Durham E-Theses

Software maintenance cost estimation with fourth generation languages

Lamb, Raymond K.

How to cite:

Lamb, Raymond K. (1997) *Software maintenance cost estimation with fourth generation languages*, Durham theses, Durham University. Available at Durham E-Theses Online: http://etheses.dur.ac.uk/4728/

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full Durham E-Theses policy for further details.

Academic Support Office, Durham University, University Office, Old Elvet, Durham DH1 3HP
e-mail: e-theses.admin@dur.ac.uk Tel: +44 0191 334 6107
http://etheses.dur.ac.uk

University of Durham

School of Engineering and Computer Science
(Computer Science)

Software Maintenance Cost Estimation With Fourth Generation Languages

The copyright of this thesis rests with the author. No quotation from it should be published without the written consent of the author and information derived from it should be acknowledged.

Raymond K. Lamb

Submitted for degree of M.Sc.
1997

0 9 MAY 1997

Abstract

Software Maintenance Cost Estimation With Fourth Generation Languages

Raymond K. Lamb

This thesis addresses the problem of allocation of software maintenance resources in a commercial environment using fourth generation language systems.

The activity of maintaining software has a poor image amongst software managers, as it often appears that there is no end product. This image will only improve when software maintenance can be discussed in business terms, one of the main reasons being that the maintenance costs can then be compared to the costs of not maintaining the system.

Software maintenance will continue to exist in the fourth generation environment, as systems will still be required to evolve.

Cost estimation is an imprecise science, as there are many variables such as human, technical, environmental and political which can effect the ultimate costs of software and the resources required to maintain it. Some of the factors appear more obvious than others, for example an experienced programmer can achieve a specific task in less time than an inexperienced one. To fully estimate software maintenance costs these factors need to be identified and weights assigned to them.

This thesis examines a means to identify these factors and their weights, and produces the first cut of an equation which will enable the software maintenance resources in a fourth generation language to be estimated.

Acknowledgements

I would like to thank my supervisor, Mr M. Munro, for his help and guidance throughout this project, and without whom none of this would have been possible.

Copyright

The copyright of this thesis rests with the author. No quotation from it should be published without his prior written consent and information derived from it should be acknowledged.

Dedication

This work is dedicated to my wife, Joan.

Disclaimer

This thesis has not been submitted for any other degree at any other institution.

CONTENTS

CHAPTER 1

Introduction

Abstract: Large backlogs of computing tasks have built up, largely because software maintenance uses the majority of data processing resources. To allow this backlog to be addressed more quickly fourth generation languages have been devised, but little work has been done to assess the effect of these products on software maintenance. Are these products, which were designed primarily to allow rapid development, going to use more resources in the maintenance phase, therefore making the situation as bad if not worse, in a few years time? To assist, software maintenance structured techniques were devised, which are aimed mainly at third generation languages, therefore, will the advantages of these methods be lost with the introduction of fourth generation languages? Are we simply replacing 77 billion lines of unstructured COBOL [PENTZOLD87] code with 77 billion lines of unstructured fourth generation language?

1.1 The Thesis Position.

Bennett [BENNETT89] stated that the activity of maintaining software is not generally regarded highly by software engineers or managers. Maintenance activity is perceived to be less creative, less complex and less challenging than new development. Managers within the computer industry currently have a poor image of software maintenance, as it often appears that there is no end product, purely the software that previously existed, and the minimum of resources are, therefore, devoted to this phase of the software life cycle. This thesis addresses the factors which influence the allocation of

resources that need to be assigned to software maintenance when using fourth generation languages.

The resources used by software maintenance has led to large backlogs of tasks, and fourth generation languages have been developed to address this problem, they have however, been devised to increase the speed at which the software can be developed, but little consideration has been given to their effect on the maintenance phase. It is possible, then, that fourth generation languages will use more software maintenance resources than third generation languages, and make the situation worse, rather than better. A 20% saving in development could be offset by a 10% increase in software maintenance, which accounts for about two-thirds of the software life cycle.

To be as competitive as possible businesses need to be aware of all their costs, otherwise resources and money will be wasted. A major problem with cost estimation for software maintenance is that it depends on a number of factors, not all of which are exact or easy to define, therefore it is often done badly or not at all. As a result of this software maintenance suffers from a lack of effective management and a poor image which will only improve when software maintenance can be discussed in business terms. One of the main requirements for this change is that the resources required for carrying out the maintenance can be defined and compared to the costs of not making the changes.

The basic premise of this thesis is that a number of factors that influence the amount of software maintenance can be identified, and tested using survey data. These influences can then be used in three ways:

- To allow the factors to be considered during the development and maintenance of the system. By considering the factors, those which increase software maintenance can be minimised and those which reduce maintenance to be maximised and thus the costs of software maintenance can be reduced to a minimum.

• To enable software maintenance costs to be estimated, thus allowing this process to be discussed in business terms, which should assist this phase of the software life cycle to gain parity with the other processes of the business. Resources could also be made available when they were required, therefore, cutting down on wastage and allowing the costs to be reduced.

• By considering the values for these factors a decision can be taken as to the best time to rewrite the system. The estimation of costs for keeping the system running can be compared with the costs of rewriting the system, and the large costs of keeping old inefficient software running can be minimised if it is cheaper in the long term to rewrite.

1.2 Criteria for Success.

The outcome of the research into fourth generation language environments and software maintenance will be to produce a means to

- Identify a list of factors which influence the amount of software maintenance in a fourth generation language environment.
- Assess the importance of the identified factors and assign a weighting factor to them.
- Combine the weights for the factors to produce the first cut of an equation for the amount of software maintenance a system written in a fourth generation language will require.

1.3 Thesis Overview.

This thesis produces the first cut of an equation to estimate the amount of software maintenance required by a particular system. It illustrates this by working through an example when discussing the techniques. It should be remembered that software

maintenance cost estimation only provides an estimate of the required resources and therefore it is likely to need adjustment for a particular installation.

In this thesis three elements are considered, software maintenance, fourth generation languages and software cost estimation. The order is to firstly introduce software maintenance, followed by fourth generation languages and a discussion of software maintenance using fourth generation languages. Software cost estimation in a third generation language environment is then included, and the factors which influence cost estimation. The factors which influence software cost estimation specifically when using fourth generation languages are then introduced, and these are then applied to software maintenance cost estimation models.

Chapter 2 introduces software maintenance and discusses the effect of fourth generation languages on software maintenance.

Chapter 3 introduces software cost estimation, and includes discussions on the need for it and the difficulties involved. This Chapter also includes a discussion on the resources that need to be scheduled to enable software to be properly managed.

In Chapter 4 data modelling and its techniques are discussed together with curve fitting. The statistical techniques used when analysing the results of the survey are also described.

In Chapter 5 the factors being tested are introduced, and the questionnaire used in this research is described together with a summary of the results obtained from the analysis of the data.

Chapter 6 brings together the conclusions from the preceding chapters, and introduces an equation for the amount of software maintenance effort required to maintain fourth generation language systems and tests the equation with results obtained from a second survey. This chapter also includes a discussion on the use of the equation.

In Chapter 7 the conclusions from the research are evaluated with respect to the criteria for success.

Summary: This chapter lays out the format for the thesis and shows an overview of the need for the research. A discussion of the criteria for success of the research is also included.

CHAPTER 2

Software Maintenance and Fourth Generation Languages

Abstract: This chapter defines Software Maintenance and the problems it causes. It also discusses the growth in Fourth Generation Languages and their use to try to resolve some of these problems. A classification of the types of Fourth Generation Languages is also included.

2.1 Software Maintenance Definition.

There are many different definitions of software maintenance, some companies using time limits as a definition, for example any change requiring more than 5 or 10 day's effort being classed as development. Other companies classify every enhancement as development and reserve maintenance purely for bug fixing. The definition that is the most widely used [ANSI90] is:

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

An alternative definition, which concentrates less on the technical aspects of software maintenance is that used by the Centre for Software Maintenance [CSM92] is:

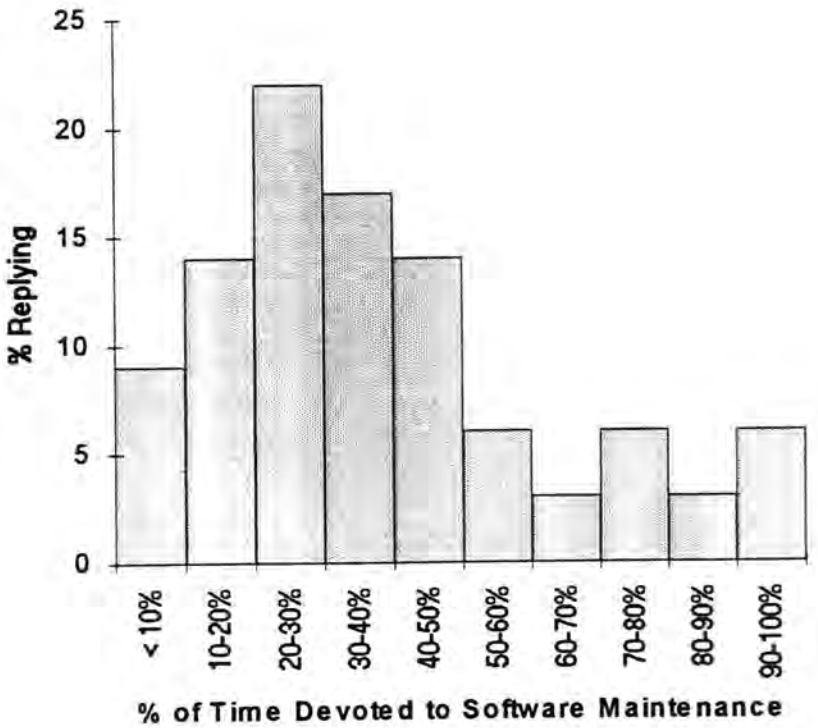
> *"Software Maintenance is the set of activities (both technical and managerial) necessary to ensure that software continues to meet organisational needs"*

Software maintenance is therefore a very broad activity that includes error corrections, enhancements of capabilities, deletion of obsolete activities, and optimisation.

Survey data [CSM87] suggests that software maintenance accounted for 49% of the total data processing budget in 1979, rising to 65% in 1986 and the indications are that it is getting worse. One estimate suggests software maintenance costs more than £1 billion in the United Kingdom per annum, and in many data processing departments the cost exceeds that of development. The Gartner Group have predicted that by the middle of the 1990s repairing old code could require over 90% of the systems processing budget [BLACK92]. In 1977 software costs in the U.S.A. were in excess of $50 billion, which represented more than 3% of the American G.N.P. for that year, and Lehman says that these costs have now more than doubled and are comparable in other developed countries [LEHMAN80]. A Hoskyns survey of 905 British installation [GILB82] concerning the percentage of time devoted to software maintenance is shown in Figure 2.1 overleaf.

The resources devoted to software maintenance are one of the main reasons for the shortage of staff, as shown by the number of companies trying to recruit experienced personnel, and large backlogs of user requests. Often users of computer systems are aware of the waiting lists for amendments and do not request desirable changes to the system, this invisible backlog usually not being measured. Many companies have a 3 to 4 year visible backlog of applications [MARTIN83] waiting for implementation. A study by the Sloan School Centre for Information Systems Research [MARTIN83] has put the invisible backlog at 164% of the declared one. The total backlog measured in this study represented 179% of the entire base of installed applications.

Figure 2.1: Survey of 905 installations



This backlog of applications has in many cases led to shortcuts being taken to produce applications more quickly and these are being paid for later by increased maintenance. Boehm [BOEHM75] quotes one case where the development cost of an avionics system was $30 per instruction but the maintenance cost was $4,000 per instruction.

All computer systems are required to evolve and will continue to do so with fourth generation languages.

There are many functions of software maintenance including:

- To support user improvements. Software requirements are not static over the life of the system and these requests for change will need including.
- Provision of mandatory upgrades due to government regulations and management decisions.

- Provision of a continuity of services, by fixing bugs, recovering from failures, and to accommodate operating system, database and equipment changes.
- To support maintenance improvements. Code documentation and databases will need to be cleaned up periodically.
- To prolong the life of the software and to enable it to be kept up to date with the requirements of the business.

Without maintenance the software in which the company has invested large amounts of time and money would soon become worthless. Besides the obvious financial costs of software maintenance, there are other less tangible costs. These costs include:

- Development opportunities may be postponed or lost because of the resources used by software maintenance.
- Development work may suffer if staff are temporarily pulled off the project to work on maintenance.
- Customers may become dissatisfied when requests for repair or modification cannot be addressed in a timely manner.
- The overall software quality reduces as a result of changes that introduce latent errors in the maintained software. This is because performing maintenance on a program is more difficult than writing it. In many cases the person who actually wrote the program has left or is not available, therefore the person performing the maintenance has to read and understand the original programmers style which is time consuming, particularly if the documentation is out of date or inaccurate.

Software never stands still, if it did it could be hardwired in microchips, the essence of most programming is therefore maintenance.

2.2 Problems Created by Software Maintenance

The traditional third generation language software life cycle model [MARTIN83] is:

Requirement Analysis	3%
Specification	3%
Design	5%
Coding	7%
Testing	15%
Operation and Maintenance	67%

Errors and omissions can occur at each stage, and the cost of correcting these increases ten fold as the project progresses, which means that an error which costs £10 to fix in the specification stage costs £100 in the design and £1,000 in coding. Cutting down on errors and omissions introduced during specification and design therefore dramatically reduces development and maintenance costs, while at the same time increasing the quality of the finished product and therefore the service to the end user. Boehm [BOEHM74] reports that in some large systems up to 95% of the code had to be rewritten to satisfy changed user requirements and also that 12% of errors discovered in a software system over a 3 year period were due to errors in the original system requirements.

Ball [BALL87] published the results of a survey he had performed concerning the problems associated with software maintenance, amongst several maintenance management issues were extensive backlogs (30%), and what data processing management wants versus what the user needs (30%). The percentages refer to the percentage of respondents who quoted the problem.

Colter [COLTER88] says that many people think that the money spent on software maintenance is wasted, it is therefore vital that maintainers learn "The business

of software maintenance". At present software is not seen as a company asset, and this needs to be addressed. This will enable software maintenance costs to be placed into the business environment. For example, the cost of software maintenance over the next 12 months will be £10,000 but the effects of not spending the money is that no price increases can be implemented, no new customers can be accommodated, etc.

2.3 Types of Software Maintenance

Swanson [SWANSON76] categorised software maintenance into three categories:

a) Perfective maintenance results primarily as a result of changes in users requirements, but also includes: changes to enhance performance, improve cost-effectiveness, improve processing efficiency, etc.

b) Adaptive maintenance adapts the software to changes in the data requirements or the processing environments, for example the central database definition may change or the hardware be altered. Adaptive maintenance includes implementation of a database management system for an existing application system, modification of a code from three to four characters, tuning a system to reduce response times, converting a system from batch to on-line operation, modification of a program to use a different terminal, etc.

c) Corrective maintenance corrects failures in the software. Traditionally it has been seen as the prime function of maintenance, that of fixing "bugs".

Zvegintzov [ZVEGINTZOV91] produced a table of comparison between software maintenance effort, and this was refined by Abran [ABRAN93] and this is shown in Table 2.1.

Table 2.1: Abran's comparison of maintenance effort:

Work Category	LIENTZ80	GUIDE85	BALL87b	DEKLOVA90	ABRAN90 See Note 1	ABRAN90 See Note 2
Corrective	22%	10%	17%	16%	22%	21%
Adaptive	59%	69%	39%	43%	57%	60%
Perfective	16%	7%	29%	28%	7%	3%
User Support	3%	14%	15%	13%	14%	15%
Total Non-Corrective	78%	90%	83%	84%	78%	79%

Note 1: This research was published in 1990 but carried out in 1989.

Note 2: This research was published in the same paper as 1, but carried out in 1990.

Even a system that is totally reliable, completely meets user requirements and is well structured will frequently be changed during the maintenance phase. Errors will have to be corrected and amended to meet new or changed end user requirements. Unless software systems are designed to be changed more easily without jeopardising their quality, maintenance of these systems will continue to be a time consuming and costly activity.

2.4 Fourth Generation Language Definition.

The term fourth generation language refers to a class of data processing languages developed in the mid 1970's that offer simplified expressions for common data processing tasks. These languages allow for systems development in significantly less time than with third generation languages.

The first generation of computer languages was machine code, the next assembler, and the third the high level languages such as COBOL, FORTRAN and PL/1.

Even these high level languages do not make life totally easy for application programmers. The programmer needs to become involved in formatting the layout of computer records, editing and validating input to the program, and organising the data that the program works with. Often the programmer has to consider that the report should have the standard heading at the top of the page, with a page number on the right and the date on the left, but may have to write 30 or 40 program statements to accomplish this.

Fourth generation languages were developed to make life easier for the application programmer. With most fourth generation languages there are a set of predefined defaults which the compiler or interpreter uses to make assumptions about the users needs. For example, it may automatically select a format for a report, put page numbers on it, select chart types for graphics display, put labels on the axes or on column headings, and ask the user in a friendly, understandable fashion when it needs more information. An assumption behind such languages is that a relatively large amount of computer power is available for compiling and interpreting.

Fourth generation language has become a blanket term applied to an array of products covering everything from specialised report writers to extended database query languages.

Codd, president of the Relational Institute and developer of the relational database has said, [CODD85] "There is no definition of a fourth generation language worth its salt, let alone any theoretical foundation . . . . Thus, any vendor can claim to provide a product that supports a fourth generation language, and there is no basis for checking or challenging such a claim".

A large portion of the degree of a systems maintainability is built in at the time it is written, as many of the factors which influence the amount of software maintenance

are decided at this time, for example the language, the use of structured techniques etc. and these can only be changed by rewriting portions of the system. One of the advantages of fourth generation languages is that it allows parts to be rewritten more quickly than with a third generation language.

Martin [MARTIN83] has said that a characteristic of a fourth generation language is that an analyst can obtain results faster than he could write specifications for a programmer. The analyst then works hand in hand with the user, creating what the user asks for and refining it in a step by step fashion to adapt it better to the user's needs.

In the United Kingdom 45% of installations were making significant use of fourth generation languages in 1986, and a further 30% were planning to introduce them in the near future [IDPM86], so in a few years software maintenance using fourth generation languages is likely to be a major factor to many companies.

2.5 Fourth Generation Language Classification

Although the term fourth generation language is in common use, they consist of a range of products, and the term fourth generation environment would be more applicable, however because it is in common use the former will be used throughout this thesis.

Fourth generation languages are not just one type of tool, they consist of a wide range of products, and to enable them to be compared, it is necessary to classify them into categories. A report by the Institute of Data Processing Managers [IDPM86] produced a list of 4 classifications of fourth generation languages and this is produced below.

a) Application builders. These products require consideration to be given to the systems design of the application being built. This means that they are for use by computer experts, who will have to code a portion of the application using third

generation like statements. The skills required will not be so great as for third generation language programming and considerable productivity gains can be achieved. Their range of application is wide, and they tackle the whole of an application. Because of the templating approach fourth generation languages limit the strategies which can be used. Application builder fourth generation languages comprise:

- Complete sets of program function packages:
    - Database packages and data dictionaries
    - Screen and report generators
    - Enquiry languages
    - A fixed processing cycle
    - A powerful third or fourth generation language for calculation and logic

Or

- Complex application packages

Examples of application builders are:

Application factory

BOS/ Speedbuilder

b) Transaction processing builders. These products are intended for professional programmers but do not have the same range of use as application builders. They do not include a database package. The target for transaction processing builders is mature installations that have already made a substantial investment in their existing database and do not wish to incur the costs of change. They comprise:

- A subset of program function packages, typically:
    - Screen and report generators
    - A fixed processing cycle
    - A powerful third or fourth generation language for calculation and logic

Examples of transaction processing builders are:

GAin

TELON

c) Management information systems. Professional programmers and end users, such as managers, financial analysts, statisticians and market analysts can use these products. Essentially they provide the facility for building and manipulating models. They comprise:

- A sophisticated database package including time-series and/ or multi-dimensional data views
- A 'what if' modelling capability
- Statistical, financial or other specialist functions
- Enquiry languages
- A sophisticated screen and report generator including graphics

Drawing data from a database is a frequent use of these products. Data processing professionals will normally build the model's database and the interface to existing databases and frequently work with the end users in other areas.

An example of a management information system is E.I.S.

d) End user products. A fourth generation language that is an end user product will require no skill for its use, or a minimum of such skills as can be acquired in, say a two day course plus a few days experience by a person with no special programming aptitude. These products are usually specialised and typically include:

- Database enquiry languages
- Report generator languages
- Spread sheet analyses
- Storage and retrieval packages
- Simple application packages

Examples of end user products are the Structured Query Language elements of Oracle and Ingres.

These categories relate to their use in development, and are not intended to relate to software maintenance.

2.6 Maintenance With Fourth Generation Languages

Over 77 billion lines of COBOL exist world-wide[PENTZOLD87] and billions more in other languages many of which will need maintenance for many years to come. The rate at which new products are developed and released is at least 2.5 times faster than the rate at which mature products become obsolete [TANG89]. Some existing systems are more than 20 years old, and COBOL will probably be around for at least another 20. The lack of trained staff and the applications backlog have caused companies to turn to fourth generation languages to enable quicker production of code, the result being more programs to maintain. The effect of fourth generation languages on software maintenance has attracted little research yet.

Chapin [CHAPIN84] has produced a list of 10 observations into the effect of fourth generation languages on software maintenance:

1. Short fourth generation language programs less than about 45 lines of source code long in total are distinctly easier, cheaper and faster to maintain than a functionally equivalent implementation in a third generation language.

2. Short fourth generation language programs less than about 45 lines of source code long in total are usually faster, sometimes easier, but rarely cheaper to maintain than a functionally equivalent implementation in a third generation report writer, report generator, file maintenance, or database access or enquiry language (e.g. RPG, MARK-IV and Easytrieve).

3. The defaults, explicit and implicit, and the tacit assumptions made in using a fourth generation language are more troublesome in maintenance than in development. They sometimes block what at first appears to be a "clean" way of getting the computer to do something.

4. Programs and systems implemented monolithically in a fourth generation language are noxious in maintenance. The worst offenders run from a few hundred to thousands of lines of fourth generation code in length.

5. Programs and systems implemented in explicit modules in a fourth generation language are better in maintenance than monolithically implemented ones.

6. Constraining data access is a major stumbling block in maintenance with fourth generation language implemented programs and modules. Module interfaces easily become both explicitly complex and extensive in such code.

7. With the technology available at the time when the article was written (late 1983), systems implemented with fourth generation languages were slower, more difficult, and more costly to maintain than with third generation languages. Inter program and inter system communications of data are often clumsy and obscure.

8. The amount of work a programmer does depends more on the power of the language to be used in the implementation than upon the number of good lines of source code written.

9. Maintenance work done with fourth generation languages appears to be largely unsuccessful in correcting a common problem arising from their use in development - they are computer resource hogs.

10. Adaptive maintenance work with fourth generation languages is being contributed to by the vendors of the languages. In particular, the lack of forward compatibility is troublesome.

Fourth generation languages programs in use are not usually as short as recommended by Chapin, and it appears that since 1983 vendors have addressed some

of the issues raised by him. Fourth generation languages now often have large amounts of compiled code called by individual commands which has improved performance and reduced the hardware resources required. It is true of all languages that modular systems are better in maintenance than monolithically implemented ones, and inter program communication with fourth generation languages has been improved since the article was written. It has been suggested that programmers who have used third generation languages do not make good fourth generation language programmers because they have more problems with the defaults implicit in the language than programmers who have never used third generation languages.

The International Data Corporation [IDC84] say that software maintenance issues do not disappear with fourth generation languages. An interview with a fourth generation user cited the following maintenance issues:

- Fourth generation language programs over 40 to 50 lines (approximately equivalent to 500 to 800 lines of COBOL) were difficult to maintain. Programs over a few hundred lines would approach being beyond maintenance.
- The operating system transparencies which make development so easy tend to make maintenance more difficult.
- Cross database access (accessing databases other than those provided with the fourth generation language, such as IDMS, TOTAL, etc.) is also a maintenance problem.
- One traditional goal of maintenance, to increase run time efficiencies is not possible; streamlining fourth generation language programs did not have much effect on hardware usage.
- New product releases are a major maintenance issue, due to the fact that forward compatibility does not often exist.

Grindley [IDPM86] reported that some companies with experience of fourth generation languages found it economically sensible to consider rewriting their systems rather than maintaining and patching existing software.

There are several types of effect which this move to fourth generation languages can have on software maintenance:

- Simple hidden errors can be avoided, a fourth generation language can deal with certain aspects of the system automatically, for example it can determine the first and last records.
- Fourth generation languages make the understandability of a program clearer, and therefore easier for maintenance by a third person.
- Many fourth generation languages disallow ill-structured program constructs which can cause trouble later.
- Many fourth generation languages are linked to data management systems with built in data dictionaries. The programmer cannot misrepresent the data or fail to declare variables.
- Many fourth generation languages are self documenting. Poor documentation is likely to be a cause of maintenance difficulties with third generation languages.

The literature therefore suggests that fourth generation languages have advantages and disadvantages in software maintenance. The advantages are:

- Software maintenance is reduced with programs of 40 to 50 lines in length
- Simple errors can be avoided because the fourth generation language can deal with certain aspects of the system automatically
- Many fourth generation languages are self documenting.

The disadvantages are:

- Assumptions made by the fourth generation language which assist in development can cause problems in maintenance

- Streamlining the software to make it run better is not possible as this is a language constraint
- Forward compatibility does not often exist with fourth generation languages.

Summary: Software maintenance is consuming vast quantities of data processing resources which has meant that new software cannot be produced quickly enough. One solution to this problem has been the use of fourth generation languages which allow software to be developed more quickly than would otherwise be the case. This change has led to an increase in the amount of software to be maintained. Little research has been carried out in this area, and therefore it is not known whether the software developed with fourth generation languages consumes more or less software maintenance resources than similar software developed in a third generation language. Companies may, therefore, be investing in fourth generation languages for short term gains, whereby quicker development is paid for later by increased maintenance.

CHAPTER 3

Software Cost Estimation

Abstract: This chapter discusses the need for software cost estimation, and the difficulties involved. It also defines the resources which need to be estimated and discusses some of the methods for software cost estimation, and goes on to include evaluating software cost estimation models. A section is included to discuss cost estimation during the software maintenance phase. The chapter goes on to include some of the attributes which may have an effect on the amount of software maintenance required.

3.1 The Need For Software Cost Estimation.

The competitive nature of business means that it is necessary to make an accurate cost estimation of a software project. In order to conduct a successful development project, it is necessary to understand the scope of the work to be done, the resources required, the tasks to be accomplished, the milestones to be tracked, the effort and costs to be expended, and the schedule to be followed.

In the early days of computing, software costs represented a small percentage of the overall cost of a computer based system. A sizeable error in estimates of software cost had relatively little impact. Today, software is the most expensive element in many computer based systems, and a large cost estimation error can make the difference between profit and loss. Cost overruns can be disastrous for the developer.

As software estimation has to be done during the planning phase there are a large number of unknowns. This makes this task difficult and error-prone. The decision to go

ahead with a particular project may be based on these estimates, and it becomes more and more difficult to abandon a project once resources and costs have been allocated to it. Cost over-runs of 300% and more are not unusual in software engineering, particularly for tasks above the level of small systems.

The costs for the Department of Social Security Operational Strategy project rocketed from £713 million to £2,000 million [HILL90]. It was also reported that it was unsure whether the project would enable staff savings to be made. This was on a project that was supposed to pay for itself by cutting over 20,000 jobs.

There are three main areas of impact of this inability to reliably plan projects:

a) Economic - these are the most obvious on projects where the estimate is grossly inaccurate. In the case of an internal systems department developing projects for its own company, the late realisation that the project will not be completed anywhere near the budget can result in the project being cancelled, with the associated waste of all work done to date. A survey by Applied Research (New Jersey USA) across 125 American companies found that 75% of all development projects were abandoned before delivery [HEWETT87], and initial misunderstandings of the scope of the project are clearly one cause of failure. In the case of an outside contractor, underestimates will result in going back to the client in an effort to secure additional funds. If the contract is on a fixed price basis, the contractor will be saddled with the cost of the overrun.

b) Technical - when the budgeted end of the project draws near, but substantial additional work remains, the tendency is to for the final tasks to suffer in order to complete the project as soon as possible. Unfortunately, the last tasks are usually testing, documentation, and training. Therefore the result is that the system is less reliable and less well received by their ultimate users. While underestimation is not the only reason for these problems, it is a contributing factor.

c) Managerial - when an unrealistic deadline draws near additional pressures are brought to bear on the staff to complete the project in a hurry. Besides the likely short

term detrimental effect on the quality of the work produced, the long term effect on morale is also costly. Personnel are taken from other assignments in order to "save" the project in trouble, often resulting in a worse problem than the original one. Brooks law [BROOKS75] states "Men and women are interchangeable commodities only when a task can be partitioned among many workers with no communication between them. This is true of reaping wheat and picking cotton; it is not even approximately true . . . . . of programming" - adding people to a late software project makes it later. If this problem is pervasive, then a sort of "crisis mentality" can develop, where only projects of this type get any managerial attention. Also staff turnover can only increase.

Software costs and how they relate to various development, systems and environmental factors need to be estimated in order that the typical commercial calculations can be undertaken. In this way, the software element of a project is a typical component for which financial resources have to be allocated, and it needs to be assessed in normal commercial terms. Without a realistic estimate of costs, software developers are in no position to tell a manager or client that the budgets and schedules are impractical. They become "locked in" to an impossible time scale that can only end in disaster and in the worst case the project is out of control from the start.

Because of the problems of estimation some organisations use a series of cost estimates. A preliminary estimate is prepared during the planning phase and presented at the project feasibility review. An improved estimate is presented at the software requirements review, and the final estimate is presented at the preliminary design review. Each estimate is a refinement on the previous one, and is based on the additional information gained as a result of the additional work activities. Sometimes several product options and associated costs are presented at the reviews. This allows the customer to choose a cost effective answer from a range of possible solutions.

3.2 The Difficulty of Software Estimation

Software cost and effort estimation will never be an exact science. Too many variables, for example human, technical, environmental and political can affect the ultimate cost of software and the effort applied to develop it. However, software project estimation can be transformed into a series of systematic steps that provide estimates with an acceptable degree of risk. Boehm [BOEHM83] has said "Today a software cost estimation model is doing well if it can estimate software development costs within 20% of actual costs for 70% of the time, and on its own turf (that is within the class of projects to which it has been calibrated . . . . .). This is not as precise as we might like, but it is accurate enough to provide a good deal of help in software engineering economic analysis and decision making".

