Managing Productivity in Systems Development

BUTLER COX

P.E.P

PEP Paper 5, April 1988

BUTLER COX

P.E.P

Managing Productivity in Systems Development

PEP Paper 5, April 1988
by Larry Putnam, Jim Greene, and Grenville Bingham

Larry Putnam

Larry Putnam is President of Quantitative Software Management Inc, a firm specialising in software investment and quality management. Mr Putnam has had extensive experience in industry and government in planning the quantitative aspects of software management including cost, schedule, and reliability determination. Mr Putnam is the developer of SLIM, an automated software estimating tool used by the Defense Department and a significant number of companies in the computer industry.

Mr Putnam has lectured extensively in the United States, Europe, Japan, and Australia about the quantitative side of software management and planning. He is the author of a tutorial book for the IEEE Computer Society, 'Software Cost Estimating and Life Cycle Control: Getting the Management Numbers' as well as numerous papers and articles on software investment management.

Jim Greene

Jim Greene is Managing Director of QSM-Europe. His career in software engineering began in 1963. He has worked for major companies and government organisations in Europe and the USA, at all management levels, over a wide spectrum of business areas.

Between 1968 and 1975 as a project manager with IBM, he managed a wide range of developments on behalf of European companies as well as software for an IBM communications product.

Working as a management consultant since 1975, he has specialised in software engineering investment strategies and introduced the Software Life Cycle Management method into Europe under a business agreement with the US developer, Larry Putnam.

Grenville Bingham

Grenville Bingham is a Principal Consultant with Butler Cox. He has extensive experience in software engineering from technical, management, and business perspectives.

He was responsible for managing the successful introduction and operation of Butler Cox's Productivity Enhancement Programme, a service to improve systems development productivity.

He started his career in software engineering in 1967 in IBM working on the design of OS and TSO. He has since worked in technical, marketing, and management roles for various software and systems houses in the United States and Europe. His experience covers applications, communications, and systems software in both the commercial and military spheres.

BUTLER COX

Published by Butler Cox & Partners Limited
Butler Cox House
12 Bloomsbury Square
London WC1A 2LL
England

Printed in Great Britain by Flexiprint Ltd., Lancing, Sussex.

BUTLER COX
P,E,P

Managing Productivity in Systems Development

PEP Paper 5, April 1988
by Larry Putnam, Jim Greene, and Grenville Bingham

Contents

(Continued)

Contents

BUTLER COX

Chapter 1

Measurement is key to managing productivity in systems development

"It is the mark of an instructed mind to rest satisfied with the degree of precision which the nature of the subject admits and not seek exactness when only an approximation of the truth is possible."
Aristotle

This paper describes the value of measurement in managing and improving the efficiency of software development. It describes various measures and approaches to applying them, and, in particular, explains the measures and approach developed by L H Putnam of Quantitative Software Management (QSM). This approach is the basis for the system development productivity assessments used in Butler Cox's Productivity Enhancement Programme (PEP) for its sponsors.

EFFECTIVE SOFTWARE INVESTMENT MANAGEMENT IS ESSENTIAL

Investment in software by commercial organisations is huge and growing. A recent study in the United States reported that high-technology companies typically spend five per cent of their gross revenue on software development (Putnam, 1987). A United Kingdom survey predicted that the proportion of total system's value that is accounted for by software would grow from 50 per cent in 1985 to 75 per cent in 1987 (EIU Informatics 1985).

Software comprises 75 per cent of systems value

Staff costs represent a large proportion of total software development costs. So you can calculate a rough measure of your own organisation's investment in software by multiplying the numbers of staff involved in software development by the average staff costs. For example, a department with 100 software development staff would cost around $6 million per year, assuming an average fully loaded staff cost of $60,000 per person per year.

Businesses are becoming increasingly dependent on IT. Systems are migrating from their non-critical back-office role to the front line of day-to-day business operations. The trend towards online systems means that reliability is assuming growing importance. The cost of failure of an electronic funds transfer system, a stock market trading system, or a factory management system can be measured in thousands of dollars per minute. Strategically important systems often need to be developed in a hurry in order to respond to a brief window of competitive opportunity. Unfortunately, pressures to reduce timescales tend to reduce reliability as well, turning advantage into disadvantage.

Software investment management is the term we use to describe methods of reducing the costs of software development and

Software investment management is important to the business

controlling the reliability of the finished product. Software investment management should not only be the concern of systems development managers. The scale of the expenditure on software and the risks of failure imply that general business management should also take a keen interest in making sure this investment is managed effectively.

IMPROVING PRODUCTIVITY IS THE MAIN GOAL OF SOFTWARE INVESTMENT MANAGEMENT

When viewing the systems development function, general management is usually concerned with the efficiency of the systems development activity and the effectiveness of systems as demonstrated by the impact that they have on the operations of the organisation. In this paper we are concerned only with the first of these objectives, efficiency.

Productivity is the key measurement of efficiency. In systems development it is a measure of how well the systems development process is carried out, that is, the *output* achieved for a given *input* (or cost).

Many different ways of measuring systems development productivity have been proposed. One commonly used measure is *system size/effort*. This seems intuitively right because *system size* is closely related to functionality and *effort* is a good indication of cost. It is a simple relationship and it is also attractive because both system size and effort are easy to measure. However, research shows that if you measure productivity in this way it will vary significantly according to the size and timescale of the project. We need to eliminate these major sources of variation so that we can compare productivity on projects of different sizes and with different timescales. In Chapter 4 we discuss how the *productivity index*, the measure developed by Putnam and used in Butler Cox's PEP, has been constructed to meet this need.

Productivity improvement is much easier to explain to general business managers. A *productivity improvement* has been made when you produce an equal or better quality system for less effort by either using fewer people, or less time, or less money (or some combination of these). This is the main goal of software investment management. One of the main aims of this paper is to help you present the results of improving system development to senior general management in these terms.

Productivity improvement is the main goal of software investment management

KEEPING NUMBERS IS ESSENTIAL TO IMPROVING PRODUCTIVITY

If you want to improve productivity, you need a way for distinguishing those projects in which the development team was relatively productive from those in which it was relatively unproductive, and for identifying those factors that have contributed to greater productivity. If you make this comparison between different projects within your own organisation you can transfer the technical and management techniques used by the most productive project teams to all the other teams to help raise the overall productivity of the systems development department. If you compare the productivity of your own projects with those of other organisations, you will be able to learn from their successes and

Productivity improvement requires measurement

failures. However, you can only make these comparisons if you collect and analyse the numbers needed to measure productivity.

Keeping numbers has other benefits. You can use them to:

— Estimate the costs of new projects.

— Plan projects on an informed and realistic basis, and so reduce cost overruns and schedule slippages.

— Assess the implications of reducing project timescales, or conversely, assess the consequences of extending the timescale and reducing the size of the team.

— Measure quality and track improvements in the quality of the systems you produce.

In our experience with PEP sponsors, the benefits of keeping the appropriate numbers far outweigh the minimal cost and effort involved.

PURPOSE AND STRUCTURE OF THIS PAPER

The aim of this paper is to explain why measurement is key to managing productivity, and in particular, the purpose and scope of the measurements used in PEP by:

— Describing the latest thinking on productivity measurement.

— Providing the PEP sponsors with a deeper understanding of the Putnam approach to productivity, which is used as the basis of the PEP measures.

— Showing how the PEP measures can be applied to managing projects and to managing the systems development function.

— Showing how to calculate the benefits to be derived from the implementation of a productivity improvement programme and how to present these benefits to senior management.

Chapter 2 deals with system development metrics — what numbers to keep, where to get them, and how to use them.

Chapter 3 describes the various approaches and models that can be used for measuring productivity.

Chapter 4 presents the Putnam approach, its history, how it was derived, what it says about managing productivity, how PEP uses it, and how you can use the PEP measures.

Chapter 5 explains how to use the Putnam approach to plan and control projects, to identify the minimum development time, to set realistic target dates, and to monitor and control progress.

Chapter 6 shows how to set productivity targets, how to improve productivity and quality, how to track productivity improvement, and how to measure the improvement.

Chapter 7 discusses the use of *return on investment* (ROI) as a powerful means of justifying the cost of your productivity improvement programme to general management.

Chapter 2

System development metrics

''But we don't keep data'' is a common response when we first discuss productivity measures, or *metrics*, with a systems development group. Indeed the PEP programme has highlighted just how few organisations have even the basic data to hand. At the other extreme some companies have kept a lot of data at great expense, only to find it difficult and time-consuming to analyse and understand.

Neither extreme makes commercial sense. If you want to improve productivity you must collect data. But the data should be readily available as a natural part of development and should be economic to collect.

Also, if such data has been proven to give value, there is an added incentive to collect it. Commercial organisations are more likely to collect the data when benefits will result.

In this chapter we set out the key numbers which you need to keep, and describe simple methods of collecting them. The numbers quantify the inputs and outputs of the development process and the environmental factors affecting the development process.

QUANTIFY THE INPUT AND OUTPUT AND IDENTIFY THE ENVIRONMENTAL INFLUENCES

The essential numbers are measures of the *input* (investment) made to generate a given *output* (the system). The input is the cost of employing a development team to create the new or enhanced system within a given timescale. These costs can be calculated by using fully loaded staff costs which include other costs such as development hardware, software tools, and so on.

Outputs from each project stage include specification and design documents, detailed software design and development, and finally, operation of the new or amended system. The output is tangible. Most simply, it can be seen as the source statements that constitute the delivered system. These statements are the end product of the input to the project, the development team. You can measure the functionality of the system encapsulated in the source statements in a number of ways. These are discussed later in this chapter.

Quality is also a vital output measure. The errors found in the systems are a simple and measurable indication of quality.

