded as a process or series of processes, and is therefore susceptible to description and measurement. Beyond that, it is an activity that takes place after initial release or implementation, and can have several objectives including the correction of faults, performance improvement or adaptation to changed circumstances. Additionally maintenance is recognised to possess both technical and managerial dimensions. Finally, as Boehm (1981) discerns, it can be distinguished from software development in that the primary objectives of the software remain unchanged. Other distinctions between development and maintenance may be observed. These include the degree of influence that users or environmental changes may have on the maintainer compared to the developer, and the obligation on the maintainer to interpret the original design intent of the developer. Additionally the operational and perhaps time-critical nature of maintenance that may prevail.

- **Software Maintainability:** Closely associated with maintenance is the concept of maintainability, which relates to the ease with which maintenance can be undertaken. Software maintenance effort estimation and risk management may both be regarded as being dependent to a degree on the maintainability of the software. Maintenance, as both Lehman (1980) and Glass (1998) have observed, is both inevitable and desirable if the software is to remain useful. Given this inevitability, the ability to quantify and improve maintainability may be a source of commercial advantage.

- **Software Maintenance Estimation:** Together with maintainability assessment, software maintenance management requires an effective effort estimation methodology. The majority of research undertaken in this field relates to estimating development effort, but many of the principles also apply to maintenance estimation. Considerable evidence exists regarding the generally disappointing performance of software estimation, and it may be that maintenance estimation is particularly problematical, given the poorer understanding of the processes that arguably pertains and the diverse range of activities it encompasses. Henry et al (1996) support this latter point. Four discrete but complementary approaches to estimation can be identified. Three of these, expert opinion, analogy and decomposition possess a
subjective element; Chatfield (1997) assigns them to a judgmental category. The fourth approach, mathematical modelling, is a more formal and structured technique, and provides an explicit relationship between input variables and the output.

- **Estimation Models:** These models are applied to the software processes at various levels of abstraction ranging from high level estimation of a complete project lifecycle through to assessing the maintainability of a particular software module. Many researchers regard the performance of these models as disappointing. Several explanations can be proposed including model complexity, inconsistent definition of the variables, insufficiently rigorous capture of process data, inadequate calibration to the local environment and superficial user training. Chatfield (1997), who offers evidence that simpler models are more robust than more complex alternatives and Somerville (2001), who addresses the issue of calibration, support this view. Regression methods have commonly been used to develop models. These typically relate resource requirements to program size, possess an exponential component designed to accommodate non-linearities that are believed to exist in the relationship between software costs and program size, and contain correction coefficients intended to minimise residual errors arising from the regression.

- **Model Performance:** It is apparent that alongside the model design and implementation, a mechanism is required to assess model performance by comparison of estimates with outcomes. This requirement is recognised by Conte et al (1986) who proposes a simple but potentially effective method. This effectiveness will be enhanced if variances are investigated and special causes taken into account.

- **Maintenance Model Application:** Software maintenance estimation methods, which may be regarded as a sub-set of the broader software estimation field, have been applied in ways consistent with the varying degrees of abstraction described above. These range from the identification of error prone modules in existing code through to quantifying the costs of a major change. Only a comparatively small proportion of the available research relates to maintenance estimation in the context of future projects. Models have been developed that consider the software product attributes, software engineering processes or a combination of both. The organisational and social aspects of software maintenance suggest that the process dimension should not be ignored, and that therefore both product and process variables are essential to the

- **Predicting Future Maintenance Costs:** An ability to estimate future maintenance costs is relevant to assessing the viability of a whole project, given the earlier discussion of the growing contribution of maintenance to total software costs. Quantification of likely future maintenance effort is therefore an operational concern for most software organisations, to enable planning of resource requirements appropriate to both development and maintenance. For a commercial organisation the issues extend further, with the possible implications for profitability. The customer service dimension is also likely to be an imperative.

- **Maintenance Cost Determinants:** Maintainability has already been identified as a dimension that is likely to influence maintenance costs. Two other likely determinants are the scale of the proposed software entity and the amount of future change that can be anticipated. Summarising work undertaken in the earlier chapters:

  \[
  \text{Future software maintenance costs are likely to be largely determined by the size of the program, the degree of change that will be made to the program and the ease with which these changes can be effected.}
  \]

This summary is supported by the work of Boehm (1981), which utilises an estimate of development effort with an annual change traffic multiplier to estimate maintenance effort. Size and level of change are considered, with ease of modification and reliability requirements reflected in correction factors that may be regarded as a surrogate for a more explicit maintainability factor. Granja-Alvarez & Barranco-Garcia (1997) have addressed this latter aspect. Having identified size, degree of change and maintainability as determinants of maintenance expenditure, the challenge is to identify those factors that influence these determinants, and quantify underlying variables for incorporation into a predictive model.

- **Maintainability Survey:** Although previous research provides evidence regarding the most important input variables for inclusion in a maintainability assessment model, a further study based on new information was regarded as essential. This conclusion reflects differing opinions over experimental and analytical methods.
Lederer & Prasad (1992), Stark & Oman (1995) and Somerville (1995) reflect this in criticism. A questionnaire-based survey was undertaken to gather data for subsequent factor analysis and regression modelling. A continuing theme during this exercise was the difficulty involved in obtaining the requested information. Several organisational and attitudinal observations arise from this experience. Despite requiring a combination of quantitative responses and qualitative judgements of a relatively basic nature, it was apparent that few organisations had implemented the management systems that would have captured the information routinely. In larger organisations information was usually provided by staff specialists, perhaps software process improvement 'champions', with a personal interest in the field. Smaller organisations tended to possess people who understood the value of the information, but had not allocated the resources needed to gather it. In neither case did the infrastructure or understanding generally appear sufficiently well developed to inculcate continuous improvement. The survey can perhaps be criticised because it inevitably contains information only from those organisations that had the data available or sufficient interest to gather it.