DeMarco [DEMARCO82] has outlined four reasons why software cost estimates are typically not accurate:

1. Developing an estimate is a complex task, requiring a significant amount of effort to do correctly. Unfortunately, a number of factors work against this. The first is that estimates are often done hurriedly, without an appreciation for the effort required to do a creditable job. In addition, it is too often the case that an estimate is needed before clear specifications of the system requirements have been produced. Therefore, a typical situation is an estimator being pressured to quickly write an estimate for a system that they do not fully understand.

2. The people developing the estimates generally do not have much experience at developing estimates, especially for large projects. Compounding this problem is the fact that few firms collect project data with which to check new estimates. Therefore, project managers often start by doing a bad job and never get any better.

The third and fourth problems are related and are:

3. An apparent human bias towards underestimation and

4. A management that asks for an estimate but really desires a goal. One problem is that an estimator is likely to consider how long a certain portion of the system would take, and then to merely extrapolate this estimate to the rest of the system, thereby ignoring the non-linear aspects of systems development including the overheads associated with co-ordinating a number of interconnected efforts. Another common underestimation problem is that the estimator, often a senior staff member, estimates the amount of time it would take them to do a task, forgetting the fact that probably large portions of the system will be written by relatively more junior staff who will require more time. These underestimates get compounded by the fact that management typically tends to want to reduce the estimate to some degree, in order to make the bid look more attractive or in order to "maximise productivity by reducing slack". Therefore, an estimate that was probably too low to begin with gets further reduced.

Estimating software maintenance costs for any particular system is very difficult. The difficulties arise because these costs are related to a number of technical factors, together with some relatively unpredictable factors which are unrelated to any technical characteristics of the system. Sommerville [SOMMERVILLE85] suggests that these include:

1. The application being supported. If the application of the program is clearly defined and well understood, the system requirements may be definitive, and perfective maintenance due to changing requirements minimised. If, on the other hand, the application is completely new, it is likely that the initial requirements will be modified as users gain experience with the system.

2. Staff stability. It is normally easier for the original writer of a program to understand and change a program rather than some other individual who must understand the program by study of its documentation and code listing. Therefore, if the programmer of a system also maintains that system, maintenance costs will be reduced. In practice, the nature of the programming profession is such that individuals change

jobs regularly and it is fairly unusual for one person to develop and maintain a program throughout its useful life.

3. The lifetime of the program. The useful life of a program obviously depends on its application. The program will become obsolete if the application becomes obsolete or if its original hardware is replaced and conversion costs exceed rewriting costs.

4. The dependence of the program on its external environment. If a program is highly dependent on its external environment it must be modified as that environment changes. For example, changes in a taxation system might require payroll, accounting and stock control programs to be modified.

5. Hardware stability. If a program is designed to operate on a particular hardware configuration and that configuration does not change during the programs lifetime, no maintenance costs due to hardware changes will be incurred. However, hardware developments are rapid and so the program may need modification to use new hardware. This process is distinct from moving the program to another computer system as the required modifications normally involve enhancing the program to make use of improved hardware or modifying assumptions built into the program about the hardware.

3.3 Methods of Software Cost Estimation

3.3.1 Development

There are many software estimation models, and some of these are discussed in the next sections.

a) Expert Judgement. A widely used cost estimation technique is expert judgement. This method relies on the experience, background, and a business sense of one or more key people in the organisation.

The biggest advantage of expert judgement, namely, experience, can be a liability. The expert may be confident that the project is similar to a previous one, but may have

overlooked some factors that make the new project significantly different. Or, the expert making the estimate may not have experience with a project similar to the present one. In order to compensate for these factors, groups of experts sometimes prepare a consensus estimate. This tends to minimise individual oversights and lack of familiarity with particular projects, and neutralises personal biases and the desire to win the contract through an overly optimistic estimate. The major disadvantage of group estimation is the effect that interpersonal group dynamics may have on individuals in the group. Group members may be less than candid due to political considerations, the presence of authority figures in the group, or the dominance of an overly assertive group member. The Delphi technique can be used to overcome these disadvantages.

b) Delphi Cost Estimation [HELMER66]. The Delphi technique was developed at the Rand Corporation in 1948 to gain expert consensus without introducing the adverse side effects of group meetings. The Delphi technique can be adapted to software cost estimation in the following manner.

1. A co-ordinator provides each estimator with the system definition document and a form for recording a cost estimate.

2. Estimators study the definition and complete their estimates anonymously. They may ask questions of the co-ordinator, but they do not discuss their estimates with one another.

3. The co-ordinator prepares and distributes a summary of the estimators' responses, and includes any unusual rationales noted by the estimators.

4. Estimators complete another estimate, again anonymously, using the results from the previous estimate. Estimators whose estimates differ sharply from the group may be asked, anonymously, to provide justification for their estimates.

5. The process is iterated for as many rounds as required. No group discussion is allowed during the process.

c) SLIM. SLIM [PUTNAM79] depends on a source lines of code estimate for the products general size, then modifies this through the use of the Rayleigh curve model to produce its effort estimates. The user can influence the shape of the curve through 2 key parameters: the initial slope of the curve (the Manpower Build-up Index {MBI}) and a productivity factor (the technology constant or productivity factor {PF}). An important and somewhat controversial feature of the model is its strict time/ effort trade-offs, where attempts at reducing SLIM's minimum time schedule are met with very large effort increases. The Rayleigh curve is an exponentially declining curve used to model a number of development processes. The curve plots effort on the vertical axis and time on the horizontal axis. The equation describes a build up followed by a slackening off for the software development cycle, where people are added where they become useful and then are transferred to other projects as the system is done, except for a decreasing number of maintenance staff.

Much of the estimation power of SLIM comes from its software equation:

$$S = cK^{1/3}t_d^{4/3}$$

Where S = source statements

c = a technology constant (productivity factor)

K = the life cycle effort

t_d = the time of peak manpower

The SLIM user has control over 2 key variables, the Manpower Build-up Index ($MBI = K/t_d^2$) and the Productivity Factor ($PF = c$) . The Manpower Build-up Index adjusts the slope of the initial part of the Rayleigh curve. The higher the value the steeper the curve and the faster the build up of staff on the project. This number establishes when t_d will be reached, and thus the "minimum" time in which the project can be completed. The larger the c value, the higher the productivity rate. The SLIM user can choose these values either by calibration of the model with data from completed

projects, or by answering a series of 22 questions from which SLIM will provide a recommended PF and MBI.

d) Function Point Analysis. One criticism of existing models is that they require a user to estimate the number of source lines of code in order to get person months and duration estimates. The function point measurement was developed by Albrecht [ALBRECHT79]. Function points are at a higher level than source lines of code, capturing information like the number of input transaction types and the number of unique reports. Albrecht believes function points offer several significant advantages over source lines of code counts. First it is possible to estimate them early in the life cycle, about the time of the requirements definition document. This can be an important advantage for anyone trying to estimate the level of effort to be required on a software development project. Secondly, they can be estimated by a relatively non-technical project member. Finally they avoid the effects of language and other implementation differences.

Function points can vary 35% from the original function counts. Once the function counts have been computed, they can be used to compare the proposed project with past projects in terms of its size. Through these comparisons an organisation can begin to develop cost estimates, first based upon analogies, and later, as additional data are collected, through statistical analysis.

e) COCOMO. The most popular software estimation model is the COnstructive COst MOdel (COCOMO), developed by Boehm [BOEHM81]. Based on his analysis of 63 software development projects, Boehm developed a model that predicts the effort and duration of a project, based on inputs relating to the size of the resulting systems and a number of "cost drivers" that Boehm believes affect productivity. COCOMO consists of a hierarchy of 3 models:

1. Basic COCOMO computes software development effort as a function of program size.

2. Intermediate COCOMO computes software development effort as a function of program size together with an effort adjustment factor derived from assessments of the product, hardware, personnel and project attributes (cost drivers).

3. Advanced COCOMO is the same as intermediate except that it has the impact of each step of the life cycle.(analysis, design, etc.)

The basic COCOMO equations take the form:

$E = a_b(KLOC)^{b_b}$

$D = c_b(E)^{d_b}$

Where

E is the effort applied in person months

D is the development time in chronological months

KLOC is the estimated number of delivered lines of code (in thousands) for the project

a_b, b_b, c_b,d_b are constants which depend on whether the organic, semidetached or embedded mode of COCOMO is being used.

The basic model can be extended to consider a set of cost driver attributes, which are factors affecting the software effort. Cost drivers are fully discussed later in this chapter.

Boehm produced a list of 15 attributes to be rated on a six-point scale ranging from "very low" to "extra high" (in importance or value). Based on the rating, an effort multiplier is determined from tables published by Boehm, and the product of all effort multipliers is an effort adjustment factor (EAF). Values for an effort adjustment factor range from 0.7 to 1.65. These factors are covered more fully in Section 3.6

The intermediate COCOMO model takes the form:

$$E = a_i (KLOC)^{b_i} * EAF$$

Where

E is the effort applied in person months

a_i, b_i are constants which depend on whether the organic, semidetached, or embedded mode of COCOMO is used.

KLOC is the estimated number of delivered lines of code (in thousands) for the project.

3.3.2 Maintenance

a) Lines of source code per programmer. A widely used estimator of personnel is the number of source lines that can be maintained by an individual programmer. Table 3.1 summarises various figures which have been published for the number of source lines of code (in thousands) an individual can maintain.

Table 3.1: Source instructions per maintenance programmer

Reference	Application Area	KSI / Full time software personnel
WOLVERTON80	Aerospace	8
FERENS79	Aerospace	10
DALY77	Real-time	10 - 30
GRIFFIN80	Real-time	12
ELLIOTT77	Business	20
GRAVER77	Business	20
LIENTZ80	Business	32
BOEHM81 25th Percentile	Numerous	10
BOEHM81 Median	Numerous	25
BOEHM81 75th Percentile	Numerous	36

An estimate of the number of full time software personnel needed for maintenance can be determined by dividing the estimated number of source instructions to be maintained by the estimated number of instructions that can be maintained by a maintenance programmer.

b) COCOMO. In a survey of 63 products in various application areas, Boehm [BOEHM81] developed a formula for estimating software maintenance costs. The estimation is calculated in terms of the Annual Change Traffic (ACT), defined as "The fraction of a software product's source instructions which undergo change during a (typical) year, either through addition or modification".

The ACT quantity is used, in conjunction with the actual or estimated development effort in person months, to derive the annual effort for software maintenance.

The COCOMO equation for estimating basic annual maintenance effort $(MM)_{AM}$, given the estimated development effort $(MM)_D$, is

$$(MM)_{AM} = (ACT)\ (MM)_D$$

Boehm uses effort adjustment factors to adjust the original calculations, these are largely the same as those for development, although some different effort multipliers are used. The annual maintenance effort is then calculated as:

$$(MM)_{AM} = (ACT)\ (MM)_{NOM}\ (EAF)_M$$

Where $(MM)_{NOM}$ is calculated from the nominal effort equations.

Boehm suggests that the maintenance effort can be estimated by the use of an activity ration, which is the number of source instructions to be added and modified in any given time period divided by the total number of instructions:

$$ACT = (DSI_{added} + DSI_{modified}) / DSI_{total}$$

c) Belady and Lehman's Model. Effort expended on maintenance may be divided into productive activities, including analysis and evaluation, design, modification, coding, etc. and "wheel spinning" activities which involves trying to understand what the code

does, trying to interpret data structure, interface characteristics, performance bounds. Belady and Lehman [BELADY72] devised the following expression as a model of maintenance effort:

$$M = p + K^{(c-d)}$$

Where

M = total effort expended on maintenance

p = productive effort (as described above)

K = an empirical constant

c = a measure of complexity that can be attributed to a lack of good design and documentation

d = a measure of the degree of familiarity with the software

This model indicates that effort (and cost) can increase exponentially if a poor software development approach was used, and the person or group that used the approach is not available to perform maintenance.

3.4 Model Evaluation

The most critical question for a manager interested in using a software estimation model is whether or not the estimates provided are sufficiently useful to justify using them. One evaluation standard is the degree to which the models estimated effort in work months (MM_e) matches the actual effort (MM_a). If the models were perfect, then for every project $MM_e = MM_a$.

Because the importance of the absolute difference between MM_e and MM_a varies with project size, for example a 6 person month error is likely to be more serious with a 10 person month project than a 1,000 person month one, a percentage error test has been recommended:

$$\frac{MM_e - MM_a}{MM_a}$$

However as errors can be of 2 types, underestimates, where $MM_e < MM_a$, and overestimates, where $MM_e > MM_a$, it makes it difficult to analyse a model's average performance over the entire set of projects. A magnitude of relative error, or MRE test is recommended as the 2 types of error do not cancel each other out when the average of multiple errors is taken.

$$\frac{|MM_e - MM_a|}{MM_a}$$

3.5 Factors Affecting Software Maintenance

In both development and maintenance there are many factors which influence the amount of effort required. Some of these cost drivers appear obvious for example a good programmer can achieve a specified task in less time than a bad one but other factors although less apparent could have as large an effect.

Various research has been carried out into factors which influence development, and these are summarised in Table 3.2.

Table 3.2 Cost estimation factors research

	a	b	c	d	e	f	g	h	i	j	k	S
DP Experience		✓	✓	✓	✓	✓				✓	✓	7
Application Experience			✓	✓	✓		✓	✓	✓	✓	✓	8
Software Experience				✓	✓	✓		✓				4
Hardware experience				✓		✓		✓				3
Capability								✓			✓	2
Education					✓						✓	2
In-house %						✓					✓	2

Table 3.2 Cost estimation factors research (continued)

	a	b	c	d	e	f	g	h	i	j	k	S
Part time %						✓					✓	2
Facility Experience					✓							1
Programmers Participation				✓								1
Age of Product					✓							1
Morale of Staff										✓		1
Scheduling Constraints						✓		✓			✓	3
Staff Load				✓		✓					✓	3
Travel									✓	✓		2
Communication		✓										1
Modern Programming Practices		✓	✓			✓		✓		✓	✓	6
Tools		✓				✓		✓		✓	✓	5
Language		✓				✓	✓			✓	✓	5
Response Time			✓			✓		✓			✓	4
Volatility			✓					✓			✓	3
Reusable Code										✓		1
Classified			✓									1
Distance	✓											1
Existing Documentation											✓	1
High Reliability								✓	✓	✓	✓	4
Required volatility		✓		✓					✓		✓	4
User Participation				✓					✓		✓	3
Number of User Organisations									✓		✓	2
User DP Knowledge									✓		✓	2

Table 3.2 Cost estimation factors research (continued)

	a	b	c	d	e	f	g	h	i	j	k	S
User Application Knowledge				✓							✓	2
Database Size								✓				1
Complexity of the Product								✓				1
Run Time Constraints								✓				1
Memory Constraints								✓				1

Table 3.2 shows a summary of the variables used in various research, each row represents a variable and each column the research. A summary column shows the number of researchers using the variable.

Where:

a = GAYLE71

b = SCOTT74

c = WOLVERTON74

d = WALSTON77

e = CHRYSLER78

f = PUTNAM78

g = ALBRECHT83

h = BOEHM81

i = RUBIN

j = JONES86

k = BANKER87

S = SUMMARY

The paper by Banker, Datar, and Kemerer [BANKER87] addresses software maintenance whereas the others address development.

From the table it is apparent that the experience of project team members, in certain areas is believed to be a critical element, the capability of the staff is often discussed but because of difficulty in measurement is rarely used.

In two areas of research the highest level of education, and amount of in-house versus outside contractor staffing were considered. There were also researchers who considered the percentage of part time workers, the amount of programmer participation, the age of the programming team and their morale.

Project management variables, including schedule constraints, staff loading, travel requirements and project communication are considered in some of the research. Project communication should be helped by the use of fourth generation languages as project teams tend to be smaller and often consist of only one or two people.

Two user variables have been considered in four of the previous papers, these are high reliability - the importance placed on avoiding system failures and required volatility - the degree to which the user-stated requirements changed over the course of the project. Other user variables which have also been considered are the degree of user participation in the project, the number of user organisations having sign off responsibility, the user's data processing knowledge, and the users application knowledge.

Technical environment variables are often included in productivity models and included in Table 3.2 are modern programming practices, the use of software tools, response time, the choice of the programming language, and the hardware/ software volatility - the amount of change in the underlying environment in which the application is being written. The use of reusable code, whether the work is classified, and in one (1971) study the distance to the machine room has also been considered. Banker, Datar and Kemerer[BANKER87] were considering the factors affecting software maintenance and therefore also included the quality of the existing documentation.

3.6 Fourth Generation Language Specific Factors.

The COCOMO model of software cost estimation is applicable to third generation languages, and the only allowance made for fourth generation languages is the use of software tools cost driver.

Besides the general factors mentioned in the previous section, certain factors which are specific to fourth generation languages or which are different with these tools are:

a) Attributes of the System.

(i) Type of application. The type of application affects the level of software maintenance as certain types of application are more prone to change than others. This attribute is specific to fourth generation languages because fourth generation languages tend to be application specific, whereas third generation languages were more general.

(ii) Original time scale for development. If the system was developed in an inadequate time the testing, standards and documentation are usually the first casualties, and this has an effect on software maintenance later. This is specific to fourth generation languages because the time scales allocated for development with these products are shorter. If the time is short the temptation is to develop the application using the fourth generation language even if it is not the type of system at which the fourth generation language is targeted.

(iii) Reliability of the underlying system. Generally fourth generation languages are associated with a database management system and errors can occur in the underlying software, rather than the user written application.

(iv) The degree of end user programming. This can effect the resources required by software maintenance.

b) Personnel Attributes.

(i) Programming language experience of the development and maintenance staff. The experience of the programming staff with the fourth generation language will effect the amount of software maintenance. This is included in the fourth generation language specific factors because fourth generation languages require less programming experience than third generation languages.

(ii) Training in the fourth generation language. Training in the use of fourth generation languages will reduce the resources required by software maintenance.

(iii) Third generation language experience. It has been reported that third generation language programmers do not make good fourth generation language programmers as they tend to make poor use of the procedural elements of the languages. Experience of third generation languages can, therefore, have a detrimental effect when using fourth generation languages.

c) Volatility.

(i) Changes in hardware. Changes in hardware may cause software changes. This is included as a fourth generation language specific factor because fourth generation languages are at least as machine specific as third generation languages.

d) Understandability of the source.

(i) Programming style, methods and standards used in development. In a survey by McClure [MCCLURE76] programmers reported that the standardisation of style introduced by structured programming conventions made programs easier to understand. IBM [BOEHM77] reported an average 40% productivity saving in real time, business application, and systems applications software products employing structured techniques. Other organisations [LYONS81] reported that maintenance costs for software development techniques are reduced by a ratio of 3:1 compared to maintenance costs for unstructured software. Error rates in tested unstructured software

averaged 1 error per 200 lines of source code, but in many structured software systems, production error rates are averaging less than 1 error per 1,000 lines of source code [MCCLURE78]. These structured techniques were designed primarily for use in the development of new software systems written in third generation languages, and as such their benefits may be lost when moving to third generation languages. The use of methods also applies to software maintenance, as the structure of the software will rapidly deteriorate if the structure is not maintained during this phase of the software life cycle.

(ii) Complexity of the source code. The first task when performing software maintenance is to understand the code before changes can be made to it. Some source code is by the nature of it is more difficult to understand than other code. Fourth generation language vendors claim that their products are easier to understand than their third generation equivalents. Chrysler [CHRYSLER78] provides evidence that increases in complexity in applications software go hand in hand with increases in the cost of maintenance and declines in the morale and productivity of programmers and analysts.

(iii) Type of fourth generation language. The fourth generation language used will greatly effect the understandability of the source, and some fourth generation languages are more applicable to certain applications than others, as described in section 2.6.

e) Management

(i) Management of the project. If a project contains a high degree of end user programming, it may be more difficult to control than if just computer professionals are involved.

f) Use of tools

(i) Prototyping. The use of prototyping can give a user a more accurate portrayal of the system they required, and therefore reduced software maintenance. If the

system is not rewritten before being used in production then maintenance may be increased.

(ii) Use of a data dictionary. A data dictionary, which is included with many fourth generation languages can reduce greatly the amount of software maintenance. A change in a record layout would just require a change to the dictionary and recompilation of the programs, rather than a change to every individual program.

(iii) Types of tools used. Whether tools were used in development or during maintenance. Software tools cover a variety of tasks, including cross referencers, test data generators and interactive debuggers can all reduce software maintenance to varying degrees.

Summary: Software is the most expensive element in many computer based systems, and as such, estimation of the costs involved are essential if resources are not to be wasted. Software estimation allows resources to be scheduled and made available when required. Although software cost estimation is an essential part of running a computer project it is a difficult and inexact task. There are a large number of software cost estimation techniques in use and most include cost drivers to adjust the effort as the size of the system is not the only factor which influences the cost. Various research has been conducted into these cost drivers, but these are mainly confined to software development using third generation languages.

CHAPTER 4

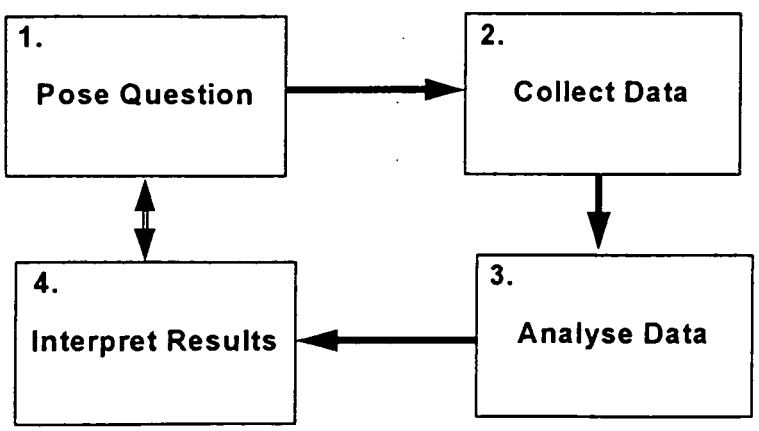
Data Modelling

Abstract: This chapter discusses data modelling and data collection methods, and describes the disadvantage of each of the methods. It also includes a discussion of statistics in general, and describes in detail some of the statistical methods which were considered for use in analysing the data. A discussion is included on curve fitting and methods of determining the fit of a particular curve. A section is included on the curve fitting software which was used in this research.

4.1 Introduction

There are 4 major stages to data modelling, and these are shown diagramatically in Figure 4.1.

Figure 4.1: The 4 stages of data modelling



4.1.1 Pose the question

The first stage is to decide exactly what question it is that the research is attempting to answer.

4.1.2 Collect data.

Having arrived at a specific question the next step is to find some data which can be analysed to enable the question to be answered.

Data can be collected in various ways, the four main ones being:

- Observation. Observing the environment of interest. This has the advantage that all the observations can be carried out by 1 person, and therefore to a particular standard, which means that different peoples opinions do not play a part, but has the disadvantage that only a limited amount of data can be collected in this way.
- Experimentation. This involves changing one or more factor and observing the effects of the change or changes. The advantage of this method is that external factors which are not part of the research can be excluded, but the disadvantage is that actually measuring something may effect it, this is known as the Hawthorn effect.
- Survey. Although the area of interest is the target population, (in this case every computer installation using fourth generation languages) it is not usually possible to collect data about the whole population. Instead a sample of the population is taken which is hoped will be representative of the population as a whole. This relies on some of the target population completing a series of questions. The advantage of surveys is that more information can be obtained in this way than by other means. The disadvantages are:
    - It relies on other peoples observations, and it is therefore difficult to set standards

- Often surveys are not completed
- The questions should be worded so that they are not leading the person completing the questionnaire towards a particular answer
- The way that the questions are worded can be different from that which was intended.

• Collection as a background task. Data is collected automatically as a background task to the ordinary activities within the project. A machine tool control system collects information about what is going on as it is being used. The advantage of this method is that it is automatic, the disadvantage being that if the system is not seen to be useful it may not be used.

Data collection has a cost in time and effort and perhaps in systems to support the collection, this cost has to be weighed against the benefits. A general requirement is therefore to minimise the costs of data collection.

Data collection methods are discussed more fully in Section 5.1.

4.1.3 Analyse the data

Using statistical techniques an analysis of the collected data is conducted to enable the answer posed to be answered, the techniques available are covered in more detail in section 4.4.

4.1.4 Interpret the results

Decide whether the results obtained satisfactorily answer the question posed, if not it may be necessary to refine the question and repeat all 4 stages again.

4.2 Population Distribution

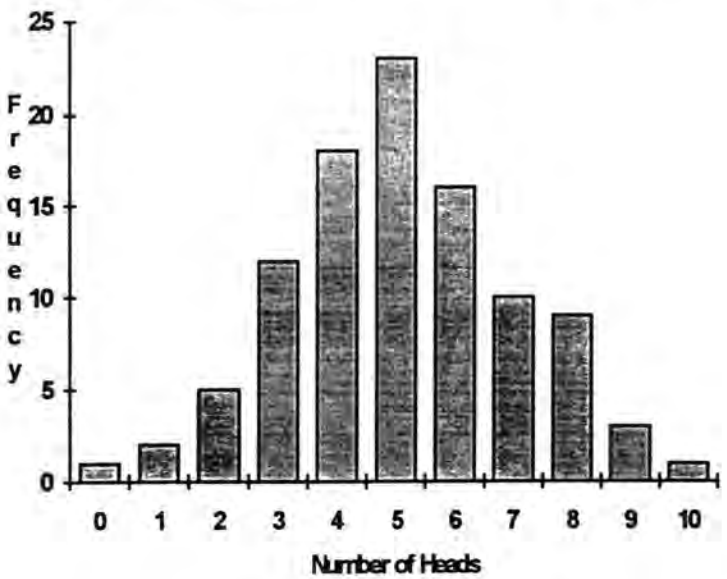
For any given statistical situation there are a number of possible outcomes, for example when tossing a coin there are 2 possible outcomes, heads and tails and each has a equal possibility. If 10 coins are tossed , the result would be expected to be 5 heads and 5 tails but on any particular experiment 6 heads and 4 tails, 3 heads and 7 tails, etc. may be achieved. If the experiment was carried out 100 times the results shown in Table 4.1 may be achieved.

Table 4.1 Results of 100 tosses of 10 coins

Number of Heads	0	1	2	3	4	5	6	7	8	9	10
Frequency	1	2	5	12	18	23	16	10	9	3	1

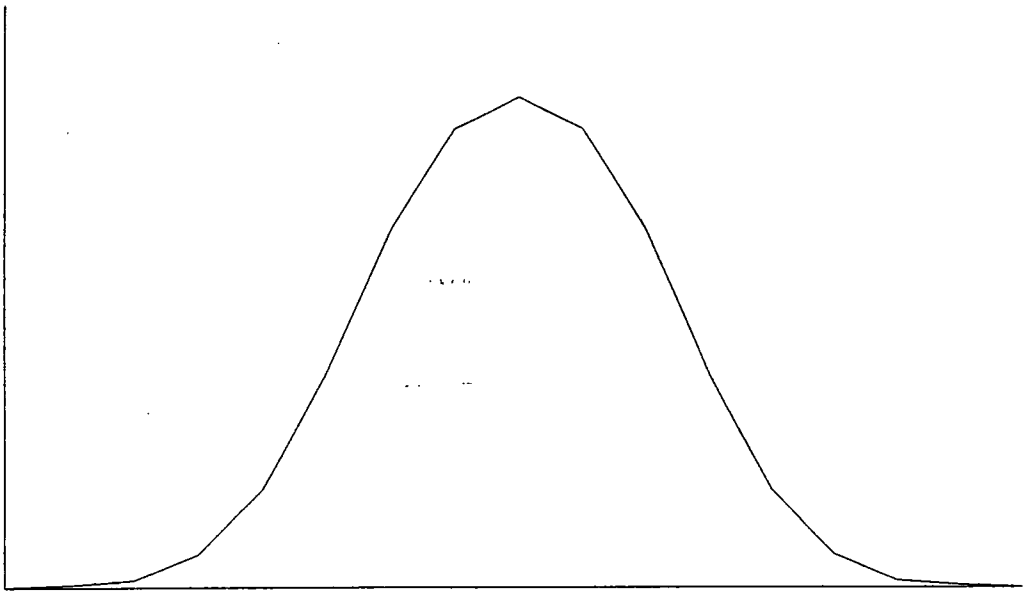
This is shown as a diagramatically in figure 4.2.

Figure 4.2 Histogram of 100 tosses of 10 coins



If this was done an increased number of times the results would have been a symmetrical bell shaped curve, such as figure 4.3.

Figure 4.3 Bell shaped curve obtained by increasing the number of results.



If this experiment was repeated using a continuous variable such as height, mass or time it would produce a similar shape although much more smooth.

A continuous random variable x having a probability density function where

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

is called a normal distribution curve.

The normal distribution is symmetrical about the mean, and over 95% of the distribution occurs within 2 standard deviations of the mean either side. The 5% most extreme outcomes of a statistical analysis are called the critical region. It includes the outcome

0[+], 1[+], 2[+] . . . . . C[+], C[-] . . . . . 2[-], 1[-], 0[-]

Where the number C is such that the combined probabilities of the outcomes add up to 1/20, i.e. approximately 2 standard deviations. The number C being the critical value at the 5% significance level.

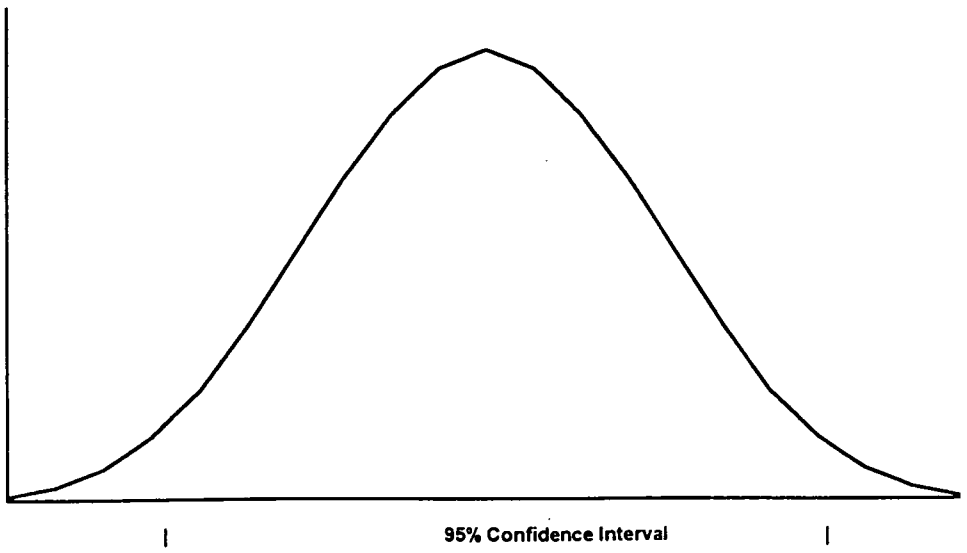
4.3 Statistics

There are various statistical techniques which can be used to analyse results and the one used in this research takes the form of a hypothesis test. A null and alternate hypothesis about the population is proposed, and the sample data is then examined to see how it fits in with this hypothesis and inferred back to the population accordingly. If the sample data, which is taken as random, is sufficiently extreme then the original hypothesis is rejected in favour of the alternate hypothesis, otherwise the conclusion is that on the evidence of the sample data, the original hypothesis appears to be reasonable.