There are many complex environmental factors at work that can influence the development team's performance and the quality of the finished product. Management style, the use of formal

The development environment is complex. Isolating the key factors is essential

methods, and machine availability are just a few examples. Some of these factors will be outside your control; others, you can influence or change.

We believe that only by calculating and analysing the productivity measures, can you identify and isolate the factors that influence productivity in your unique environment. However you must first record information on the key environmental factors so that you can use the productivity measures as the magnifying glass to show you which factors have a significant influence.

THE STAFFING PROFILE IS A SIMPLE METHOD OF CALCULATING THE INPUT

The straightforward way to calculate the total man-months of effort is to use the staffing profile for each stage of the project. The project manager can usually sketch this out in five to ten minutes. There is no need to go into detail in distinguishing between productive and non-productive contributions as these broad-brush figures are adequate for projects with 18 man-months effort or more. We find the balance of total productive versus non-productive man-days per person in a year is usually very stable, the average being around 200 to 210 productive man-days. The staffing profile is straightforward, requires little effort to collect, and it is readily available.

You could assemble much more detailed data, usually at great cost, by calculating individual resource contributions, and distinguishing between productive and non-productive time (holidays, training, and so on). This approach relies on a time-recording system and, usually, cost accounting. Based on our experience, we are now cautious about using cost-accounting data. The information is often inaccurate, too detailed, and lacking suitable summaries. To analyse this data usually takes much longer than asking the project managers to sketch the staffing profile.

Simple, broad-brush metrics are sufficient for measuring input

As an instance of the broad-brush numbers that can be collected, Figure 2.1 sets out the staffing profile constructed over the key development stages *feasibility, specification and design, and main build.* Total staff used each month are shown without detailing who was on holiday, absent, or engaged in non-productive

Figure 2.1 System development stages: staffing profile

This figure shows the relative magnitudes of staff sizes used in the development stages defined for PEP analyses.

Number of staff
Time (months)
Feasibility
Specification and design
Main software development build: detailed design/code/unit test/ integration/system test
Date live
©QSM

activities. You will notice that some of the development stages overlap. The extent of this overlap needs to be assessed (but only approximately) because separate figures are required for the effort used in each phase.

EFFECTIVE LINES OF CODE OR FUNCTION POINTS CAN BE USED TO MEASURE OUTPUT

Counting the number of *effective lines of code* (ELOC) is one of the easiest ways of measuring the output end product. Many PEP sponsors automate the collection of the ELOC statistics by writing a program that scans the appropriate libraries to count lines of code based on the following rules:

— Lines are indicated by delimiters.

— Only executable lines are counted, not expansions.

— Comments are not counted.

— Delivered lines only are counted (those eventually thrown away are ignored).

— New or amended lines only are counted, not unchanged lines.

— Data definitions are counted once only.

Alternatively, you can take a sample of programs or transactions. Numbers accurate to the nearest 1,000 ELOC are adequate for meaningful analysis. This consistent collection of sizing data quantifies the output produced by the development team.

Effective lines of code (ELOC) is the easiest measurement of size

The 200-plus projects analysed to date in PEP account for a total output of some 22 million ELOC. Cobol and PL/1 make up 18 million of them and fourth-generation languages another 2.5 million ELOC.

PEP sponsors are able to gather this information on ELOC quickly and, at the macroscopic level, accurately. Few sponsors have any other measure of the developed functionality available.

Another measure of the end product that is used by some PEP sponsors is the total number of *function points*. This approach was developed by Albrecht and involves counting external user inputs, enquiries, outputs, and master files to be delivered by the development project. Guidelines are available for counting these *function points* (which Albrecht considers to be the outward manifestation of any application). However, counting is not readily automated although you can use a spreadsheet to sum the values. We return to the use of function points in the next chapter.

ERRORS CAN BE USED TO MEASURE OUTPUT QUALITY

The technical quality of the end product can be measured by counting the number of errors. In practice it is useful to distinguish between three categories of error:

— Statement of Requirement Errors (SORs), essentially highlighting errors found in the requirements specifications.

— Software Errors, (SERs), usually those bugs found during the integration and system test.

— No Faults, (NOFs), a null category which is invaluable to identify misunderstanding by testers or end users of what the system is required to do.

Within these categories you need to distinguish levels of seriousness. Usually, three levels are sufficient:

— Critical.

— Serious.

— Moderate.

Integration and test staff usually log the software errors they find and pass them to the programmers to correct. The log is updated as errors are corrected. By analysing the log, a count of total software errors can be made. (If no log is kept this highlights an immediate action you can take to improve quality.)

Errors found is the easiest measurement of technical quality

You can keep more detailed information on errors, for example, relating errors to test data and to test plans. However, the three categories SOR, SER, and NOF, and the three levels of seriousness, are adequate to give useful measures of the quality of the development output.

GATHERING ADDITIONAL INFORMATION ON THE DEVELOPMENT ENVIRONMENT

Besides measuring the inputs and outputs of the development process we need to collect further data on the environmental factors that could influence the efficiency of the development process. There could be tens, or even hundreds of such factors. To contain the costs and efforts of collecting such data, they should be limited to only the most important factors.

In PEP we extend the information we collect on projects to an additional 40 or so items on the project-development environment. These are listed on the PEP questionnaire. In consulting assignments for individual organisations, QSM use an extended questionnaire that seeks information on some 80 characteristics. The data collected from the extended questionnaire includes:

Collect data on only the most important environmental factors

— Development policy.

— Formal methods and supporting techniques.

— Computer-based aids.

— Application complexity.

— Skills.

— Machine availability, response times, and turnaround times.

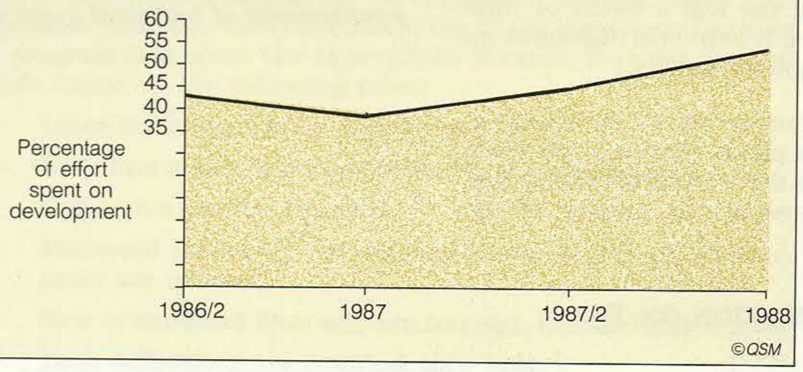
This data is analysed and interpreted across all projects to determine the essential characteristics of the development environment. Powerful insights result when these qualitative analyses are combined with quantitative measures. We can determine those factors unique to the organisation that are influencing productivity and quality, and make the business case for tackling those factors reducing productivity.

HOW THE ESSENTIAL NUMBERS CAN BE USED

If you collect data on *all* significant development projects, you can consolidate the effort to give the total effort consumed in actual development. If you subtract this from the total development staff employed, the balance shows you how much work is taking place in maintenance and general support. As a percentage the figures can be used as a baseline to monitor use of resources on a regular six-monthly or annual basis. Over time the figures give an understanding of whether effective development effort is increasing or not. Figure 2.2 illustrates this analysis.

Figure 2.2 Development effort

This figure shows how the percentage of available effort that is spent on development changes over time.



The basic input data can also be used to determine the proportions of effort and time used over the feasibility, functional design, and main build phases. PEP includes this as a key analysis. You can use this analysis to compare your own projects with the data from other organisations where such data is held in a reference database.

Using both the input and the output data you can explore the complex relationships between size, time, effort, and quality in the main build stage. The way this is done in PEP is described in Chapter 4. This can provide insights into the behaviour of the development team in response to time pressure or staffing constraints. Again, comparisons can be made with similar reference data from other organisations.

The analysis may highlight an exceptional project with significantly different development characteristics from your other projects. These exceptional projects frequently provide very convincing evidence of the interactions between size, time, effort, and quality: for example, the cost (in effort and quality) of time compression.

Another benefit from positioning the projects against measures from a reference database is that you have a baseline for comparing plans for new developments. This comparison can help to ensure that your planning assumptions are realistic and consistent with previous projects. Taking this a step further, as new developments are completed, they can be compared against the original baseline. In this way you can produce evidence to measure and quantify real productivity and quality gains as well as update your own baseline for future comparisons.

Metrics can be used as an aid for planning, managing, and improving

Hence, armed with the essential numbers, and with access to similar reference data, you can gain an informed understanding of development team behaviour within your own organisation. You can model your development process and use the model to help you manage the new software investments more effectively.

In the next chapter we describe some of the established reference databases and the models which exist of the development process.

Chapter 3

Modelling the software development process

In the context of this paper, the purpose of modelling the software development process is to understand what determines development effort and time. You can then use this knowledge in two ways. First, you can improve productivity by changing those factors which are having a negative effect. Second, you can use the model to estimate the costs of new projects and to determine realistic timescales. The payoffs can be huge as we illustrate in Chapter 7.

There are two conceptual approaches to modelling. The *empirical* approach first gathers the data from a large number of projects, and then looks for patterns in it. The patterns are isolated and progressively refined until a model can be constructed which closely fits the collected data. The acid test of a model produced empirically is to see whether new data conforms to the same patterns. In contrast, the *theoretical* approach involves proposing a theory about the way the software development process works and about the factors influencing the productivity of development teams. The next step is to collect data from a number of projects that can be analysed to prove (or disprove) the theory.