- **Maintainability Modelling**: The output from the survey consists of 69 numerical responses relating to input variables and outcomes for 34 projects. A data reduction exercise was undertaken using factor analysis. This resulted in sixteen factors being generated, which were then subjected to linear regression analysis. One of the factors, which reflected the maintenance outcomes, was treated as the dependent variable with the other fifteen considered as the independent variables. The adjusted Coefficient of Determination ($R^2_{adj}$) ratio was used to interpret the strength of the relationship between the maintenance outcomes and the independent variables. A maximum value of 40% was obtained using a combination of four factors that related to development performance, project characteristics, the design process and code complexity. These four factors were subsequently used to initiate the identification of the underlying input variables. It had been concluded that identifying these individual variables was desirable for distinct two reasons, one statistical and the other operational. The former relates to the elimination of error embedded in the factors and a consequent improvement in model performance. The latter reflects a belief that the most useful model is one in which the variables are both easy to obtain and understandable. Factors do not satisfy this latter criterion.
This further regression analysis resulted in a linear model containing five variables and an improved $R^2_{adj}$ performance of approximately 65%. This model is regarded as a predictor of maintainability. The variables relate to the computer architecture, domain experience of the development team, quality of the design specification, planned location of the subsequent maintenance and anticipated success in meeting the development budget. The content of the model is consistent with observations made by Haworth et al (1992), Alkhatib (1992) and Somerville (1995). The conclusion that is implicit in the structure of the model was summarised as:

_Ease of maintenance is likely (65%) to depend on the development project having started with a good design specification and been completed within budget on an open/distributed architecture by engineers with high domain knowledge, with the subsequent maintenance performed at the site of the operational system._

One limitation of the model is a dependence on the success of the development phase of the overall project in meeting budget, which is inconsistent with the objective of estimating development effort prior to the start of development. This problem was addressed through a further regression analysis designed to identify those input variables likely to be determinants of an on-budget completion to software development. Two important variables that were identified related to the experience of the project manager and the degree of software reuse. Of these the latter was found to be the more powerful. Together they were included in a predictive model with an $R^2_{adj}$ performance of 34%. In the case of dynamic prediction, progressive cost reviews during the development phase remove this limitation.

Two variables that do not feature in the predictive model but might have been expected are those that relate to code complexity and program age. In the former case the organisational norms regarding complexity are appear to weaken its influence over the maintenance outcome. However, the code complexity factor that is incorporated into the initial model derived from the factor analysis does still exert some influence over the performance of the variable-based maintainability predictor. This is achieved through one of the code complexity variables, software reuse, which features in the on-budget prediction described above. In the latter case, where more recent programs might be expected to be the more maintainable and therefore influence the predicted maintenance outcome, the absence of an age variable may be
explained by the limited age range within the sample and possibly by the architecture acting as a surrogate.

- **Model Validation**: Validation of the conclusions implied by the model structure was pursued through two separate strategies, described respectively as extrinsic and intrinsic. The former consisted of a series of interviews with contributors to the survey, resulted in an emphasis on the managerial and human aspects to a greater extent than the technical, and broadly agreed with the content of the model. Additionally, their comments provided some support for a causal relationship between the input variables and the maintenance outcome. One limitation of these interviews lies in the subjects being drawn from the population of contributors, who could reasonably be expected to provide comments that align with the questionnaire content. A follow-up exercise with a previously uninvolved sample may provide some additional insights. However, the views of these contributors do align with the observations of Van Genuchten (1991) and Potts (1993). The latter strategy involved the use of hold-out samples to test the genericity of the model by comparing predictions made by two different sub-sets of the total sample. It was concluded that no significant difference existed between these samples and that the model may be suited to more generalised application.

- **Further Model Validation**: Structural equation modelling (SEM) was used to explore inter-dependencies between the input variables. Given that the model development process began with a factor analysis that should have removed many of the possible collinearities, the expectation was that a SEM analysis might only produce a marginal improvement. This was confirmed with a small improvement arising from the application of a more complicated modelling method. Given the advice of Chatfield (1997) favouring simplicity as tending to facilitate robustness, this marginal benefit was discounted. The five-variable maintainability model described earlier was adopted as a basis for further predictive work.

- **Static Prediction**: Two forms of static prediction were primarily explored: categorical and continuous. Some further work was also undertaken relating to estimation arising from discretionary maintenance.

- **Categorical Prediction**: The analysis involved the use of logistic regression to assign predictions to either a 'good' or 'bad' category, based upon some predetermined
acceptance criterion. This is represented in a defined numerical value for the maintenance outcome. Various models were explored, all based on the five-variable maintainability predictor discussed above, and a hold-out sample test was used to evaluate performance. This test concluded that a high level of agreement between prediction and outcome was achievable and that a useful management tool might therefore be developed. Several requirements and limitations were also identified. A rigorous and consistent means of assigning values to the model variables is required. The robustness of the method to changes in the acceptance criterion is also an issue, but based on the sample data an acceptable performance was obtained. Scale was explored and it was concluded that below a certain size the performance of the model would deteriorate, because other factors begin to exert an influence. The sensitivity of the model to changes in the values of the input variables was investigated, with the architecture and success in meeting the development budget proving to be the most powerful determinants of the predicted maintenance outcome.

- **Continuous Prediction:** In this case the objective is to generate an estimate of likely maintenance costs. This primarily relates in the first instance to corrective maintenance, where only limited discretion exists as to whether or when to undertake the work. Numerical data relating to corrective maintenance from a survey of a further six projects within a single organisation was introduced. A comparison between the two surveys confirmed the software to have similar attributes, although the programs in the second survey tended to be larger. The relationships that were explored during this particular analysis were those between maintenance cost and development scale, and between maintenance cost and software maintainability. The size of the program and the ease with which changes can be made have already been identified as likely determinants of maintenance costs. Boehm (1981) and Granja-Alvarez & Barranco-Garcia (1997) both support this hypothesis. Analysis of the sample data for the six projects confirmed the existence of a relatively linear relationship between the scale of development effort and corrective maintenance effort. The maintainability aspect was explored by consideration of the predicted maintenance outcome variable ($V_{pmo}$), which is a predictor of maintainability. A high level of agreement was obtained between these predictions and the maintenance outcome factor (MO1) for the second survey. Given the relationship between development and corrective maintenance effort, a standardised effort statistic ($R_{mv/d}$) was defined to take account of the scale aspect. The relationship between $R_{mv/d}$ and $V_{pmo}$ was explored and modelled. This model allowed the calculation of a predicted
standardised effort statistic ($\hat{R}_{m/d}$) for each of the six projects. Close agreement between $\hat{R}_{m/d}$ and $R_{m/d}$ was achieved. Recognition that a prediction is likely to lie within certain limits was reflected in the application a confidence interval to these results.

Given an ability to quantify maintainability through $V_{p\text{mo}}$ and the existence of an estimate of development effort, the generation of a prediction of corrective maintenance effort is possible. The method builds on the approach previously taken to categorical prediction, and again requires a structured method for the assignment of values to the input variables associated with the calculation of $V_{p\text{mo}}$. The calibration of this model to the characteristics of a particular organisation may serve to improve predictive performance. It is concluded that this method has the potential to be developed into a useful management tool.