To test if the sample data is sufficiently extreme a test statistic is calculated, this depends on the form of the sample data and the hypothesis test to be used.

In a hypothesis test the probability distribution of all the possible values that the test statistic could take if the null hypothesis is true, then the 5% rejection rule is used to decide whether or not to reject the null hypothesis at the 5% significance level. That is, the null hypothesis is rejected in favour of the alternate hypothesis if the value of the test statistic obtained from the sample is outside the 95% confidence interval, although this includes the highest and lowest 2½%. Figure 4.4 shows this diagramatically.

Figure 4.4 Normal Distribution Curve.



Therefore there is a 1 in 20 chance of a hypothesis being rejected when in fact it should be accepted.

With each of the statistical techniques a test statistic is calculated and compared to a table of values, calculated from this 5% significance level.

In a survey using a random sample and inferring from data about this sample back to the population the results may contain 2 types of error:

- Sampling error: the natural variation between individuals and the random method of their selection can produce samples which are not representative of the population. Data from an unrepresentative sample will give inaccurate information about the population.
- Bias: this is the name given to all other errors and will consist of various types:
  - There is a possibility that a true null hypothesis has been rejected, because at the 5% significance level there is a 5% chance of rejecting the null hypothesis when it is true.
  - A null hypothesis may not have been rejected even though it was incorrect, this type of error could be reduced by increasing the significance level but this would increase the probability of the first type of bias error.
  - If the sample size is very small then it is quite possible that there is insufficient evidence to reject the null hypothesis even when it should be. Generally, if not influenced by other factors, a large sample size gives more accurate results, but also leads to higher costs.

In this survey a sampling error could have occurred because medium to large business organisations were targeted, although replies received showed organisations

ranging in size from 34 to 80,000 employees, and Data Processing departments from 1 to 900 employees.

When considering software maintenance cost factors, comparisons are made at the level of lines of source code per person month, this relies on an accurate estimation of the system size and amount of time allocated to software maintenance, if this data is not known, or is inaccurate, then the replies to the other questions will be invalid.

The accuracy level of the reply needs to be considered as often for political reasons the person completing a survey replies in a particular way, they may have been responsible for the purchase of a particular tool and therefore reply that it was more useful than it really was.

The wording of questions needs to be carefully considered so that the respondent understands the question being asked and is not led to a particular answer.

Certain factors may be connected, and therefore when the results are analysed the independence of factors may need to be considered.

If a sampling method is used the target population also has to be considered to ensure that the whole population is being sampled, and not an atypical subset.

Errors would also have occurred because of:

- the person completing the survey could be either too hard or too soft on the organisation.
- the person completing the survey could have misunderstood the question.
- organisations have different levels of user support, as often maintenance and support are classed together.
- Because the 5% significance level is used there is a 5% chance that the statistics produce a result that the factor does not affect software maintenance when in fact it does.

The statistical techniques considered for use in the analysis of the surveys obtained in this research are described more fully in Section 4.4.

4.4 Statistical Techniques

The actual test to be chosen depends on a number of factors including:

- the type of data
  - Categorical data which is based on mutually exclusive categories i.e. Labour, Conservative, Liberal Democrat or Other voters.
  - Ordinal data which contains more quantitative information than categorical data i.e. rating a brand of soap powder on a scale 1 to 10.
  - Interval data which is actual measurements i.e. length in metres, etc.
- the number of samples
- the size of the samples
- if there are 2 samples, whether the data is paired i.e. the sample selected is matched in pairs so that each pair possesses the same characteristics (e.g. sex, age, etc).
- if the sample data can be assumed to have a normal distribution, in which case a parametric test should be used i.e. t-test. If a normal distribution cannot be assumed a non-parametric test should be used i.e. Krustal-Wallis H test.

A number of tests can be used and the main ones considered for this research are described in the following sections.

4.4.1 t-test

The t-test (students t-test) is a test which can be used if the data is:

- The data is interval is based on actual quantities in definite units, in this research lines of code.
- The population distribution can be assumed to be normally distributed.
- The data should not be paired in any way, i.e. not matched by systems of similar characteristics.
- The standard deviations of the populations can be assumed to be approximately equal.
- The sample sizes are small.

The test statistic is given by:

$$t = m_x - m_y / \sqrt{((Sc^2/n_x) + (Sc^2/n_y))}$$

The test statistic for the t-test is given by:

$$t = m_x - m_y / \sqrt{((Sc^2/n_x) + (Sc^2/n_y))}$$

where:

$$Sc^2 = \Sigma(x - m_x)^2 + \Sigma(y - m_y)^2 / ((n_x - 1) + (n_y - 1))$$

m_x = mean of sample x (The arithmetic mean is a measure of the level of a batch; it is given by the sum of all the data values in the batch, divided by the batch size and is often called average).

m_y = mean of sample y

n_x = number of samples in x

n_y = number of samples in y

The degrees of freedom is then given by:

$$(n_x - 1) + (n_y - 1)$$

The test statistic is then compared against a table of critical values (t_c) (the values for the t-test are shown at Appendix A) at the 5% significance level. If the test statistic is greater than or equal to t_c or less than or equal to $-t_c$ then the hypothesis that the 2 values are equal is rejected in favour of the hypothesis that the 2 values are different.

4.4.2 Analysis of Variance

Analysis of variance is a statistical technique which may be used for making many simultaneous comparisons , and it makes it possible to compare 3 or more samples without comparing each of the values using the t-test. At the 95% significance level each test has a 1 in 20 chance of the null hypothesis being rejected when in fact it is true, and therefore a large number of tests is inadvisable.

The first step in the analysis of variance is to make two estimates of the variance of the hypothesised common population: the within samples variance estimate, and the between samples variance estimate.

The within samples variance estimate is calculated according to the following equation:

$$\sigma w^2 = (\Sigma\Sigma(X-\bar{X})^2)/(N-k)$$

The between samples variance estimate can now be calculated as:

$$\sigma_b^2 = (\Sigma n\Sigma(\bar{X}-\bar{X}_G)^2)/(k-1)$$

Where:

k = total number of samples

N = total number of data elements

$\bar{X}G$ = Grand Mean

A test statistic known as the F ratio is now calculated as follows:

F = between samples variance estimate / within samples variance estimate

The value of F is compared to tables of critical values based on the degrees of freedom, the values for the 5% significance level are shown in Appendix B. If the test statistic is greater than or equal to t_c or less than or equal to $-t_c$ then the hypothesis that the 2 samples are equal is rejected in favour of the hypothesis that the 2 are different.

4.4.3 Krustal-Wallis H Test

The Krustal-Wallis H test is a test for deciding whether there is a significant difference between three or more samples, and is a useful alternative to the analysis of variance since it is a non-parametric test and therefore does not rely on assumptions about the distribution of the variable.

In order to apply the H test, the data must be ranked, from lowest to highest, with identical values being given the mean of the value they would otherwise have received, e.g. if 2 values are equal and ranked 9 and 10 they would both be given the value 9.5.

The sums of the ranks are then found for each sample. This information can now be used to calculate H from the following equation:

$$H = (12/N(N + 1)\Sigma(R^2/N)-3(N + 1)$$

Where:

R = Sum of the rankings for each sample.

N = Total number of data elements

The Krustal-Wallis H test is not used in this research as a normal distribution curve can be assumed as the factor being analysed was lines of code per person month.

4.5 Curve Fitting

The process of forming an equation to fit (satisfy) given data is called curve fitting, for example given the points in Table 4.2.

Table 4.2 Set of points

x	y
0	0
1	1

There are infinitely many equations that will satisfy a specified criteria. Examples of criteria are that the equation goes through each point, or that the sum of the difference between the squares is minimised.

In this research a computer program which computes the line of best fit was used. The software examines the points for 25 equations and computes equation coefficients, correlation coefficients, and best fit.

Included in the 25 fitted equations are:

- Y = A + B*X - Straight line.
- Y = A + B*X = C/X - Combined linear and reciprocal.
- Y = A + B/X + C/X*X - First and second order hyperbola.

- Y = A + B*X + C*X*X - Parabola.
- Y = A*X^B - Power.
- Y = A*B^(1/X) - Root.
- Y = A*X^(B*X) - Super geometric.
- Y = A*e^(B*X) - Exponential.
- Y = A + B*ln(X) - Logarithmic.
- Y = A*B^X*X^C - Hoerl's equation.
- Y = A*e^(((X-B)^2/)/C) Normal distribution (Gaussian).
- Y = A*e^((ln(X)-B)^2/C) - Logarithmic normal distribution.

There a several ways to determine the line of best fit including:

a) Inspection. Examining the pattern of dots on a scatter diagram and drawing the line which seems to fit the pattern.

b) The arithmetic means method. This method uses groups of data to find the arithmetic mean of parts of the scatter, and finds the line of best fit by joining up these partial means. Since the minimum number of means required to draw a straight line is three points it is sometimes called the three-point method.

c) The method of least squares. The least squared method utilises the fact that the equation of a straight line is always in the form:

$$y = a + bx$$

where x and y are dependent and independent variables.

The least squares method provides the equation of the straight line that is the best fit to the points in a scatter diagram. In trying to find the best fit to the data the points should be scattered on either side of any line drawn. The best line will be the one which minimises the deviations of the scattered points from the line. As some of the points are above the line and have positive values, whilst others are below the line and have negative values, it is necessary to square the deviations to eliminate the minus signs. The deviations are measured as vertical distances from the line.

The equation can be found with the formulae:

$$a = \bar{y} - b\bar{x}$$

where:

a is the intercept on the axis

$\bar{y}$ and $\bar{x}$ are the arithmetic means of the data series

b is the slope of the line

and

$$b = \frac{n \Sigma xy - \Sigma x \Sigma y}{n \Sigma x^2 - (\Sigma x)^2}$$

where:

b is the slope of the line

n is the number of data pairs.

d) The coefficient of correlation. If there is a linear correlation between two sets of data it will either be positive or negative correlation. If the observed values increase together or decline together there is positive correlation. If one increases as the other decreases there is a negative correlation. The correlation coefficient (r) is calculated to give a value from -1 to 1, and if there is perfect positive correlation between the 2 sets of data r = +1. If there is no correlation between the 2 sets of data r = 0, while if there is perfect negative correlation, the correlation coefficient is -1.

The formula to calculate the correlation coefficient is:

$$r = (\Sigma (x - \bar{x}) (y - \bar{y}) / n) / ([\sqrt{\Sigma} (x - \bar{x})^2 / n] [\sqrt{\Sigma} (y - \bar{y})^2 / n])$$

where:

$\bar{y}$ and $\bar{x}$ are the arithmetic means of the data series

n is the number of pairs of data

This formula can be written as:

$$r = \frac{\text{covariance of x and y}}{\text{standard deviation of x multiplied by standard deviation of y}}$$

Summary: This chapter describes the 4 stages of data modelling:

- *Pose the question*
- *Collect data*
- *Analyse the data*
- *Interpret the results*

A basic discussion of population distribution and statistics was included with the methods which were considered in the analysis of data. Curve fitting was also discussed in this chapter together with some techniques which can be used to check for the fit of the curve.

CHAPTER 5

Survey

Abstract: This chapter discusses the factors which were being tested, the design of the questionnaire, the reasoning behind the questions and the information they were trying to obtain. The general results which show the parameters of the research in terms installation type and size are included in summary form in this chapter. A discussion on the normalisation of the data into types of fourth generation languages is included and the survey data from this research is normalised. A summary of the results obtained from the research is also included.

5.1 Survey Design

In Chapter 3 the need for software cost estimation was discussed. Section 3.3 stated that the estimation of software costs cannot be an exact science as there are a large number of factors which are not known. These unknown factors mean that the best way to estimate costs for a particular project is that information from as many similar projects as possible should be considered and averaged to allow for any special attributes of that project, which are not being considered.

The first stage was therefore to define the questions which the research was trying to answer. As the question this research was trying to answer was which factors influence the amount of software maintenance when using fourth generation languages, it was not possible to test this directly. It was necessary to produce a list of factors which were considered may influence software maintenance and to pose a question for each of these.

By refining the factors introduced in Sections 3.5 and 3.6 a list of factors was produced and these are:

- Attributes of the system
  - The number of errors in the software
  - The age of the product
  - The required reliability
  - The complexity of the product
  - The type of application
  - The size of the system
  - The original time scale for development
    - * elapsed time
    - * person months
  - The reliability of the underlying system
  - The number of user written programs in the system

- Personnel
  - Staff ability
    - * development staff at time system was written
    - * maintenance staff
  - Morale and motivation of staff
  - Experience of staff in 4 main areas:
    - * The fourth generation language
    - * The computer environment
    - * The type of application
    - * Any third generation language
  - Training of software maintenance staff in 3 key areas:
    - * Software maintenance

        * The fourth generation language
        * Any third generation language
    - Change of staff
    - Organisation of software maintenance
- Volatility
    - Built in flexibility
    - Number of requests for enhancements
    - Hardware changes
- Understandability of the source
    - Documentation
    - Programming style
    - Use of methodologies
    - Use of site standards
    - Use of meaningful names in the software
    - Complexity of the source code
    - The type of fourth generation languages used
- Management issues
    - Management attitude to software maintenance
    - The policy to recruit software maintenance staff
    - The resources allocated to software maintenance
    - The workload of the software maintenance staff
    - The time allowed for individual changes
- Use of tools
    - The use of tools in development and maintenance
    - The use of prototyping
    - The use of a data dictionary

Data is necessary if an environment is to be understood and controlled. It was then necessary to collect data which could be analysed to enable a judgement to be made on the effect of these factors. There were two of the four ways in which the necessary data could be collected - experimentation or survey. Experimentation was ruled out because it would have been almost impossible to develop multiple systems which vary only in the maximisation or minimisation of one factor. To fully consider the impact of these factors on software maintenance the systems would then need to be kept running for years to enable them to be considered over the whole software life cycle. Using experimentation it would not have been possible to consider more than one or two factors, because of these difficulties.

It was decided that the crucial element in this research was to obtain information from as many sources as possible and because of this it was decided that the best method would be by the use of a questionnaire.

The best method of completing the survey would have been through personal contact with one person conducting an interview and completing a form. This, however, had the disadvantage that again, this would reduce the number of responses which could be obtained over a postal survey. To maximise the responses the surveys were despatched by mail, targeted at:

- Medium to large companies, as these are the ones more likely to use fourth generation languages
- Business applications, because these are the main targets for fourth generation languages

The disadvantage of postal surveys is that they can have a limited success as often the person receiving them, does not complete them, and a completion rate of less than 10% is not uncommon.

After the questionnaire was produced a few contacts were asked to complete the survey and were questioned and the questionnaire was refined in response to their

replies. They were then asked to complete the survey again to ensure that the information they were providing was that which the question was trying to obtain. The final survey is shown in Appendix C. After this process 235 surveys were despatched in 2 batches with the results being analysed after receipt of the first batch to enable any necessary changes to be included before the second batch was sent out, although no changes were actually made.

The survey consisted of two parts, the first being general questions about the company and its organisation, together with questions concerning the position of the person completing the questionnaire, the second being specific questions about one fourth generation system which was being maintained.

For the majority of questions the respondent was asked for their rating on a scale 1 to 5. This was convenient for three reasons:

- If numeric information was required exact figures may not be at hand, for example the number of errors in the software.
- Currently software maintenance resources are likely to be allocated by expert judgement, and it is likely that the view of the software is likely to affect this judgement.
- Once the factors are identified and a weight attached to them, it is likely that the person estimating the resources will be providing the information based on their judgement. If the research gathers the information in this way then it is as close as possible to its final use.

As the final aim of the research is to provide a weight for a range of values it was decided that for as many questions as possible a rating of 1 (low values) to 5 (high values) would be used. Again, this is because it is as close as possible to the way the factors are likely to be used in the final method.

In some cases the questions were left open ended, for example question 43 "What tools are used in software maintenance?" because it was thought that at a later stage different categories of tools could be examined separately, if it was thought necessary.

Due to the length of the original survey no questions were included to cross-check previous answers, as it was thought that the length of the survey may deter people from completing them.

The questions asked fall into the following categories:

- Background
- Attributes of the system
- Personnel
- Volatility
- Understandability of the source
- Management issues
- Use of tools

a) Background.

These questions were aimed at information which will enable the results to be interpreted within the context of the research. The questions which fell into this category were:

- The category to which the department and the organisation served by it belong.
- The size of the DP department and the company.
- Whether the person completing the survey believes that fourth generation languages have reduced software maintenance over similar systems developed without them. This is a subjective opinion, and may not provide an actual answer as the users expectations may increase with the use of fourth generation languages and therefore make more requests for change. The person completing the survey may therefore

say that fourth generation languages have not reduced software maintenance, when in fact, what has happened is that since the introduction of these tools the service provided by the data processing department may have improved.

b) Attributes of the System.

Nine attributes of the system were considered and examined in the survey. They were:

- The number of errors in the software. The information that was actually required was the number of errors in the software which were discovered in the last 12 months. As it was not possible to examine this factor directly, it was therefore decided that the best question to ask was how reliable the software had been in the past. Questions asked for the reliability of the system over the last 12 months and the historic reliability rated on a scale 1 (very unreliable) to 5 (very reliable). It also allowed for the choice of 6 meaning a new system, in this case asking for a rating of 1 to 5 for the expected reliability. The expected reliability was a difficult question to answer for a new system but often a person who was involved in the production of a system has some idea of how reliable it is likely to be. This is based on their knowledge of how much testing has been carried out etc. It is also possible that some assumption of reliability has been assumed when allocating resources for software maintenance over the next 12 months. This question also asks about the history of the product, as often if a product has been unreliable in the past there is a reluctance to make changes, or is tested more thoroughly.
- The age of the product. A question asked how long the software has been in operation.

- The required reliability. The survey asked how serious the consequences of failure are, 5 categories were given for this which were:

  i) No real problem

  ii) Problems

  iii) Serious problems

  iv) Financial disaster

  v) Life critical

- The complexity of the product. The questionnaire asked for the complexity of the tasks undertaken by the product to be rated from 1 (very easy) to 5 (very complex).
- The type of application. Three questions asked for this information, the main one being simply what type of application is it? for example financial. The name of the system and a brief description of what the software does, were also asked as this would allow the classifications to be broken down further if it was thought to be necessary.
- The size of the system. This is the major factor which was assumed to influence the amount of software maintenance, and this will be used as a baseline against which all the other factors will be compared. Various measures of system size were asked for to try and include the maximum number of systems, including:

  - Total lines of source code
  - Total lines of executable code (excluding comments and declarations)
  - Number of entity relationships or similar (depending on the Computer Aided Software Engineering tool used)
  - The number of function points
  - The number of lines of job control

- The number of entries in a data dictionary.

These measures of system size were asked for both the system as a whole and for the largest program to show if the system consisted of one main program and very little else, as this may have an effect on the amount of software maintenance.

- Original time scale for development. The questionnaire asked for three pieces of information on the original time scale for development:
  - Time scale in person months/ years
  - Elapse time in months/ years
  - The number of staff employed on the project.

  Armed with this information and the system size it should be possible to determine whether the original time scale was too short. If this was the case the testing and implementation phases are likely to be compressed with an increase in software maintenance.
- The reliability of the underlying system. The survey asked how reliable the fourth generation language is, this is because if failures with the system are occurring this is likely to use software maintenance resources.
- The number of user written programs in the system. The survey asked how many user written programs there are in the system.

c) Personnel.

Various personnel factors were considered in the survey. These were:

- Staff ability. One question addressed the ability level of the staff for each of development and maintenance. The average (mean)ability level of the staff was asked to be rated on a scale 1 (very poor) to 5 (very good). The question for development stated that the rating should be assessed for the time the system was written, and this question allowed

a rating of 6 (not known), as the system may have been written years ago and the information no longer available.

- Morale and motivation of staff. A rating of software maintenance staff on a scale 1 (very bad) to 5 (very good) was requested. An indirect measure, for instance, the level of sick leave, staff turnover rates etc. was considered but as this may be affected by other factors, therefore a direct question was asked.
- Experience of staff. The survey examined the experience of both maintenance and development staff in 4 areas:
    - The fourth generation language
    - The computer environment
    - The type of application
    - Any third generation language

  The average (mean) experience of the staff was classified into one of three categories:
    - Less than 6 months
    - 6 to 12 months
    - 1 to 2 years
    - 2 to 3 years
    - More than 3 years.

  Whilst the bands are not exactly equal, it was felt that they reflect stages of experience, equating to trainee, beginner, average, good and expert.
- Training of software maintenance staff. Training in 3 key areas was examined in the survey:
    - Software maintenance
    - The fourth generation language
    - Any third generation language.

The areas were rated under the following categories:

- None
- Insufficient
- Adequate
- More than adequate.

- Change of staff. The questionnaire asked if the staff who developed the system are the same people who are responsible for its maintenance. The 3 possibilities were:
  - None
  - Some of them
  - All of them.
- Organisation of software maintenance. The survey covered the organisation of the software maintenance teams within the company. The question asked whether the teams were:
  - Separate development and maintenance teams
  - Mixed maintenance and development
  - Both policies

  If either the second or third options were answered the survey asked what percentage of time was spent on software maintenance.
- Staff workload. The survey asked for a rating of 1 to 5 ranging from vastly underworked to vastly overworked.
- Time allowed for individual changes. The survey asked for a 5 point rating of whether the time allowed for individual changes were very inadequate to very generous.

d) Volatility.

Three volatility factors were considered in the survey:

- Built in flexibility. A rating, again on a scale of 1 (none) to 5 (lots) of the flexibility for future changes was requested in the survey.
- Number of requests for enhancements. The survey asked for the number of changes to the system in the last 12 months. Again this was rated on a scale 1 (none) to 5 (lots).
- Hardware changes. The survey asked whether any hardware changes were expected within the next 12 months, and if so were they likely to require changes to the software.

e) Understandability of the source.

Factors which influence the understandability of the source were included in the survey:

- Documentation. The rating of the standard of the documentation was requested, again using the scale 1 (non-existent) to 5 (excellent).
- Programming style, methodologies, and site standards in documentation and maintenance. The questionnaire asked if any of the following were used:
  - Meaningful names in the software
  - Comments in the source code
  - Site standards
  - A measure of complexity
  - A methodology in development
  - A methodology in maintenance.

  Often companies use these methods but not in their purest form, therefore with all these questions a middle ground was allowed besides the definite Yes and No answers.

The survey also asked for a classification of an overall understandability of the source code on a scale from 1 (very muddled) to 5 (very clear).

- Complexity of the source code. The survey also included a question about the complexity of the code itself, rather than the tasks undertaken by it. Again the scale 1 (very easy) to 5 (very complex) was used.
- The type of fourth generation language used. The questionnaire asked which fourth generation was used, and this could be used to define the category of the language.

f) Management issues.

Management issues were considered in the survey:

- The management attitude to software maintenance. A question was included in the survey asking how management see software maintenance in relation to development, 5 categories of answers which were:
    - Much less important than development
    - Less important than development
    - About the same
    - More important than development
    - Much more important than development.

  As with some of the previous questions it may be necessary to consider the role within the organisation of the person completing the survey.
- The policy to recruit software maintenance staff. A question was included asking what the strategy for the recruitment of software maintenance staff was employed within the organisation. The following categories were used:
    - Staff are recruited specifically for maintenance
    - Trainees are used and then moved to development

- Other, which the respondent was asked to specify.

- The resources allocated to software maintenance. The survey asked for the resources which have been allocated to software maintenance within the previous, current and next 12 months. A question was also included as to whether the time allocated to individual changes was sufficient, again using the scale 1 (very inadequate) to 5 (very generous). As this is only part of the picture a question also asked about the workload of the staff involved in software maintenance, again on the scale 1 (vastly underworked) to 5 (vastly overworked). The survey also asked a question which of the following did the person completing the survey include in software maintenance. The following list was provided:

  - Emergency repairs
  - Changes to correctly reflect the specifications or correctly utilise system resources
  - Upgrades to adapt to changes in processing requirements
  - Amendments to adapt to changes in regulations
  - Growth amendments performed to adapt to changes in data requirements, or to the addition of new programs, new users etc.
  - Enhancements for changes requirements
  - Support for users of the system
  - Changes which take less than 1 day
  - Changes which take less than 3 days
  - Changes which take less than 5 days
  - Changes which take less than 10 days
  - Changes which take less than 20 days

  This question could be used to determine the tasks expected from the software maintenance resources.

g) The Use of tools

Three factors related to the use of tools were included in the survey:

- The use of tools in development and maintenance. Three questions were included concerning the use of tools, the first asked whether any Computer Aided Systems Engineering (CASE) tools were in use in the company. The others asked what tools were used during development and maintenance.
- Prototyping. A question asked whether the system was prototyped when it was written, and if so was it rewritten before being used in production.
- The use of a data dictionary. The survey asked if a data dictionary was used on the system and if so, what for.

5.2 Normalisation of the Data

The first stage in analysing the results of the survey was to convert the surveys into a common measurement which could be used for comparison. As previously stated in Chapter 3 the traditional method has been Lines of Source Code per Person Month, and it was considered appropriate to use a measure of system size per person month in this case. The survey asked for a number of measures of system size, and these were converted to one common measure using the replies from the others. Of the 47 replies, 8 did not use a fourth generation language, 9 gave no indication of the system size and 4 gave no indication of the resources allocated to software maintenance these were excluded from the analysis of the results, meaning that in the analysis of the factors 26 of the surveys were used.

The replies are shown in Table 51.

Table 5.1: Replies to question on system size.

Lines of Code	Lines of Executable Code	Entity Relat'nships	Function Points	Lines of Job Control	Entries in data Dictionary
		25			210
27,400	15,840			52	
85,400	60,000			2,000	
1,000	1,200				
88,000			2,600	1,000	450
100,000	80,000				
		140			
			7,800		
800,000	300,000				110
15,000	14,500				
		60			
			1,430		
35,000				400	5,000
69,000	65,000				
1,000,000					
155,540				1,232	400
6,000,000	3,000,000				
1,125,000				5,000	500
		30			
40,000		78			450
			1,500		10,000

Table 5.1: Replies to question on system size (continued).

Lines of Code	Lines of Executable Code	Entity Relat'nships	Function Points	Lines of Job Control	Entries in data Dictionary
			1,500		
2,500	2,150	5			
400	400				30
45,000	25,000			500	
71,000	53,000				
48,000					107
120,000	100,000			10,000	
685,000					
2,000		5			

As 22 of the replies specified the system size in lines of code, and this was by far the most common method specified, it was decided to use this as the measure of system size. It should be noted that it appears that in most cases the number of lines of code has been rounded to the nearest thousand, or even guessed less accurately. This rounding should not have a significant effect on the end results providing that it is not too inaccurate, as the intention of the research is to provide an estimate of the amount of software resources, to within 20%.

Three of the systems were specified in both number of entities and lines of code and these calculated to be:

	40,000 / 78 = 512.8
	2,500 / 5 = 500
	2,000 / 5 = 400
Total	44,500 / 88 = 506

Therefore if the system size was only given in the number of entity relationships the number of entity relationships was multiplied by 500.

Only one of the systems specified the size in function points and lines of source code and this calculated to be:

$$88{,}000 / 2{,}600 = 34$$

Therefore for the conversion 34 lines of code per function point was used.

To enable the system to be compared all systems were therefore converted into lines of code per person month (LOC/PM). The figure used was the amount of software maintenance required by the system for the previous 12 months, as this was thought to be the most reliable. In many cases the amount of software maintenance resources allocated to the system was being greatly reduced in the following financial year, but it was thought that this was probably due to a recession and as software maintenance has a low priority in management's view it was receiving the largest cuts. In many cases the staff who were maintaining the system were overworked and the resources were being cut to a half or even one third of those for the previous financial year. It was not known at this stage whether the resources could be reduced by this amount and the system still be maintained.

The number of lines of code being maintained is shown in Table 5.2

Table 5.2 Number of lines of code per person month being maintained - replies from survey.

Language	LOC/PM	LOC/PM	LOC/PM	LOC/PM	Language Average
Smart/400	850				850
SYNON/2	972				972
Oracle	1,250	1,250	1,500	2,160	1,540
Ingres	1,972	2,083	2,222		2,092

Table 5.2 Number of lines of code per person month being maintained - replies from survey (continued).

Language	LOC/PM	LOC/PM	LOC/PM	LOC/PM	Language Average
Linc	2,200				2,200
Natural	2,210				2,210
Pro-IV	2,125				2,125
Focus	1,000	1,945	4,000		2,315
Nomad	1,250	3,558	6,666		3,825
Ramis	4,566				4,566
Ideal	3,333	6,000			4,667
Mantis	21,666				21,666
Unisys Link	15,930				15,930
Ingrev	19,800				19,800
SQL Windows-Forms	25,000				25,000
Telon	27,780				27,780

It can be seen from this table that there is a major difference between the amount of software maintenance a system requires depending on the fourth generation language used. It was also noted that there appeared to be 3 groups consisting of:

Group A - Smart/400, SYNON/2, Oracle, Ingres, Linc, Natural, Pro-IV and Focus.