There are two approaches to modelling: theoretical and empirical

Some empirical studies have identified as many as 176 variables that correlate with measures of software development output. However, a model which included that many variables would not be practical or economic to use. Pareto's 80/20 principle applies; it is much better to construct a simple model which accounts for most of the variance than to include all the variables which are seen to have some effect, however small.

Furthermore, the observed correlations are not always simple proportional or linear ones. Relationships between the variables usually involve power functions.

DATA SOURCES FOR MODELLING

We now discuss some of the databases which can be used either for empirical modelling or for testing theories. Of those databases known to us, that assembled by QSM (including some of the data from PEP assessments) is the largest, and most representative source of reference data. The data in the PEP database is rapidly growing and details development experience mainly in business systems.

QSM DATABASE

QSM has accumulated data on over 4,000 systems, and from these has compiled a carefully validated database of about 1,500 systems. This database of systems encompasses a very wide range of application types, in particular, business, scientific, operating systems, telecommunications, process control, command and control, radar, avionics, and realtime embedded firmware (ROM) Microcode. Of the 1,500 systems, 60 per cent, or about 900 are classed as management information or business systems.

Several hundred systems per year are being added to the database as a result of consulting activities, research, and PEP assessments. The database now contains statistics on the development of a total of 117,000,000 lines of code in 76 different languages resulting from 39,272 man-years of effort. In 1987 alone, 293 systems were added of which 127 were business systems. The average size of the systems that were added in 1987 was about 65,000 lines of Cobol.

Rolling deletion is being started to maintain the relevance of the data to current systems development environments and practices. The first cut will remove the systems that pre-date 1983, approximately 15 per cent of the total.

Data that is now being added include a number of qualitative items as well as quantitative measures. Details of the Productivity Analysis Database System (PADS) data questionnaire can be seen in the PEP data-capture instruction manual.

Although QSM do not currently provide an analysis of the data by industry, an enhancement to allow such analyses is on the development agenda. We would expect to see baselines for some industries available in 1989.

The QSM database contains statistics on development projects from Europe, the USA, Japan, and Australia. To our knowledge, the QSM database is the largest and most comprehensive collection of software data that has been analysed and the results published.

PEP DATABASE

The PEP database is relevant, extensive, and evolving

The Butler Cox PEP database follows the same format as the QSM database. It now has over 200 systems that have been submitted for analysis by PEP sponsors. Of these almost all are business systems. We expect more than 400 systems to be added in 1988.

The Butler Cox PEP database currently contains statistics on the development of 22,000,000 lines of code in 58 languages. Cobol is reported as the primary language in over 50 per cent of the projects. This data reports on over 1,100 man-years of effort. It has been drawn from organisations in Belgium, France, Ireland, the Netherlands, and the United Kingdom.

Many of these systems are being anonymously included in the QSM database (only Butler Cox retains a key to the identification and this only in order to allow correction of the data).

RADC DATABASE

Richard Nelson collected a large software database at the Rome Air Development Center (RADC) covering more than 400 systems (Putnam 1980). The systems presented an enormous range of sizes from less than 100 source statements to more than a million. Project duration ranged from less than a month to more than six years. The projects varied in size from one man-month to 20,000 man-months of effort. The simple measure of source statements per man-month ranged from 10 to several thousand. The data spanned a wide range of applications.

This data was analysed in various ways and project durations, total man-months, and average number of staff correlated significantly with the number of source statements. It is interesting to note that no correlation was found between the source statements per man-month and the size of the system.

IFPUG DATABASE

The International Function Point Users' Group has gathered data on 292 projects (Emrick 1987). This includes data from 12 companies and encompasses 1,022 man-years of effort. The average length of project was 10.5 months. The data is very detailed and even includes the square-feet of working space used by the development team.

The systems submitted covered a total of 26 languages. Cobol was used as the primary language in 66 per cent of projects and no other language exceeded five per cent of the total as the primary language. Cobol was used as a secondary language in 27 per cent of projects, followed by MARK IV (12 per cent), assembler (9 per cent), and RAMIS (7 per cent).

Database management systems were used by 55 per cent of the projects. IMS was the most widely used (77 per cent of systems), followed by System 2000 (13 per cent), and IDMS-IDMS/R (6 per cent).

SOFTWARE DATA LIBRARY

The Alvey Programme in the United Kingdom is running an ambitious project in conjunction with the National Computer Centre and a consortium of other organisations. This project, called the Software Data Library, is collecting extensive data on software development projects. Unfortunately, publicly available information about the database is limited, but when the data is available for research, we believe it will provide a valuable basis for examining the characteristics of software development in the United Kingdom.

SYSTEMS DEVELOPMENT MODELS

Boehm (who developed the COCOMO model discussed later) lists 10 evaluation criteria with which to judge models. These are shown in Figure 3.1 (overleaf).

When you are choosing a model for practical use you also need to consider the extent to which the model has been validated with data from real projects. If it is an empirical model, has it been verified with a substantial set of alternate real data? If it is a more theoretical model, has it been thoroughly tested with a large enough database of real data?

In addition to the model developed by Putnam, used for PEP, there are two other widely known approaches. These are the COCOMO model which was derived empirically from data on a limited set of projects, and the function-point model which has a more theoretical basis.

THE COCOMO MODEL

The COnstructive COst MOdel (COCOMO) was developed by Barry W Boehm of TRW, Inc, (Boehm, 1981). It is a model for software

Figure 3.1 Boehm's evaluation criteria for software development models

"1. Definition. Has the model clearly defined which cost it is estimating, and which costs it is excluding?

2. Fidelity. Are the estimates close to the actual costs expended on the projects?

3. Objectivity. Does the model avoid allocating most of the software cost variance to poorly calibrated subjective factors (such as complexity)? That is, is it hard to jigger the model to obtain any result you want?

4. Constructiveness. Can a user tell why the model gives the estimates it does? Does it help the user understand the software job to be done?

5. Detail. Does the model easily accommodate the estimation of a software system consisting of a number of subsystems and units? Does it give (accurate) phase and activity breakdowns?

6. Stability. Do small differences in inputs produce small differences in output cost estimates?

7. Scope. Does the model cover the class of software projects whose costs you need to estimate?

8. Ease of Use. Are the model inputs and options easy to understand and specify?

9. Prospectiveness. Does the model avoid the use of information which will not be well known until the project is complete?

10. Parsimony. Does the model avoid the use of highly redundant factors, or factors which make no appreciable contribution to the result?"

cost estimation that was based on a carefully screened sample of 63 projects representing business, industry, government, and commercial software-house organisations. It estimates the cost of developing a proposed software product by:

- Estimating nominal development effort as a function of the product's size in thousands of delivered source instructions.
- Determining a set of effort multipliers from the product's ratings on a set of 15 attributes which Boehm refers to as 'cost drivers'.
- Multiplying the estimate of nominal effort by all of the product's effort multipliers to yield the estimated development effort.
- Applying additional factors to the development effort estimate to determine dollar costs, computer costs, annual maintenance costs, and other cost elements.

There are three levels of complexity of the COCOMO model but the fundamental approach is the same in each.

There are other popular models that have been derived empirically in a similar way to Boehm's. They include Price-S, developed by RCA; SPQR, developed by Capers Jones; and Estimacs developed by Howard Rubin. In all cases the approach is similar to Boehm's with differences in the number and nature of cost drivers and the complexity of the calculations. However Price-S claims to model all application types whereas SPQR and Estimacs are restricted to business systems.

THE FUNCTION-POINT APPROACH

In modelling the development of business software, another approach is well known. Using the theoretical approach to modelling, Albrecht has developed a method for validating estimates of the amount of effort needed to design and develop custom application software (Albrecht and Gaffney, 1983). The approach involves listing and counting the number of external user inputs, inquiries, outputs, and master files to be delivered by the development project.

Each of the categories of input and output are counted individually and then are weighted by numbers reflecting the relative value of the function to the user/customer. The weighted sum of the inputs and outputs Albrecht calls 'function points'. Albrecht's weights were determined by 'trial and debate'.

The premise of Albrecht's approach is that the amount of function to be provided by the application can be estimated from an itemisation of the major components of data to be used or provided by it. Albrecht hypothesises that this estimate of function is correlated with the amount of delivered lines of source code to be developed and the effort needed. Indeed, independent data made available to us supports Albrecht's hypothesis that there well may be a stable linear relation, a ratio, between function points and lines of code. Furthermore, this data suggests the ratio is dependent on the language employed and the nature of the application.

These findings imply that function points are a good measure of system size. Thus effort and duration should bear similar *nonlinear* relationships to the number of function points as they have been demonstrated to do with lines of source code. An interim report on the analysis of The International Function Point Users' Group (IFPUG) database, presented at the Atlanta meeting of IFPUG in August 1987 by Ronald Emrick of GTE, appears to support this. A summary comment he made at the meeting (as reported by Tony Reid of QSM) was that the correlation of function points per man-month with other productivity-related measures was "just miserable." The final version of this paper will be read by Emrick at the IFPUG meeting in Dallas, Texas, on the 16 to 19 May 1988.

Function points are a good measure of system size. Function points per man-month is not a good measure of productivity

We recommend strongly that those organisations using or contemplating the use of the function-point model obtain copies of the final Emrick paper. We intend to cover the full results and the appropriate use of the function-point approach in a subsequent PEP publication.

We believe that the Putnam model, based on the QSM database, provides the most practical tool for assessing system development productivity in the current state of development of such models. We describe it in more detail in the next chapter.

Chapter 4

The Putnam approach

The Putnam approach is at the heart of PEP productivity assessments. To get full value from PEP you need to understand how the model underlying it was developed, how the outputs are derived, and how PEP uses them. This chapter covers the basic principles of Putnam's approach. Full technical details can be obtained from his published papers which are listed in the bibliography.