- **Discretionary Maintenance:** The application of continuous prediction to non-corrective maintenance was explored. This situation requires the introduction of the third dimension to maintenance estimation, relating to the degree of change that can be anticipated, which was identified above together with scale and maintainability as the major determinants of maintenance cost. The applicability and limitations of the annual change traffic measure described by Boehm (1981) was explored. Concerns over the use of degree of code change as a surrogate were discussed, and it was concluded that within the norms of a particular organisation it is valid. Outwith these norms a correction factor might be required. A means of applying this methodology within an organisation was discussed.

- **Dynamic Prediction:** This form of prediction takes account of new information, as it becomes available during the software development phase. Data arising from periodic reviews of software development effort requirements associated with the six projects considered above was the source information.

Initially only variations in forecast development effort over time were used in conjunction with the predicted standardised effort ratio, $\hat{R}_{m/d}$, to generate revised corrective maintenance forecasts. This was undertaken for each of the six projects, and confidence intervals applied. The methods advocated by Conte et al (1986) to measure the quality of predictions were developed to quantify the success of applying the upper confidence level during the development phase as a prudent predictor of
maintenance effort. This was found to be a useful approach, although the acceptance criterion associated with predictive performance requires to be established within the context of the particular organisation.

Dynamic prediction was subsequently enhanced by considering updates to both the software development effort forecast and the predicted maintainability of the software. A ‘storyboard’ approach was adopted, utilising an illustrative example, in the absence of any factual data relating to changes in predicted maintainability. The impact of this determinant on the forecast was noted and it was concluded that it may be insufficient only to consider changes to the development effort forecast. Limitations to the deployment of maintainability updates are primarily organisational. Beyond having processes established to assign values to the input variables associated with $V_{pmo}$, management processes are required to provide revised values where appropriate.

- **Summary:** Key points that have been noted include:
  - Software project management is difficult. Effort estimation and risk identification are important aspects.
  - Software maintenance is a major area of software endeavour.
  - Software maintenance possesses features that distinguish it from software development. Managerial, organisational and human aspects are important.
  - Software maintainability is a determinant of maintenance effort. Quantifying maintainability is therefore important.
  - Software maintenance effort estimation is problematical. Mathematical modelling is the most structured approach. Regression is a recognised method of model development.
  - Quantification and review of model performance is important.
  - Comparatively little research has been devoted to estimating maintenance costs prior to development of the software.
- In addition to maintainability, other determinants of maintenance costs are the overall software size and the degree of change that will pertain.

- Five variables were identified as the most powerful predictors of maintainability. A maintainability prediction model was developed, which can be used to generate a maintainability predictor parameter, $V_{pmo}$.

- A categorical predictive tool was developed to classify projects as potentially 'good' or 'bad' early in the development lifecycle, enabling remedial action.

- A continuous predictive tool was developed to quantify maintenance costs. This tool utilises the relatively linear scale relationship between development and maintenance effort requirements that was identified, and provides a ratio, $\hat{R}_{md}$.

- Application of the continuous predictive tool to non-corrective maintenance requires a metric to quantify the level of change. Annual change traffic was identified as a suitable metric, within a particular organisation.

- Dynamic prediction was explored to consider updates to maintenance predictions, as information becomes available during software development. The Dynamic Linear Model and 'Steady State Model with Intervention' were most useful.

- Each of the tools should be calibrated to reflect the norms of an organisation.

- Business benefits will accrue from the deployment of these tools.

6.3 Maintainability Definition

Arising from the research is an improved understanding of software maintainability, and the related determinants of this important attribute. A consequence of this work is the opportunity to provide a revised maintainability definition. Objectives in creating this definition include simplicity, brevity and completeness; to this end it is believed that the definition should accommodate product, process and personnel characteristics.
These requirements have been incorporated within:

*The ease with which software can be accessed for the purposes of correction, perfection, defect prevention or adaptation, recognising product attributes and organisational (process and personnel) capabilities.*

A three-dimensional space can be conceived to describe the maintainability envelope. This is illustrated below in Figure 6.1.

![Figure 6.1 - Maintainability Envelope](image)

The model recognises three dimensions of the maintainability paradigm, two of which represent the organisational aspects of maintainability. This element of the model utilises the work of Wickens (1998). Progression along either of the axes represents increasing capability, with objective being to achieve ascendancy, which requires developed processes and trained, motivated personnel. Progression along the vertical axis relates to the increasing accessibility of the product. The lower level represents the extreme of an abandoned program, for which no organisational memory in the form of either people or documentation is available to support maintenance.
Within a particular organisation it is possible that objective performance metrics could be devised, relating to all three axes and designed to promote the maintainability of future software deliverables.

For the process dimension it is likely that the business processes under consideration are generic, rather than relating to a particular software project. An exception to this generalisation may be where a large project is synonymous with the enterprise, but even here it is probable that the processes would be drawn from established practice elsewhere. The objective of process measurement is to ensure that acceptable outcomes can be achieved with a high degree of repeatability; some concepts embedded within CMM are applicable. Measurement can be undertaken utilising established statistical process control (SPC) techniques, and enable calculation of a process capability index, $C_{pk}$. The general application of these techniques is described by Owen (1989), and is a development of the ideas of Middleton (1995) regarding the deployment of established management methods within the software arena. A general description of the application of these methods to software engineering processes has been provided by Pressman (1997), and specifically with regard to control chart applications is provided by Haworth (1996). For an agreed set of processes an organisation would have to specify the minimum acceptable $C_{pk}$ value, probably relating to the boundary between the 'apathetic' and 'alienated' zones along the process axis of the organisational plane. Acceptance levels would initially have to be assigned subjectively, based on prior experience of earlier projects. Given achievement of a minimum acceptance $C_{pk}$ threshold for each of the nominated processes, it may then be appropriate to calculate an average $C_{pk}$ to characterise the process performance of the organisation and, by implication, the project. An alternative to the average $C_{pk}$ might involve assigning weightings to the processes, but given the sequential nature of the processes with software development and maintenance, this may be inappropriate. More acceptable may be the calculation of a 'rolled $C_{pk}$', that quantifies the capability of the aggregate software process. The approach taken may vary between organisations, but should enable the process capability to be described numerically. Where the $C_{pk}$ falls below an acceptable level, corrective action will be required to ensure that a project is to be delivered within required area of the maintainability envelope.
Quantifying attributes associated with the personnel axis may be applicable at either an organisational or project level, or indeed both. The discussion of the 'team experience' quantification in 4.2 can be developed into a scheme design with the goal of acquiring an objective view of personnel capability. A variety of possible approaches exist but one possible method is outlined below:

- The scheme applies within a particular organisation. A key input is the regular personal performance appraisal, which is applied at either the 'engineer' or 'manager' level.