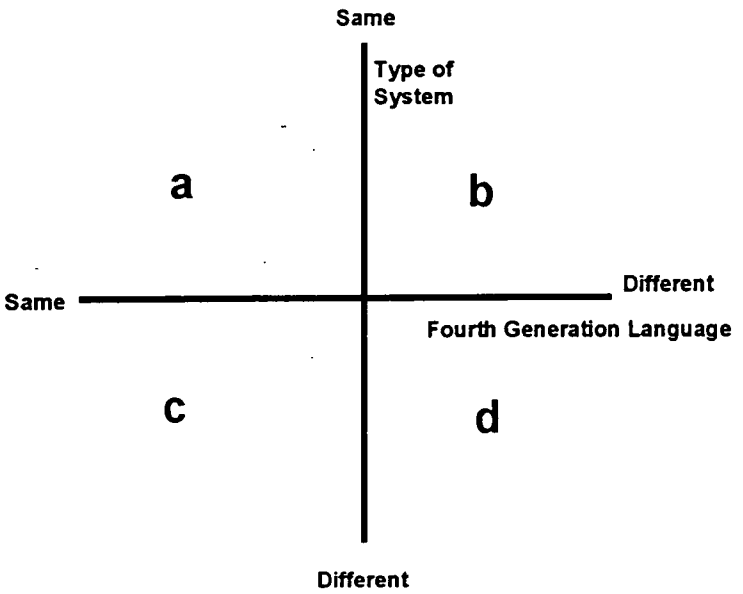
Group B - Nomad, Ramis and Ideal.

Group C - Mantis, Unisys Link, Ingrev, SQL Windows - Forms and Telon.

These groupings are based on the average number of lines of code being maintained for a particular language. They do not relate to the classification of fourth

generation languages discussed in Chapter 2 as these were development categories and here a maintenance category is being considered. In order to verify that this is a reasonable grouping the lines of code were related to the type of application. In considering this there are four possibilities as shown in Figure 5.1

Figure 5.1: Lines of code related to type of language



a) the systems are of the same type and written in the same language in which case the amount of code being maintained per person month will be approximately equal.

b) the systems are of the same type and written in the different languages in which case the amount of code being maintained per person month will not be expected to be the same.

c) the systems are not of the same type and are written in the same language in which case the amount of code being maintained per person month will not necessarily be the same.

d) the systems are not of the same type and written in different languages in which case the amount of code being maintained per person month will be expected to be different.

As financial systems are the largest category in the survey, this has been used to compare the above possibilities. Table 5.3 shows the number of lines of code for financial systems together with their languages.

Table 5.3 Lines of code maintained per person month for financial systems.

Language	Lines of Code	Lines of Code
Smart/400	850	
SYNON/2	972	
Oracle	1,250	1,500
Ingres	1,972	2,083
Natural	2,210	
Focus	1,945	
Nomad	3,558	
Ramis	4,556	
Ideal	6,000	
Mantis	21,666	
Ingrev	19,800	
SQL Windows-Forms	25,000	
Telon	27,780	

a) There are 2 financial systems written in both Oracle and Ingres, and in Oracle 1,250 and 1,500 lines of code are maintained in each of their systems. and 1,972 and 2,083 in Ingres.

b) Table 5.3 shows that shows that 1,250 lines of code are being maintained with Oracle and 4,566 are being maintained with Ramis and 27,780 with Telon.

c) In Oracle non financial and financial systems are written and it shows that 1,250 and 1,500 lines are maintained per person month in the financial systems and 2,160 in a stock control system.

d) 2,160 lines of Oracle code are being maintained in a stock control system and 6,000 in an Ideal financial system and 15,930 in a Unisys Link M.I.S. system.

All these results confirm that the languages appear to fall into three distinct categories.

In this case:

Group A has 15 samples with a mean of 1,849.

Group B has 6 samples with a mean of 4,229

Group C has 5 samples with a mean of 22,035

The comparisons are therefore shown in Table 5.4.

Table 5.4: Analysis of Variance table for types of language.

	Variance Estimate	Degrees of Freedom
Between Samples	780,423,154.49	2
Within Samples	73,516,075.42	23

The calculated F ratio is therefore 10.615, and as can be seen from Appendix B the critical value for the appropriate degrees of freedom is 3.42, and therefore statistically these figures show that at the 5% significance level it was considered that there were three classes of fourth generation language.

To enable comparisons between the types of language to be made a conversion was made. The mean of the three types was used to obtain a weighting factor by dividing the mean by the mean of Group A (the lowest value), these are shown in Table 5.5.

Table 5.5 Multipliers for each fourth generation language group

Group	A	B	C
Average	1,849	4,229	22,035
Conversion	1.0	4,229/1,849 = 2.3	22,035/1,849 = 11.9

This conversion has been applied to the lines of code per person month from the surveys and the results are shown in Table 5.6.

Table 5.6 Number of lines of code per person month being maintained - replies from survey after conversion .

Language	LOC/PM	LOC/PM	LOC/PM	LOC/PM	Language Average
Smart/400	850				850
SYNON/2	972				972
Oracle	1,250	1,250	1,500	2,160	1,540
Ingres	1,972	2,083	2,222		2,092
Linc	2,200				2,200
Natural	2,210				2,210
Pro-IV	2,125				2,125
Focus	1,000	1,945	4,000		2,315
Nomad	547	1,556	2,915		1,673

Table 5.6 Number of lines of code per person month being maintained - replies from survey after conversion (continued).

Language	LOC/PM	LOC/PM	LOC/PM	LOC/PM	Language Average
Ramis	1,997				1,997
Ideal	1,458	2,624			2,041
Mantis	1,818				1,818
Unisys Link	1,337				1,337
Ingrev	1,662				1,662
SQL Windows-Forms	2,098				2,098
Telon	2,331				2,331

After the conversion these three types of language can be shown to be approximately equal, as can be seen from Table 5.7.

Table 5.7: Comparison of the 3 types of language after conversion.

	Group A	Group B	Group C
Minimum value	850	547	1,337
Maximum value	4,000	2,915	2,331
Mean value	1,849	1,831	1,849

5.3 General Results

Forty seven replies were received from the survey, and the results in this section are all classified by the number these replies, and set the scope of the survey. A number of industrial categories were given and the number of responses for each of these are shown in Table 5.8.

Table 5.8 The industrial categories of the departments and organisations.

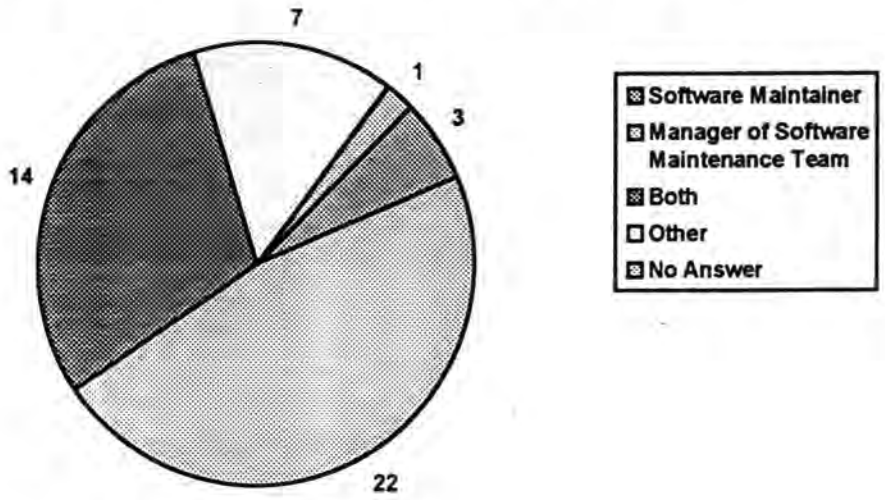
Industrial Category	Department	Organisation
Aerospace engineering	1	1
Chemical/ Allied products	1	1
Computer hardware	0	1
Computer software	23	6
Food/ Drink/ Tobacco	0	1
Instruments/ Electrical/Electronics	1	1
Mechanical engineering	1	2
Petroleum/ Coal/ Rubber	0	2
Textiles/ Leather goods/ Footwear/ Clothing	1	1
Transportation equipment/ Vehicles	1	1
Banking/ Credit agency	3	7
Business and professional services	1	0
Communications and information media	1	2
Distribution and associated trades	0	1
Health services	0	2
Insurance/ Assurance	5	5
Leisure and recreational services	1	2
Public administration	1	1
Transport/ Travel and supporting services	5	5
Retail	0	2
Public utilities (Gas/ Electricity/ Water)	1	3
Total	47	47

One of the respondents did not supply a company size, and excluding this the minimum was 34, the maximum 80,000 and the average size was 6,578.63.

The maximum DP Department size was 900, minimum 1 and average 243.04.

The survey asked whether the respondent was a software maintainers or a manager of a software maintenance team and the results are shown in Figure 5.2.

Figure 5.2 The position of the respondents to the survey.



The survey included a list of categories which are classed as software maintenance under the definition given in Chapter 1. A question then asked which of these categories were classed as software maintenance, the number of replies for each of these categories are given in Table 5.9. Where a time limit was applied the lower categories were also assumed to be included, for instance if changes which take less than 10 days was given, changes which take less than 1, 3 and 5 days were also included, even if they were not marked.

Table 5.9 Number of replies who included category as software maintenance.

Description	Count
Emergency repairs	44
Changes to correctly reflect the specifications or correctly utilise system resources	36
Upgrades to adapt to changes in processing requirements	37
Amendments to adapt to changes in regulations	34
Growth amendments performed to adapt to changes in data requirements, or to the addition of new programs, new users etc.	36
Enhancements for changed requirements	31
Support for users of the system	37
Changes which take less than 1 day	39
Changes which take less than 3 days	36
Changes which take less than 5 days	34
Changes which take less than 10 days	27
Changes which take less than 20 days	24

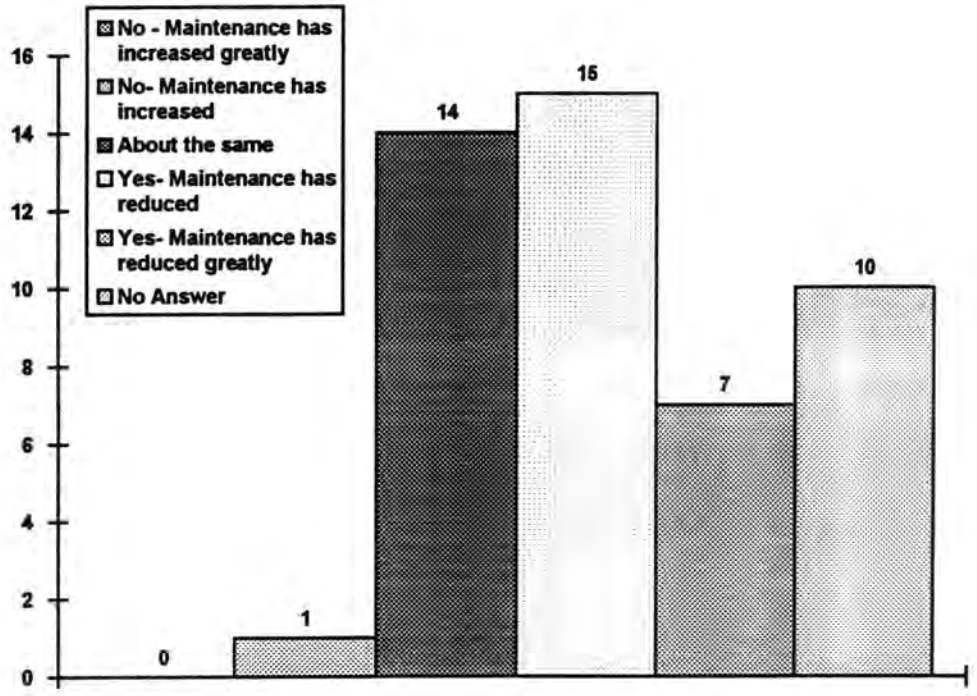
All of these categories are included in the types of maintenance introduced at section 2.3, but the results show that approximately one-third of the respondents did not include perfective as maintenance, whilst most (94%) include emergency repairs.

Of the respondents 39 used a fourth generation language and 8 did not, this percentage is artificially high as the survey was targeted at companies who do.

Twenty five respondents used a Computer Aided Software Engineering (CASE) tool and 21 did not. One did not answer this question

The survey asked whether the respondent thought that the fourth generation language had reduced software maintenance, and these results are shown in Figure 5.3.

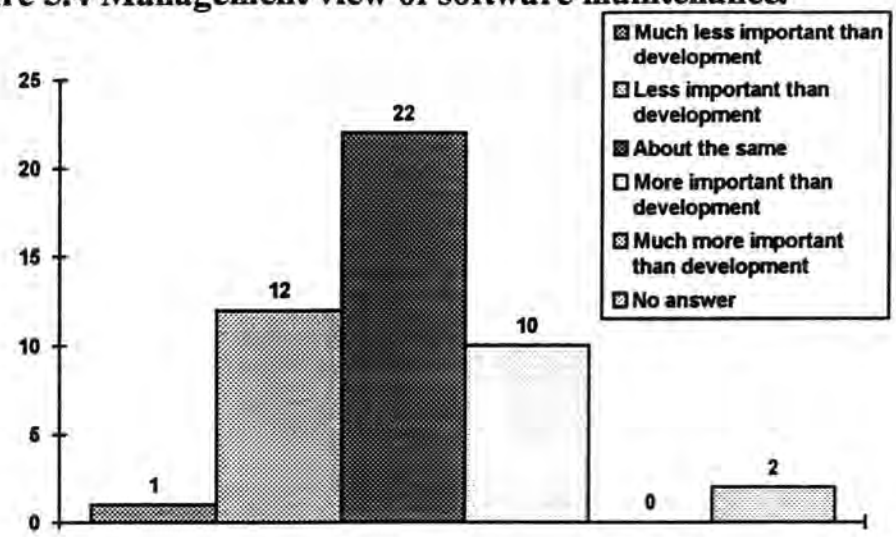
Figure 5.3 Whether respondents thought the fourth generation language had reduced software maintenance.



These results show that the majority of the respondents (22) feel that fourth generation languages have reduced software maintenance, with only one reporting an increase and 14 answering about the same.

The replies to the question about management's view of software maintenance is shown in Figure 5.4.

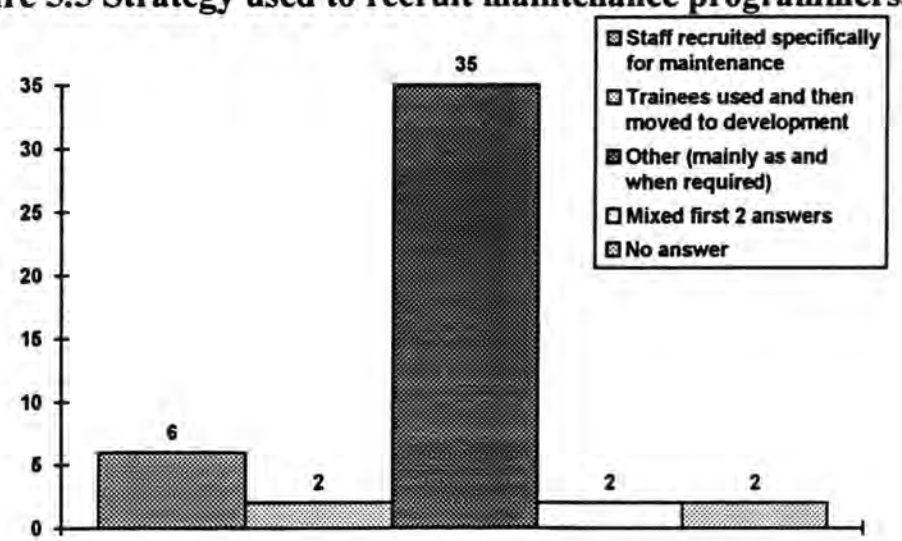
Figure 5.4 Management view of software maintenance.



These results show that, on average, managers do rate software maintenance on a par with development, 13 seeing it as less important, 10 as more important and 22 about the same.

Replies to the question on strategy of recruitment of software maintenance programmers is shown in Figure 5.5.

Figure 5.5 Strategy used to recruit maintenance programmers.



The results show that companies normally recruit software maintainers as and when required.

5.4 Survey Results

All results in this section were calculated by lines of code per person month and were initially analysed using the t-test. This is because the number of categories to be chosen from i.e. the scale 1 to 5 was an arbitrary choice, and another scale could have been chosen for example, 1 to 3, or even a YES/ NO answer. The t-test was therefore used as a first cut, to see if any of the categories could be combined, with the resulting combined categories being analysed through the appropriate test, t-test or analysis of variance, depending on whether there were more than 2 combined categories.

Following the first cut adjacent groups with no significant difference between them were combined i.e. if the initial results showed that there was no significant difference between 1 and 2 and 4 and 5 then the analysis of variance would be used to compare 3 samples (1 and 2 combined, 3, and 4 and 5 combined). If when the groups were combined only 2 categories remained the t-test would be used against these 2 samples. Combinations of non-adjacent groups were not considered logical and therefore were never combined unless the intermediate group or groups were combined.

Any survey which did not answer a particular question was excluded from the results for that question. Section 5.1 introduced the list of factors to be tested and the rationale behind the survey. A copy of the survey is shown at Appendix C and the full results are at Appendix D, together with the rationale behind the hypothesis and a discussion of the results. The raw data is shown in Appendix E.

The analysis of the results are shown in Table 5.10.

5.10 Summary of Survey Results.

Hypothesis - The amount of software maintenance is dependant upon . . .	Number of surveys analysed	Difference between factors Hypothesis proved	Difference between factors Hypothesis not proved
The history of the reliability of the system	22		✓
The reliability of the system in the last 12 months	25		✓
The age of the system	26		✓
The required reliability of the system	26		✓
The complexity of the tasks undertaken by the system	26		✓
The type of application	26		✓
Development effort in person months compared to estimates of development effort	22	✓	
The elapse time of development when compared to estimates of development time	23	✓	
Reliability of the fourth generation language	25		✓
The number of user written programs	26	✓	
The mean ability level of the staff at the time the system was developed	26		✓
The average ability level of the software maintenance staff	26		✓
The morale of the software maintenance staff	26		✓

5.10 Summary of Survey Results (continued).

Hypothesis - The amount of software maintenance is dependant upon . . .	Number of surveys analysed	Difference between factors Hypothesis proved	Difference between factors Hypothesis not proved
The experience of the development staff with the fourth generation language	23		✓
The experience of the development staff with the computer environment	23	✓	
The experience of the development staff with the type of application	23		✓
The experience of the development staff with any third generation language	22		✓
The experience of the software maintenance staff with the fourth generation language	23	✓	
The experience of the software maintenance staff with the computer environment	23		✓
The experience of the software maintenance staff with the type of application	23		✓
The experience of the software maintenance staff with any third generation language	22		✓
The level of training provided specifically for software maintenance	25		✓
The level of training provided for the fourth generation language	25	✓	

5.10 Summary of Survey Results (continued).

Hypothesis - The amount of software maintenance is dependant upon . . .	Number of surveys analysed	Difference between factors Hypothesis proved	Difference between factors Hypothesis not proved
The level of training provided for any third generation language	26	✓	
Whether the staff maintaining the system are those who developed it	26	✓	
The workload of the software maintenance staff	25		✓
The time allowed for individual changes	26		✓
Whether any flexibility for future changes was built into the system at the time it was written	26	✓	
The number of enhancements included in the system in the previous 12 months	26		✓
Expected hardware changes	26		✓
The quality of the documentation	25		✓
The use of site standards	26		✓
The use of comments in the source code	26		✓

5.10 Summary of Survey Results (continued).

Hypothesis - The amount of software maintenance is dependant upon . . .	Number of surveys analysed	Difference between factors Hypothesis proved	Difference between factors Hypothesis not proved
The use of meaningful names in the software	25	✓	
The use of a methodology in the development of the system	26		✓
The use of a methodology in the maintenance of the system	26		✓
The understandability of the source code	26		✓
The complexity of the code	26		✓
Management's view of software maintenance against their view of development	25		✓
Whether the system was prototyped during development	26	✓	
Whether the prototype was rewritten before being used in production	7		✓
Whether a data dictionary is used in the system	25		✓
The use of tools in the development of the system	23	✓	
The use of tools in software maintenance	24		✓

Summary: The data to test the factors which influence the amount of software maintenance was collected by means of a survey, and the analysis was done initially by the use of the t-test, to enable classes to be combined. Three categories of fourth generation language were identified and the amount of software maintenance per person month was adjusted to provide a common value to enable the different classes of language to be compared. The hypotheses were examined in this chapter to see which of the factors discussed in section 5.1 had a significant influence on the level of software maintenance. The original list was refined to 12 factors which did significantly influence the resources required by software maintenance. These 12 factors are:

Attributes of the system

- *Development effort in person months being adequate*
- *Elapsed development time being adequate*
- *The number of user written programs in the system*

Personnel attributes

- *The development staff having more than 6 months experience of the computer environment*
- *The software maintenance staff having more than 6 months experience with the fourth generation language*
- *Provision of training for the fourth generation language*
- *Training provided for any third generation language*
- *Drawing the software maintenance staff from the development staff*

Volatility

- *Building in flexibility for changes when the system is written*

Understandability of the source code

- *Always using meaningful names in the software*

Use of tools

- *The use of prototyping during the development of the system*
- *Using tools during the development of the system*

CHAPTER 6

Application of the Results

Abstract: This chapter uses the factors which influence the amount of software maintenance, applies a weight to them and produces an equation to calculate the amount of software maintenance a system written in a fourth generation language will require. The chapter also discusses the use of the factors and the equation.

6.1 Introduction.

Chapter 5 introduced a list of factors which were considered may have an influence on the amount of software maintenance resources when using fourth generation languages. The chapter then examined these factors to see whether the evidence showed that they actually have a significant effect on software maintenance. The original list was reduced to 12 factors which are:

- Development effort in person months being adequate.
- Elapsed development time being adequate.
- The number of user written programs in the system.
- The development staff having more than 6 months experience of the computer environment.
- The software maintenance staff having more than 6 months experience of the fourth generation language.
- Provision of training for the fourth generation language.
- Training provided in any third generation language.

- Drawing the software maintenance staff from those who originally developed the system.
- Building in flexibility for changes when the system was written.
- Always using meaningful names in the software.
- The use of prototyping during the development of the system.
- Using tools during the development of the system.

6.2 Factors Weights.

The identified factors will not influence the amount of software maintenance equally and a weight was assigned from the mean of the normalised value of software being maintained i.e. the mean of all the surveys which gave that answer. These means are shown in Table 6.1.

Table 6.1 Mean of amount of software being maintained for each factor.

Factor	Yes	No
Development effort in person months being at least that predicted by the COCOMO basic organic model based on the current system size.	2,010	942
Development elapse time being at least that predicted by the COCOMO basic organic model based on the current system size.	2,039	1,423
Number of user written programs being less than 100	2,130	1,218
Development staff experience of the computer environment less than 6 months	905	2,106

Table 6.1 Mean of amount of software being maintained for each factor (continued).

Factor	Yes	No
Software maintainers experience of the fourth generation language less than 6 months	1,350	2,090
Provision of training for the fourth generation language at least adequate	2,018	842
Training provided for any third generation language at least adequate	1,988	790
All software maintenance staff drawn from developers of the system	2,355	1,698
Built in flexibility at the time the system was developed more than average	2,143	1,449
Always using meaningful names in the software	2,103	1,158
The use of prototyping during the development of the system	2,363	1,660
The use of tools in the development of the software	2,371	1,645

These factors show that the means of these factors are different from each other and a weight can be derived from these figures by division of these means and these are shown in Table 6.2 with the highest mean being given a value of one.

Table 6.2 Weights for the factors identified.

Ref.	Factor	Yes	No
(i)	Development effort in person months being at least that predicted by the COCOMO basic organic model based on the current system size.	1.0	2,010/942 = 2.1
(ii)	Development elapse time being at least that predicted by the COCOMO basic organic model based on the current system size.	1.0	2,039/1,423 = 1.4
(iii)	Number of user written programs being less than 100	1.0	2,130/1,218 = 1.7
(iv)	Development staff experience of the computer environment less than 6 months	2,106/905 = 2.3	1.0
(v)	Software Maintainers experience of the fourth generation language less than 6 months	2,090/1,350 = 1.5	1.0
(vi)	Provision of training for the fourth generation language at least adequate	1.0	2,018/842 = 2.4
(vii)	Training provided for any third generation language at least adequate	1.0	1,988/790 = 2.5
(viii)	All software maintenance staff drawn from developers of the system	1.0	2,355/1,698 = 1.4
(ix)	Built in flexibility at the time the system was developed more than average	1.0	2,143/1449 = 1.4
(x)	Always using of meaningful names in the software	1.0	2,103/1,158 = 1.8
(xi)	The use of prototyping during the development of the system	1.0	2,363/1,660 = 1.4
(xii)	The use of tools in the development of the software	1.0	2,371/1,645 = 1.4

These weights can then be used by reading off the figure from the columns corresponding to the answer from the survey.

6.3 Software Maintenance Equation.

Section 6.2 showed the weights for the factors identified in Chapter 5, and section 5.2 showed that there were three distinct maintenance classes of fourth generation language. A curve fitting program was used to derive an equation for the best fit. As many of the surveys had the same effort adjustment factor, and the criteria was to produce an equation which would fit to within 20% of the obtained points, the values were combined where the effort adjustment factors were equal. The computer program produced an equation for the best fit for the allocation of software maintenance resources in person months:

NSS * f(EAF)

where NSS = Normalised System Size

f(EAF) = function of effort adjustment factor

and:

NSS = SS / LAF

where SS = System Size

LAF = Language Adjustment Factor

The Language Adjustment factor is that derived in section 5.2, and is:

Group A - 1.0

(Smart/400, SYNON/2, Oracle, Ingres, Linc, Natural, Pro-IV and Focus)

Group B - 2.3

(Nomad, Ramis and Ideal)

Group C - 11.9

(Mantis, Unisys Link, Ingrev, SQL Windows - Forms and Telon)

$$f(EAF) = \ln(EAF) * 0.0002 + 0.0001$$

EAF is that derived from the 12 factors identified in Chapter 5 and weighted as shown in Table 6.2. It is simply all the weights multiplied together.

The computer program produced a value for the correlation coefficient r of 0.96, and the fit of this equation to the obtained data is shown later in this chapter in figure 6.3.

The full format of the equation is:

Annual software maintenance resources =

$$(SS/LAF) * (\ln(EAF) * 0.0002 + 0.0001)$$

The EAF is in the range:

1.0 to 715.4

and the range of ln(EAF) is:

0 to 6.573

Multiplying this range by 0.0002 is the same as dividing by 5,000 and provides the range:

0 to 0.001315

Adding the 0.0001 ensures that f(EAF) is not zero, otherwise if all factors were 1.0 i.e. software maintenance was reduced to a minimum the formula would provide an estimate of zero man months to maintain the system. After this calculation the new range is:

0.0001 to 0.001415

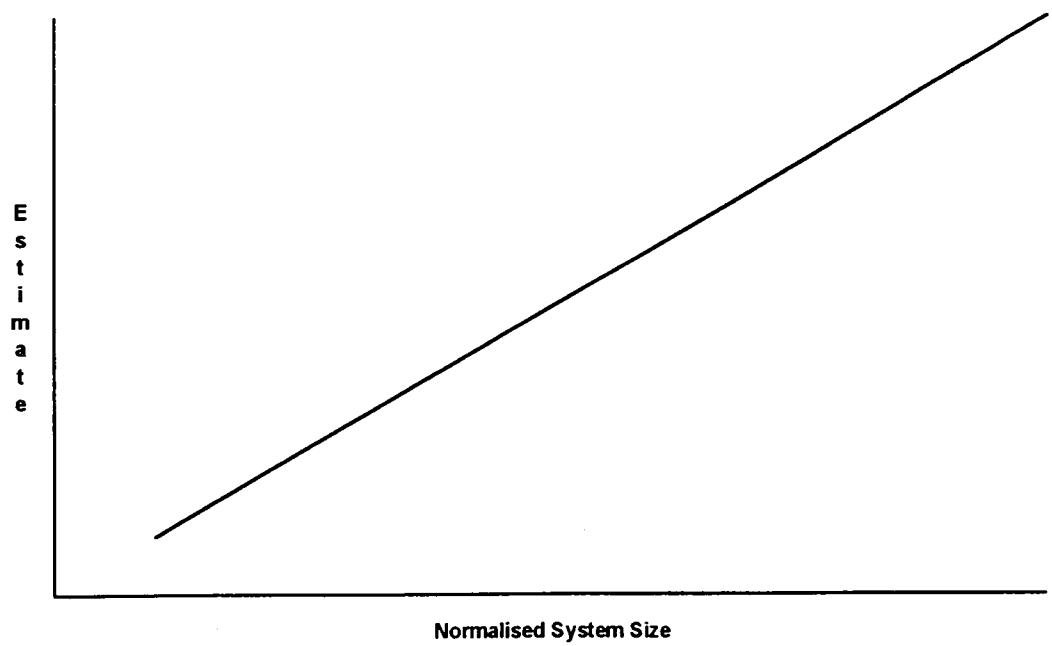
Although these figures are small, the highest value is greater than 14 times the smallest, showing that these factors do have a significant effect on the amount of software maintenance resources required by a system.

From these figures the range of software maintenance resources in man months required annually by systems of particular sizes are:

1,000 lines	0.1	to	1.41
5,000 lines	0.5	to	7.07
10.000 lines	1.0	to	14.1
20,000 lines	2.0	to	28.2
50,000 lines	5.0	to	70.7
100,000 lines	10.0	to	141.5

These figures show that the estimate in man months related to system size is linear, i.e. it takes the form:

Figure 6.1. Normalised System Size plotted against Estimate



The equation shows that the EAF has a logarithmic relation to the estimate:

Figure 6.2 Effort Adjustment Factor plotted against Estimate

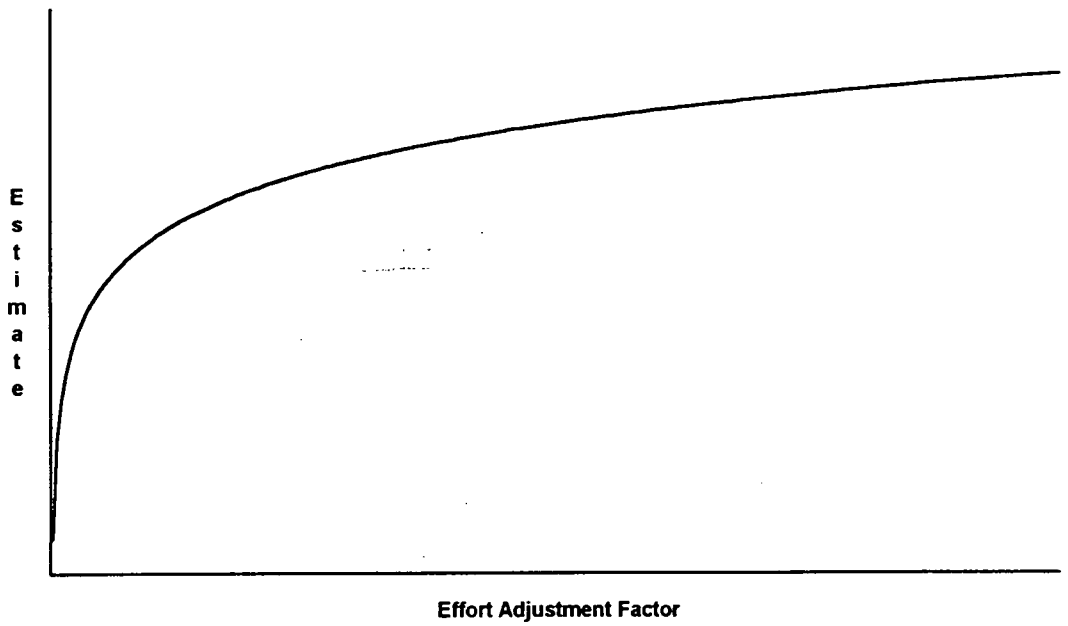


Table 6.3 shows the individual weights for the fifteen of the surveys (numbered 1 to 15) who supplied all the required information. The factors i to xii are those shown in Table 6.2 and the weights are those appropriate to the individual answers. All these individual scores have been multiplied together to give an Effort Adjustment Factor (EAF), also shown in Table 6.3.