Initially software development managers regarded manpower and time as interchangeable. If they wanted to complete a project in half the time they put twice as many people on it. During the 1970s it became clear from the work of several researchers that this approach does not work and that the relationships between the factors that influence productivity are not simple linear functions:

Manpower and time are not directly interchangeable

— Fred Brooks showed, in his book *The Mythical Man Month* published in 1975, that manpower and time are indeed not interchangeable.

— Peter Norden of IBM showed that hardware development projects are composed of overlapping phases, and that these phases have a well-defined manning profile that matches a mathematical function — the Rayleigh curve.

— In 1976 Joel Aron of IBM recognised that the manpower in large developments builds up in a characteristic pattern and identified complexity and duration as key elements affecting development productivity.

— In 1977 Walston and Felix of IBM collected consistent data on 60 completed software developments. They show that the variables of interest appear as complex power functions of the size of the system.

— Larry Putnam extended the earlier work by Peter Norden of IBM on hardware development to software projects. He found that the Rayleigh curve also fitted not only the individual components of a software development project but the entire project. He also refined the power functions described by Felix and Walston.

All this research was empirically based.

HOW THE PUTNAM APPROACH IS DERIVED

The Putnam approach divides development projects into three basic phases. These phases can accommodate the development processes in most organisations. The three basic phases are:

— *Feasibility study*, which stops at the point where the outline requirements specification and the project plans are approved. The data recorded in this phase are time and effort.

- *Functional design*, which continues to the point when all functional design specifications, test plans, and management plans are approved. It can overlap the main build phase. In this phase, time, effort, peak staffing, and overlap with the main build phase are recorded.
- *Main build*, which begins at the start of detailed logic design and ends when the system reaches full operational capability. Full operational capability is defined to be the point at which all system and integration tests are successfully completed. In the main build phase a substantial amount of data can be recorded. Key items are time, effort, peak staffing, size of the system, and errors from the start of integration test to first operational capability.

Once the system is operational, additional data may be collected. (This is sometimes referred to as the *maintenance and operational* phase.) The data includes mean-time-to-failure, errors in the first month after first operational use, and cumulative time and effort spent by development staff in operations and maintenance.

Main build — 75 per cent of development effort

In Putnam's approach, his engineering analysis is applied to the main build phase, where on average over 75 per cent of the development effort reported by PEP sponsors is expended.

Putnam chose to use an empirical approach to identify the relationships between the key management numbers. He found that the most useful way of analysing the data is to relate six project measures to the size of the project, expressed as effective source lines of code (ELOC). These measures are:

- Duration, the time taken in months.
- Effort, in man-months (cost).
- Average manpower, defined as effort/duration.
- Average code production rate, in ELOC per month.
- Error rate.
- Productivity, in ELOC per man-month.

Figures 4.1 to 4.3 (overleaf), show the development project duration, effort, and error rate plotted as a function of system size for a wide variety of projects.

These graphs are plotted against logarithmic scales to accommodate the wide range of values, and to cater for the nonlinear relationships involved.

The slopes of the correlation lines demonstrate there are nonlinear relationships between the dependent variables and the independent variable, system size.

The graphs also show the wide variability on the data for any given size of system. It indicates the absence of any simple pattern based on a small number of variables. This is because of the great variation in application types and in the time period of the developments. However, a large database can be partitioned to eliminate some of the major sources of variation including application type, time, and developer efficiency. (Figure 4.4 overleaf, shows the application types used to partition the QSM database.) Putnam's results show that when this is done very useful behaviour patterns do emerge.

Figure 4.1 Software development schedule for main software construction phase

This figure shows that the time taken increases with the size of the system, but there is a large variability in time for any given size.

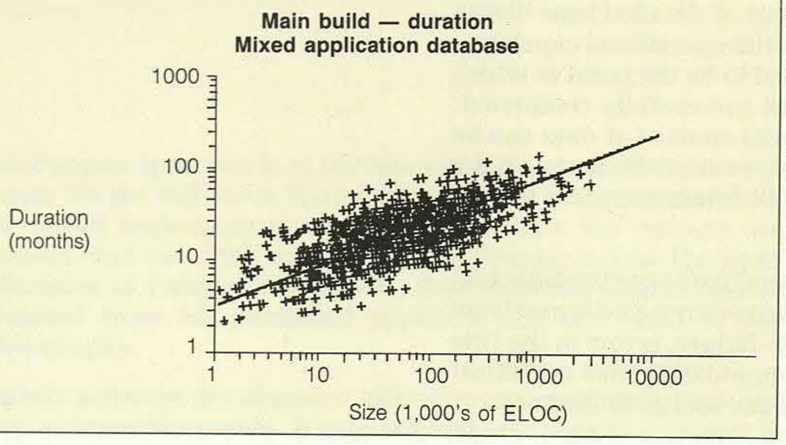


Figure 4.2 Development effort for main software construction phase

This figure shows that the effort increases with the size of the system, but there is wide variability in effort for any given size.

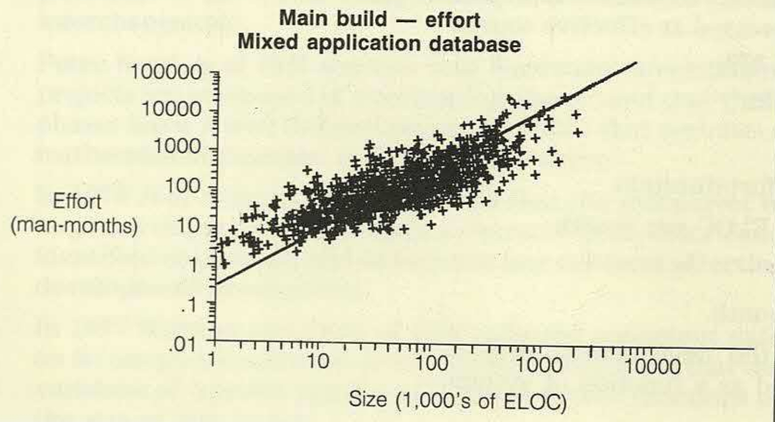


Figure 4.5 shows the correlation between effort and system size for an application type in the form of a trend line together with two other lines. The middle line represents the least squares best fit. The upper line is plus one standard deviation and the lower minus one standard deviation. If the variability follows a normal distribution 67 per cent of the data will be expected to lie between plus and minus one standard deviation.

Standard slopes for the trend lines have been determined using a combination of statistical curve fitting, and bootstrap statistical simulation. The intercepts are determined directly from the appropriate data set. Slopes and intercepts are then verified for reasonable closeness of fit from all the available data in the specified data set.

This procedure is necessary because pure curve fitting may produce poor results when the data set is sparse, noisy, poorly

Figure 4.3 Errors discovered from start of systems integration testing through to full operational capability

This figure shows that the number of errors increases with the size of the system, but there is wide variability.

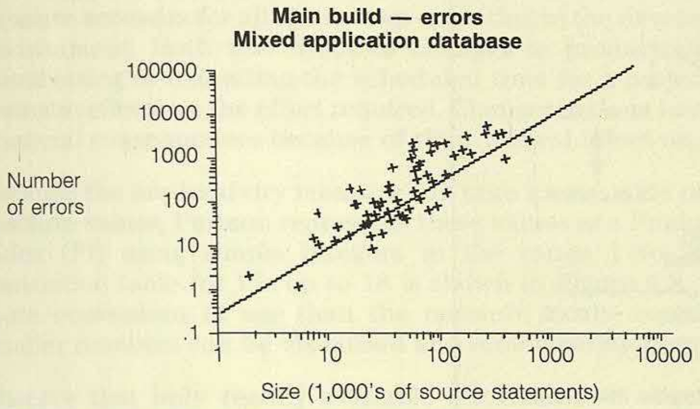


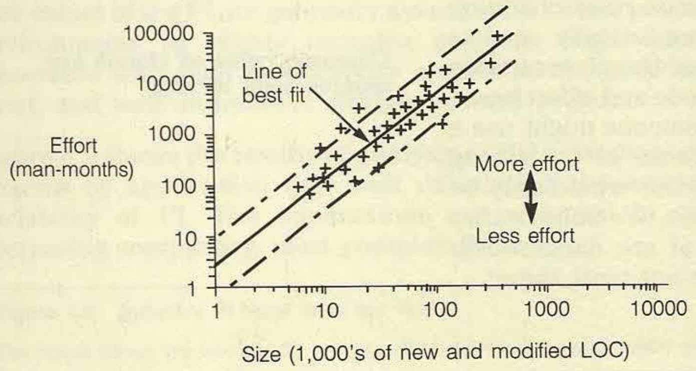
Figure 4.4 Application types

Listed below are the 11 information system application types covered by the QSM database.

- Microcode
- Firmware (ROM)
- Realtime embedded systems
- Avionics systems
- Radar systems
- Command and control systems
- Process control systems
- Telecom systems
- Systems code systems
- Scientific systems
- Business systems

Figure 4.5 Correlation between effort and system size for a single application type

This figure shows the better correlation obtained once the data is partitioned into different kinds of system.



distributed, and so on — all quite common. The trend lines have been verified by independent data sets, worldwide, over a period of seven years. They are continuing to be refined. They are reliable and well represent software behaviour within our current recording and measurement accuracy.