- At either level several key attributes are assessed with strengths and improvement opportunities identified, and a score assigned by the appraiser.

- For the 'engineer' these attributes might be:
  - Job Knowledge - design methods, languages...
  - Dependability - meeting specifications, achieving deadlines...
  - Communication - external, documentation...
  - Customer Focus - responsiveness, personal appearance...
  - Change Flexibility - adaptability, time management...
  - Personal Qualities - team working, attendance...

- An aggregate score can be calculated for each individual.

- These scores would then be factored to reflect the experiential dimension:
  - basic $\times 1$
  - graduate $\times 2$
  - professional $\times 3$

- A team score can be summed for a given project.

- An average team member score (ATMS) can then be calculated, by dividing the aggregate by the team size.

The ATMS value can then be plotted on the personnel axis, again with a defined acceptance threshold wherever the organisation defines the boundary to lie.
Given acceptable scores on both the process and personnel axes, the project would be positioned within the desirable ascendant zone of the organisational plane.

Positioning of a future project on the product axis requires the introduction of a complexity or functionality metric. Compared with the previous discussion of the development of process and personnel metrics, this is a well-developed area, notwithstanding the disagreement described in Chapter 2 over which metrics are the most meaningful. Fenton & Pfleeger (1997) discuss the merits of several alternative complexity measures and also introduce function points as a functionality metric. Despite the controversy over whether any single metric provides a comprehensive description, it may be that a sufficient understanding of the product dimension can be achieved through a single value to enable positioning within the maintainability envelope. Khoshgoftaar & Munson (1990) provide an example of such a metric. To an extent the selection of a metric may be a matter of organisational preference, perhaps with metrics already in place.

This model can be applied as an improvement tool, with progress along the axes likely to result in an improving $V_{pmo}$ prediction for successive projects.

6.4 Novelty of Approach

There are several aspects of the research that possess a degree of novelty. These are noted below.

- The scale and content of the survey is large. Many previous surveys are more oriented towards software development. Where software maintenance is the theme, most surveys tend to be confined to a single organisation. Additionally, the application of factor analysis to the survey results seems to be uncommon.

- The development of a five variable linear regression model to predict maintainability is unusual. Previous approaches have included more complex non-linear methods or matrix methods.

- Recognition that software development performance is an explicit and powerful determinant of maintainability is not apparent in much of the earlier research.
• Use of Structural Equation Modelling to explore the performance of the maintainability prediction model appears to be a new approach.

• Application of the Logistic Model as a non-linear predictor of standardised maintenance effort, with the maintainability predictor embedded as an independent variable, is not evident in previous work.

• Dynamic updates to maintenance forecasts, utilising input from the software development phase, is not an area that has been considered previously to a great extent. The use of the Steady State Model and Dynamic Linear Model both seem to be novel applications of the tools.

• Formalised approaches to assessing the quality of estimates have been explored. The application of this approach, utilising confidence intervals, to dynamic prediction is new.

• Provision of a revised maintainability definition has been undertaken, together with an illustrative model.

6.5 Business Implications

Several implications relating to both the business environment and organisation arise from the research.

• Application of the continuous prediction technique discussed as part of Static Prediction requires the organisation to deploy an effective software development estimation process. This is essential, given that the maintenance estimate is expressed in standardised form, as a proportion of the development effort.

• Categorical prediction, again discussed within Static Prediction, has the potential to be applied as a project management tool, ensuring that that an acceptable level of predicted maintainability pertains from early in the project life. Acceptance criteria need to be specified for this to operate.
• For Dynamic Prediction to be implemented satisfactorily, project management structures need to be established to provide frequent updates that can be fed into the maintenance estimation model.

• Calibration of models is likely to improve the predictive performance of the models. This aspect is considered further in 6.5.

• Quantification of maintenance costs can be regarded as a source of competitive advantage in a commercial market. The business that understands the costs associated with supporting the product is better able to quote competitively, or alternatively avoid entering into profitless contracts. In the non-commercial environment frequently associated with certain government contracts, pricing is determined using formulae that require cost estimates as inputs. In both these circumstances the models that have been developed can provide an advantage.

6.6 Recommendations

• **Deployment**: To develop the findings discussed in 6.2 as useful operational tools requires the implementation of management processes and the calibration of models. These requirements relate to the application of established project management structures and the calibration of maintainability, categorical and continuous effort predictive models.

• **Effective Project Management**: The related processes are important to both software development and maintenance phases. These processes have both organisational and methodological attributes. Organisational aspects include clear authority and accountability protocols; this is particularly critical in an environment that possesses matrix characteristics, in which both project and functional reporting lines pertain. Defined project review mechanisms are also needed, reflecting specified update methods, information inputs, and frequencies. Methodological issues reflect the generic software project lifecycle. Requirements capture and specification should be established and codified. Similar imperatives apply to the creation of design and test specifications. Work breakdown structures and risk management mechanisms are important. Detailed estimation and resource planning follow. It is likely that the complexity of these arrangements will increase with organisational size. Training of
users in these principles should complement technical education. A culture that supports effective project management practice may be regarded as a necessary pre-condition for the implementation of maintenance estimation processes. Given this, model calibration and application can proceed.

- **Quantification of Software Maintainability**: This has been identified as essential to maintenance estimation, and the maintainability predictor \( V_{pmo} \) has been developed in response to this need. This model can be calibrated to reflect organisational performance. This means that five input variables, with two further inputs to predict the development budget aspect, require to be quantified through a structured and consistent scoring regime. Alternative approaches to developing such a regime were explored in 4.2. These input variables should then be scored for a population of past projects from within the organisation; 20 projects are a desirable population. The inputs should be related to the dependent variable, the maintenance outcome, which again requires to be quantified through a similarly structured mechanism. Maintenance frequency and difficulty are the dimensions identified to score this outcome. Linear regression can then be applied to assign parameter values to the variables within the model. The resultant model should then be validated against a hold-out sample of a further five projects.

- **Categorical Prediction**: This can be performed using the initial project conditions that pertain. Logistic regression can be applied, using the maintainability predictor derived above. Issues that must be considered include assignment of a cut value to differentiate acceptable and unacceptable outcomes, and confirmation of a size range through which the model performance is valid. Model performance should confirmed through a hold-out sample. Figure 4.4 summarises this process. Beyond categorical development is the requirement to train users, and subsequently to monitor results to facilitate model refinement and re-calibration.