Table 6.3 Calculated effort adjustment factors.

	i	ii	iii	iv	v	vi	vii	viii	ix	x	xi	xii	EAF
1	1.0	1.0	1.7	2.3	1.5	2.4	2.5	1.4	1.4	1.8	1.4	1.4	243.33
2	1.0	1.0	1.7	2.3	1.5	2.4	2.5	1.4	1.4	1.8	1.4	1.4	243.33
3	2.1	1.4	1.7	1.0	1.0	1.0	1.0	1.4	1.0	1.8	1.4	1.4	24.69
4	2.1	1.4	1.0	1.0	1.5	1.0	1.0	1.4	1.0	1.0	1.4	1.4	12.10
5	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.4	1.4	1.0	1.4	1.4	11.29
6	2.1	1.4	1.7	1.0	1.0	1.0	1.0	1.4	1.0	1.0	1.0	1.4	9.80
7	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.0	1.4	1.4	8.07
8	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.0	1.4	5.76
9	2.1	1.0	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.0	1.4	1.4	5.76
10	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.0	1.4	1.0	5.76
11	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.0	1.4	1.0	5.76
12	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.0	1.4	1.0	5.76
13	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.4	4.12
14	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.4	4.12
15	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.4	1.0	4.12

The System Size (SS) and Language Adjustment Factor (LAF) derived in Table 5.5 is used to determine a Normalised System Size (NSS) by the formula NSS = SS/LAF. This Normalised System Size together with the Effort Adjustment Factor (EAF) can be used to provide an estimate of the resources required to maintain the system. This can then be compared to the actual resources allocated, given by the survey. The difference between the two values, together with the actual value can be used to provide a percentage accuracy for the estimate. This is shown in Table 6.4.

Table 6.4 Estimates versus actual software maintenance resources.

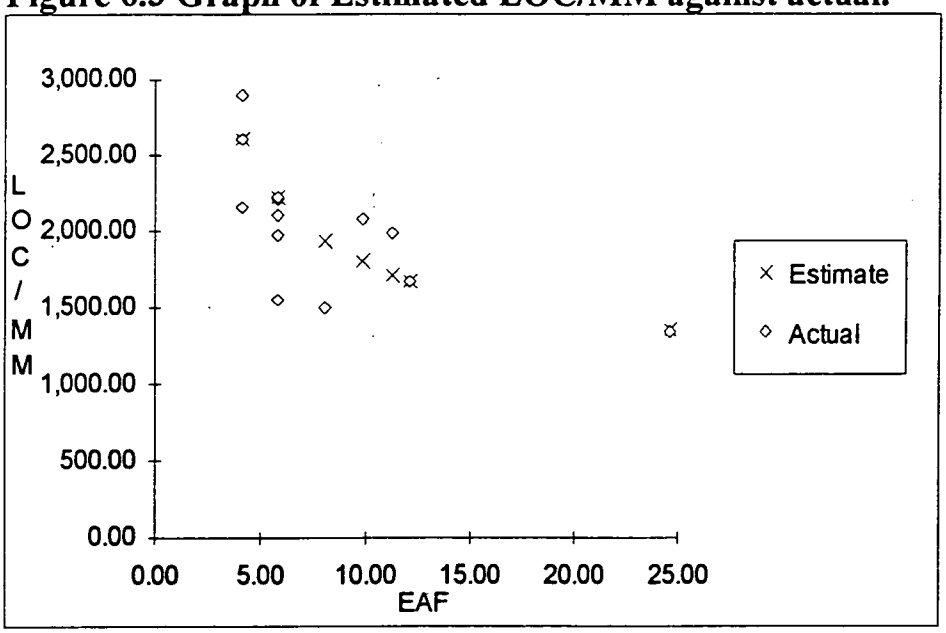
No.	System Size from survey	L.A.F.	Normalised System Size	E.A.F.	Estimate	Actual	% Difference
1	2,500	2.3	1,087	243.33	1.30	2	35.00
2	35,000	1.0	35,000	243.33	41.96	36	16.56
3	685,000	11.9	57,563	24.61	42.67	43	0.77
4	12,500	11.9	1,050	12.10	0.63	0.63	0.00
5	27,400	2.3	11,913	11.29	6.97	6	16.17
6	300,000	1.0	300,000	9.80	166.94	144	15.93
7	45,000	1.0	45,000	8.07	23.29	30	22.37
8	2,000	1.0	2,000	5.76	0.90	0.9	0.00
9	71,000	1.0	71,000	5.76	31.96	36	11.22
10	85,400	2.3	37,130	5.76	16.72	24	30.33
11	100,000	11.9	8,403	5.76	3.78	4	5.50
12	265,000	1.0	265,200	5.76	119.39	120	0.51
13	48,000	2.3	20,870	4.12	8.00	8	0.00
14	40,000	2.3	17,391	4.12	6.66	6	11.00
15	155,540	1.0	155,540	4.12	59.60	72	17.22

Table 6.4 shows the calculated resources required by software maintenance against the actual resources 12 of the 15 are within 20% of the actual, this calculates to 80% of the tested systems. Of the other three systems, one is very small giving an estimate of 1.3 as opposed to 2 person months, and another provides an estimate of 23.29, whereas in this case the 20% range is from 24 to 36 person months.

It can be seen from Table 6.4 that systems 2 and 9 are written in the same class of language, and both use 36 man months effort to maintain, however system 2 is 35,000 lines and system 9 is 71,000 lines. This difference is accounted for by the difference in EAF (243.33 and 5.76).

A graph of the actual values compared against the estimates is shown in Figure 6.3. As there were 3 factors Effort Adjustment Factor, Normalised System Size and either the estimated or actual person months effort the Normalised System Size / Man Months was plotted against the Effort Adjustment Factor. There are 2 values with an Effort Adjustment Factor of 243.33 which are not included on this graph, as the scale required would affect the detail which could be seen for the other values.

Figure 6.3 Graph of Estimated LOC/MM against actual.



Boehm [BOEHM83] has said, "Today a software cost estimation model is doing well if it can estimate software development costs within 20% of actual costs for 70% of the time, and on its own turf (that is within the class of projects to which it has been calibrated . . . . .)". The cost estimation model developed here refers to software maintenance but it is within this range, and it is thought that the prediction of software maintenance costs is more difficult than development costs because of problems with definition of software maintenance.

6.4 Confirmation of the Results.

Having obtained the equation, the next stage was to test its effectiveness against data which had not been used to derive it. A reduced survey was prepared containing just those questions relating to the factors which had been identified as having a significant effect on the amount of software maintenance. This survey contained questions 1, 11(relevant part), 17, 22, 23, 24, 29(relevant part), 30, 31, 33(relevant part), 42 and 44 from the original survey. The other questions were removed in the hope that due to the reduction in size of the survey more responses would be received. Eighty five shortened surveys were despatched and 14 were received, of these 9 contained all the relevant information to test the equation. Table 6.5 shows the calculation of the effort adjustment factor for these 9, the roman numerals refer to those weights in Table 6.2.

Table 6.5 Calculated effort adjustment factors for the additional data

	i	ii	iii	iv	v	vi	vii	viii	ix	x	xi	xii	EAF
1	2.1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.10
2	1.0	1.4	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.8	1.4	1.0	4.94
3	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.4	1.4	1.0	1.4	1.0	8.07
4	2.1	1.0	1.0	1.0	1.0	1.0	1.0	1.4	1.4	1.0	1.4	1.0	5.76
5	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.4	1.4	5.76
6	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.4	1.4	1.0	1.0	1.0	5.76
7	2.1	1.0	1.7	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.4	1.0	6.99
8	2.1	1.0	1.0	1.0	1.0	1.0	2.5	1.4	1.4	1.0	1.0	1.0	10.29
9	2.1	1.4	1.0	1.0	1.0	1.0	1.0	1.0	1.4	1.0	1.0	1.0	4.12

An estimate of the amount of software maintenance was calculated for each of these 9 new surveys using the derived equation and the appropriate Language Adjustment Factor. The results are shown in Table 6.6. The percentage difference is calculated in the same way as that in Table 6.4.

Table 6.6 Estimates versus actual software maintenance resources for the additional data

No.	System Size from survey	L.A.F.	Normalised System Size	E.A.F.	Estimate	Actual	% Difference
1	3,000	1.0	3,000	2.10	0.75	0.75	0.00
2	650	1.0	650	4.94	0.27	0.25	7.41
3	15,000	1.0	15,000	44.45	12.88	12.0	6.83
4	331,133	2.3	143,971	4.66	58.71	48.0	18.24
5	20,000	11.9	1,681	8.07	0.87	1.0	14.94
6	5,000	2.3	2,174	5.76	0.98	1.0	2.04
7	15,000	1.0	15,000	6.99	7.33	7	4.50
8	9,000	1.0	9,000	10.29	5.10	5.00	1.96
9	21,000	1.0	21,000	4.12	8.05	8	0.62

These figures confirm the validity of the equation within the limits of 20% of the actual costs for 70% of the time.

6.5 Application of Results.

The 12 factors which were identified in Chapter 5 to have a significant effect on software maintenance in fourth generation language systems can be classified as either:

- fixed at the time of development
- changeable during maintenance.

The fixed factors are:

- the time of development in person months
- the elapsed time of development
- the number of user written programs
- development staff experience with the computer environment
- the amount of flexibility in the system
- the use of prototyping during development
- the use of tools in development

The changeable factors are:

- the fourth generation language experience of the software maintenance staff
- fourth generation language training
- third generation language training
- maintainers drawn from development
- use of meaningful names in the software. This is a changeable factor as sections of the code could be rewritten to change this factor.

System 2 from the original data can be used to show how varying the factors it can influence the estimate of software maintenance resources required. This system is a good one to illustrate the use of the equation as with the exception of factors (i) and (ii) all the weights are greater than 1 showing that improvements could be made in these areas. Table 6.7 shows how the estimate changes when a factor is changed.

Table 6.7 Adjustment of the factors.

Changed Factor	Adjusted EAF	Adjusted Estimate	Saving - person months
original system	243.33	41.96	-
(iii)	143.14	38.25	3.71
(iv)	105.80	36.13	5.83
(v)	162.22	39.12	2.84
(vi)	101.39	35.83	6.13
(vii)	97.33	35.55	6.41
(viii)	173.81	39.61	2.35
(ix)	173.81	39.61	2.35
(x)	135.19	37.85	4.11
(xi)	173.81	39.61	2.35
(xii)	173.81	39.61	2.35
All changeable factors	10.73	20.11	21.85
All fixed factors	22.68	25.35	16.61
All factors	1	3.5	38.46

If factors (i) and (ii) were inadequate as well as all the other factors, this would produce the worst case, and revise the estimate upwards to 49.51 person months.

This example illustrates that varying the factors, the estimate for maintaining this system varies from 3.5 to 49.51 person months. Assuming that the system would be the same size if it was rewritten COCOMO estimates a 100 person month development (although the actual was 660 person months) consideration can now be given to rewriting this system.

If a 10,000 line system was taken which was being maintained, and has all the factors greater than 1 (i.e. the software maintenance was maximised). If a decision was taken to reduce software maintenance and various factors were taken and addressed the maintenance could be reduced as shown in Table 6.8.

Table 6.8 Estimates for a 10,000 line system varying the EAF

Factor	EAF	Estimate in Man Months
Initial system	715.4	14.14
Provide adequate training for the 4GL	298.08	12.4
Provide adequate 3GL training	119.23	10.56
Use staff with at least 6 months experience of the 4GL	79.49	9.76
Use maintainers who developed the system	56.78	9.08
Rewrite to always use meaningful names in the software	31.54	7.9

Table 6.8 shows that by addressing these factors the software maintenance effort required to keep the system running has been reduced from 14.14 man months to 7.9 a saving of 6.24 man months of 44%.

It should be considered when using this equation that it provides and estimate of the amount of software maintenance resources required, and it may therefore need to be adjusted for any particular installation.

*Summary: The weights for each of the 12 factors can be multiplied together to obtain an effort adjustment factor (EAF). The system size can be adjusted by a Language Adjustment Factor to give a normalised system size (NSS). These two factors can be combined into NSS * f(EAF) to provide an estimate in person months of the*

*software maintenance resources required by a system written in a fourth generation language. The f(EAF) is ln(EAF) * 0.0002 + 0.0001, and therefore the full equation is:*

Annual software maintenance effort in person months =

*(SS/LAF) * (ln(EAF) * 0.0002 + 0.0001).*

CHAPTER 7

Evaluation and Conclusion

Abstract: This chapter summarises the results of the research.

7.1 Introduction.

Software maintenance uses over half of the data processing resources with traditional third generation languages, which has led to large backlogs of computing tasks. To alleviate this problem fourth generation languages have been designed which enable software to be developed more quickly than previously.

Software maintenance will not disappear with systems written in fourth generation languages as these programs will have to evolve and require changes to correct faults, etc. Little research has yet been done to assess the effect of fourth generation languages on software maintenance. They were designed to allow rapid development, but if they use more resources in the maintenance phase the situation could be at least as bad, if not worse in a few years.

Software maintenance is often seen as producing nothing, merely maintaining the status quo, because of this often managers have a low opinion of software maintenance and tend to view development as a higher priority.

This situation could be improved if software maintenance could be discussed in business terms. To enable this to be done a means of estimating software maintenance costs needs to be used. When department budgets are set the Data Processing manager needs to be able to quote the costs for keeping a system running and the consequences

of not paying them. This would enable software maintenance to be treated on an equal status as all the other business functions.

There are various cost estimation models, mainly for estimating development costs, and most of these have cost drivers related to various factors. In this research the methods to produce an equation were introduced, together with an actual worked example. A survey was carried out and the results analysed, initially using the t-test and then either the t-test or the analysis of variance. This enabled the first cut of a list of cost drivers which significantly influence the amount of software maintenance when using fourth generation languages to be produced. From these factors an equation was formulated to provide an estimate of the software maintenance resources required to keep a system running for twelve months. It should be noted that this equation is only the first draft and will require further refinement. It is included only as a worked example, and as the sample size was small another survey may produce significantly differing results.

These factors can be used in the following ways:

- To allow the factors to be considered during the development and maintenance of the system to enable software maintenance resources to be minimised.
- To enable the correct resources to be assigned to software maintenance, this would also enable business decisions to be made concerning software maintenance, as when a system was designed the development and maintenance costs could be considered.
- Consideration could be given to these factors and a decision can be taken to decide the best time to rewrite the system. The estimation of costs for keeping the system running can be compared to the costs for rewriting.

7.2 Comparison to Criteria for success.

The expected outcome of the research into fourth generation language environments and software maintenance was to produce a means to:

- Identify a list of factors which influence the amount of software maintenance in a fourth generation language environment.
- Assess the importance of the identified factors and assign a weighting factor to them.
- Combine the weights for the factors to produce the first cut of an equation for the amount of software maintenance a system written in a fourth generation language will require.

7.2.1. Identification of Factors.

An analysis of the results of this survey showed that there were three software maintenance categories of fourth generation languages. The amount of code being maintained by each of these categories was significantly different. The means of these classes were manipulated to produce a language adjustment factor, which could be used to produce a normalised system size.

A number of factors which were considered may have a influence on the amount of software maintenance were examined and from the survey results this was reduced to a list of twelve, these being:

- Attributes of the system
  - Development effort in man months being adequate when compared to the estimate from the basic COCOMO organic model
  - Elapsed development time being adequate when compared to the estimate from the basic COCOMO organic model

- The number of user written programs in the system being less than 100

- Personnel attributes
  - The development staff having more than 6 months experience of the computer environment
  - The software maintenance staff having more than 6 months experience with the fourth generation language
  - Provision of adequate training for the fourth generation language
  - Adequate training for any third generation language
  - Drawing the software maintenance staff from the development staff
- Volatility
  - When the system is written building in flexibility for future changes
- Understandability of the source code
  - Always using meaningful names in the software
- Use of tools
  - The use of prototyping during the development of the system
  - Using tools during the development of the system

7.2.2. Assignment of Weights.

Weights were assigned to each of these factors, based on the mean of the amount of software being maintained per person month from the answers to the survey. The highest value being assigned a value of one, and the lower value assigned a weight greater than this. The value of these weights was shown in Table 6.2

7.2.3. Equation for the Estimation of Software Maintenance.

Section 5.2 showed that from the survey results obtained there were three distinct software maintenance categories of fourth generation languages and these together with the system size were used to produce a normalised system size (NSS). The weights for the identified factors, derived in section 6.2, when multiplied together produced an Effort Adjustment Factor (EAF). These were then be used in the equation:

Annual maintenance effort in person months =

Normalised System Size * function (Effort Adjustment Factor).

where:

Normalised System Size = System Size / Language Adjustment Factor

function(Effort Adjustment Factor) = ln(EAF) * 0.0002 + 0.0001

The actual equation developed from the survey results produced an estimate of the required annual software maintenance resources was therefore:

(System Size /Language Adjustment Factor) * (ln(EAF) * 0.0002 + 0.0001)

This equation produced estimates within 20% of the actual values for 80% of the cases examined from the survey.

7.3 Evaluation of Research.

In the previous section the research was compared to the criteria for success outlined in Chapter 1. This section takes a more objective view of the research and the results obtained. The research was a success because of the results obtained. However, the research has a number of weaknesses some of which could be corrected by further work. The weaknesses are as follows:

a) The form of the questionnaire was taken from that by Lientz and Swanson[LIENTZ80], and was modified for this research. It was clear from some of the returned questionnaires that some of the questions were either ambiguous or difficult to answer. This could have been cleared up by replacing questions, or by follow up telephone enquiries. At the front of every questionnaire was a definition of software maintenance, which was an attempt to force the respondents to use this so that similar factors were being compared. However, question 5 of the questionnaire asked which factors were included in software maintenance, and the answers varied widely with some including all the factors, and others only including those changes taking less than 5 days. These factors could account for some of the differences in the resources allocated to the software maintenance of different systems.

b) In the original survey questions to cross check with previous questions could have been included, this was not done as the questionnaire was already a considerable length, however, information as to how certain questions had been interpreted would have been obtained, an example of this is, the definition of adequate training in a third generation language in a fourth generation language system.

c) The sample population may have been in error because only medium to large business organisations were targeted. A sample covering the full range of businesses by size and activity would have had far more confidence in the results. This would highlight if large companies and small companies at in different ways when using fourth generation languages. Fourth generation languages, however, are not widely used in small companies and therefore the questionnaire was targeted at medium to large companies.

d) The number of returned questionnaires was small (26) leading to a small sample size for statistics. In all 47 replies were recorded, 8 did not use fourth generation languages, 9 gave no indication of system size and 4 gave no indication of resources allocated to software maintenance. A larger sample size would have given more confidence in the results, particularly as some of the classes being analysed contained only 1 or 2 responses. In all statistical research a larger sample size is better than a small one, however there are problems in obtaining the extra data. The original survey was sent to 235 businesses, but only 26 usable responses were received, but this could have been increased by 50% if the 9 had been contacted for an indication of system size and 4 for the resources for software maintenance. The lack of responses is however, almost always a major problem with postal surveys.

e) The main driver for comparison of systems and effort was lines of code. It was hoped to use a standard measure of system size independent of the language (e.g. function points). This is because the number of lines of code is not particularly relevant to some fourth generation languages. The questionnaire therefore asked for system sizes based on:

- Total lines of source code
- Total lines of executable code (excluding comments and declarations)
- Number of entity relationships, diagrams of similar (depending on CASE tool used)
- Function points
- Lines of job control
- Number of entries in the data dictionary

Most respondents used total lines of code as a measure which may have been interpreted differently by different respondents and a more precise definition of lines of code, together with examples could have been given. This, coupled with the different

verbosity of the languages surveyed, led to a wide range of values of lines of code. The natural solution to this was to normalise the data to equalise the differences caused by the use of different languages and this was successfully done and justified in Section 5.2 of this thesis. However, this normalisation process was based on a small sample and requires a much larger sample to validate the process.

f) The three groups of fourth generation language that resulted from the normalisation process on the lines of code should have been examined closely and compared with the classification of fourth generation languages carried out by Grindley[IDPM86]. The 3 groups obtained from the survey in section 5.2 (shown in table 5.5) relate to software maintenance and those specified by Grindley in section 2.5 are a development classification. The classification derived within this thesis may be a classification which may be related to the amount of code produced to accomplish a particular task whereas the Grindley classification may be a classification of the attributes of the tools used in development.

g) The validation of the results was conducted against a small sample size (9 of 14 returned questionnaires). This sample is too small, and a large sample would have given more credence to the results, again problems were caused by businesses not completing the questionnaire. The length of the questionnaire was reduced to try to solicit more responses, however, as explained in d) the lack of responses is almost always a major problem with postal surveys.

h) When calculating the equation in 6.3, because it was impossible for the formula to pass through every point exactly (some systems were the same size after normalisation, but used a different amount of software maintenance), the formula was therefore calculated to pass within 20% of as many points as possible. A different

formula would have been obtained by excluding certain points to make it exactly pass through as many points as possible, or if the outliers had been removed.

7.4 Further Research.

Further research can to be carried out in the following areas:

- The three groups of fourth generation languages identified in section 5.2 need to be considered to decide what factors of the language influence the grouping. This would enable a test to decide into which of the classes any specific fourth generation language would fall.
- These three software maintenance classes of fourth generation language need further research to consider whether the factors of the language which influence the amount of software maintenance could be incorporated into their design.
- Further validation of the 12 factors from the research by a survey and in-depth interviews to confirm that they are the only factors and that they do play a significant part in the required amount of software maintenance resources.
- Further validation of the weights assigned to the factors by a survey and in-depth interviews to determine the accuracy of the weights and enable them to be further refined.
- The equation was a first cut and was formulated from the respondents of the survey, and the results may be better for these cases than in the general case. Research, therefore needs to be carried out to find out if this equation holds for all fourth generation language systems.
- The results of this research need to be compared with third generation languages to determine whether fourth generation languages offer any maintenance savings over these traditional languages.

APPENDIX A

Critical Values for the t-test at the 5% Significance Level

Degrees of Freedom	Critical Value t_c	Degrees of Freedom	Critical Value t_c
1	12.706	21	2.080
2	4.303	22	2.074
3	3.182	23	2.069
4	2.776	24	2.064
5	2.571	25	2.060
6	2.447	26	2.056
7	2.365	27	2.052
8	2.306	28	2.048
9	2.262	29	2.045
10	2.228	30	2.042
11	2.201	31	2.040
12	2.179	32	2.037
13	2.160	33	2.035
14	2.145	34	2.032
15	2.131	35	2.030
16	2.120	36	2.028
17	2.110	37	2.026
18	2.101	38	2.024
19	2.093	39	2.023
20	2.086	40	2.021

APPENDIX B

Critical Values for the analysis of variance at the 5% significance Level

Degrees of freedom for between samples variance estimate

Degrees of freedom within samples	1	2	3	4	5	6	7	8	9
1	161.4	199.5	215.7	224.6	230.2	234.0	236.8	238.9	240.5
2	18.51	19.00	19.16	19.25	19.30	19.33	19.35	19.37	19.38
3	10.13	9.55	9.28	9.12	9.01	8.94	8.89	8.85	8.81
4	7.71	6.94	6.59	6.39	6.26	6.16	6.09	6.04	6.00
5	6.61	5.79	5.41	5.19	5.05	4.95	4.88	4.82	4.77
6	5.99	5.14	4.76	4.53	4.39	4.28	4.21	4.15	4.10
7	5.59	4.74	4.35	4.12	3.97	3.87	3.79	3.73	3.68
8	5.32	4.46	4.07	3.84	3.69	3.58	3.50	3.44	3.39
9	5.12	4.26	3.86	3.63	3.48	3.37	3.29	3.23	3.18
10	4.96	4.10	3.71	3.48	3.33	3.22	3.14	3.07	3.02
11	4.84	3.98	3.59	3.36	3.20	3.09	3.01	2.95	2.90
12	4.75	3.89	3.49	3.26	3.11	3.00	2.91	2.85	2.80
13	4.67	3.81	3.41	3.18	3.03	2.92	2.83	2.77	2.71
14	4.60	3.74	3.34	3.11	2.96	2.85	2.76	2.70	2.65
15	4.54	3.68	3.29	3.06	2.90	2.79	2.71	2.64	2.59
16	4.49	3.63	3.24	3.01	2.85	2.74	2.66	2.59	2.54
17	4.45	3.59	3.20	2.96	2.81	2.70	2.61	2.55	2.49
18	4.41	3.55	3.16	2.93	2.77	2.66	2.58	2.51	2.46
19	4.38	3.52	3.13	2.90	2.74	2.63	2.54	2.48	2.42
20	4.35	3.49	3.10	2.87	2.71	2.60	2.51	2.45	2.39
21	4.32	3.47	3.07	2.84	2.68	2.57	2.49	2.42	2.37
22	4.30	3.44	3.05	2.82	2.66	2.55	2.46	2.40	2.34
23	4.28	3.42	3.03	2.80	2.64	2.53	2.44	2.37	2.32
24	4.26	3.40	3.01	2.78	2.62	2.51	2.42	2.36	2.30
25	4.24	3.39	2.99	2.76	2.60	2.49	2.40	2.34	2.28
26	4.23	3.37	2.98	2.74	2.59	2.47	2.39	2.32	2.27
27	4.21	3.35	2.96	2.73	2.57	2.46	2.37	2.31	2.25
28	4.20	3.34	2.95	2.71	2.56	2.45	2.36	2.29	2.24
29	4.18	3.33	2.93	2.70	2.55	2.43	2.35	2.28	2.22
30	4.17	3.32	2.92	2.69	2.53	2.42	2.33	2.27	2.21
40	4.08	3.23	2.84	2.61	2.45	2.34	2.25	2.18	2.12
60	4.00	3.15	2.76	2.53	2.37	2.25	2.17	2.10	2.04
120	3.92	3.07	2.68	2.45	2.29	2.17	2.09	2.02	1.96
∞	3.84	3.00	2.60	2.37	2.21	2.10	2.01	1.94	1.88

Degrees of freedom for between samples variance estimate

Degrees of freedom within samples	10	12	15	20	24	30	40	60	120	∞
1	241.9	243.9	245.9	248.0	249.1	250.1	251.1	252.2	253.3	254.3
2	19.4	19.41	19.43	19.45	19.45	19.46	19.47	19.48	19.49	19.50
3	8.79	8.74	8.70	8.66	8.64	8.62	8.59	8.57	8.55	8.53
4	5.96	5.91	5.86	5.80	5.77	5.75	5.72	5.69	5.66	5.63
5	4.74	4.68	4.62	4.56	4.53	4.50	4.46	4.43	4.40	4.36
6	4.06	4.00	3.94	3.87	3.84	3.81	3.77	3.74	3.70	3.67
7	3.64	3.57	3.51	3.44	3.41	3.38	3.34	3.30	3.27	3.23
8	3.35	3.28	3.22	3.15	3.12	3.08	3.04	3.01	2.97	2.93
9	3.14	3.07	3.01	2.94	2.90	2.86	2.83	2.79	2.75	2.71
10	2.98	2.91	2.85	2.77	2.74	2.70	2.66	2.62	2.58	2.54
11	2.85	2.79	2.72	2.65	2.61	2.57	2.53	2.49	2.45	2.40
12	2.75	2.69	2.62	2.54	2.51	2.47	2.43	2.38	2.34	2.30
13	2.67	2.60	2.53	2.46	2.42	2.38	2.34	2.30	2.25	2.21
14	2.60	2.53	2.46	2.39	2.35	2.31	2.27	2.22	2.18	2.13
15	2.54	2.48	2.40	2.33	2.29	2.25	2.20	2.16	2.11	2.07
16	2.49	2.42	2.35	2.28	2.24	2.19	2.15	2.11	2.06	2.01
17	2.45	2.38	2.31	2.23	2.19	2.15	2.10	2.06	2.01	1.96
18	2.41	2.34	2.27	2.19	2.15	2.11	2.06	2.02	1.97	1.92
19	2.38	2.31	2.23	2.16	2.11	2.07	2.03	1.98	1.93	1.88
20	2.35	2.28	2.20	2.12	2.08	2.04	1.99	1.95	1.90	1.84
21	2.32	2.25	2.18	2.10	2.05	2.01	1.96	1.92	1.87	1.81
22	2.30	2.23	2.15	2.07	2.03	1.98	1.94	1.89	1.84	1.78
23	2.27	2.20	2.13	2.05	2.01	1.96	1.91	1.86	1.81	1.76
24	2.25	2.18	2.11	2.03	1.98	1.94	1.89	1.84	1.79	1.73
25	2.24	2.16	2.09	2.01	1.96	1.92	1.87	1.82	1.77	1.71
26	2.22	2.15	2.07	1.99	1.95	1.90	1.85	1.80	1.75	1.69
27	2.20	2.13	2.06	1.97	1.93	1.88	1.84	1.79	1.73	1.67
28	2.19	2.12	2.04	1.96	1.91	1.87	1.82	1.77	1.71	1.65
29	2.18	2.10	2.03	1.94	1.90	1.85	1.81	1.75	1.70	1.64
30	2.16	2.09	2.01	1.93	1.89	1.84	1.79	1.74	1.68	1.62
40	2.08	2.00	1.92	1.84	1.79	1.74	1.69	1.64	1.58	1.51
60	1.99	1.92	1.84	1.75	1.70	1.65	1.59	1.53	1.47	1.39
120	1.91	1.83	1.75	1.66	1.61	1.55	1.50	1.43	1.35	1.25
∞	1.83	1.75	1.67	1.57	1.52	1.46	1.39	1.32	1.22	1.00

APPENDIX C

Copy of Survey

SOFTWARE MAINTENANCE

Software maintenance is defined as work done on a software system after it becomes operational. Therefore software maintenance includes:

* understanding and documenting existing systems

* extending existing functions

* adding new functions

* finding and correcting bugs

* answering questions for users and operations staff

* rewriting, restructuring, converting and purging software

* managing the software of an operational system

* other activities which go into running a successful software system

Software maintenance is therefore a very broad activity that includes error corrections, enhancements of capabilities, deletion of obsolete capabilities, and optimisation.