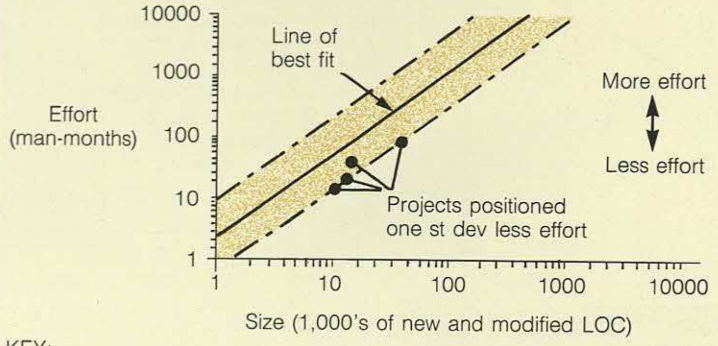
Putnam asserts that all six measures have a distinct characteristic behaviour as system size increases. Duration, cost, manpower, code production, and errors all increase with size. Productivity, expressed as ELOC/man-month decreases with system size. All these relationships are nonlinear.

Since Putnam's six project measures increase with system size it follows that all comparisons should take size into account. Figure 4.6 shows four projects of different sizes positioned with respect to the trend lines.

We conclude that these four projects required about one standard deviation less effort than the average for other systems of the same

Figure 4.6 Example of four systems positioned with respect to the effort trend lines

This figure shows four projects of different size positioned relative to the trend lines. They all require about one standard deviation less effort than the average.



KEY:
– —— – These lines represent ±1 standard deviations from the line of best fit. Two-thirds of projects should lie between these boundaries.

size in the database. You can make similar comparisons for duration, manpower, code production rate, productivity, and error rate.

The trend lines also dispel myths about common rules-of-thumb that abound in the industry and that are simplistically used to make multimillion-dollar decisions. The rules-of-thumb invariably assume some constant ratio between lines of code and effort based on a very small data sample. For example, someone might use a rate of 180 ELOC per man-month. Yet because of the strong variation of ELOC per man-month with size, this would only be reasonably valid for a very small range of sizes. We believe that most of the rules-of-thumb we are aware of are dangerously wrong, especially ELOC per man-month. Do not trust them!

Common rules of thumb are dangerously wrong

PUTNAM'S SOFTWARE EQUATION

Putnam's research shows that it is possible to derive a mathematical relationship between the size of a project and the time and effort needed to complete it. But the relationship is nonlinear, not a constant ratio, and involves a parameter which is a measure of productivity. From the relationship (the 'software equation') we can obtain a computational formula for a productivity measure which allows us to compare the productivities of different development projects even if they are of different size or duration. It is of the form:

The relationships between system size and effort and time are nonlinear

$$\text{Productivity measure} = \frac{\text{size}}{(\text{effort}/B)^{1/3} \times (\text{time})^{4/3}}$$

where time is in years, effort is in man-years, and B is a special skills factor directly related to size (see Figure 4.7).

Given a certain level of productivity, the equation may be rearranged to calculate the effort required to complete a project of a given size.

Figure 4.7 Relationship between the skills factor, B, and size of project

This figure shows how B, the special skills factor, varies with system size.

Size (ELOC)	B
5-15k	0.16
20k	0.18
30k	0.28
40k	0.34
50k	0.37
<70k	0.39

$$\text{Effort} = \frac{\text{B x size}^3}{\text{time}^4 \text{ x (productivity measure)}^3}$$

This equation shows that effort required for a project depends on the duration as well as the size, and on the productivity measure which applies in the particular development environment. The measure accounts for all the factors operating in the development environment. Both the effect of changes in productivity and compressing or extending the scheduled time for a project have dramatic effects on the effort required. Changes in them have large financial consequences because of their critical effect on effort.

Because the productivity measure can take a very wide range of absolute values, Putnam represents these values as a Productivity Index (PI) using simple integers in the range 1 to 25. The translation table for PIs up to 18 is shown in Figure 4.8. This is more convenient to use than the measure in the equation as smaller numbers can be visualised and remembered more easily.

Information required to calculate PI is readily available

Observe that only readily available information is required to calculate the PI for completed projects and those in progress:

— The number of new and modified source lines of code (which also allows us to determine B).

— Total man-months of effort.

— Total elapsed calendar months.

Low values of the PI are generally associated with low productivity environments or highly complex projects. High values are associated with high productivity environments, good management, and well-understood straightforward projects.

Figure 4.9 shows the results of analysing QSM's database of 1,500 systems by application type and calculating the average and variability of PI. The applications are arranged in order of decreasing complexity, most complex first.

Figure 4.8 Productivity index (PI) and the associated productivity measure

This figure shows the translation from productivity measures to productivity index.

Productivity measure	PI
754	1
987	2
1220	3
1597	4
1974	5
2584	6
3194	7
4181	8
5186	9
6765	10
8362	11
10946	12
13530	13
17711	14
21892	15
28657	16
35422	17
46368	18

Figure 4.9 Industry PI base lines for 1987

This figure shows the average PI and standard deviation by application type.

Application type	Average	Std. deviation
Microcode	2	+/−1
Firmware (ROM)	4	+/−2
Realtime embedded systems	5	+/−2
Avionics systems	5	+/−2
Radar systems	7	+/−3
Command & control systems	8	+/−3
Process control systems	9	+/−3
Telecom systems	11	+/−3
Systems code systems	12	+/−3
Scientific systems	12	+/−3
Business systems	15	+/−4

INCREASES IN THE PRODUCTIVITY INDEX MEAN LARGE IMPROVEMENTS IN PROJECT COSTS

Small increases in productivity can generate enormous savings

Any movement in the PI has a dramatic impact on the time, total effort, and hence total cost of development. The PI embraces all the environmental factors impacting development. If you have a low PI, you may find there are bottlenecks that are acting as brakes to efficient production. Figure 4.10 shows a simple example of the economic value of a PI increase. Note that this economic value is high. An increase of one PI for a 30,000 line Cobol system saves close to two hundred thousand dollars. When you invest in tools, techniques, and management practices that relieve bottlenecks, the PI goes up. The effects of high and low values for the PI are summarised in Figure 4.11. An increase in the PI reflects decreases in time, effort, manpower, cost, and errors.

Figure 4.10 Impact of changing PI

This figure shows the financial and time impact of increasing the PI.
Modest system of 30,000 Cobol SLOC
Labour rate — $60,000 man-year

PI	Man-months	Time (months)	Cost ($)
9	146	17	$729,000
10	108	16	$537,000
11	80	14	$400,000

Impact: higher PI = higher productivity

Figure 4.11 PI Impact

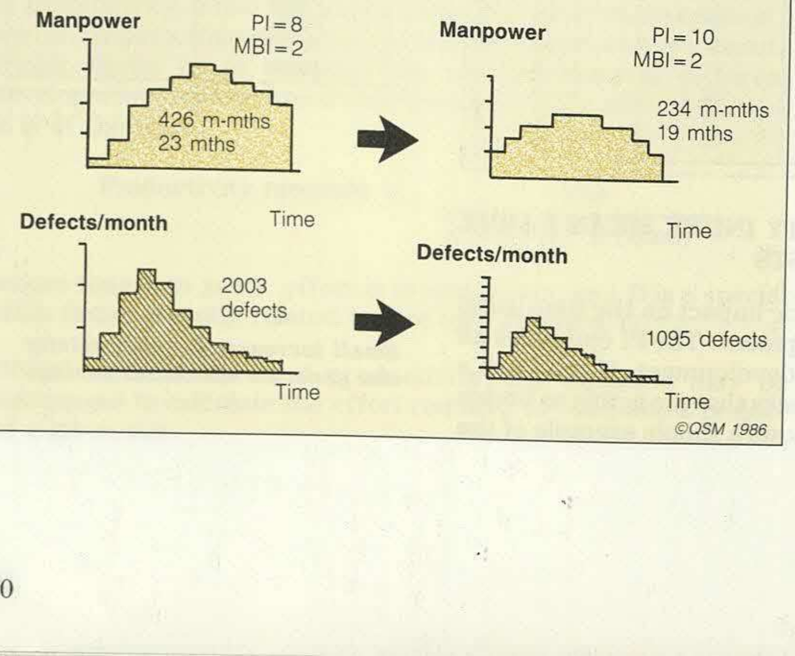
This figure summarises the impact of changes in the PI upon the project.

High PI	Low PI
Less time	More time
Less effort	More effort
Fewer defects	More defects
Higher MTTD	Lower MTTD
Fewer people	More people
More SLOC/month	Less SLOC/month
More SLOC/man-month	Less SLOC/man-month

Figure 4.12 shows the effect on the resource and error profile of a capital investment that boosts the PI from 8 to 10. Clearly the PI is a measurement which management needs to understand and exploit for its impact on a company's profitability.

Figure 4.12 Impact of capital investment on resources and defect profiles

This figure shows the reduction in resources and defects as a result of investments which raise the PI from 8 to 10 for a product of a given size.



THE MANPOWER BUILDUP INDEX (MBI) REFLECTS TIME PRESSURES

The software equation also accounts for compression or extension of the project schedule. The software equation shows that when you compress the timescale for a project the effort increases substantially. One of the reasons for this is that as you overlap tasks you need more staff to work on the project, which means more communications paths and more overheads. Putnam represents the effect of time compression by using a measure which he calls the Manpower Buildup Index (MBI). (In mathematical terms, this is the manpower acceleration of the Rayleigh curve.)

Compressed timescales cause substantial increases in effort

He defines the MBI parameter as Effort/(B x Time^3) where effort is in man-years, time is in years, and B is the same special skills factor as in the software equation. As with the PI, Putnam expresses the MBI as a simple integer value (level) which is more readily appreciated by business managers. The relationship between the MBI parameter and the integer MBI levels is shown in Figure 4.13.

Level 1 represents a slow staff buildup. The project will take the longest and cost the least. Usually, it reflects a limited number of staff available for development. Level 6 can be described as the 'throw people at it' approach. It is characterised by attempting totally parallel task execution, with no staff or money limitations, and assumes all design issues are well understood from the outset. Level 6 is the fastest and most expensive staffing profile.