- **Continuous Prediction**: This method can also be utilised in conjunction with the initial project conditions. In this case the objective is to generate quantitative estimates of corrective maintenance effort. An initial step should be to confirm the availability, provenance and quality of estimates of software development effort. Quality can be assessed through the Conte et al (1986) method described in 2.10, with a minimum acceptable level identified; a \( \text{PRED}(0.25) \) of 0.75, meaning that 75% of estimates are within 25% of the outcome was regarded as acceptable by these
researchers. If, however, this level of performance is being sought for the eventual maintenance estimate then a better performance will be required from this input. Given the availability of adequate development estimates for future projects, which is an essential pre-requisite, then calibration of the maintenance estimation model can proceed. The first step in this process is to confirm the linearity of the scale relationship between development and maintenance effort requirements; evaluation of results from past projects within the organisation will facilitate this confirmation. This enables calculation of the standardised effort statistic \( \frac{R_m}{d} \). A relationship can then be modelled between \( \frac{R_m}{d} \) and \( V_{pmo} \) utilising the logistic model. The predicted standardised effort statistic \( \hat{\frac{R_m}{d}} \) can then be calculated for a hold-out sample of a further five projects to validate model performance. Figure 4.9 outlines this process.

User training, refinement and re-calibration may be regarded as continuing requirements. Given the above, the organisation has access to a management tool that can be used to estimate probable corrective maintenance costs very early in the life of a project. Decisions can then be made regarding the acceptability of these costs, and the effect of changes to the input variables can be modelled, if required.

**Discretionary Maintenance:** The area of maintenance estimation arising from discretionary maintenance requires further analysis. This is discussed in 6.2.

**Dynamic Prediction:** This may be regarded as an extension to continuous prediction. Having undertaken the calibration discussed above, issues relating to the management processes require attention. Development effort and maintainability updates were identified as being essential. Reviewing the software development effort forecasts to completion requires a disciplined approach involving analysis of the progress to date and the effort expended on work packages within the work breakdown structure. A risk analysis should be available and also subject to review, with any changes reflected into the development effort forecast. Mechanisms are also required to review the values assigned to those input variables associated with the maintainability predictor. A regular monthly review and confirmation using the same methods described to assign the original values will help promote consistency. Where a change occurs to one of these inputs a revised value for \( V_{pmo} \) should be calculated, and through \( \hat{\frac{R_m}{d}} \) an updated estimate of maintenance effort can be determined.
6.7 Recommendations for Further Research

- **Overview:** An important thrust of further research should relate to converting some general observations and conclusions into a suite of management tools suitable for deployment within a software organisation. Differences between organisations that could usefully be explored relate to the sizes of programs being considered, the application areas, and the size and culture of any given organisation. For brevity, much of the discussion below is written in the context of a single organisation.

- **Maintainability Modelling:** The major determinants of software maintenance cost have been identified as being the scale of the software entity, the amount of maintenance that is required and the ease with which this maintenance can be undertaken. The scale of the software project may be dependent on the application area or the requirements of the user, and the degree of maintenance is to an extent at the discretion of the user. The ease of maintenance is related to the maintainability of a program and is of critical importance because it has a major effect on maintenance costs, it is heavily influenced by decisions taken before or during the development phase and may be expensive to change afterwards. Quantification of maintainability has been effected through the maintainability predictor, $V_{pmo}$, which itself depends on the ability of an organisation to meaningfully quantify several input variables. These input variables were identified through factor analysis and linear regression methods that were applied to a population of projects drawn from several organisations. Several areas requiring further research can be identified.

  The general applicability of the input variables within a single organisation should be confirmed by undertaking a further factor analysis and linear regression on a population of past projects.

This analysis would serve to confirm the validation undertaken in **Chapter 3**, in which results drawn from a single organisation within the survey population were compared to a random sample, with no significant difference identifiable. Of particular interest may be the effects of code complexity and particularly program age. It may be that within one organisation over a longer time span, during which methods and technologies have changed, that this variable exerts greater influence.
• **Input Variable Assessment:** Validation of the input variables should complemented by the development of adequate scoring systems within a target organisation, and exploration of the sensitivity of the maintainability value to variations in input values.

Robustness of model design within the target organisation should be explored through a sensitivity analysis of the input variables. Management systems designed to score these inputs should take account of the comparative sensitivities of the inputs.

• **Model Validation:** The outcome of these analyses should be supported by further confirmatory interviews.

The single organisation maintainability predictor model design should be validated by interviews with practitioners within the target organisation. Interviewees should preferably not have been involved in completing questionnaires. Emphasis should be placed upon identifying causal relationships between the putative inputs and maintenance outcomes.

• **Scale Relationships:** Subsequent to considering maintainability prediction, further attention should be applied to maintenance estimation. The assumed relationship between development and maintenance scale could be explored further, and particularly might be characterised for a single organisation. Given this relationship, confirmation of the genericity of the effort estimation model should be obtained.

*Development and corrective maintenance scale relationships should be explored within a target organisation. The association between standardised effort and maintainability should be confirmed and the suitability of the logistic model as an adequate descriptor of the relationship verified. The sensitivity of \( \hat{R}_{mid} \) to variations in \( V_{pmo} \) should be analysed for the organisation, and consideration given to the implications for both the designs of the maintainability predictor and standardised effort statistic predictor models.*

• **Discretionary Maintenance:** The area of non-corrective maintenance has been identified as one where an outline approach requires further research. Of particular
interest is the use of the annual change traffic metric to quantify anticipated levels of non-corrective maintenance.

For a population of past projects within a given organisation a comparison should be undertaken between forecast numbers of code changes, actual changes and maintenance costs arising from the work. The objective should be to model the relationship between change traffic and maintenance costs, given a prior understanding of the scale and maintainability determinants. The issue of sensitivity of predicted outcome to the quality of the input, as represented by the forecast of annual change traffic, should be explored.

• **Sensitivity Analysis:** Beyond this, the overall confidence that can be assigned to the predictions should be explored, given an understanding of the sensitivities associated with each phase of the analysis.

    An overall sensitivity analysis should be undertaken for the predictive system that has been developed. For a given organisation this should take account of variations in maintainability predictor input variables, accuracy of development estimates and, where appropriate, annual change traffic.