CONFIDENTIAL SURVEY
ALL INFORMATION WILL BE TREATED IN THE STRICTEST CONFIDENCE
Name _____
Position in company _____ _____
Company name _____

1. In which industrial categories do the department, and the organisation served by the department, primarily belong? (Enter the one letter corresponding to the most appropriate answer.)

Manufacturing industries:

a. Aerospace engineering
b. Brick /Pottery/ Glass/ Cement etc.
c. Chemical/ Allied products
d. Computer hardware
e. Computer software
f. Construction and civil engineering
g. Food/ Drink/ Tobacco
h. Instruments/ Electrical/ Electronics
i. Mechanical engineering
j. Metal goods/ Machinery
k. Paper/ Paper products/ Printing/ Packaging
l. Petroleum/ Coal/ Rubber
m. Ship building and marine engineering
n. Textiles/ Leather goods/ Footwear/ Clothing
o. Timber/ Furniture
p. Transportation equipment/ Vehicles

Non-manufacturing industries:

q. Agriculture/ Forestry/ Fishing
r. Banking/ Credit agency
s. Business and professional services
t. Communications and information media
u. Distribution and associated trades
v. Education
w. Extraction and processing of natural resources
x. Health services
y. Insurance/ Assurance
z. Leisure and recreation services
aa. Public administration
ab. Transport/ Travel and supporting services
ac. Software house

Department: (Enter letter or specify other) _____
Organisation: (Enter letter or specify other _____

2. How many people are employed in the company? _____

3. How many people are employed in the DP department? _____

4. What is your role in the DP organisation?
Are you:
   a. Software maintainer
   b. responsible for the management of a software team
   c. both

5. Which of the following do you include in software maintenance?
   a. emergency repairs
   b. changes to correctly reflect the specifications or correctly utilise system resources
   c. upgrades to adapt to changes in processing requirements
   d. amendments to adapt to changes in regulations
   e. growth amendments to adapt to changes in data requirements, or to the addition of new programs, new users etc.
   f. enhancements for changed requirements
   g. support for users of the system
   h. changes which take less than 1 day
   i. changes which take less than 3 days
   j. changes which take less than 5 days
   k. changes which take less than 10 days
   l. changes which take less than 20 days

6. How are the software teams organised within the company?
   a. separate software maintenance and development teams
   b. mixed maintenance and development
   c. both policies

7. Do you use a fourth generation language (4GL) in your company?
YES/NO

   - If YES which one _____

   If YES how reliable is the fourth generation language itself?

Very Unreliable		Average		Very Reliable
<				>
1	2	3	4	5

8. Do you use a Computer Aided Software Engineering (CASE) tool in your company?

YES/NO

- If YES which one _____

9. Do you consider that your fourth generation language has reduced software maintenance over similar systems developed without these tools?

a. no - increased maintenance greatly
b. no - increased maintenance
c. about the same
d. yes - maintenance has reduced
e. yes - maintenance has reduced greatly

10. How would you describe the morale of the staff involved in software maintenance?

Very Bad		Average		Very Good
<				>
1	2	3	4	5

11. What training has been/ is provided specifically for:

	None	Insufficient	Adequate	More than adequate
Software Maintenance				
The Fourth Generation Language				
A Third Generation Language				

12. How does management see software maintenance?

a. Much less important than development
b. Less important than development
c. About the same
d. More important than development
e. Much more important than development

13. What is the strategy for the recruitment of maintenance programmers? e.g. are trainees used and then moved to development, etc.

a. staff recruited specifically for maintenance
b. trainees are used and then moved to development
c. other, specify _____

CHOOSE A SYSTEM DEVELOPED USING THE FOURTH GENERATION LANGUAGE

14. What is the name of the system? _____

15. What does the software do? _____

16. What type of application is it? e.g. financial? _____

17. How many user written programs are there in the system? _____

18. How long has the software been in operation? _____

19. In what mode is the fourth generation language used? e.g. systems programs only, use of user written programs, etc. _____

20. How reliable has your software been in the past?

HISTORY:

Very Unreliable		Average		Very Reliable	New System
< 1	2	3	4	5	6 >

LAST 12 MONTHS:

Very Unreliable		Average		Very Reliable	New System
< 1	2	3	4	5	6 >

- If this is a new system how reliable do you expect it to be in the next 12 months? (if possible to estimate)

Very Unreliable		Average		Very Reliable
< 1	2	3	4	5 >

21. How serious are the consequences of failure?

a. No real problem
b. Problems
c. Serious problems
d. Financial disaster
e. Life critical

22. System size - fill in the values where appropriate:

System		**Largest Program**
_____.	Total lines of source code	_____
_____.	Total lines executable code (excluding comments and declarations)	_____
_____.	Number of entity relationships, diagrams or similar (depending on the CASE tool used)	_____
_____.	Function points	_____
_____.	Lines of job control	_____
_____.	Number of entries in data dictionary	_____

23. What was the original timescale for the development of this system?

_____. Man months/ years

_____. Elapse time in months/ years

_____. Number of staff employed on the project

24. How many man months have been allocated to the software maintenance of this project:

Previous 12 months _____

Current 12 months _____

Next 12 months _____

25. What was the average (mean) ability level of the development staff at the time the software was developed?

Very Poor		Average		Very Good	Not Known
<					>
1	2	3	4	5	6

26. Are the staff involved in software maintenance:

Vastly Underworked Overworked		Average		Vastly
<				>
1	2	3	4	5

27. Is the time allocated for individual changes generally:

Very Inadequate		Average		Very Generous
<				>
1	2	3	4	5

28. What is the average (mean) ability level of the maintenance staff?

Very Poor		Average		Very Good
<				>
1	2	3	4	5

29.

	Less than 6 months	6 to 12 months	1 to 2 years	2 to 3 years	More than 3 years
a) Development staff: Before working on the project what was the average (mean) experience of:					
(i) The Fourth Generation Language					
(ii) The Computer Environment					
(iii) Type of Application					
(iv) Third Generation Language (e.g. COBOL)					
b) Maintenance Staff. What is the average (mean) experience of:					
(i) The Fourth Generation Language					
(ii) The Computer Environment					
(iii) Type of Application					
(iv) Third Generation Language (e.g. COBOL)					

30. Are the staff who developed the system the same people who are responsible for its maintenance?

a. None
b. Yes some of them
c. Yes all of them

31. When the system was developed was any flexibility for future changes built in?

None Average Lots

< _____ >

1 2 3 4 5

32. How many enhancements were included in the software in the last 12 months? (If this is a new system how many are expected in the next 12 months)

None Average Lots

< _____ >

1 2 3 4 5

33. Are any of the following used?

	YES	NO	SOME
Data dictionary			
Meaningful names in software			
Comments in source code			
Site standards			

34. If you use a data dictionary, what do you use it for? _____

35. How would you describe the overall understandability of the source code?

Very Muddled Average Very Clear

< _____ >

1 2 3 4 5

36. How would you describe the complexity of the tasks undertaken by the software?

Very Easy Average Very Complex

< _____ >

1 2 3 4 5

37. How would you describe the complexity of the code itself (as opposed to the tasks undertaken by it)?

Very Easy — Average — Very Complex

< _____ >

1 2 3 4 5

38. Do you use a measure of complexity?

YES/ NO

- If YES, Which _____

39. Were any methodologies used in development?

YES/ YES BUT NOT PURE/ NO

- If "YES" or "YES BUT NOT PURE" which methodologies? _____

40. Are any methodologies used in maintenance of the software?

YES/ YES BUT NOT PURE/ NO

- If "YES" or "YES BUT NOT PURE" which methodologies? _____

41. How good is the documentation for software maintenance?

Non-existent — Average — Excellent

< _____ >

1 2 3 4 5

42. What tools were used in the development of the software? _____

_____

43. What tools are used in software maintenance? _____

_____

44. Was the system prototyped when it was written?

YES/ NO

- If YES was it rewritten before being used in production?

YES/ NO

45. Are any major changes expected to the hardware in the next 12 months?

YES/ NO

- If YES, what _____

If so, will they require software changes?

YES/ NO

- If YES, what _____

APPENDIX D

Hypotheses and Results

All results discussed in this appendix were calculated by lines of code per person month and were analysed using the t-test. Any values which did not receive any answers in the surveys are excluded from the tables. Section 5.1 introduced the list of factors to be tested and the rationale behind the survey. A copy of the survey is shown at appendix C.

D.1 Attributes of System

a) The number of errors in the software.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the history of reliability of the system.

(ii) The amount of software maintenance required by a system is dependant upon the history of reliability of the system.

Rationale and Results

This factor was included because if a system has a history of unreliability there may be a resistance to change the software, or extra testing may be carried out. Question 20 of the survey asked for information concerning the history of reliability of the system.

The analysis of the results for this factor is shown in Table D.1.

Table D.1 Analysis of results of the history of reliability of the system.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	3	8	13	5	10	15
Critical Value t_c	3.182	2.306	2.160	2.571	2.228	2.131
Calculated Test Statistic	2.698	1.074	1.033	-2.096	-1.345	0.224

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) and therefore it does not appear that the history of reliability of the system has a significant effect on software maintenance resources.

Hypothesis

(iii) The amount of software maintenance required by a system is independent of the reliability of the system in the previous 12 months.

(iv) The amount of software maintenance required by a system is dependant upon the reliability of the system in the previous 12 months.

Rationale and Results

This factor is equivalent to the amount of corrective maintenance required by the system, which is usually referred to as 'bug fixing'. Question 20 of the survey asked for information concerning the number of errors discovered in the software.

The analysis of the results for the reliability of the system in the previous 12 months are shown in Table D.2.

Table D.2 Analysis of the results of the reliability of the system in the previous 12 months.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	1	5	15	6	16	20
Critical Value t_c	12.706	2.571	2.131	2.447	2.120	2.086
Calculated Test Statistic	0.723	0.802	0.180	-0.315	-0.783	-0.899

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (iii) in favour of (iv) indicating that the reliability of the system in the previous 12 months did not have a significant effect on the amount of software maintenance.

b) The age of the product.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the age of the system.

(ii) The amount of software maintenance required by a system is dependant upon the age of the system.

Rationale and Results

Question 18 of the survey asked for the age of the system. This factor was tested as during the life of a system it undergoes three main stages:

- after installation errors are being discovered and the users are undergoing training and are not used to the system

- main phase where the system is used
- breakdown of the system when it becomes ready for rewriting

The analysis of the results for the age of the system are shown in Table D.3.

Table D.3 Analysis of the age of the system.

	1 and 2	1 and 3	1 and 4	2 and 3	2 and 4	3 and 4
Degrees of Freedom	10	16	12	10	6	12
Critical Value t_c	2.228	2.120	2.179	2.228	2.447	2.179
Calculated Test Statistic	0.618	1.652	0.968	0.454	0.067	-0.584

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the age of the system does not have an significant effect on the amount of software maintenance.

c) The required reliability of the system.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the consequences of the failure of the system.

(ii) The amount of software maintenance required by a system is dependant upon the consequences of the failure of the system.

Rationale and Results

This factor was included because if the consequences of failure are more serious then more testing and checking will be carried out using more resources. Question 21 of the survey asked how serious the consequences the failure of the system are.

The analysis of the results of the consequences of failure of the system are shown in Table D.4.

Table D.4 Analysis of the consequences of failure of the system.

	2 and 3	2 and 4	3 and 4
Degrees of Freedom	22	10	14
Critical Value t_c	2.074	2.228	2.145
Calculated Test Statistic	0.766	0.737	0.725

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the required reliability of the system does not have a significant effect on the amount of software maintenance.

d) The complexity of the product

Hypothesis

(i) The amount of software maintenance required by a system is independent of the complexity of the tasks undertaken by it.

(ii) The amount of software maintenance required by a system is dependant upon the complexity of the tasks undertaken by it.

Rationale and Results

This factor was included as the more complex the tasks undertaken by the system the more software maintenance resources it will require. Question 36 of the survey asked for a rating of the complexity of the tasks undertaken by the system.

The analysis of these results are shown in Table D.5.

Table D.5 Analysis of the complexity of the tasks undertaken by the system.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	11	8	3	19	14	11
Critical Value t_c	2.201	2.306	3.182	2.093	2.145	2.201
Calculated Test Statistic	0.810	0.086	0.774	-1.510	0.104	1.122

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the complexity of the tasks undertaken by the system does not have a significant effect on the amount of software maintenance it requires.

e) The type of application.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the type of application.

(ii) The amount of software maintenance required by a system is dependant upon the type of application.

Rationale and Results

The rationale behind this factor is that different types of system require different amounts of maintenance. Question 16 of the survey asked for the type of application.

The analysis of the results for the type of factor are shown in Table D.6.

Table D.6 Analysis of the type of application

	1 and 2	1 and 3	1 and 4	2 and 3	2 and 4	3 and 4
Degrees of Freedom	4	17	7	15	5	18
Critical Value t_c	2.776	2.101	2.447	2.120	2.776	2.101
Calculated Test Statistic	-0.726	-0.138	-0.299	0.958	0.205	-0.442

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the type of application does not have a significant effect on the amount of software maintenance.

f) The size of the system

This is assumed to be an essential element relating to the amount of software maintenance, and in this research as in previous papers this is used as an element to enable the systems to be compared. Question 22 of the survey asked for information concerning the size of the system.

g) The original time scale for development

The factors that the research was trying to test was not what the original time scale for development was, but whether it was sufficient, or whether the development time was compressed, thus reducing development but increasing maintenance.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the development effort in person months compared to estimates of development effort.

(ii) The amount of software maintenance required by a system is dependant upon the development effort in person months compared to estimates of development effort.

Rationale and Results

Question 23 of the survey asked for information about the original timescale for development.

It was necessary to set a baseline against which to measure, and for this the Basic COCOMO model was chosen. The Basic model was used to avoid using any cost estimation factors which may introduce errors in the research. The organic mode of the model was used because it was considered that most of the software was developed in house, albeit that some of the departments were acting as a software house within the company. The model was not calibrated in any way due to the difference between third and fourth generation languages as the figure was used only as a baseline.

It is appreciated that the figures being used in this calculation are the actual size of the existing system, not that at the time it was written. This was thought appropriate as it is likely that if large parts were added to the system after it was used in production this could have an adverse effect on software maintenance. The COCOMO estimates for the

effort in person months and elapse time for development against the development time are shown in Table D.7.

Table D.7 Estimated development time against actual

D.S.I	COCOMO PM	COCOMO TDEV	ACTUAL PM	ACTUAL TDEV	% EST/ ACT PM	%EST/ ACT TDEV
2,500	6.28	5.03	14	8	44.87	62.82
35,000	100.34	14.40	660	18	15.20	80.03
685,000	2,278.70	27.19	120	36	1,898.91	131.09
45,000	130.64	15.92	12	6	1,088.69	265.40
27,400	77.60	13.06	8	2	969.97	653.21
1,200	2.91	3.75	3	3	96.88	124.99
85,400	256.00	20.56	72	12	355.56	171.35
100,000	302.14	21.90	15	4	2,014.28	547.47
71,000	210.88	19.10	36	12	585.77	159.18
800,000	2,681.98	50.21	360	24	744.99	209.19
48,000	139.80	16.34	6	9	2,330.04	181.55
1,000,000	3,390.09	54.88	216	10	1,569.49	548.81
155,540	480.45	26.12	24	6	2,001.86	435.33
88,000	264.19	20.81	51	9	518.02	231.22
2,000	4.97	4.60	4	6	124.23	76.63
40,000	155.44	15.19	24	6	481.02	253.22
400	0.92	2.42	6	2	15.28	120.95
265,200	841.32	32.32	600	12	140.22	269.31
51,000	148.99	16.74		48		34.87
48,620	141.70	16.42	14	6	1,012.14	273.72

Table D.7 Estimated development time against actual (continued).

D.S.I	COCOMO PM	COCOMO TDEV	ACTUAL PM	ACTUAL TDEV	% EST/ ACT PM	%EST/ ACT TDEV
12,500	34.04	9.55	8	4	425.48	238.8
30,000	85.35	13.55	216	36	39.51	37.63
15,000	41.22	10.27	24		171.75	
51,000	148.99	16.74		14		119.57

The table shows a wide range of percentages estimated/ actual person months, ranging from 15.20 to 2,330.04, these results could be caused because of the power of the language or large amounts of code have been added to the system since development, as COCOMO calculates the development time based on the delivered system size where the survey is based on current system size.

Table D.8 shows the analysis of the COCOMO model against the effort being equal to, or more than predicted.

Table D.8 Analysis of comparison between Basic COCOMO organic model estimated effort against actual.

	Less than 100% and 100% plus
Degrees of Freedom	20
Critical Value t_c	2.086
Calculated Test Statistic	-4.723

Discussion

These results show that software maintenance is increased if the actual development effort is less than that predicted by the Basic COCOMO model. This result could be interpreted, given the discussion of COCOMO above that fast development leads to increased maintenance.

Hypothesis

(iii) The amount of software maintenance required by a system is independent of the elapse time of development when compared to estimates of development time.

(iv) The amount of software maintenance required by a system is dependant upon the elapse time of development when compared to estimates of development time.

Rationale and Results

Table D.9 shows the analysis of the results of the actual time of development being equal to or more than that predicted by the Basic COCOMO model.

Table D.9 Analysis of comparison between Basic COCOMO organic model development elapse time against actual.

	Less than 100% and 100% plus
Degrees of Freedom	18
Critical Value t_c	2.101
Calculated Test Statistic	-2.366

Discussion

These results show that software maintenance is increased if the actual elapsed time is less than that predicted by the basic COCOMO model. This result could also be interpreted as fast development leads to increased maintenance.

h) The reliability of the underlying system

Hypothesis

(i) The amount of software maintenance required by a system is independent of the reliability of the fourth generation language.

(ii) The amount of software maintenance required by a system is dependant upon the reliability of the fourth generation language.

Rationale and Results

This factor was included because if the fourth generation language itself is unreliable extra software maintenance resources may be required to keep the system running. Question 7 of the survey asked for the reliability of the fourth generation language.

The analysis of the reliability of the fourth generation language are shown in Table D.10.

Table D.10 Analysis of results of the reliability of the fourth generation language.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	5	8	8	13	13	16
Critical Value t_c	2.571	2.306	2.306	2.160	2.160	2.120
Calculated Test Statistic	-0.284	-0.606	-0.683	-1.093	-0.850	0.616

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the reliability of the fourth generation language does not have a significant effect on the amount of software maintenance.

i) The number of user written programs

Hypothesis

(i) The amount of software maintenance required by a system is independent of the reliability of the number of user written programs.

(ii) The amount of software maintenance required by a system is dependant upon the number of user written programs.

Rationale and Results

The rationale behind this factor is that software maintainers give advice to users for their programs and this time is recorded against software maintenance. Question 17 of the survey asked for the number of user written programs.

The analysis of systems with less than 100 user written programs and more than 100 are shown in Table D.11.

Table D.11 Analysis of the number of user written programs.

	Less than 100 and 100 plus
Degrees of Freedom	24
Critical Value t_c	2.064
Calculated Test Statistic	3.651

Discussion

These results show that software maintenance is increased if the system contains more than 100 user written programs.

D.2 Personnel Attributes.

a) Staff ability.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the mean ability level of the staff at the time the system was developed.

(ii) The amount of software maintenance required by a system is dependant upon the mean ability level of the staff at the time the system was developed.

Rationale and Results

This factor has been included because it is assumed that a good development programmer can produce a more maintainable system than a bad one. Question 25 of the survey asked for a rating of the ability of the development staff.

The analysis of the results for the average ability level of the development staff at the time the system was developed is shown in Table D.12.

Table D.12 Analysis of results of average ability level of the staff at the time the system was developed.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	11	9	4	18	13	11
Critical Value t_c	2.201	2.262	2.776	2.101	2.160	2.201
Calculated Test Statistic	0.621	-0.099	-0.259	-1.286	-1.531	-0.241

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the ability level of the development staff does not have a significant effect on the amount of software maintenance.

Hypothesis

(iii) The amount of software maintenance required by a system is independent of the mean ability level of the software maintenance staff.

(iv) The amount of software maintenance required by a system is dependant upon the mean ability level of the software maintenance staff.

Rationale and Results

The rationale behind this factor is that a good software maintenance programmer will perform software maintenance more effectively and quicker than a poor one. Question 28 of the survey asked for information concerning the ability of the software maintenance staff.

The analysis of the results for the average ability level of the software maintenance staff is shown in Table D.13.

Table D.13 Analysis of results of the average ability level of the software maintenance staff.

	2 and 3	2 and 4	3 and 4
Degrees of Freedom	19	11	16
Critical Value t_c	2.093	2.201	2.120
Calculated Test Statistic	-0.671	0.613	1.084

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the ability level of the software maintenance staff does not have a significant effect on the amount of software maintenance required by the system.

b) Morale and motivation of staff.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the morale of the software maintenance staff.

(ii) The amount of software maintenance required by a system is dependant upon the morale of the software maintenance staff.

Rationale and Results

This factor was included because it is possible that happy and motivated staff produce better quality software quicker than badly motivated ones. Question 10 of the survey asked for information concerning the morale and motivation of software maintenance staff.

The analysis of the results for the morale of the software maintenance staff is shown in Table D.14.

Table D.14 Analysis of results of the morale of the software maintenance staff.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	12	11	3	19	11	10
Critical Value t_c	2.179	2.201	3.182	2.093	2.201	2.228
Calculated Test Statistic	0.404	-0.109	-0.448	-0.603	-0.580	-0.063

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) showing that the morale and motivation of the staff does not have a significant effect on the amount of software maintenance a system requires.

c) The experience of the staff.

Various factors are included for experience of staff and all of them are based on the premise that staff with experience will produce a better product. The experience level is measured for both development and maintenance staff for:

- The fourth generation language
- The computer environment
- The type of application
- Any third generation language - this is included because experience of any third generation language can lead to a particular way of thinking which causes problems with the defaults of a fourth generation language. Question 29 of the survey asked for a rating of the experience of both development and maintenance staff in these areas.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the experience of the development staff with the fourth generation language.

(ii) The amount of software maintenance required by a system is dependant upon the experience of the development staff with the fourth generation language.

Rationale and Results

Table D.15 shows the analysis of the results for the experience of the development staff with the fourth generation language.

Table D.15 Analysis of results of the experience of the software maintenance staff with the fourth generation language.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	11	18	12	7	1	8
Critical Value t_c	2.201	2.101	2.179	2.365	12.706	2.306
Calculated Test Statistic	0.849	0.966	0.587	-0.602	-0.849	0.543

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the experience of the software maintenance staff with the fourth generation language does not have a significant effect on software maintenance.

Hypothesis

(iii) The amount of software maintenance required by a system is independent of the experience of the development staff with the computer environment.

(iv) The amount of software maintenance required by a system is dependant upon the experience of the development staff with the computer environment.

Rationale and Results

Table D.16 shows the analysis of the results for the experience of the development staff with the computer environment.

Table D.16 Analysis of the development staff with the computer environment at the time the system was written.

	1&2	1&3	1&4	1&5	2&3	2&4	2&5	3&4	3&5	4&5
Degrees of Freedom	6	7	4	10	7	4	10	5	11	8
Critical Values t_c	2.447	2.365	2.776	2.228	2.365	2.774	2.228	2.571	2.201	2.306
Calculated Test Statistic	-4.411	-3.302	-3.361	-4.059	-0.456	0.927	0.498	0.917	1.501	-0.221

The results show a difference between those systems where the developers had less than 6 months experience, and those systems where the developers had more than 6 months experience, and an analysis of just these values is shown in Table D.17.

Table D.17 Analysis of development staff with less than 6 months experience compared with more.

	1 and 2	1 and 3	1 and 4	1 and 5
Degrees of Freedom	6	11	13	21
Critical Value t_c	2.447	2.201	2.160	2.080
Calculated Test Statistic	-4.411	-3.824	-3.750	-3.820

Discussion

These results show that software maintenance is reduced if the staff maintaining the system have at least 6 months experience of the computer environment.

Hypothesis

(v) The amount of software maintenance required by a system is independent of the experience of the development staff with the type of application.

(vi) The amount of software maintenance required by a system is dependant upon the experience of the development staff with the type of application.

Rationale and Results

The analysis of the results for the experience of the development staff with the application at the time the system was written are shown in Table D.18.

Table D.18 Analysis of the development staff with the type of application at the time the system was written.

	1&2	1&3	1&4	1&5	2&3	2&4	2&5	3&4	3&5	4&5
Degrees of Freedom	8	9	4	6	11	6	8	7	9	4
Critical Values t_c	2.306	2.262	2.776	2.447	2.201	2.447	2.306	2.365	2.262	2.776
Calculated Test Statistic	-0.117	0.135	-0.198	-2.175	0.240	-0.023	-0.764	-0.176	-1.026	-1.983

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (v) in favour of (vi) suggesting that the experience of the development staff with the type of application does not have a significant effect on the amount of software maintenance.

Hypothesis

(vii) The amount of software maintenance required by a system is independent of the experience of the development staff with any third generation language.

(viii) The amount of software maintenance required by a system is dependant upon the experience of the development staff with any third generation language.

Rationale and Results

Table D.19 shows the analysis of the results for the experience of the development staff with any third generation language.

Table D.19 Analysis of results of the experience of the development staff with any third generation language at the time the system was developed.

	1 and 3	1 and 4	1 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	1	6	10	7	11	16
Critical Value t_c	12.706	2.447	2.228	2.365	2.201	2.120
Calculated Test Statistic	-0.044	0.066	0.192	0.160	0.353	0.142

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (vii) in favour of (viii) suggesting that the third generation language experience of the development staff does not have a significant effect on the amount of software maintenance.

Hypothesis

(ix) The amount of software maintenance required by a system is independent of the experience of the software maintenance staff with the fourth generation language.

(x) The amount of software maintenance required by a system is dependant upon the experience of the software maintenance staff with the fourth generation language.

Rationale and Results

Table D.20 shows the analysis of the results for the experience of the software maintenance staff with the fourth generation language.

Table D.20 Analysis of the software maintenance staff with the fourth generation language.

	1&2	1&3	1&4	1&5	2&3	2&4	2&5	3&4	3&5	4&5
Degrees of Freedom	12	8	8	5	10	10	7	6	3	3
Critical Values t_c	2.179	2.306	2.306	2.571	2.228	2.228	2.365	2.447	3.182	3.182
Calculated Test Statistic	-2.488	-1.416	-1.141	-0.926	0.873	1.396	0.410	0.520	-0.079	-0.681

The results show a difference between less than 6 months and 6 to 12 months. Table D.21 shows an analysis of less than 6 months experience compared with the other values.

Table D.21 Analysis of systems where the software maintainers have less than 6 months experience of the fourth generation language.

	1 and 2	1 and 3	1 and 4	1 and 5
Degrees of Freedom	12	16	20	21
Critical Values t_c	2.179	2.120	2.086	2.080
Calculated Test Statistic	-2.488	-2.394	-2.298	-2.354

Discussion

The results show that software maintenance is reduced if the software maintenance staff have at least 6 months experience of the fourth generation language.

Hypothesis

(xi) The amount of software maintenance required by a system is independent of the experience of the software maintenance staff with the computer environment.

(xii) The amount of software maintenance required by a system is dependant upon the experience of the software maintenance staff with the computer environment.

Rationale and Results

Table D.22 shows the analysis of the results for the experience of the software maintenance staff with the computer environment.

Table D.22 Analysis of results of the software maintenance staff with the computer environment.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	7	8	10	9	11	12
Critical Value t_c	2.365	2.306	2.228	2.262	2.201	2.179
Calculated Test Statistic	1.729	0.209	1.274	-1.207	-1.328	0.594

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (xi) in favour of (xii) suggesting that the experience of the software maintenance staff with the computer environment does not have a significant effect on the amount of software maintenance.

Hypothesis

(xiii) The amount of software maintenance required by a system is independent of the experience of the software maintenance staff with the type of application.

(xiv) The amount of software maintenance required by a system is dependant upon the experience of the software maintenance staff with the type of application.

Rationale and Results

Table D.23 shows the analysis of the results for the experience of the software maintenance staff with the type of application.

Table D.23 Analysis of results of software maintenance staff with the type of application.

	1 and 2	1 and 3	1 and 4	2 and 3	2 and 4	3 and 4
Degrees of Freedom	9	10	10	9	9	10
Critical Value t_c	2.262	2.228	2.228	2.262	2.262	2.228
Calculated Test Statistic	0.242	0.292	-0.139	-0.117	-0.315	-0.439

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (xiii) in favour of (xiv) suggesting that the experience of the software maintenance staff with the type of application does not have a significant effect on the level of software maintenance.

Hypothesis

(xv) The amount of software maintenance required by a system is independent of the experience of the software maintenance staff with any third generation language.

(xvi) The amount of software maintenance required by a system is dependant upon the experience of the software maintenance staff with any third generation language.

Rationale and Results

The analysis of the results for the experience of the software maintenance staff with any third generation language are shown in Table D.24.

Table D.24 Analysis of results of the software maintenance staff with any third generation language.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	1	7	10	8	11	17
Critical Value t_c	12.706	2.365	2.228	2.306	2.201	2.110
Calculated Test Statistic	0.485	0.134	0.549	-0.674	-0.480	0.675

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (xv) in favour of (xvi) suggesting that the experience of the software maintenance staff with any third generation language does not have a significant effect on the level of software maintenance.

d) The training of staff.

The training of staff is included because it is assumed that someone who has received an adequate level of training will produce a better product than someone who has received no or little training. Training was separated into three factors:

- specifically for software maintenance
- for the fourth generation language
- for any third generation language

Question 11 of the survey asked for information concerning the training of staff in these areas.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the level of training provided specifically for software maintenance.

(ii) The amount of software maintenance required by a system is dependant upon the level of training provided specifically for software maintenance.

Rationale and Results

Table D.25 shows the analysis of the results for training provided specifically for software maintenance.

Table D.25 Analysis of the level of training provided specifically for software maintenance.

	1 and 2	1 and 3	2 and 3
Degrees of Freedom	11	19	14
Critical Value t_c	2.201	2.093	2.145
Calculated Test Statistic	1.176	0.883	-0.944

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the level of training provided specifically for software maintenance does not have a significant effect on the level of software maintenance.