The lower the rate of staff buildup, the lower the cost of the project

Figure 4.14 shows the economic impact of an increase in the MBI. If we were to increase the MBI from one to three in an effort to compress the schedule it would more than double the effort, and hence cost.

Figure 4.15 shows why the cost increases so dramatically. The number of human communication paths for the Level 3 MBI is

Figure 4.13 Manpower buildup index

This figure shows the relationship between the simple MBI scale and the computed values of the MBI parameters from the data.

MBI parameter	MBI level
7.3	1
14.7	2
26.9	3
55.0	4
89.0	5
233.0	6

Figure 4.14 Economic impact of MBI

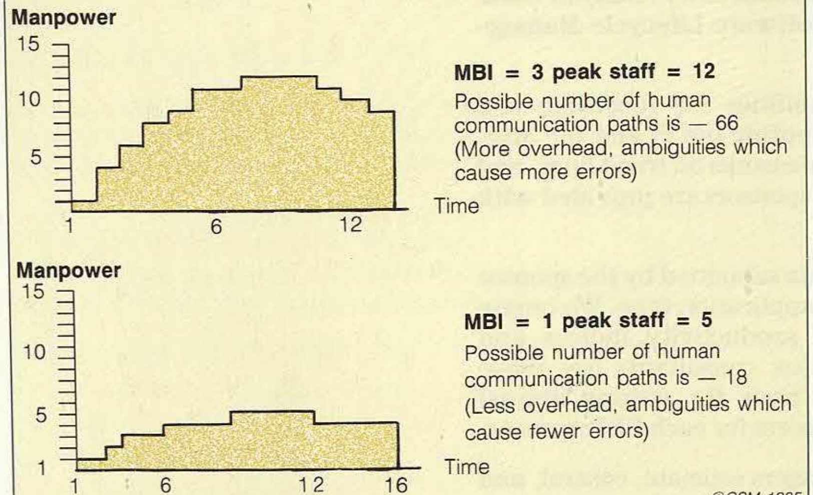
This table shows the dramatic effect that changing the MBI has on the effort required.

(30,000 Cobol; PI = 11)

MBI	Time (months)	Effort (man-months)
1	16	55
2	14	80
3	13	120
4	12	180
5	11	235

Figure 4.15 Human communication paths

This figure illustrates why the cost increases so dramatically with changes in the MBI.



about six times that for a Level 1 MBI. This also manifests itself directly in quality terms by causing exponentially more defects.

Clearly, schedule compression is very expensive. This is important to recognise because it is very commonly done, with little appreciation of the consequences. Figure 4.16 summarises the effects of high and low values of the MBI. The MBI is a parameter that managers can influence enormously since it is within their immediate control. The figure shows how modest changes in schedule have a great impact on cost and quality.

Figure 4.16 MBI Impact

This figure summarises the impact of changing the MBI upon the project.

Low MBI	High MBI
Longer time Fewer people Less effort Fewer defects Longer MTTD Fewer LOC/month Higher LOC/man-month	Shorter time More people More effort More defects Less MTTD More LOC/month Lower LOC/man-month

USING THE PUTNAM APPROACH TO ANALYSE THE PEP PROJECT DATA

PEP uses the Putnam approach to prepare the annual individual productivity assessment reports for each PEP sponsor. This also enables Butler Cox to create a PEP reference database which increasingly reflects the particular characteristics of the PEP group. Compared with the QSM database, the projects submitted to PEP are much more homogenous. The applications are all broadly similar and all were developed recently. As a result, we are seeing great consistency between projects. This implies that the predictions we can make for PEP sponsors on potential cost savings will become even more accurate. Our analysis of this database will also help us to identify the characteristics of the sponsor group, and to be more responsive to their problems and needs.

PEP uses two commercially available tools that implement the Putnam approach: QSM's PADS, the Productivity Analysis Database System, and QSM's SLIM, the Software Lifecycle Management Methodology.

PADS is a tool that provides capabilities for recording and analysing software project data, computing the PI and the MBI, displaying the data against reference measures on trend lines, and consolidating the recorded data. PEP sponsors are provided with the PADS data-collection modules.

We use PADS to position the project data submitted by the sponsor on the appropriate trend lines for the application type. We create consolidation graphs, and compute productivity indices and manpower buildup indices. Butler Cox consultants use these graphs, tables, and indices as the basis for writing annual individual productivity assessment reports for each PEP sponsor.

SLIM was designed to help senior managers estimate, control, and measure software developments. It is a strategic planning and capital budgeting tool for development management. It allows 'what if' analysis of different development plan parameters.

We use SLIM to compute tables for each individual PEP assessment that show the effects of changing the PI and the MBI in the context of the sponsor's development environment. These tables have two purposes: first, to help PEP sponsors understand the costs and benefits of their current approach to systems development; second, to show the concrete benefits that can be achieved by taking management actions to increase the productivity index and reduce the manpower buildup index. We discuss in Chapter 7 how these numbers will permit the sponsors to compute the return on investment in software development.

Chapter 5

You can use the Putnam model to plan and control projects

In Chapter 4, we described the Putnam approach and how this uses the Productivity Analysis Database Systems (PADS) to compute two fundamental measures, the Productivity Index (PI) and the Manpower Buildup Index (MBI). These measures are calculated using the basic data of development time, effort, and size from completed projects.

Using the PI and MBI as inputs to Putnam's software equation together with the estimated size, you can plan the development time and effort for new projects. You can also use 'what if' analyses to investigate management actions such as the effects of shortening or lengthening the timescales. The SLIM software product incorporates the software equation and makes the calculation of practical alternatives easy.

To determine the development time and effort requires three parameters:

Project planning requires only three parameters

— The PI, determined from the PI calculated for similar projects completed within the organisation's development environment.

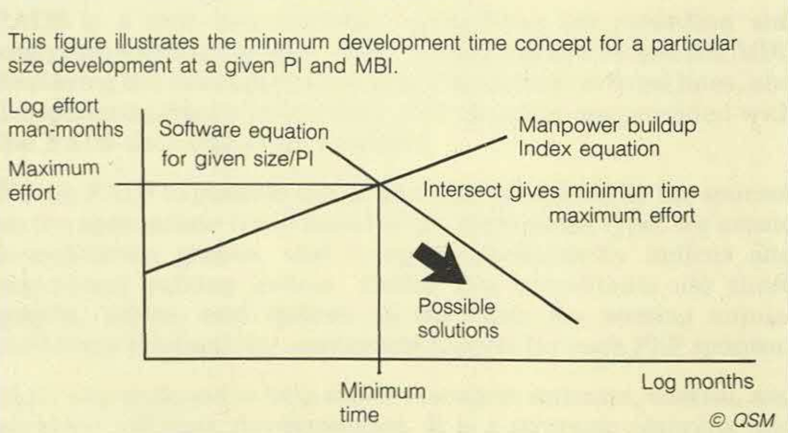
— The MBI, which reflects the typical rate of manpower buildup on previous projects, or that proposed for the new project.

— The size of the system to be developed in ELOC, including estimates of the upper and lower limits of the expected size.

Figure 5.1 shows how these parameters can be used to plot equations which relate effort to time both for the software equation and for the MBI. Where they intersect represents the *minimum* development time consistent with the given PI, MBI, and mean size. The project can be planned to take longer with consequent reduction in effort and cost.

Figure 5.1 SLIM minimum time calculation

This figure illustrates the minimum development time concept for a particular size development at a given PI and MBI.


Log effort man-months
Software equation for given size/PI
Manpower buildup Index equation
Maximum effort
Intersect gives minimum time maximum effort
Possible solutions
Minimum time
Log months
© QSM

Our software equation line is determined by the estimated mean size of the software. In addition, Monte Carlo simulation (random sampling based on different sizes within the uncertainty range) determines a series of time and effort results. These techniques enable the uncertainty in size to be reflected in uncertainties in estimating development time and effort. Development always involves uncertainty and risk. This approach quantifies risk throughout all development stages since the estimates of size can be updated as the project progresses.

'WHAT IF?' ANALYSIS

Once we have determined the minimum time for a given project we can examine a number of practical alternatives to the development plan and their effects on costs, quality, and timescale. For example:

- Setting new schedules beyond the minimum time to exploit the time/effort trade-off.
- Imposing management constraints on time, cost, and resources to determine if development is feasible within these constraints.
- Determining the size of system that can be developed in a shorter timeframe.
- Quantifying the uncertainty in the plans to ensure the software can be developed within a given timescale at a specific level of risk.
- Specifying reliability goals, since SLIM also models software errors.
- Evaluating the probability of the software development being completed within the planned budgeted time and price.

The comparison we make of the model estimates with the appropriate reference database trendlines (an integral part of the SLIM product) alerts the manager when plans are moving outside normal development limits.

The model does not find an optimum development solution. Rather it allows the user to explore practical alternatives rapidly and arrive at planning estimates that are consistent with the specific objectives and constraints faced by the project.

SET REALISTIC TARGET DATES

Our experience has shown that plans can be wildly unrealistic. Over-optimistic plans are the most common but we frequently find examples of over-conservative plans.

Using the software equation you can identify realistic target dates and effort for the main build phase that are consistent with achievable productivity, and project size. You can also take into account management constraints, reliability, and risk.

Use the software equation to set realistic target dates

SLIM outputs key milestone dates in the main build phase for:

- Reviewing all design elements, including detailed program logic.
- Completing the initial coding (when all code can be expected to be written but not yet unit tested, integrated, and system tested).