• **Conclusion:** The outputs from the above could form part of a suite of soft tools, possibly utilising spreadsheets, that an organisation would apply to develop a maintenance estimation system. This requires further work.
<table>
<thead>
<tr>
<th>FACTOR</th>
<th>NOTES</th>
<th>SCORE</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. System attributes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1 Code complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.1 Size</td>
<td>enter total system KDSI into score box; see note 2.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.2 Modularity</td>
<td>enter no. of CSCIs into score box; see note 3. for a definition of a CSCI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.3 Development</td>
<td>enter no. of CSCIs requiring new development into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.4 Re-use</td>
<td>enter no. of CSCIs re-used without development into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Third-party</td>
<td>enter no. of CSCIs consisting of third-party software into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2 Environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.1 Language</td>
<td>enter principal language into comments; if more than one then add description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.2 Hardware</td>
<td>specify principal manufacturer, approx. CPU and disk capacities in comments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.3 Resources</td>
<td>specify approx. % of available memory used in score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.4 Architecture</td>
<td>specify configuration (eg. open vs. proprietary; distributed vs. centralised) in comments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3 Design Process</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.1 life-cycle</td>
<td>specify any formal model used in comments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.2 quality of SRS</td>
<td>see note 4.; should reflect degree to which the SRS (System Requirements Spec.) responds to the customer requirement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.3 quality of SDS</td>
<td>see note 4.; should reflect degree to which the SDS (System Design Spec.) responds to the SRS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.4 quality of design reviews</td>
<td>see note 4.; should reflect degree to which ISO9000-compatible procedures were applied</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.5 quality of user documentation</td>
<td>see note 4.; should reflect utility and timeliness</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FACTOR</td>
<td>NOTES</td>
<td>SCORE</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>1.3.6 quality of software project plan</td>
<td>see note 4.; should reflect degree of detail specified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.7 approp. Syst. DA experience</td>
<td>see note 4. (Syst. DA is the system design authority)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.8 approp. SDA experience</td>
<td>see note 4. (SDA is the software design authority)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.9 ave. domain experience of team</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.10 ave. language experience of team</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.11 ave. requirements analysis experience of team</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.12 ave. design methodology experience of team</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.13 ave. development system experience of team</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.14 ave. target system experience of team</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.15 % code review cover</td>
<td>enter percentage into score</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.16 ave. code review experience</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.17 quality of module test</td>
<td>see note 4.; should reflect structural coverage of design and availability of suitable tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.18 SCNs</td>
<td>enter approx. no. of SCNs (software change notes) raised during development into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.19 quality of integration test</td>
<td>see note 4.; should reflect structural coverage of design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.20 quality of defect logging</td>
<td>see note 4.; reflect logging rigour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.21 quality of config. management</td>
<td>see note 4.; should reflect formality of process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.22 quality of design to meet ARM objectives</td>
<td>see note 4.; should reflect the degree to which ARM objectives were recognised in the s/w design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FACTOR</td>
<td>NOTES</td>
<td>SCORE</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>--------------------------------------------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>1.4 Project characteristics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.1 month started</td>
<td>enter into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.2 year started</td>
<td>enter into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.3 planned duration (months)</td>
<td>enter into score box; include extensions arising from Customer changes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.4 actual duration (months)</td>
<td>enter into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.5 duration (months) at customer site prior to acceptance</td>
<td>enter into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.6 size of team (ave.)</td>
<td>enter into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.7 level of staff turnover over duration</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.8 effort</td>
<td>enter total man-months of effort into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.9 software effort</td>
<td>enter software engineering man-months of effort into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.10 software subcontracting</td>
<td>see notes 4. &amp; 5.; should reflect proportion s/c (1=none; 9=all)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.11 approp. project mngr. experience</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.12 quality of project tracking and review</td>
<td>see note 4.; should reflect frequency and formality of project reporting and mngr. review</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.13 location</td>
<td>enter into comments box; secondary locations should be also entered under comments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.14 project type</td>
<td>see note 6.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.15 customer type</td>
<td>see note 7.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.16 application area</td>
<td>see note 8.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Maintenance policies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1 experienced s/w maint. staff exist</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FACTOR</td>
<td>NOTES</td>
<td>SCORE</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>2.2 maint. staff familiar with system design</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3 maint. staff familiar with language</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4 sufficient maint. staff exist</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5 maint. team have worked together before</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.6 maint. team have adequate access to the development Syst.DA</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7 maint. team have adequate access to the development SDA</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.8 maint. team have adequate access to the development HDA</td>
<td>see note 4. (HDA is the hardware design authority)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.9 maint. team have adequate access to dev. documentation</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.10 reproducibility</td>
<td>is the most recent software build reproducible from scratch? comments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.11 maint. team have access to replica system</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.12 ISO9000 procedures applied</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.13 ave. no. of system maintainers</td>
<td>enter into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.14 maint. undertaken at customer location</td>
<td>enter into score box (100% site=1, 100% home=9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.15 maint. subcontracted</td>
<td>enter yes/no into score box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Outcomes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1 Development</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1.1 on schedule completion</td>
<td>enter yes/no into score box; if not, indicate approx. % overrun in comments box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FACTOR</td>
<td>NOTES</td>
<td>SCORE</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>3.1.2 on budget completion</td>
<td>enter yes/no into score box; if not, indicate approx. % overrun in comments box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1.3 on budget completion (including any contingency)</td>
<td>enter yes/no into score box; if not, indicate approx. % overrun in comments box</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2 Maintenance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2.1 Frequency</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2.2 Degree of difficulty</td>
<td>see note 4.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2.3 Availability of absolute frequencies and/or costs incurred</td>
<td>enter yes/no into score box</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Explanatory Notes

1. ‘Maintenance’ refers to changes to the system after initial development and acceptance. A useful definition of software maintenance is ‘the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.’

2. KDSI means ‘thousand lines of deliverable source instruction’

3. A computer software configuration item (CSCI) may be defined as “a named, separately compilable file containing source code. It will typically, though not necessarily, perform a single logical task or set of tasks.”

4. This factor should be assessed on a scale of one to nine, where five represents a nominal, typical or median result. One reflects a better than expected or ‘good’ result while nine is at the inadequate or unsatisfactory end of the scale. The value selected should be entered into the score box. If available and appropriate (e.g. range of experience within a team) then the upper and lower values should be entered into comments. Although the results are necessarily subjective, they do have value when a statistically significant sample size is considered.

5. Subcontract relates to the use of a third party to produce bespoke software (probably on a fixed price), but does not include the use third party proprietary software or contractors employed to operate as members of the software team.

6. Project type should be scored as follows:

   new development - %
   re-use of proven code - %
   extension of an existing system - %

   The value(s) selected should be entered into the comments box.