Hypothesis

(iii) The amount of software maintenance required by a system is independent of the level of training provided for the fourth generation language.

(iv) The amount of software maintenance required by a system is dependant upon the level of training provided for the fourth generation language.

Rationale and Results

The analysis of the results for the level of training provided for the fourth generation language are shown in Table D.26.

Table D.26 Analysis of results of the level of training for the fourth generation language.

	1 and 2	1 and 3	1 and 4	2 and 3	2 and 4	3 and 4
Degrees of Freedom	2	15	4	17	6	19
Critical Value t_c	4.303	2.131	2.776	2.110	2.447	2.093
Calculated Test Statistic	0.649	-1.449	-2.191	-2.939	-4.380	-0.547

The results show a difference between insufficient and adequate and insufficient and more than adequate. Table D.27 shows analysis of the comparison between none/ less than adequate and at least adequate.

Table D.27 Comparison of level of training for the fourth generation language being at least adequate.

Degrees of Freedom	24
Critical Value t_c	2.064
Calculated Test Statistic	-3.658

Discussion

The results show that software maintenance is reduced if at least adequate training is provided for the fourth generation language.

Hypothesis

(v) The amount of software maintenance required by a system is independent of the level of training provided for any third generation language.

(vi) The amount of software maintenance required by a system is dependant upon the level of training provided for any third generation language.

Rationale and Results

The analysis of the results for training provided for any third generation language are shown in Table D.28.

Table D.28 Analysis of results of the level of training for any third generation language.

	1 and 2	1 and 3	1 and 4	2 and 3	2 and 4	3 and 4
Degrees of Freedom	1	19	4	18	3	21
Critical Value t_c	12.706	2.093	2.776	2.101	3.182	2.080
Calculated Test Statistic	-1.042	-2.608	-3.146	-1.440	-1.917	-0.723

The results show a difference between none and adequate. Table D.29 shows the comparison of none and insufficient and at least adequate.

Table D.29 Training provided for any third generation language none/ insufficient and at least adequate.

Degrees of Freedom	24
Critical Value t_c	2.064
Calculated Test Statistic	-3.173

Discussion

The results show that software maintenance is reduced if at least adequate training is provided for any third generation language.

e) The Change of staff.

Hypothesis

(i) The amount of software maintenance required by a system is independent of whether the staff maintaining the system are those who developed it.

(ii) The amount of software maintenance required by a system is dependant upon whether the staff maintaining the system are those who developed it.

Rationale and Results

This factor is included because it is assumed that the staff who developed the system would have some knowledge of the system which would enable the software maintenance to be done more efficiently. Question 30 of the survey asked for this information.

Table D.30 shows the analysis of the results for whether the staff who are responsible for the maintenance of the system were involved in its development.

Table D.30 Analysis of whether the staff who developed the system are those who are maintaining it.

	1 and 2	1 and 3	2 and 3
Degrees of Freedom	18	14	14
Critical Value t_c	2.101	2.145	2.145
Calculated Test Statistic	0.750	-1.369	-3.126

The results show a difference between some and all. Table D.31 shows the analysis of a comparison between All and none/ some.

Table D.31 Analysis of whether all the staff maintaining the system were involved in its development

Degrees of Freedom	22
Critical Value t_c	2.074
Calculated Test Statistic	-2.094

Discussion

The results show that software maintenance is reduced if all the people who maintain the system were involved in its development.

f) Whether the staff are overworked or underworked.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the workload of the software maintenance staff.

(ii) The amount of software maintenance required by a system is dependant upon the workload of the software maintenance staff.

Rationale and Results

This factor was included as this obviously affects the output of a software maintainer. Question 26 of the survey asked for information concerning whether the software maintenance staff are overworked or underworked.

Table D.32 shows the analysis of the results for the workload of the software maintenance staff.

Table D.32 Analysis of results of the workload of the software maintenance staff.

	1 and 3	1 and 4	1 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	7	13	1	20	8	14
Critical Value t_c	2.365	2.160	12.706	2.086	2.306	2.145
Calculated Test Statistic	0.358	0.188	0.667	0.014	1.575	1.010

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the workload of the software maintenance staff does not have a significant effect on the level of software maintenance.

g) The time allowed for individual changes.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the time allowed for individual changes.

(ii) The amount of software maintenance required by a system is dependant upon the time allowed for individual changes.

Rationale and Results

This factor was included because if changes are done quickly testing and checking is not adequately done thereby increasing software maintenance. Question 27 of the survey asked for information on whether the time allowed for individual changes was adequate.

Table D.33 shows the analysis of the results for the time allowed for individual changes.

Table D.33 Analysis of time allowed for individual changes.

	2 and 3	2 and 4	3 and 4
Degrees of Freedom	20	7	19
Critical Value t_c	2.086	2.365	2.093
Calculated Test Statistic	0.037	0.153	0.074

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the time allowed for individual changes does not have a significant influence on the amount of software maintenance.

D.3 Volatility.

a) Built in flexibility.

Hypothesis

(i) The amount of software maintenance required by a system is independent of whether any flexibility for future changes was built into the system at the time it was written.

(ii) The amount of software maintenance required by a system is dependant upon whether any flexibility for future changes was built into the system at the time it was written.

Rationale and Results

This factor was included as it is possible to build flexibility into a system to enable certain future changes to be made more quickly. Question 31 of the survey asked for information on whether any flexibility for future changes had been built in.

Table D.34 shows the analysis of the results for the flexibility at the time the system was written.

Table D.34 Analysis of the amount of flexibility for future changes.

	1&2	1&3	1&4	1&5	2&3	2&4	2&5	3&4	3&5	4&5
Degrees of Freedom	6	4	14	3	6	16	5	14	3	13
Critical Values t_c	2.447	2.776	2.145	3.182	2.447	2.120	2.571	2.145	3.182	2.160
Calculated Test Statistic	1.097	-0.401	-1.066	-1.756	-1.770	-2.629	-3.414	-0.676	-1.788	-0.911

The results show a difference between the category "minor" and categories "some" and "lots", an analysis of these results is shown in Table D.35.

Table D.35 Analysis of whether the flexibility for future changes was more than average.

	Yes and No
Degrees of Freedom	24
Critical Value t_c	2.064
Calculated Test Statistic	-2.735

Discussion

The results show that software maintenance is reduced if more than average flexibility for changes is built into the system at the time it is written.

b) The number of enhancements.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the number of enhancements included in the system in the previous 12 months.

(ii) The amount of software maintenance required by a system is dependant upon the number of enhancements included in the system in the previous 12 months.

Rationale and Results

User enhancements use the largest amount of software maintenance resources under third generation languages and therefore this factor was tested for fourth generation languages. Question 32 of the survey asked for information concerning the number of user enhancements in the system.

The analysis of the results for the number of enhancements included in the system in the previous 12 months is shown in Table D.36.

Table D.36 Analysis of the number of enhancements included in the last 12 months.

	1&2	1&3	1&4	1&5	2&3	2&4	2&5	3&4	3&5	4&5
Degrees of Freedom	3	5	6	7	8	9	10	11	12	13
Critical Values t_c	3.182	2.571	2.447	2.365	2.306	2.262	2.228	2.201	2.179	2.160
Calculated Test Statistic	-0.145	0.674	0.046	0.453	1.044	0.410	0.905	-0.993	-0.506	0.663

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the number of user enhancements does not significantly effect the amount of software maintenance with fourth generation languages. This was a surprising result and may have been caused because the majority of the organisations were software houses or were operating as software houses and may therefore exclude user enhancements from the definition of software maintenance under their contracts.

c) Expected hardware changes.

Hypothesis

(i) The amount of software maintenance required by a system is independent of expected hardware changes.

(ii) The amount of software maintenance required by a system is dependant upon expected hardware changes.

Rationale and Results

This factor was included because fourth generation languages tend to be machine specific and therefore hardware changes may mean changes to the system. Question 45 of the survey asked for information concerning expected hardware changes.

Table D.37 shows the analysis of the results for whether hardware changes were expected.

Table D.37 Analysis of whether any hardware changes were expected.

	Yes and No
Degrees of Freedom	24
Critical Value t_c	2.064
Calculated Test Statistic	-0.385

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that hardware changes does not have a significant effect on the amount of software maintenance.

D.4 Understandability of the Source Code.

a) Documentation.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the quality of the documentation.

(ii) The amount of software maintenance required by a system is dependant upon the quality of the documentation.

Rationale and Results

In most surveys where software maintainers are asked what they need to improve software maintenance the most common answer is better documentation. Question 41 of the survey asked for information concerning the quality of the documentation.

The analysis of the results for the quality of the documentation are shown in Table D.38.

Table D.38 Analysis of results of the quality of the documentation.

	1 and 2	1 and 3	1 and 4	2 and 3	2 and 4	3 and 4
Degrees of Freedom	5	10	8	13	11	16
Critical Value t_c	2.571	2.228	2.306	2.160	2.201	2.120
Calculated Test Statistic	-0.892	0.536	-1.051	-0.190	-0.537	-0.141

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that software maintenance does not have a significant effect on the level of software maintenance. This is a surprising result but could be explained by a number of reasons including:

- fourth generation languages are easier to understand and therefore software maintenance requires less documentation than with third generation languages
- programmers have found alternatives to documentation because they expect it to be inaccurate.

- the documentation is used to enable the change to be made more quickly, but there is an overhead keeping it up to date.

b) Programming style, methodologies and site standards.

The rationale behind these factors is that they improve the understandability of the software, thus making maintenance easier.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the use of site standards.

(ii) The amount of software maintenance required by a system is dependant upon the use of site standards.

Rationale and Results

Question 33 of the survey asked for information concerning the use of site standards, and Table D.39 shows the analysis of the results.

Table D.39 Analysis of the use of site standards.

	1 and 2	1 and 3	2 and 3
Degrees of Freedom	22	21	3
Critical Value t_c	2.074	2.080	3.182
Calculated Test Statistic	-0.695	-0.684	0.023

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the use of site standards does not have a significant effect on the amount of software maintenance.

Hypothesis

(iii) The amount of software maintenance required by a system is independent of the use of comments in the source code.

(iv) The amount of software maintenance required by a system is dependant upon the use of comments in the source code.

Rationale and Results

Question 33 of the survey asked for information concerning the use of comments in the source code and the analysis of the results is are shown in Table D.40.

Table D.40 Analysis of the use of comments in the source code.

	1 and 2	1 and 3	2 and 3
Degrees of Freedom	23	21	2
Critical Value t_c	2.069	2.080	4.303
Calculated Test Statistic	0.816	-0.072	-0.584

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (iii) in favour of (iv) suggesting that the use of comments in the source code do not have a significant effect on the amount of software maintenance.

Hypothesis

(v) The amount of software maintenance required by a system is independent of the use meaningful names in the software.

(vi) The amount of software maintenance required by a system is dependant upon the use of meaningful names in the software.

Rationale and Results

Question 33 of the survey asked for information concerning the use of meaningful names in the software and Table D.41 shows the analysis of the results.

Table D.41 Analysis of the use of meaningful names in the source code.

	1 and 2	1 and 3	2 and 3
Degrees of Freedom	19	20	5
Critical Value t_c	2.093	2.086	2.571
Calculated Test Statistic	3.069	2.217	-1.300

These results show a difference between Yes and the other values. Table D.42 shows the analysis of a comparison of Yes and the other values.

Table D.42 Analysis of Meaningful Names Used in the Software.

Degrees of Freedom	23
Critical Value t_c	2.069
Calculated Test Statistic	3.526

Discussion

The results show that software maintenance is reduced if meaningful names are always used in the software.

Hypothesis

(vii) The amount of software maintenance required by a system is independent of the use of a methodology in the development of the system.

(viii) The amount of software maintenance required by a system is dependant upon the use of a methodology in the development of the system.

Rationale and Results

The analysis of the results for the use of a methodology in the development of the system are shown in Table D.43 which were derived from Question 39 of the survey.

Table D.43 Analysis of the use of a methodology in the development of the system.

	1 and 2	1 and 3	2 and 3
Degrees of Freedom	13	14	19
Critical Value t_c	2.160	2.145	2.093
Calculated Test Statistic	0.378	-0.545	-1.020

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (vii) in favour of (viii) suggesting that the use of a methodology in the development of a system does not have a significant influence on the level of software maintenance.

Hypothesis

(ix) The amount of software maintenance required by a system is independent of the use of a methodology in the maintenance of the system.

(x) The amount of software maintenance required by a system is dependant upon the use of a methodology in the maintenance of the system.

Rationale and Results

Question 40 of the survey asked about the use of a methodology in the maintenance of the system and Table D.44 shows the analysis of these results.

Table D.44 Analysis of the use of a methodology in software maintenance.

	1 and 2	1 and 3	2 and 3
Degrees of Freedom	5	21	20
Critical Value t_c	2.571	2.080	2.086
Calculated Test Statistic	-0.221	-0.205	-0.061

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (ix) in favour of (x) suggesting that the use of a methodology during the maintenance of a system does not significantly influence the amount of software maintenance.

Hypothesis

(xi) The amount of software maintenance required by a system is independent of the understandability of the source code.

(xii) The amount of software maintenance required by a system is dependant upon the understandability of the source code.

Rationale and Results

The analysis of the results for the understandability of the source code are shown in Table D.45 which were derived from question 35.

Table D.45 Analysis of results of the understandability of the source code.

	2 and 3	2 and 4	2 and 5	3 and 4	3 and 5	4 and 5
Degrees of Freedom	11	13	4	18	9	11
Critical Value t_c	2.201	2.160	2.776	2.101	2.262	2.201
Calculated Test Statistic	-1.279	-1.170	-0.513	0.812	0.550	0.271

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (xi) in favour of (xii) suggesting that the understandability of the source code does not significantly influence the amount of software maintenance.

c) Complexity of the source code.

Hypothesis

(i) The amount of software maintenance required by a system is independent of the complexity of the code.

(ii) The amount of software maintenance required by a system is dependant upon the complexity of the code.

Rationale and Results

Question 37 of the survey asked for information concerning the complexity of the code, and Table D.46 shows the analysis of the results.

Table D.46 Analysis of the complexity of the source code.

	1&2	1&3	1&4	1&5	2&3	2&4	2&5	3&4	3&5	4&5
Degrees of Freedom	1	7	12	4	6	11	3	17	9	14
Critical Values t_c	12.706	2.365	2.179	2.776	2.447	2.201	3.182	2.110	2.262	2.145
Calculated Test Statistic	3.853	0.149	0.949	0.643	-0.962	-1.861	-1.430	0.589	0.291	-0.125

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the complexity of the source code does not have a significant effect on the level of software maintenance.

D.5 Management.

a) Management attitude to software maintenance.

Hypothesis

(i) The amount of software maintenance required by a system is independent of management's view of software maintenance as opposed to their view of development.

(ii) The amount of software maintenance required by a system is dependant upon management's view of software maintenance as opposed to their view of development.

Rationale and Results

Management's attitude to software maintenance can have a significant effect on the software maintainers view of their tasks, and thus influence the maintainers attitude to their tasks and reduce their effectiveness.

Table D.47 shows the analysis of the results for management's view of software maintenance as opposed to their view of development taken from question 12 of the survey.

Table D.47 Analysis of results of management's attitude to software maintenance.

	a and b	a and c	a and d	b and c	b and d	c and d
Degrees of Freedom	7	10	4	17	11	14
Critical Value t_c	2.365	2.228	2.776	2.110	2.201	2.145
Calculated Test Statistic	-1.030	-1.657	-2.157	-1.972	-0.453	1.409

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that management's attitude to software maintenance does not significantly influence the amount of software maintenance.

b)The resources allocated to software maintenance.

Question 23 asked for the resources allocated to software maintenance. This is assumed to be an essential element relating to the amount of software maintenance, and in this research as in previous papers this is used as an element to enable the systems to be compared.

D.6 Use of Tools.

a) The use of prototyping.

Hypothesis

(i) The amount of software maintenance required by a system is independent of whether the system was prototyped during development.

(ii) The amount of software maintenance required by a system is dependant upon whether the system was prototyped during development.

Rationale and Results

The use of prototyping can significantly reduce errors introduced in the early stages of the development life cycle. Question 44 of the survey asked for information concerning the use of prototyping.

The analysis of the results for the use of prototyping is shown in Table D.48.

Table D.48 Analysis of the use of prototyping.

	Yes and No
Degrees of Freedom	24
Critical Value t_c	2.064
Calculated Test Statistic	2.419

Discussion

The results show that software maintenance is reduced if the system is prototyped.

Hypothesis

(iii) The amount of software maintenance required by a system is independent of whether the prototype of the system was rewritten before being used in production.

(iv) The amount of software maintenance required by a system is dependant upon whether the prototype of the system was rewritten before being used in production.

Rationale and Results

Prototyping can lead to parts of the system developed independently and added to the system at a later date which can make software maintenance difficult, however because the user is involved at all stages of analysis the quality of the completed system can be greatly improved.

In all the cases surveyed the prototype was never rewritten before being used in production, therefore hypothesis (iii) cannot be rejected in favour of (iv).

b) The use of a data dictionary.

Hypothesis

(i) The amount of software maintenance required by a system is independent of whether a data dictionary is used in the system.

(ii) The amount of software maintenance required by a system is dependant upon whether a data dictionary is used by the system.

Rationale and Results

The use of a data dictionary can reduce the amount of software maintenance as the dictionary can be changed and the programs recompiled, instead of making all the

changes individually. Question 33 of the survey asked whether a data dictionary was used on the system.

The analysis of the results for the use of a data dictionary are shown in Table D.49.

Table D.49 Analysis of the use of a data dictionary.

	Yes and No
Degrees of Freedom	23
Critical Value t_c	2.069
Calculated Test Statistic	1.401

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (i) in favour of (ii) suggesting that the use of a data dictionary does not have a significant influence on the level of software maintenance.

c) The use of tools.

Hypothesis

(i) The amount of software maintenance required by a system is independent of whether tools were used in the development of the system.

(ii) The amount of software maintenance required by a system is dependant upon whether tools were used in the development of the system.

Rationale and Results

This factor was included because the use of tools can reduce the amount of software maintenance.

Question 42 of the survey asked for information concerning the use of tools in the development of the system and Table D.50 shows the analysis of the results.

Table D.50 Analysis of the use of tools in the development in the software.

	Yes and No
Degrees of Freedom	21
Critical Value t_c	2.080
Calculated Test Statistic	2.164

Discussion

All of these results show that software maintenance is reduced if tools were used in the development of the system.

Hypothesis

(iii) The amount of software maintenance required by a system is independent of whether tools are used in the maintenance of the system.

(iv) The amount of software maintenance required by a system is dependant upon whether tools are used in the maintenance of the system.

Rationale and Results

The use of tools in the maintenance of the system can reduce the amount of software maintenance.

The analysis of the results for the use of tools in the maintenance of the system are shown in Table D.51 and this is taken from question 43 of the survey.

Table D.51 Analysis of the use of tools in the maintenance of the software.

	Yes and No
Degrees of Freedom	22
Critical Value t_c	2.074
Calculated Test Statistic	0.490

Discussion

All of these results show that there is insufficient evidence to reject hypothesis (iii) in favour of (iv) suggesting that the use of tools in software maintenance does not have a significant impact on the level of software maintenance required.

APPENDIX E

E.1 Individual Survey Responses - Initial Survey

Survey Number

Question number	1	2	3	4	5
1 Dept. Org.	c c	y y	aa aa	e r	e x
2	1,000+	10,000	16,000	11,000	130
3	35	420	130	850	90
4	c	b	c	b	c
5	a,c-j	a,b,d,e,f,j,k,l	a,b,c,e,g	ALL	a,b,c,e,i
6	c- 40%	a	b - 20%	c - 70-80%	b - 70%
7 Which Reliability	YES NOMAD 3	YES NATU-RAL 4	YES INGREV 4	YES NOMAD 3	YES INGRES 5
8	NO	YES - SSADM ENGINEER.	YES AUTOMATE +	YES SPEEDBUILDER, PDF	YES-AUTOMATE +
9	d	c	c	d	c
10	4	4	3	4	3
11- S/w Maint. 4GL 3GL	Adequate More than Adequate None	None Adequate Adequate	None Adequate Adequate	None More than Adequate More than Adequate	None Insufficient Adequate
12	d	c	c	e	b
13	c	c	c	c	c
16	Plant data	Financial	Admin	Marketing	M.I.S.
17	100+	20-30	25	None	40
18	3 years	5 years	9 months	18 months	3 years
19	System	Application Programs	Both	Systems Programs	User written
20	5,5	2,4	4,4	5,5	5,5
21	c	c	b	b	c

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	6	7	8	9	10
1 Dept. Org.	e Pub. Util.	y y	e t	e Retail	e Pub. Util.
2	80,000	2,000+	3,000	50	5,500
3	70	180	3,000	50	60
4	b	b	a	b	a
5	a,c-l	a,b,c,e	a-f,h-l	ALL	a,e-i
6	b- varies	a	b - 20%	c - 85%	b
7 Which	YES ORACLE	YES IDEAL	YES INGRES	YES INGRES	YES SQL WINDOWS FORMS
Reliability	5	4	5	4	3
8	YES - ORACLE* CASE	NO	NO	YES	YES - ORACLE* CASE
9		e		c	c
10	5	3	2	4	3
11- S/w Maint.	Adequate	Adequate	Adequate	Adequate	Adequate
4GL	Adequate	Adequate	Insufficient	Adequate	Adequate
3GL	Adequate	Adequate	Adequate	Adequate	Adequate
12	c	c	c	c	c
13	c	c	c	a	c
16	Stock Control	MDS Support	Admin & Financial	Labour Manag.	Asset Manag.
17	71	None	400+	50	11
18	6 months	6 months	1 year	3 years	6 months
19	Application Programs	System Programs		Systems Programs	User written
20	5,5	5,5	4,4	4,3	2,2
21	c	b	c	c	b

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	11	12	13	14	15
1 Dept.	r	ab	e	e	e
Org.	r	ab	g	r	z
2	4,000	5,000+	30,000	5,000	400
3	220	30	140	200	28
4	c	c	c	b	c
5	ALL	a-e,g-j	c-f,j	ALL	a-g,j
6	c-85%	c	a	b-90%	b-30%
7	YES	YES	YES	YES	YES
Which	RAMIS	ORACLE	NOMAD	UNISYS LINK	SYNON/2
Reliability	4	4	2	4	3
8	NO	YES- TOPCASE	NO	NO	NO
9	c	d	b	c	c
10	2	4	2	3	3
11- S/w Maint.	None	Adequate	Insufficient	None	Adequate
4GL	More than Adequate	Adequate	More than Adequate	Adequate	None
3GL	More than Adequate	Adequate	More than Adequate	Adequate	Insufficient
12	d	d	c	d	b
13	c	c	c	c	a/b
16	Financial	Financial	Payroll	Alpha Information	Accounting
17	68	None	None	145	300
18	5 years	18 months	3 months	3 years	Due to go live
19	Appliation Programs	System Programs	Systems Programs	User written Application	Bespoke software only
20	5,4	5,5	5,5	4,4	3,n/a
21	d	b	b	c	d

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	16	17	18	19	20
1 Dept. Org.	e l	y y	e Retail	e i	e u
2	600	2,000	2,000	600	600
3	50	250	120	32	550
4	b	c	b	b	b
5	a-c,e,g-j	a,d,g	a-g,k	ALL	a,g,h
6	b-30%	b-25 - 30%	c - 30%	b - 15-25%	c - 40%
7 Which Reliability	YES FOCUS 5	YES LINC 5	YES TELON 5	YES PRO-IV 3	YES FOCUS 3
8	YES - EXCELERATOR	YES- LBMS SYSTEM ENGINEER	NO	YES - IEW	YES- CSP
9	c	e	e	c	c
10	3	3	4	3	4
11- S/w Maint. 4GL 3GL	None Adequate Adequate	Insufficient Adequate Adequate	 More than Adequate	Adequate Adequate Adequate	None Adequate Adequate
12	c	c	b	d	c
13	c	c	c	c	a
16	Financial	Customer database	Financial	Stock Control	Stock Control
17	None	20-30	None		40
18	2 years	4 years	2 months	Due to go live	6 months
19	Systems programs		Systems Programs	Application Programs	User Application
20	5,5	4,5	2,5	5,4	5,5
21	b	c	c	c	b

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	21	22	23	24	25
1 Dept. Org.	e l	s ac	y y	e Retail	y y
2	600	75	1,600	1,700	260
3	600	45	150		50
4	b	b	b	c	b
5	ALL	ALL	ALL	ALL	ALL
6	b - 80%	c - 20%	a	a	b-40%
7 Which Reliability	YES FOCUS 5	YES IDEAL 4	YES MANTIS 5	YES INGRES	YES ORACLE 5
8	YES - IEF	NO	YES- LBMS SYSTEM ENGINEER	YES - SEVERAL	YES CASE* DESIGNER
9	c	d	d	e	d
10	3	4	5	4	4
11- S/w Maint. 4GL 3GL	Adequate Adequate Adequate	Adequate Adequate Adequate	Adequate Adequate Adequate	None Adequate Adequate	Insufficient Adequate Adequate
12	b	d	b	b	b
13	mix a & b	a	a	c	c
16	Financial	Market research	Insurance	Stock Control	Financial
17	80	150	30	100	120
18	3 years	10 years	7 years	4 years	5 years
19		System Programs	Application Programs		Systems Programs
20	5,5	2,5	5,5	5,5	4,4
21	b	c	c	c	c

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	26	27	28	29	30
1 Dept. Org.	e ac	r r	e ac	a a	e e
2	500	6,000	40	27,000	150
3	450	400	15	800	70
4	c	c	b	None	None
5	a-e,g-l	a,c-l	a,b,g	a,c-l	ALL
6	a	b	b-25%	a	b - 10%
7 Which Reliability	YES SMART/400 3	YES IDEAL 5	YES IDEAL 4	YES MANTIS 5	YES INGRES
8	NO	YES- HPS (ICASE)	NO	YES - IEF	YES - TEAMWORK
9	d	e	d	e	
10	3	3	4	3	2
11- S/w Maint. 4GL 3GL	Insufficient Adequate None	Adequate Insufficient Adequate	Adequate More than Adequate More than Adequate	Adequate Adequate Adequate	Adequate Adequate Adequate
12	a	b	d	c	c
13	c	b	c	c	c
16	Insurance	Financial	Commercial Statistics		Operator Interface
17	1,000+	275	> 100		Many
18	3 years	2 years	7 years		Still in development
19	Systems Programs		Development		Screen Presentation
20	3,4	5,5	5,5	5,5	3,4
21	b	c	b		c

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	31	32	33	34	35
1 Dept. Org.	e d	e r	e e	ab ab	Public Util. Public Util.
2	240	4,500	650	65	4,000
3	11	250	600+	5	225
4	b	b	None	c	None
5	a-k	a-d,f-l	ALL	a,b,h	a,,b,g,h,i,j
6	b- 50%	b- 80%	c - 90%	b- 50%	b - 5%
7 Which Reliability	YES IMPLEMENTOR 3	YES LINK 5	YES SYNON 5	NO	YES CORVISION 5
8	YES - IMPLEMENTOR	YES- SYSTEM ENGINEER	YES -BIS IPSE	NO	YES - IEW
9	d	d	c		e
10	3	4	3	3	4
11- S/w Maint. 4GL 3GL	None Adequate Adequate	Adequate More than Adequate More than Adequate	Adequate Adequate Adequate	Insufficient Insufficient Adequate	Adequate Adequate Adequate
12	c	d	d	b	
13	c	c	a	c	
16	Marketing	Financial	Financial		Real time measured values
17	140	600	3,000+		None
18	Still in development	2 years	6 years		Being implemented
19	Application Development	Application Development	Control of Data		Application Development
20	3,4	5,5	4,5		3,5
21	b	d	d		c

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	36	37	38	39	40
1 Dept.	z	ab	i	ab	ab
Org.	z	ab	i	ab	ab
2	400	43	3,500	130	34
3	1	4	20	16	1
4	None	b		b	a
5	ALL	e	a-g,l	a-g,l	a-c,g
6			b - 60%	b-100%	b
7 Which Reliability	NO	NO	NO	NO	NO
8	NO	NO	NO	NO	NO
9					
10			3	3	3
11- S/w Maint.	None		None	None	None
4GL	None			None	None
3GL	None		Adequate	Adequate	None
12	c	b	c		d
13			c	b	c
16					
17					
18					
19					
20					
21					

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	41	42	43	44	45
1 Dept. Org.	e x	e r	n n	p p	h h
2	350	16,800	15,000	35,000	1,900
3	6	700	200	900	50
4	b	c	None	None	b
5	a-g	a-d,f,g	a-d,g-j	ALL	a-f,h
6	a	b - 60%	b - 70%	b - 60%	b - 25%
7 Which Reliability	YES	NO	NO	YES FOCUS 5	YES PROGRESS 2
8		NO	NO	YES - IEF	NO
9				e	d
10	2			3	3
11- S/w Maint. 4GL 3GL	Insufficient	Adequate	Adequate Adequate	None Adequate Adequate	None Adequate More than Adequate
12	d	c	b	c	c
13	a	a		c	
16					Financial
17					None
18					18 months
19					System Programs
20					1,2
21					c

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	46	47
1 Dept. Org.	t t	r r
2	1,500	5,000+
3	49	300+
4	b	c
5	a-f,l	a,g-k
6	b - 30-40%	c
7 Which Reliability	YES PRO-IV 4	YES NOMAD 4
8	NO	YES -IEW
9	d	d
10	3	3
11- S/w Maint. 4GL 3GL	None Adequate More than Adequate	Adequate Adequate Adequate
12	c	e
13	c	c
16	Financial	MIS
17	120+	None
18	5-6 years	2.5 years
19	Systems Programs	All programs
20	5,5	5,5
21	c	b