— Beginning the integration of the software units.
— Beginning the user-orientated system test.
— Installing the software on the operational hardware.
— Achieving full operational capability. (Based on empirical analysis, this is the point at which 95 per cent of software errors have been found and fixed.)
— The points at which 99 per cent and 99.9 of the total software errors have been found and fixed.

The estimates for the two earlier project phases, feasibility and specification/design, are formed based on the standard ratios expected for the development size, time, and effort.

Once the development is planned at the macro level, the project manager must break down the work and allocate this to individual team members. The milestones provide a framework for drawing up the detailed plan using such tools at the Project Manager Workbench (PMW) or ARTEMIS.

MONITOR AND CONTROL PROGRESS

You can use the plans to track progress and exercise high-level control. If you quantify the uncertainty in the plans you can track performance within these uncertainty bounds. Where the reported actuals exceed the uncertainty bounds then actions can be taken either to replan the development or get the project back on track.

Milestones are essential for project tracking. If you miss a milestone, it is often very difficult to catch up without reducing the functionality of the system. You can use the model to replan the project based upon the actual milestone achieved so that you can evaluate the consequences in terms of the impending time and cost overruns. You can then decide whether to reduce functionality.

Milestones slippage is often caused by a substantial growth in the requirements specification that has not been taken into account in estimating the size range.

The high-level plans enable you to track:
— Total staff and cost.
— Cumulative staff and cost.
— End-product code.
— Cumulative end-product code.
— Software errors.
— Cumulative software errors.
— Mean-time to defect.

The data you need to track these measures is usually readily available from the project managers on a monthly basis, without incurring any additional cost or effort. Provided the monthly and cumulative values are within the uncertainty bounds, and the milestones are met, the project can be expected to complete within the upper limits set on the schedule and budget.

The data needed to track progress is readily available, without additional cost

Chapter 6

How to manage a productivity improvement programme

If you want to manage productivity improvement, the first step is to use an objective method to measure your current productivity. You can then compare your own performance against external measures to determine how great an improvement is likely to be achievable. Armed with this quantified information on your current productivity, you can begin to find out which factors significantly affect productivity in your own development environment.

Use productivity measures to decide:
— where you are
— what is achievable
— what are the important factors affecting productivity

As we discussed in Chapter 2, there are many environmental factors that influence productivity. However, QSM has found that there are two factors that seem to have the most significant effect in practice:

— The management of user involvement in the development process.

— Policy and commitment to use formal methods.

In evaluating development groups, QSM have found only a few cases where purely technical factors have been the key to productivity improvement. It also seems that the experience and skills of the development team are not usually major factors. QSM's research indicates that extremely good productivity results can be achieved by relatively inexperienced people provided investment is made in good methods underpinned by a sound and consistent policy regarding their use, and in adequate staff training. Conversely, QSM have seen results that are well below average from organisations that have very experienced technical staff but no formal methods. These findings are encouraging because they show that in many cases there are clear *management* actions you can take to improve productivity.

Productivity is not a technical issue

It is seldom necessary to introduce *new* methods. In any development organisation of a reasonable size, we normally find evidence of individual projects which have been successful in using good, practical methods. The key is to build on your successes and adopt these proven methods throughout. If you are too willing to try all possible methods without ever stabilising your development environment, productivity will suffer.

Once you have identified the factors that influence your own productivity, you can select and implement specific improvements. You can then measure the effect of these improvements to see whether productivity has improved, and by how much. Moreover, you are able to set realistic targets, to track improvements, and to demonstrate the improvement.

SETTING REALISTIC TARGETS

Our experience shows that if you take informed steps to improve, the Productivity Index can be expected to increase by approximately one point every one and a half years. However, we only find this rate of improvement occurring in organisations where a measurement programme has been in place for several years: that is, in organisations that have already taken a strong initiative to manage productivity.

In our analysis of projects we also find characteristic values for the Manpower Buildup Index, MBI, within an organisation. When projects are developed with low MBI values this is often because there are specific constraints on staffing. Occasionally, we find development groups that have a uniform style of time compression measured with high values of the MBI. In these cases we can illustrate the cost of this management style in terms of increased effort and increased errors. Organisations with this time-compression style usually have no clear understanding of the negative impact on cost and reliability.

The main costs are in education and training

The main costs of making significant improvements in productivity and quality are predominantly incurred in education and training. These costs are not simply of a financial nature, but include the effort of changing management policy and attitudes when it comes to dealing with the development processes including relationship with users. The mutual understanding of the processes and relationships can be put on an objective footing if the users and developers can both appreciate how their contributions impact productivity and quality.

As an incentive to bring about change in your development organisation, you can use the current average Productivity Index to determine the minimum time and cost for an average-sized development, consistent with your typical MBI. PEP uses SLIM to calculate the benefits you would achieve in this typical project by increasing the Productivity Index by one. We normally expect savings from this increase to be at least $90,000 per project on a typical PEP sponsor project costing $485,000. It is a straightforward calculation to determine the total potential benefits to be gained by considering all the projects which you develop annually as we show in Chapter 7.

These large financial incentives can motivate everyone involved in the development process, users and system development staff alike, to work together in order to bring about major savings. By continuing to use the measures the savings can be demonstrated. This is important since users naturally like to be shown that improvement initiatives really do produce tangible benefits.

It should also be recognised that some initiatives to improve productivity may involve little or no investment. For example one large development group introduced a policy of not exceeding more than 15 people on any software development team. Over the last two years this policy has been shown to be effective by the measures made using our techniques.

If we can identify the most significant negative factors in the development environment and cost out the implications of changing them, we can then build a business case that considers the costs of the improvements offset against the benefits. In the next chapter we show how the return on investment can be calculated and used to justify the initiatives you would like to make to bring about improved productivity and quality.

TRACKING PRODUCTIVITY IMPROVEMENT

The initial measures for each PEP sponsor provide a set of baselines against which changes in project development productivity can be tracked. As each new project is planned the Productivity Index assumed in the plan may be verified against the reference database.

Completion of each project gives the essential input and output numbers. You can find out the final development time, total man-months of effort, and the size of the software from the project records which PEP uses to compute your Productivity Index and Manpower Buildup Index. The project post-implementation review can identify which factors were being addressed during the development to improve productivity, and use the measures to judge the effectiveness of these initiatives.

Each completed project can be entered into the database. As new projects accumulate, they can be analysed to find out the overall performance improvement within the organisation. For example, you can select projects by their date of completion to demonstrate changes in productivity and quality over time.

With the tracking process it is practical to tell whether productivity initiatives and investments are indeed producing benefits. We believe that if you are able to measure and demonstrate the outcome of your productivity investments, management will become more supportive and will help you to achieve further improvements.

DEMONSTRATING THE IMPROVEMENT

Using the PEP measures, sponsors are able to demonstrate improvement gains against:

- The QSM industry baselines.
- The PEP sponsors baselines.
- Previous years' measures for their own projects.

PEP is now building a very substantial data processing reference database for projects. This provides baselines for independent comparison of productivity measures.

Communicate with senior management using business terms

These reference measures, in particular with your own position, provide clear evidence of the gains you are making. As we show in Chapter 7 the benefits from these improvements can be quantified in business terms. In our experience senior business managers, with responsibility for investments in the systems development area, readily respond to arguments based upon an informed business case. While they may have no background or understanding of systems development it is not difficult to interest them in the management information that PEP provides. It is worthwhile getting these measures understood and accepted by senior management, as a means of monitoring and controlling the company's investment in software.

Chapter 7

Calculating and presenting the benefits

If you want to sell the idea of software investment management to general business management, you must speak in terms they recognise and understand. In our experience, terms like function points, lines of code, and similar IT-related jargon cause most business managers to lose interest.

The primary aim is to demonstrate that money spent on improving the development department has produced benefits by increasing productivity. A secondary aim is to demonstrate any benefits achieved through management actions that have not involved investment such as taking action to reduce high MBIs.

Cash savings and return on investment are the keys to communicating

We suggest that you use two simple methods of presenting the benefits: *cash savings* and *return on investment.*

In this chapter we work through an example calculation based on project averages from data submitted by PEP sponsors.

CASH SAVINGS

In the PEP assessment report we supply two tables that you can use to present cash savings. They relate *your* average productivity index and the manpower buildup index alternatives to the cost of developing average size projects. The tables are based on the data that your organisation submitted and reflect your unique development environment. These tables can be used to show:

— How well the current systems are being developed. (Most PEP sponsors have a PI which is above the average in the QSM database.)

— What payback could be achieved from a change in policy to reduce the time pressure (lower the MBI). This is particularly relevant in cases where high MBIs have become the 'style' of development without business justifications.

— What payback would be achieved by investment to improve the productivity index.

Figures 7.1 and 7.2 illustrate these tables for an 'average' PEP sponsor. We have used the average project size, PI and MBI calculated from the PEP database to give the baseline costs against which the savings are calculated.

From Figure 7.1 we can see that the typical PEP sponsor could save an average of $140,000 per system, a reduction of 29 per cent, by taking action to reduce the MBI from 3 to 2. Since this action relates only to the staffing profile, it can take effect immediately on all subsequent systems you develop.

Assuming this initiative is taken, then from Figure 7.2 we see that the typical PEP sponsor could save a further $90,000, a reduction of a further 19 per cent, by taking actions to increase the productivity index from 16 to 17.

Figure 7.1 The impact of changing the MBI at a PI of 16

This figure shows the result of changing the MBI for a system size of 70487 ELOC and PI of 16. Figures are based upon the averages in 193 business systems in the PEP database with a cost per man-month of $5,000.