7. Customer type should be scored as follows:

   UK military end-user - 1
   UK military prime - 2
   Overseas military end-user - 3
   Overseas military prime - 4
   UK civil end-user - 5
   UK civil prime - 6
   Overseas civil end-user - 7
   Overseas civil prime - 8
8. **Application area** (e.g. avionics, business system, CBT, simulation....) should be described in the *comments* box. If predominantly real time (e.g. sensor systems) enter 1 into *score* box, if predominantly database orientated (e.g. business systems) enter 3 into *score* box; if neither predominates enter a 2 into *score*.
**SOFTWARE MAINTENANCE ESTIMATION PROJECT - DAVID MORRISON**

Some explanatory notes:

- The project relates to the field of software maintenance, and specifically to the estimation of probable future maintenance costs on a software project which is still at the feasibility stage. It is regarded as an important field because of the ubiquitous nature of software, and the fact that the bulk of software engineering expenditure now relates to the maintenance of existing programs rather than developing new ones; it was reckoned to have cost £26.5Bn during 1990 in Western Europe alone. Despite this, most formal estimation tools remain development oriented, or relate to measuring the maintainability of existing code.

- The current phase of the project involves analysing metrics relating to past software development projects, to demonstrate a relationship between these factors and the subsequent maintenance of the software. The objective is to validate the thinking provided by the literature in the field, and also reflected in some current maintenance models.

- It is important to collect information from a variety of environments and application areas to genericise the conclusions, and ultimately the utility of the model.

- This is the purpose of the survey document which needs to be completed for each past development project for which information is available, and for which a perspective on the current maintenance status exists.

- If required, the questionnaire is available in a soft format (Microsoft Word).
APPENDIX 2

ANALYSIS OF VARIANCE

The Null Hypothesis:

'No systemic difference exists between predictions made by random – and selected – sample models'  

Using the test statistic for the Null Hypothesis:

\[ H_0 : \ 1 = \ 2 \]

Sum of squares for treatments:

\[ SST = \sum_{i=1}^{2} n_i (\overline{X}_i - \overline{X})^2 \]

\[ = 0.9 \]

Sum of squares for error:

\[ SSE = \sum_{i=1}^{2} \sum_{j=1}^{n_i} (X_{ij} - \overline{X}_i)^2 \]

\[ = 20.04 \]

Mean squares for treatment:

\[ MST = \frac{SST}{1} = 0.900 \]

\[ MSE = \frac{SSE}{66} = 0.304 \]

\[ F = \frac{MST}{MSE} = 2.96 \]

\[ F_{\alpha} \Omega 4.00 \text{  For } \alpha = 0.05 \]

\[ 1 = 2 \]

Accept Null Hypothesis
<table>
<thead>
<tr>
<th>Author</th>
<th>Title</th>
<th>Publisher/Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anon.</td>
<td>THE NATIONAL BULLETIN BOARD <a href="http://www.it2000.com">www.it2000.com</a> (October 1997)</td>
<td></td>
</tr>
<tr>
<td>Author(s)</td>
<td>Title</td>
<td>Details</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Mary B 1997</td>
<td></td>
<td>Model Uncertainty and Forecast Accuracy</td>
</tr>
<tr>
<td>Chatfield, Chris 1997</td>
<td></td>
<td>Forecasting in the 1990s The Statistician 46, No 4, 461-473 (1997)</td>
</tr>
<tr>
<td>Dettmer, Roger 1999</td>
<td>Liberte, fraternite and Linux</td>
<td>IEE Review, May 1999</td>
</tr>
<tr>
<td>Dunn, John 1997</td>
<td>Professional Engineering</td>
<td>Vol. 10 no. 8, 16 (1997)</td>
</tr>
<tr>
<td>Financial Times (2) 1998</td>
<td>Banks will “not be ready to join EMU until 2005”</td>
<td>The Financial Times Ltd., 13 March 1998</td>
</tr>
<tr>
<td>Author / Title / Year</td>
<td>Details</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>Financial Times 1999</td>
<td>New euro software gets cold shoulder&lt;br&gt;The Financial Times Ltd., 27 May 1999</td>
<td></td>
</tr>
<tr>
<td>Financial Times 2000</td>
<td>Delays to MoD procurement increase&lt;br&gt;The Financial Times Ltd., 6 July 2000</td>
<td></td>
</tr>
<tr>
<td>GEC-Marconi 1993</td>
<td>Software Estimator’s Handbook&lt;br&gt;ASET Report No. 78</td>
<td></td>
</tr>
<tr>
<td>Haworth, Dwight A 1996</td>
<td>Regression Control Charts to Manage Software Maintenance&lt;br&gt;Software Maintenance: Research and Practice&lt;br&gt;John Wiley &amp; Sons Ltd.&lt;br&gt;Vol.8, 35-48 (1996)</td>
<td></td>
</tr>
</tbody>
</table>
Henry Joel, Blasewitz Robert & Kettinger David 1996
Software Maintenance Process
Software Maintenance: Research and Practice
John Wiley & Sons Ltd.
Vol. 8, 79-100 (1996)

Henry Joel, Henry Sallie, Kafura Dennis & Matheson Lance 1994
Improving Software Maintenance at Martin Marietta, IEEE Software, July 1994

Henry, Sallie & Wake, Steve 1991
Predicting Maintainability with Software Quality Metrics
Software Maintenance: Research and Practice
John Wiley & Sons Ltd.

Hetzel, William C 1995
The Sorry State of Software Practice
Measurement and Evaluation
Journal of Systems & Software
Elsevier Science Inc.

Hillson, David 1998
Managing risk
IEE Review
January 1998 31

Hinley, David S 1996
Software evolution management:
a process-oriented perspective
Information and Software Technology
Elsevier Science B.V.
North Holland
Vol. 38, 723-730 (1996)

Hops, Jonathon M & Sherif, Joseph S 1995
Development and Application of Composite Complexity Models and a Relative Complexity Metric in a Software Maintenance Environment
Journal of Systems & Software
Elsevier Science Inc.

