Managing Software Maintenance

BUTLER COX
P,E,P

PEP Paper 8, November 1988

BUTLER COX

P,E,P

Managing Software Maintenance

PEP Paper 8, November 1988
by Rob Moreton

Rob Moreton

Rob Moreton is an associate consultant with Butler Cox, where he specialises in systems development methods and project management. In the past eight years, he has worked on both consultancy and research projects for the company. He has contributed to Butler Cox Foundation Reports on cost-effective systems development and maintenance, expert systems, and trends in information technology. He was also responsible for a study of the market for system-building tools in Europe. A common theme of his work has been tracking and evaluating developments in information technology and forecasting the potential impact of these developments.

In addition to his work for Butler Cox, Rob Moreton is a principal lecturer at the City of Birmingham Polytechnic. As Director of Studies within the Department of Computing, he is responsible for the academic quality of a range of professional and degree courses in computing and information technology.

He has a masters degree in computer science from Brunel University and is a member of the British Computer Society.

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 Software Maintenance

PEP Paper 8, November 1988
by Rob Moreton

Contents

Chapter 1

The problem of software maintenance

Software maintenance accounts for a significant proportion of most companies' systems development efforts. The more effort that goes on maintenance, the less is available for developing new systems. Yet, despite the obvious importance of maintenance, both in its own right and in the context of productivity enhancement, the attitude of many systems managers to the subject is strangely ambivalent.

SOFTWARE MAINTENANCE IN PERSPECTIVE

Software maintenance is a huge consumer of resources

As any manager in a systems development department knows, software maintenance is a huge consumer of resources, accounting for about the same amount of time and effort as developing new systems in the first place. A major survey of software maintenance conducted amongst businesses in the United States in 1981 showed maintenance accounting for 50 per cent of all programming and analysis effort. A survey in the United Kingdom, conducted in 1987 by K3 Software Services, put the maintenance proportion as high as 65 per cent. The growing investment by businesses in computer systems and the labour-intensive nature of maintenance work mean that maintenance is becoming increasingly expensive.

Software maintenance is much more than merely correcting errors in coding. It embraces all of the programming and analysis activities required to keep a system operational and effective after it has been accepted and placed in production. The purpose of maintenance is to protect a company's investment in systems by prolonging their useful life and improving the contribution that they make.

Figure 1.1 Three broad categories of maintenance

Category	Concern
Corrective maintenance	Resolving errors
Adaptive maintenance	Enhancing and extending systems
Perfective maintenance	Improving performance

There are, in fact, three broad categories of maintenance, which are summarised in Figure 1.1. The first is *corrective maintenance*, which is concerned with resolving errors. Corrective maintenance is a reactive process, usually requiring rapid action. The second is *adaptive maintenance*, which is about enhancing and extending systems to incorporate the evolving needs of users. The third is *perfective maintenance* (sometimes called preventive maintenance), which consists of changes to the structure of software to improve its performance and maintainability.

There is widespread disagreement over whether adaptive maintenance should be considered as part of software maintenance, or as part of new systems development. This is important, because adaptive maintenance is by far the largest maintenance activity. Some companies adopt a clear definition, one way or the other. For others, it depends on scale — if the effort exceeds six man-months, for instance, the work is considered to be new systems development. It is for this reason that reports of maintenance as a proportion of overall systems development work vary widely.

In the survey that we undertook for this paper (described on page 3), the proportion ranged from as low as 5 per cent to as high as 90 per cent, with an average of about 40 per cent.

Maintenance is often, by nature, more difficult and demanding than new systems development. Maintenance staff do not start with a clean sheet of paper. Often, they have to work to short timescales, particularly for corrective maintenance. Testing can be more demanding when the system being maintained has to fit, as is often the case, into the tight constraints of surrounding hardware and software, and when the methods and tools available to help with maintenance are not as well developed.