MBI	Time (months)	Effort (man-months)	Cost per system ($)
6	8	333	1,655,000
5	9.3	192	960,000
4	10	146	730,000
3	11	97	485,000
2	12	69	345,000
1	13.3	46	230,000

Figure 7.2 The impact of changing the PI at an MBI of two

This figure shows the result of changing the PI for a system size of 70487 ELOC. The data is based on the average of 193 business systems in the PEP database. The MBI is one below the PEP average and cost per man-month is $5,000.

PI	Time (months)	Effort (man-months)	Cost per system ($)
18	10	37	185,000
17	11	52	260,000
16	12	70	350,000
15	13.5	95	475,000
14	15	120	600,000

RETURN ON INVESTMENT

Just as business managers expect to know how their systems development department compares with other, similar departments, they also want to know how the returns on investment in different business units compare. To illustrate this we can calculate the *return on investment* (ROI).

ROI is a yardstick applied in financial management. The ROI is used in conjunction with other business factors to help managers make sound judgements on investing in improvements to the business.

You can use the ROI in two ways:

— First, you can demonstrate the potential ROI of any expenditure you decide to make in the systems development environment.

— Second, using the results of the second and subsequent years, you can demonstrate the actual ROI you achieved from this expenditure.

This use of the ROI is intended to demonstrate the benefits of *software investment management*, that is, the return on investment in the software development environment. It is not intended to show the benefit to the company of the systems being delivered, or the related benefit of how much 'business function' is being delivered.

CALCULATING THE ROI

ROI is calculated by taking the investment in the systems development environment and dividing it into the benefits achieved (or the expected benefits). In its simplest form it is the annual net savings expressed as a percentage of the investment made.

Using the PEP tables the benefits can be calculated from the reduction in systems development costs that can be achieved for an average-sized system when you increase your typical PI by one. This reduction would then be multiplied by the average number of systems that have been (or are expected to be) completed in the year to give the total benefits. Greater precision is possible by making these calculations on a project-by-project basis.

In Figure 7.3 we show an example of computing the ROI, using the data in Figures 7.1 and 7.2, and making assumptions about cost of development per man-month.

Figure 7.3 Computing the ROI

This is a simple illustration of how to calculate the ROI on new investment in the systems development department. Sponsors should note that the precise way in which such calculations are made, and their results presented, vary with the accounting conventions of each organisation.

In the illustration, we assume that the PEP sponsor has 100 staff in the systems development department, and the fully loaded cost of each is $5,000 per month. This means that the total fully loaded staff cost is about $6,000,000 per annum.

After reviewing the PEP assessment results and identifying the factors most likely to result in productivity improvements, systems development management decides to improve the development environment. The once-off cost of the improvements is $600,000, an amount equal to 10 per cent of the staff budget.

The department produces, on average, five systems per year, and being the absolutely typical PEP sponsor, has an average system size of about 70,000 lines of code, an average PI of 16, and has already reduced its MBI to 2 from the average of 3.

Based on current QSM experience, an effectively managed productivity improvement programme can improve the PI by 1 in about one and a half years.

The PEP sponsor can calculate the expected ROI as follows:

The average cost of developing a typical sized system is found in the table in Figure 7.2. We are currently at a PI of 16 and have 17 as our goal.

PI	Cost per system ($)	Benefit per system from an increase in PI ($)
16	350,000	
17	260,000	90,000

Since it will take a year and a half to achieve the full PI point improvement we shall achieve only about half the benefit in the first year. We shall also assume a 'write off' period of four years for the investment, so we shall only look at the benefits in the next four years. Thus the benefits over the current costs for the five systems each year for four years are:

Year	Total benefit ($)
1	225,000
2	450,000
3	450,000
4	450,000
Total cost saving	$1,575,000

The cost of making this improvement was $600,000. Hence, the average annual return on that investment is:

$$\frac{1{,}575{,}000 - 600{,}000}{600{,}000} \times \frac{100}{4} \text{ \% per annum}$$

$$= 40 \text{ per cent per annum}$$

The benefit will be manifest in faster development of the systems with fewer staff needed. This means that capacity has been made available for other uses.

PRESENTING THE BENEFITS

We suggest you present the benefits in the following way:

- First present the PEP tables in the assessment report, reflecting your present productivity position. Present the PI as the capital investment measure, and the MBI as a measure of staffing buildup (or time pressure).
- Explain that the PI increases in response to investment in the departmental environment, and measures the overall productivity of the development team.
- Present the benefits as actual (or expected) cost reductions.
- Present the achieved (or potential) increase in throughput due to new investment and other changes you have made, explaining that this means with improved productivity you can deliver more system functionality in less time.
- Finally show the ROI you have computed. Restate that this has been calculated based on empirical analysis of your data.

We find that this form of presentation is remarkably effective in impressing general management. You must, of course, be prepared to justify the linkages between the investments and the results achieved.

By using quantitative measures of development productivity in this way, and for planning and controlling both individual projects and the system development process as a whole, you should be able not only to achieve substantial business benefits but also to demonstrate their impact to general managers.

References

Albrecht, Allan and Gaffney, John. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. Transactions of Software Engineering. IEEE Vol SE-9, No. 6, November 1983.

Boehm, Barry. Improving Software Productivity. IEEE. 1981.

Boehm, Barry. Software Engineering Economics. Prentice Hall Inc, Englewood Cliffs, New Jersey. 1981.

EIU Informatics report, quoted in the Financial Times. 4 October 1985.

Emrick, Ronald. Presentation by Ronald Emrick at the August 1987 International Function Point Users' Group, Atlanta. 1987. (Final version of the paper to be presented and published at the May 1988 International Function Point Users' Group Meeting at Dallas.)

Putnam, Lawrence H. Progress in Modelling the Software Life Cycle in a Phenomenological Way to Obtain Engineering Quality Estimates and Dynamic Control of the Process: Tutorial; IEEE Computer Society Publication EHO 165-1, pp 183-206, Computer Society Press. October 1980.

Putnam, Lawrence H. Key Issues in Managing Software Cost and Quality. QSM, McLean Virginia. 1987.

BUTLER COX
P.E.P

Butler Cox

Butler Cox is an independent international consulting group specialising in the application of information technology within commerce, industry and government.

The company offers a unique blend of high-level commercial perspective and in-depth technical expertise: a capability which in recent years has been put to the service of many of the world's largest and most successful organisations.

The services provided include:

Consulting for Users
Guiding and giving practical support to organisations trying to exploit technology effectively and sensibly.

Consulting for Suppliers
Guiding suppliers towards market opportunities and their exploitation.

The Butler Cox Foundation
Keeping major organisations abreast of developments and their implications.

Multiclient Studies
Surveying markets, their driving forces and potential future.

Public Reports
Analysing trends and experience in specific areas of widespread concern.

PEP

The Butler Cox Productivity Enhancement Programme (PEP) is a participative service whose goal is to improve productivity in application system development.

It provides practical help to system development managers and identifies the specific problems that prevent them from using their development resources effectively. At the same time, the programme keeps these managers abreast of the latest thinking and experience of experts and practitioners in the field.

The programme consists of individual guidance for each subscriber in the form of a productivity assessment, and also publications and forum meetings common to all subscribers.

Productivity Assessment
Each subscribing organisation receives a confidential management assessment of its system development productivity. The assessment is based on a comparison of key development data from selected subscriber projects against a large comprehensive database. It is presented in a detailed report and subscribers are briefed at a meeting with Butler Cox specialists.

PEP Papers
Four PEP papers are produced each year. They focus on specific aspects of system development productivity and offer practical advice based on recent research and experience.

Meetings
Each quarterly PEP forum meeting and annual symposium focuses on the issues highlighted in the PEP papers, and permits deep consideration of the topics. They enable participants to exchange experience and views with managers from other subscriber organisations.

Topics in 1988

Each year PEP will focus on four topics directly relating to improving systems development and productivity. The topics will be selected to reflect the concerns of the subscribers while maintaining a balance between management and technical issues.

The topics to be covered in 1988 are:

— Managing productivity in systems development.

— Tools for planning and managing systems development.

— Staffing issues in systems development.

— Managing the maintenance mountain.

BUTLER COX

Butler Cox & Partners Limited
Butler Cox House, 12 Bloomsbury Square,
London WC1A 2LL, England
☎ (01) 831 0101, Telex 8813717 BUTCOX G
Fax (01) 831 6250

Belgium and the Netherlands
Butler Cox BV
Burg Hogguerstraat 791
1064 EB Amsterdam
☎ (020) 139955, Fax (020) 131157

France
Butler Cox SARL
Tour Akzo, 164 Rue Ambroise Croizat,
93204 St Denis-Cedex 1, France
☎ (1) 48.20.61.64, Télécopieur (1) 48.20.72.58

Germany (FR)
Butler Cox GmbH
Richard-Wagner-Str. 13
8000 München 2
☎ (089) 5 23 40 01, Fax (089) 5 23 35 15

United States of America
Butler Cox Inc.
150 East 58th Street, New York, NY 10155, USA
☎ (212) 891 8188

Australia/New Zealand
Mr J Cooper
Butler Cox Foundation
3rd Floor, 275 George Street, Sydney 2000, Australia
☎ (02) 236 6161, Fax (02) 236 6199

Ireland
SD Consulting
72 Merrion Square, Dublin 2, Ireland
☎ (01) 766088/762501, Telex 31077 EI,
Fax (01) 767945

Italy
SISDO
20123 Milano, Via Caradosso 7, Italy
☎ (02) 498 4651, Telex 350309, Fax (02) 481 8842

The Nordic Region
Statskonsult AB
Stora Varvsgatan 1, 21120 Malmo, Sweden
☎ (040) 1030 40, Telex 12754 SINTABS

Spain
Associated Management Consultants Spain SA
Rosalía de Castro, 84-2°D, 28035 Madrid, Spain
☎ (91) 723 0995