Hughes, RT, Cunliffe, A & Young-Martos F 1998
Evaluating software development effort model-building techniques for application in a real-time telecommunications environment
IEE Proceedings- Software

Jardine, AKS & Buzacott, JA 1985
Equipment Reliability & Maintenance
European Journal of Operational Research
Elsevier Science Publishers BV, North Holland

Jelinski, Z & Moranda, PB 1972
Statistical Computer Performance Evaluation
Academic Press
New York (1972)
Jones, Vernon Dale, Bretschneider, Stuart & Gorr, Wilpen L 1997
Organisational Pressures on Forecast Evaluation: Management, Political and Procedural Influences
John Wiley & Sons

Jorgensen, Magne 1995
Experience with the Accuracy of Software Maintenance Task Effort Prediction Models
IEEE Trans. on Software Eng.
Vol.21, 8, 674-681 (1995)

Kan, Stephen H 1995
Metrics and Models in Software Quality Engineering
Addison Wesley Longman

Kemerer, Chris F & Slaughter, Sandra A 1997
Determinants of Software Maintenance Profiles: An Empirical Investigation
Software Maintenance: Research and Practice
Vol. 9, 235-251 (1997)

Khoshgoftaar, Taghi M & Lanning, David L 1995
A Neural Network Approach for Early Detection of Program Modules Having High Risk in the Maintenance Phase
Journal of Systems & Software
Elsevier Science Inc.

Khoshgoftaar, TM & Munson, JC 1990
Predicting Software Development Errors using Complexity Metrics
IEEE Journal of Selected Areas in Communications 8(2), 253-261 (1990)

Lano, Kevin & Haughton, Howard 1993
Reverse Engineering and Software Maintenance

Lederer, Albert L & Prasad, Jayesh 1992
Nine Management Guidelines for Better Cost Estimating

Lee, Anita, Chun Hung Cheng & Balakrishnan, Jaydeep 1998
Software development cost estimation: Integrating neural network with cluster analysis
Information & Management
Elsevier Science BV 34 (1998) 1-9

Lehman, MM 1980
Programs, lifecycles and the laws of software evolution
Proc. IEEE, 19, 1980, 1060 - 1076

Lientz, BP & Swanson, EB 1980
Software Maintenance Management
Addison-Wesley
<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Publisher/Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Londeix, Bernard</td>
<td>Cost Estimation for Software Development</td>
<td>Addison-Wesley (1990)</td>
</tr>
<tr>
<td>Makridakis, Spyros</td>
<td>Accuracy measures: theoretical and practical concerns</td>
<td>International Journal of Forecasting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Elsevier Science (1993), 527-529</td>
</tr>
<tr>
<td>Martin, Doug E</td>
<td>TPM Experiences of DuPont Fibers</td>
<td>Total Productive Maintenance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proceedings of First International Conf. (1992)</td>
</tr>
<tr>
<td>Middleton, Peter</td>
<td>Maintenance Management: From Product to Process</td>
<td>Software Maintenance: Research and Practice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New York (1976)</td>
</tr>
<tr>
<td>Owen, Mal</td>
<td>SPC and Continuous Improvement</td>
<td>IFS Publications, Bedford (1989)</td>
</tr>
</tbody>
</table>
Patton, Joseph D 1980
Maintainability and Maintenance Management
Instrument Society of America
North Carolina (1980)

Pfleeger, Shari Lawrence 1995
Maturity, Models and Goals
How to Build a Metrics Plan
Journal of Systems & Software
Elsevier Science Inc.

Pole Andy, West Mike & Harrison Jeff 1994
Applied Bayesian Forecasting and
Time Series Analysis (1994)

Potts, Colin 1993
Software Engineering Research Revisited
Chapman & Hall, New York

Pressman, R S 1997
Software Engineering: A Practitioner’s Approach

Putnam, Lawrence H & Myers, Ware 1997
How Solved Is the Cost Estimation Problem?

Putnam, LH 1978
Example of an Early Sizing, Cost and Schedule
Estimate for an Application Software System
Proceedings of COMPAC 78
IEEE (1978)

Rook, Paul 1990
Software Reliability Handbook

Sakthivel, Sachidanandam 1994
A Decision Model to Choose between Software
Maintenance and Software Redevelopment
Software Maintenance: Research and Practice
Vol. 6, 121 - 143 (1994)

Schneidewind, N F 1987
The State of Software Maintenance
Vol. 13, 303 - 310 (1987)

Schach, Stephen R 1994
The Economic Impact of Software Reuse on
Maintenance
Software Maintenance: Research and Practice
Vol. 6, 185 - 196 (1994)

Sharma, Subhash 1996
Applied Multivariate Techniques

Shooman, Martin L 1988
Software Engineering
McGraw-Hill
Singapore (1988)
205
<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sommerville, Ian</td>
<td>Software Engineering</td>
<td>2001</td>
</tr>
<tr>
<td>Spirtes Peter, Richardson Thomas, Meek Christopher, Scheines Richard &amp; Glymour Clark</td>
<td>Using Path Diagrams as a Structural Equation Modelling Tool</td>
<td>1998</td>
</tr>
<tr>
<td>Stark, George E &amp; Oman, Paul</td>
<td>A Survey Instrument for Understanding the Complexity of Software Maintenance</td>
<td>1995</td>
</tr>
<tr>
<td>Stevens, Alastair</td>
<td>The BUGS Project</td>
<td>2000</td>
</tr>
<tr>
<td>Stork, David G</td>
<td>Hal's Legacy</td>
<td>1997</td>
</tr>
<tr>
<td>Swanson, EB &amp; Beath, CM</td>
<td>Maintaining Information Systems in Organisation</td>
<td>1989</td>
</tr>
<tr>
<td>Tucker, Paul &amp; Griffiths, John</td>
<td>Measuring and Targetting Overall Equipment Effectiveness</td>
<td>1992</td>
</tr>
<tr>
<td>Tinham, Brian</td>
<td>Manufacturing Computer Solutions</td>
<td>2000</td>
</tr>
<tr>
<td>Turver, Richard J &amp; Munro, Malcolm</td>
<td>An Early Impact Analysis Technique for Software Maintenance</td>
<td>1994</td>
</tr>
<tr>
<td>Vallabhaneni, S. Rao</td>
<td>Auditing the Maintenance of Software</td>
<td>1987</td>
</tr>
</tbody>
</table>

206
Van Genuchten, Michiel 1991
Why is Software Late? An Empirical Study of Reasons For Delay in Software Development

Walston, CE & Felix, CP 1997
A method of programming measurement and estimation
IBM Systems Journal, 16(1), 54-73, 1977

Wei Li 1997
Software Reuse in Reconstructive Maintenance

West, Richard 1996
Improving the Maintainability of Software

Wheatley, Malcolm 1999
E-commerce and manufacturing: closing the UK-US gap

Wickens, Peter D 1998
The Ascendant Organisation

Wigington, J C 1980
A Note on the Comparison of Logit and Discriminant Models of Consumer Credit Behaviour

Wilmott, Peter 1992
TPM-The Western Way
Total Productive Maintenance Proceedings of First International Conf. (1992)

Xie, M 1993
Software Testing, Verification & Reliability