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	1	2	3	4	5
22 - LOC	2,500			40,000	2,000
Lines exec.	2,150				
Code					
Entity relations.	5		25	78	5
F.P.'s		7,800			
J C L Lines					
Data Dict Entries			210	450	
23	8m,14m,2	50y,12m, 50	8m,4m,3	24m,6m,4	4m,6m,4
24	2,3,2	120,36,24	-,10days, 5 days	-,2m,6m	0.9m,0.9m, 0.9m
25	3	3	3	2	3
26	5	4	3	4	3
27	3	3	2	3	4
28	5	3	3	4	3
29 a) (i)	<6 mths	< 6 mths	< 6 mths	< 6 mths	<6mths
(ii)	< 6 mths	> 3 years	6 - 12 mths	6 - 12 mths	6-12mths
(iii)	6-12mths	6-12mths	2-3years	6-12mths	<6mths
(iv)		>3years	2-3years	>3years	2-3years
b) (i)	1-2years	6-12mths	<6mths	<6mths	<6mths
(ii)	1-2 years	2-3 years	6-12mths	6-12mths	6-12mths
(iii)	1-2years	2-3years	2-3years	6-12mths	6-12mths
(iv)		2-3years	2-3years	>3years	6-12mths
30	b	a	b	c	a
31	2	5	4	5	1
32	4	5	3	4	4
33 Data Dict.		SOME	NO	YES	NO
Mean. Names	SOME	YES	YES	YES	YES
Comments in Code	YES	YES	YES	YES	SOME
Site Standards	YES	YES	YES	YES	NO
34		Doc X- ref		Repository & impact analysis	

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	6	7	8	9	10
22 - LOC	155,540	48,000	800,000	71,000	100,000
Lines exec.	11,251			53,000	80,000
Code blocks					
Entity relations.					
F.P.'s					
J C L Lines	1,232				
Data Dict Entries	400	107	110		
23	24m,6m,4	6m,9m,2	360m,24m, 20	36m,12m,3	180m,48m, 6
24	0,72,36	0,0,8	240,-,144	24,36,48	4 months
25	4	5	3	4	5
26	4	4	1	5	3
27	2	3	4	2	3
28	4	5	3	4	5
29 a) (i)	<6mths	1-2years	<6mths	2-3years	1-2years
(ii)	1-2years	>3years	6-12mths	>3years	2-3years
(iii)	>3years	>3years	<6mths	6-12 mths	2-3years
(iv)	>3years	>3years	<6mths	2-3years	>3years
b) (i)	<6mths	1-2years	2-3years	2-3years	6-12mths
(ii)	1-2years	>3years	2-3years	>3years	6-12mths
(iii)	>3years	>3years	2-3years	6-12mths	1-2years
(iv)	>3years	>3years	2-3years	2-3years	2-3years
30	c	c	b	c	b
31	4	4	4	3	4
32	4	3	5	5	5
33 Data Dict.	YES	YES	YES	NO	YES
Mean. Names	YES	YES	YES	YES	YES
Comments in Code	YES	YES	YES	YES	YES
Site Standards	YES	YES	YES	YES	YES
34	Maintain definitions	Database definitions	Database definitions		

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number		11	12	13	14	15
	22 - LOC	27,400	45,000	85,400	685,000	35,000
	Lines exec.	15,840	25,000	60,000		
	Code Entity		14			
	relations.					
	F.P.'s					
	J C L Lines	52	500	2,000		400
	Data Dict Entries					5,000
	23	7.5m,1.5m, 5	12m,6m,3	6m,1m,8	10y,3y,5	53y,18m, 30
	24	6,6,6	60,30,30	-,6,24	600,680, 500 (man days)	36
	25	4	4	5	4	2
	26	3	4	4	4	4
	27	4	3	3	2	4
	28	4	4	4	4	4
	29 a) (i)	>3years	1-2years	1-2years	6-12mths	1-2years
	(ii)	1-2years	>3years	2-3years	>3years	2-3years
	(iii)	>3years	<6mths	6-12mths	1-2years	1-2years
	(iv)	6-12mths	>3years	2-3years	>3years	1-2years
	b) (i)	>3years	2-3years	1-2years	6-12mths	2-3years
	(ii)	>3years	>3years	2-3years	>3years	2-3years
	(iii)	2-3years	6-12mths	6-12mths	1-2years	1-2years
	(iv)	>3years	>3years	2-3years	>3years	1-2years
	30	b	a	b	b	a
	31	1	4	4	4	2
	32	1	3	4	3	5
	33 Data Dict.	NO	YES	NO	SOME	YES
	Mean. Names	YES	YES	YES	SOME	YES
	Comments in Code	YES	YES	YES	SOME	YES
	Site Standards	YES	YES	YES	SOME	YES
	34				Part of 4GL	Field repository

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	16	17	18	19	20
22 - LOC	1,000	88,000	1,000,000		400
Lines exec.					400
Code					
Entity					
relations.					
F.P.'s		2,600			
J C L		1,000			
Lines					
Data Dict		450			30
Entries					
23	3m,3m,2	51m,9m,5	18y,10m, 11-43	-,4y,16	6m,2m,3
24	1,1,0.5	30,40,30+	36	24	0.1
25	3	4	3	4	4
26	3	4	3	3	4
27	3	2	3	3	3
28	3	4	3	4	4
29 a) (i)			1-2years	<6mths	<6mths
(ii)			1-2years	>3years	1-2years
(iii)			>3years	>3years	1-2years
(iv)			1-2years	>3years	2-3years
b) (i)			1-2years	<6mths	<6mths
(ii)			1-2years	>3years	2-3years
(iii)			>3years	>3years	1-2years
(iv)			1-2years	>3years	2-3years
30	b	b	c	c	a
31	4	4	4	3	3
32	3	4	2	4	2
33 Data Dict.	NO	YES	YES	YES	YES
Mean. Names	NO	YES	SOME		YES
Comments in Code	SOME	YES	YES	YES	YES
Site Standards	SOME	YES	YES	YES	YES
34		Consistency	Impact analysis	File definitions	Names & data types

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	21	22	23	24	25
22 - LOC		120,000	69,000		
Lines exec.		100,000	65,000		
Code					
Entity				30	60
relations.					
F.P.'s	1,430				
J C L		10,000			
Lines					
Data Dict					
Entries					
23	14m,6m,3			2m,-,4	18m,3m,6
24	25,6,3	36,36,36	6,6,3	1,1,1	24,24,24
25	3	4	4	3	4
26	3	4	4		4
27	3	3	3	3	3
28	3	4	4	5	5
29 a) (i)	<6mths	2-3years	<6mths		<6mths
(ii)	1-2years	>3years	1-2years		<6mths
(iii)	1-2years	<6mths	1-2years		1-2years
(iv)	1-2years	2-3years	>3years		2-3years
b) (i)	<6mths	2-3years	6-12mths		1-2years
(ii)	>3years	>3years	2-3years		1-2years
(iii)	1-2years	6-12mths	>3years		2-3years
(iv)	>3years	>3years	>3years		2-3years
30	a	a	a	a	b
31	1	2	2	3	4
32	5	5	5	2	2
33 Data	YES	YES	NO	YES	YES
Dict.					
Mean.	NO	YES	YES	YES	YES
Names					
Comments	NO	YES	YES	YES	YES
in Code					
Site	NO	YES	YES	YES	YES
Standards					
34		Data definitions		Part of 4GL	Central repository

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	26	27	28	29	30
22 - LOC					
Lines exec.					
Code					
Entity					
relations.					
F.P.'s	1,500				
J C L					
Lines					
Data Dict	10,000				
Entries					
23	-,14m,6		18m,24m,2		6m,8m,1
24	60,120,120		1,-,1		0.5
25	3	3	4	3	4
26	4	4	3		4
27	3	3	3		2
28	3	3	4	3	4
29 a) (i)	1-2years	6-12mths	<6mths		<6mths
(ii)	1-2years	>3years	<6mths		>3years
(iii)	1-2years	>3years	>3years		<6mths
(iv)	>3years	2-3years	<6mths		>3years
b) (i)	1-2years	6-12mths			<6mths
(ii)	1-2years	>3years			>3years
(iii)	>3years	>3years			<6mths
(iv)	>3years	2-3years			>3years
30	a		c	b	b
31	5	4	5	3	3
32	5	3	2		3
33 Data	YES	YES	YES	YES	NO
Dict.					
Mean.	SOME	YES	YES	YES	YES
Names					
Comments	YES	YES	YES	YES	YES
in Code					
Site	YES	YES	YES	YES	YES
Standards					
34	Part of 4GL	Part of 4GL		Corporate data model	

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	31	32	33	34	35
22 - LOC		1,000,000	6,000,000		
Lines exec.			3,000,000		
Code Entity relations.					140
F.P.'s					
J C L Lines		5,000			
Data Dict Entries		600			
23	4m,6m,1	15m,15m,30	100+m,18m,up to 100		-,5m,10
24	-,-,1				None,5%,5%
25	4	2			3
26	4	5	4		3
27	3	2	3		3
28	4	3			3
29 a) (i)	<6mths	<6mths	6-12mths		<6mths
(ii)	>3years	<6mths	1-2years		>3years
(iii)	>3years	>3years	1-2years		<6mths
(iv)	>3years	>3years	1-2years		>3years
b) (i)	<6mths	6-12mths	<6mths		6-12mths
(ii)	>3years	6-12mths	2-3years		>3years
(iii)	>3years	>3years	2-3years		6-12mths
(iv)	>3years	>3years	2-3years		>3years
30	c	c	c		b
31	4	5	3		5
32	3	5	5		1
33 Data Dict.	YES	YES	YES		YES
Mean. Names	YES	YES	YES		NO
Comments in Code	YES	YES	YES		NO
Site Standards	YES	YES	YES		YES
34		Repository	Field reference		Repository

E.1 Individual Survey Responses - Initial Survey (Continued)

Question number		Survey Number 45	46	47
	22 - LOC			
	Lines exec.			
	Code			
	Entity			
	relations.			
	F.P.'s			
	J C L			
	Lines			
	Data Dict			
	Entries			
	23		26m,10m,4	12m,15m,2
	24	3,3,3	1,1.5,1	2.5,2.5,1.5
	25		3	4
	26	3	4	2
	27	3	2	3
	28	4	3	4
	29 a) (i)	<6mths	<6mths	<6mths
	(ii)	<6mths	2-3years	6-12mths
	(iii)	>3years	6-12mths	<6mths
	(iv)	>3years	>3years	6-12mths
	b) (i)	<6mths	2-3years	6-12 mths
	(ii)	<6mths	>3years	6-12mths
	(iii)	>3years	2-3years	6-12mths
	(iv)	>3years	>3years	6-12mths
	30	a	b	b
	31		4	5
	32	5	2	2
	33 Data	YES	YES	NO
	Dict.			
	Mean.	YES	YES	YES
	Names			
	Comments	YES	YES	YES
	in Code			
	Site	YES	YES	YES
	Standards			
	34		Repository	

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	1	2	3	4	5
35	4	4	4	3	2
36	3	5	3	3	2
37	3	4	4	3	4
38	NO	YES-F.P's	NO	NO	NO
39	YES BUT NOT PURE - YOURDON /SSADM	YES BUT NOT PURE - SSAMD	YES BUT NOT PURE - SSAMD	YES BUT NOT PURE - JSDEV SYSTEMCRAFT	NO
40	NO	NO	NO	AS 39	NO
41	3	3	4	4	4
42		CASE	INGRES 4GL		NONE
43		NONE	AS 42		NONE
44	YES/NO	NO	YES/NO	YES/NO	NO
45	NO	NO	NO	NO	NO

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	6	7	8	9	10
35	5	4	4	3	3
36	4	4	4	3	3
37	4	5	4	3	4
38	NO	NO	NO	YES-IN HOUSE	NO
39	YES BUT NOT PURE - SSADM	NO	NO	NO	YES-CASE
40	NO	NO	NO	NO	YES-CASE
41	4	3	3	2	2
42	ORACLE* CASE	DATACOM DB	DEBUGGER	INFORMIX	CASE*DICT & DESIGNER
43	AS 42	AS 42	AS 42	AS 42	AS 42
44	NO	YES/NO	YES/NO	YES/NO	NO
45	NO	NO	NO	YES - Move to any UNIX platform. Software impacts.	NO

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	11	12	13	14	15
35	3	4	5	4	2
36	4	3	4	4	3
37	5	3	4	3	4
38	NO	NO	NO	NO	NO
39	NO	YES BUT NOT PURE - ORACLE	YES- MODUS	YES BUT NOT PURE - LSDM	NO
40	NO	AS 39	AS 39	NO	NO
41	2	2	4	2	1
42	DEBUG	SQL*PLUS	INHOUSE TOOLS	NONE	DESIGN AID
43	AS 42	AS 42	TEST DATA CREATOR	NONE	
44	NO	YES/NO	NO	NO	NO
45	YES - porting to UNIX	NO	NO	NO	NO

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	16	17	18	19	20
35	2	4	3	3	3
36	3	5	3	4	4
37	4	4	1	4	3
38	NO	NO	NO	NO	NO
39	YES BUT NOT PURE - STRUCTURED	YES - MODUS	YES BUT NOT PURE - SSAMD	NO	NO
40	NO	AS 39	NO	NO	NO
41	3	4	3		3
42	NONE	IPSE	FLOW-CHART		
43	NONE	AS 42	AS 42		
44	YES/NO	NO	NO	NO	YES/NO
45	NO	NO	NO	YES - Move to new hardware	NO

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	21	22	23	24	25
35	4	3	4	3	5
36	3	4	3	3	5
37	1	5	3	5	4
38	NO	NO	NO	NO	NO
39	YES-INHOUSE	NO		YES - INHOUSE	YES BUT NOT PURE - ORACLE
40	NO	- NO	NO	AS 39	AS 39
41	1	4	3	4	3
42		MVS		PMW	FORMS
43		MVS	NONE		FORMS
44	YES/NO	NO	NO	NO	NO
45	NO		NO	NO	YES-NEW PROCESSOR

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number		26	27	28	29	30
	35	3		4	3	4
	36	5	3	3	3	4
	37	2	3	3	3	3
	38	NO	NO	NO	NO	NO
	39	YES BUT NOT PURE - REVERSE ENG.	YES - CACI	NO	YES - IEW	YES - INHOUSE
	40	NO	AS 39	NO	AS 39	AS 39
	41	3	4	4	3	3
	42	NONE	DATACOM	NONE	NONE	
	43	ERROR MONITOR-ING	DATACOM	NONE	NONE	
	44	NO	YES/YES	NO	NO	NO
	45	NO	NO	NO	NO	NO

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question number	31	32	33	34	35
35	4	2	2		5
36	4	4	5		5
37	4	4	5		3
38	NO	NO	NO		NO
39	YES BUT NOT PURE - YOURDON	NO	YES BUT NOT PURE - SSAMD		NO
40	NO	NO			NO
41	2	2	3		4
42	IMPLEM-ENTOR	NONE	NONE		IEW
43	AS 42	NONE	NONE		AS 42
44	YES/NO	YES/NO	NO		YES/NO
45	NO	NO	NO		NO

E.1 Individual Survey Responses - Initial Survey (Continued)

Survey Number

Question no.		45	46	47
	35	3	3	4
	36	3	4	4
	37	3	3	3
	38	NO	NO	
	39	NO	NO	YES - INHOUSE
	40	NO	NO	AS 39
	41	1	4	3
	42	NONE		
	43	NONE		
	44	NO	YES/YES	YES/NO
	45	NO		NO

E.2 Individual Survey Responses - Supplementary Survey

Survey Number

Question number		1	2	3	4	5
	4GL	Ingres	Oracle	Oracle	Nomad	Telon
	22 -LOC	3,000	650	15,000	331,133	20,000
	Lines exec. code	3,000		13,500		18,000
	Entity relations				285	9
	F.P.'s				350	80
	JCL lines					820
	Data Dict Entries	1,400			2,439	40
	23	6m,9m,3	2m,2m,1	12m,9m,2	13m,4m,5 to 8	4m,6m,3
	24	0.75m,0.75m, 0.75m	0.25	12m,12m, 12m	-,48m,48m	1m,1m,1m
	17	NONE	20	None	70	40
	29 a) (ii)	>12mths	>12mths	6-12mths	>12mths	>12mths
	b) (i)	6-12mths	>12mths	6-12mths	>12mths	>12mths
	11 -4GL	Adequate	Adequate	Adequate	Adequate	Adequate
	3GL	Adequate	Adequate	Adequate	Adequate	Adequate
	30	YES	NONE	NONE	SOME	YES
	31	5	4	2	3	4
	33 Mean. Names	YES	SOME	YES	YES	YES
	44	YES	NO	NO	NO	NO
	42	Uniface, Silverhorn	Debugger	Test data generator	PMW	

E.2 Individual Survey Responses - Supplementary Survey (Cont'd)

Survey Number

Question number	6	7	8	9	10
4GL	Ideal	Natural	Focus	Ingres	Oracle
22 -LOC	5,000	15,000	9,000	21,000	20,000
Lines exec. code	4,500				
Entity relations					
F.P.'s					
JCL lines					
Data Dict Entries					
23	2m,3m,2	24m,24m,2	10m,10m,3	18m,9,2	
24	-,1m,1m	7m	5m	8m,8m,6m	
17	None	150	68	70	None
29 a) (ii)	>12mths	6-12mths	6-12mths	>12mths	<6mths
b) (i)	>12mths	6-12mths	6-12mths	>12mths	<6mths
11 -4GL	Adequate	More than Adequate	Adequate	More than Adequate	Adequate
3GL	Adequate	Adequate	Adequate	Adequate	Adequate
30	SOME	YES	NONE	ALL	YES
31	3	3	2	3	3
33 Mean. Names	YES	NO	YES	YES	YES
44	YES	NO	NO	NO	NO
42	PMW	Inhouse	Inhouse	Test data generator	

E.2 Individual Survey Responses - Supplementary Survey (Cont'd)

Survey Number

Question number		11	12	13	14
	4GL	Nomad	Ingres	Pro-IV	Oracle
	22 -LOC			17,000	25,000
	Lines exec. code				
	Entity relations				
	F.P.'s				
	JCL lines				
	Data Dict Entries				
	23			3m,3m,3	
	24	2m			
	17	25	5	23	None
	29 a) (ii)	<6mths	6-12mths	>12mths	>12mths
	b) (i)	>12mths	>12mths	6-12mths	>12mths
	11 -4GL	Adequate	Adequate	Adequate	Adequate
	3GL	None	Adequate	None	Adequate
	30	YES	NO	SOME	YES
	31	3	5	2	3
	33 Mean. Names	YES	YES	YES	YES
	44	YES	YES	NO	NO
	42			Inhouse	

REFERENCES

[ABRAN90] Abran A and P.N. Robillard. Proceedings of the CIPS 1990 Conference, Canadian Information Processing Society, Ottawa, Canada, May 1990.

[ABRAN93] Abran A and H. Nguyenkim. Measurement of the maintenance process from a demand-based perspective. Journal of Software Maintenance: Research and Procedure Volume 5 Number 2, June 1993, pp 63 - 90.

[ALBRECHT79] Albrecht A.J. Measuring application development productivity. Proceedings of the IBM Application Development Symposium. Guide/ share. October 1979. pp. 83 -92.

[ALBRECHT83] Albrecht, A.J. and J. Gaffney Jr. Software function, source lines of code, and development effort prediction: A software science validation. IEEE Transactions on Software Engineering. SE-9, Number 6, November 1983, pp. 639 - 648.

[ANSI90] American national standard IEEE standard glossary of software engineering terminology, ANSI/ IEEE Standards. 610.12. 1990.

[BALL87] Ball R.K. Aerobic polls and broad definitions: The Calgary connection. Software Maintenance News Volume 5 Number 7, July 1987. Page 12.

[BALL87b] Ball R.K. 1987 Annual software maintenance survey: Survey Results. Software Maintenance Association Vallejo, California. 1987.

[BANKER87] Banker R.D., S.M. Datar, and C.F. Kemerer. Factors affecting software maintenance productivity: an exploratory study. Proceedings of the eighth international conference on information systems. Pittsburgh PA, December 6 - 9 1987. pp. 160 - 175.

[BATE87] Bate J. St.J. and D.B. Vadhia Fourth generation languages under DOS and UNIX. BSP Professional Books. Oxford, England. 1987.

[BELADY72] Belady L. and M. Lehman. An introduction to growth dynamics statistical computer performance evaluation. W. Frieberger, ed. Academic Press, 1972. pp. 503 - 511.

[BENNETT89] Bennett K.H. The software maintenance of large software systems: Management, methods and tools. CSR 6th Annual Conference on Large Software Systems. 1989.

[BLACK92] Black G. Maintenance: The never ending nightmare. Software Management April 1992. pp 4 - 6.

[BOEHM74] Boehm B.W. Some steps towards formal automated aids to software requirements analysis and design. IFIP74. Amsterdam, Holland. pp 192 -197.

[BOEHM75] Boehm B.W: The high cost of software. Practical strategies for developing large software systems. Ed E. Horowitz Addison- Wesley Reading MA. 1975.

[BOEHM76] Boehm B.W. Software engineering. IEEE Trans. Comp. Volume 25, Number 12, December 1976. pp 1226 - 1242.

[BOEHM77] Boehm B.W. Seven basic principles of software engineering. Infotech state of the art reports: Software Engineering Techniques. Maidenhead. 1977. pp 77 - 113.

[BOEHM81] Boehm B.W. Software engineering economics. Englewood Cliffs N.J. 1981.

[BOEHM83] Boehm B.W. The economics of software maintenance. IEEE. December 1983.

[BROOKS75] Brooks F. The mythical man month. Addison-Wesley, Reading MA. 1975.

[CACCAMESE86] Caccamese, A., L. Cappello, and G. Dodero. A comparison of SLIM and COCOMO estimates versus historical man-power and effort allocation. Unpublished paper. (Test of COCOMO and SLIM against three projects at Olivetti in Italy). 1986.

[CAP87] CAP. ICL productivity under control: Calipso 1987.

[CHAPIN84] Chapin N. Software maintenance with fourth generation languages. ACM Sigsoft software engineering notes Volume 9 Number 1. January 1984. Pp 41 - 42.

[CHRYSLER78] Chrysler E. Some basic determinants of computer programming productivity. Communications of the ACM, Volume 21, Number 6, June 1978. pp 472 - 483.

[CODD85] Codd E.F. Viewpoint. Computerworld Volume 19 Number 50. 16 December 1985. Page 18.

[COLTER88] Colter M. The business of software maintenance. Second software maintenance workshop notes. Centre for Software Maintenance, Durham, England. 13 - 14 September 1988.

[CSM87] Centre for Software Maintenance. Centre for Software Maintenance, Durham, England. 1 October 1987.

[CSM92] Centre for Software Maintenance, Durham England. 1992.

[DALY77] Daly E. Management of software engineering. IEEE Transactions on Software Engineering, Volume SE-3, Number 3, May 1977.

[DEKLEVA90] Dekleva S. 1990 Annual software maintenance survey: Survey results. Software Maintenance Association, Vallejo California 1990.

[DEMARCO82] DeMarco T. Controlling software projects. Yourdon Press, N.Y. 1982.

[ELLIOTT77] Elliott I. Life cycle planning for a large mix of commercial systems, Proceedings U.S. Army ISRAD software workshop, August 1977.

[FERENS79] Ferens D. and R. Harris. Avionics computer software operation and support cost estimation, Proceedings NAECON79, Dayton, Ohio, May 1979.

[GAYLE71] Gayle J.B. Multiple regression techniques for estimating computer programming costs. Journal of Systems Management, Volume 22, Number 2, February 1971, pp 13 - 16.

[GILB82] Gilb T. Software metrics. Studentlitteratur. Lund. 1982.

[GILB83] Gilb T. Design by objectives: Maintainability. Tutorial on software maintenance IEEE computer society. pp. 167 - 179.

[GOLDEN81] Golden J.R., J.R. Mueller and B. Anselm. Software cost estimating: craft or witchcraft. Database 12(3) Spring 1981. pp 12 - 14.

[GRAVER77] Graver C. Cost of reporting elements and activity cost trade-offs for defence system software, General research corporation, Santa Barbara, CA. March 1977.

[GRIFFIN80] Griffin E. Real time estimating. Datamation. June 1980.

[GUIDE85] GUIDE International. Maintenance productivity improvements through matrices and measurements, GUIDE Publications, GPP-130. 1985.

[HELMER66] Helmer-Heidelberg O. Social technology. Basic Books, New York. 1966.

[HEWETT87] Hewett J. and T. Durham. Computer aided software engineering: commercial strategies. Ovum Ltd. London, England. 1987.

[HILL90] Hill S. DSS fails to see the benefit of £2 billion system. Computer Weekly 24 May 1990. Page 1.

[IDC84] International Data Corporation. Fourth generation language: information generators to meet information needs. IDC ISPS report Number 2563. Framingham Ma. 1984.

[IDPM86] Institute of Data Processing Managers. Fourth generation languages - how to use them. The Grindley Report. London. 1986.

[JEFFERY87] Jeffery D.R. Time sensitive cost models in the commercial MIS environment. IEEE transactions on software engineering. SE-13(7). July 1987. pp 852 - 859.

[JONES86] Jones C. Programming productivity. McGraw-Hill Book Company, New York, 1986.

[JONES88] Jones C. A ten year history of software engineering in the ITT Corporation. Software Productivity Research Inc. Cambridge, Mass.

[KEMERER87] Kemerer C.F. An empirical validation of software cost estimation models. Communications of the ACM. 30(5). May 1987. pp 416 - 429.

[KEMERER88] Kemerer C.F. Software cost estimation models. September 1988.

[KITCHENHAM84] Kitchenham B. and N.R. Taylor. Software cost. models. ICL technical journal May 1984. pp 73 - 102.

[KITCHENHAM85] Kitchenham B. and N.R. Taylor. Software project development cost estimation. Journal of systems and software 5(4) November 1985. pp 267 - 278.

[LEHMAN76] Lehman M.M. and L.A. Belady. A model of large program development. IBM Systems Journal. 15(3). pp. 225 - 252.

[LEHMAN80] Lehman M.M. and L.A. Belady. Programs, life cycles and the laws of software evolution. Proceedings IEEE 68(9). pp 1060 - 1076.

[LIENTZ80] Lientz B. and E.B. Swanson. Software maintenance management: A study of the maintenance of computer application software in 487 data processing organisations. Addison-Wesley, Reading, MA. 1980.

[LIU76] Liu C.C. A look at software maintenance. Datamation Volume 22, Number 11. November 1976. pp 51 - 55.

[LYONS81] Lyons M.J. Salvaging your software asset (Tools based maintenance). AFIPS Conference Proceedings of 1981 National Computer Conference. Chicago. Volume 50. 4 - 7 May 1981. pp 337 - 342.

[McCLURE76] McClure C.L. Normalization and application structured programming and program complexity. PhD dissertation. Illinois Institute of Technology. 1976.

[McCLURE78] McClure C.L. Reducing COBOL complexity through structured programming. New York, Van Nostrand Reinhold Co. 1978.

[MARTIN83] Martin J. and C. McClure. Software maintenance: The problem and its solutions. Englewood Cliffs N.J. 1983.

[MIYAZAKI85] Miyazaki Y. and K. Mori COCOMO evaluation and tailoring. Proceedings of the eighth International Conference on Software Engineering. pp 292 - 299. 1985.

[MOHANTY81] Mohanty S. Software cost estimation: Present and future. software - practice and experience. Number 11. 1981. pp 103 - 121.

[PARIKH82] Parikh G. Some tips, techniques and guidelines for program and system maintenance. Techniques of Program and System Maintenance. Winthrop Publishers, Cambridge MA. 1982. pp 65 - 70.

[PATKAU83] Patkau B.H. A foundation for software maintenance. PhD Thesis, Department of Computer Science, University of Toronto. December 1983.

[PETZOLD87] Petzold K. The COBOL maintenance crisis. First software maintenance workshop notes. Durham, England. 8 - 9 September 1987.

[PUTNAM78] Putnam L.H. General empirical solutions to the macro software sizing and estimating program. IEEE Transactions on software engineering. Volume 4, 1978. pp 345 - 361.

[PUTNAM79] Putnam L.H. and A. Fitzsimmons. Estimating software costs. Datamation, September 1979 pp 189 - 198, October 1979 pp 171 - 178 and November 1979 pp137 - 140.

[RATHBONE89] Rathbone M. On the straight and narrow. Computer Weekly 16 March 1989. pp 40 - 41.

[RUBIN] Rubin H.A. Using ESTIMATICS E. Management and computer services, Inc., Valley Forge, PA.

[RUBIN85] Rubin H.A.(Chairman). A comparison of cost estimation tools (A panel session). Proceedings of the eighth International Conference on Software Engineering. IEEE Computer Society Press. 1985. pp 174 - 180.

[SCOTT74] Scott R.F. and D. Simmons Programmer productivity and the Delphi technique. Datamation, Volume 20, Number 5, May 1974. pp 71 - 73.

[SHARPLEY77] Sharpley W.K. Software maintenance planning for embedded computer systems. Proceedings IEEE COMPSAC77. November 1977. pp 520 - 526.

[SOMMERVILLE85] Sommerville I. Software engineering. Second Edition. Wokingham, England. 1985.

[SWANSON76] Swanson E.B. The dimensions of maintenance. Second international conference on software engineering proceedings. San Francisco. 13 -15 October 1976. pp 492 - 497.

[TANG89] Tang R. Third party software maintenance. Third Software Maintenance Workshop Notes, Centre for Software Maintenance, Durham, England. 19 - 21 September 1989.

[TINNIRELLO85] Tinnirello P.C. Software maintenance in fourth generation language environments. Data management. March 1985. Pp 38 - 43.

[VERNER88] Verner J. and G. Tate. Estimating size and effort in fourth generation development. IEEE Software, July 1988. pp 15 - 22.

[WALSTON77] Walston C.E. and C.P. Felix. A method of programming measurement and estimation. IBM Systems journal, Volume 16, Number 1. 1977. pp 54 - 73.

[WIENER84] Wiener-Ehrlich, W.K., J.R. Hamrick and V.F. Rupolo. Modelling software behaviour in terms of a formal life cycle curve: implications for software maintenance. IEEE Transactions for software engineering. July 1984. pp 376 - 383.

[WOLVERTON74] Wolverton W.R. Cost of developing large scale software. IEEE Transactions on computers, Volume 23, June 1974. pp 615 - 634.

[WOLVERTON80] Wolverton R. Airborne systems software acquisition engineering guidebook: software cost analysis and estimating, U.S. Air Force ASD/EN, Wright- Patterson AFB, OH. February 1980.

[WRIGLEY88] Wrigley C. and A. Dexter. A model for estimating information systems requirements size: Preliminary findings. Proceedings of the Ninth International Conference on Information systems. November 30 - December 3 1988. pp 245 - 255.

[YAU78] Yau S., J.S. Collofello and T. MacGregor. Ripple effect analysis of software maintenance. Proceedings. IEEE COMPSAC78, Chicago Il. November 1978. pp 60 - 65.

[YOURDON86] Yourdon E. Tooling up for 1995. Datalink. 17 November 1986. Page 10.

[ZVEGINTZOV91] Zvegintzov N. Real maintenance statistics. software Maintenance News. Volume 9 Number 2. 1991.