Maintenance is often more demanding than new systems development

The ambivalence of many managers towards maintenance is a further complication. Management attitudes to maintenance are not determined by maintenance levels. Five times as many managers in our survey reported that new systems development is more demanding of their time than maintenance, than vice versa. This is out of proportion with the division of effort, and points to a need for managers to pay greater attention to the maintenance function.

PURPOSE AND STRUCTURE OF THE PAPER

Software maintenance is an expensive and complicated business that every PEP sponsor is having to deal with. This paper is intended particularly for systems managers who have a responsibility for operational systems, and for their maintenance and enhancement. It will also be of interest to managers responsible for allocating systems development resources and budgets.

It is clear from our research that software maintenance is generally undermanaged. Because it is undermanaged, staff are not usually allocated in an optimum way, staff motivation is often a problem, system changes are not always properly controlled, and there is insufficient recognition of the benefits to be derived from the newer methods and tools. The purpose of this paper is to describe these deficiencies, and to suggest how they might be rectified.

Software maintenance is generally undermanaged

Management has traditionally given little thought to the allocation of appropriate resources to the maintenance function. Software-maintenance requirements are often omitted from systems design criteria, and maintenance is perceived as requiring little skill and enjoying little prestige. As software costs increase, this situation is set to change, and, in Chapter 2, we discuss the growing need for managers to turn their attention to the resource-allocation question. As systems get older, they become harder to maintain (and to operate) and their replacement becomes more pressing. Whether to continue maintaining a system or to replace it is a question that should be kept under continual review. The procedure for making the decision should be the same as that for evaluating new applications — that is, existing systems should be evaluated as part of an organisation's applications portfolio and assessed in terms of their contribution to the business and their level of maintainability. In this way, maintenance will be guaranteed its rightful share of the budget. A maintenance-rating process can be a great help in establishing costs and setting priorities.

Our survey indicated that the nature and efficiency of maintenance work is independent of the way maintenance is organised. There was, however, some evidence that staff morale improves when maintenance is organised as a separate function, rather than undertaken within project teams. Despite the prevailing view to the contrary, we believe that maintenance work is intrinsically highly motivating. To reap the potential, though, staff need to be carefully selected and trained, and management commitment is essential. The broad question of organisation and staffing of the maintenance function is discussed in Chapter 3.

The maintenance process consists of a series of steps, the first of which, change management, formalises the relationship between maintenance staff and the user. Most PEP sponsors claim to have some form of change management in place but few have fully formalised or implemented the remaining steps. We believe this to be an area requiring attention and discuss the implications of it in Chapter 4.

Chapter 5 is about the new methods and tools that are aimed at the maintenance process and are now reaching the market. Enforcing standards and using modern development methods are both ways of improving the quality of new systems, so that maintenance becomes easier when the systems are operational. Other contemporary tools are designed to be directly beneficial to the maintenance process. Their growing availability is opening up possibilities for more users to maintain systems for themselves in future.

Our suggested improvements to the maintenance environment are summarised, in the form of a checklist, in Chapter 6. The procedures set out in that list are the foundation of a management approach that is applicable to a broad range of companies, irrespective of particular implementation issues.

RESEARCH SOURCES

We have drawn on the research material for PEP Paper 7, for which Butler Cox conducted a survey of more than 600 staff in seven PEP sponsoring organisations, during the spring of 1988. Specifically for *this* paper, we conducted a further questionnaire survey of PEP sponsors. Twenty-four organisations, forming a representative sample of PEP sponsors, responded, although not all respondents answered every question. The number of programmers and analysts in the systems development departments that we surveyed ranged between 20 and 300, 15 departments having fewer than 99 programmers and analysts, and the remaining nine departments having 100 or more. The total headcount in the departments ranged between 50 and 550, with 11 having 200 or more. Annual departmental budgets ranged up to $50 million, nine of them exceeding $10 million. In addition to the questionnaire, we reviewed published research material and interviewed selected specialists, both within and outside the PEP sponsorship.

Chapter 2

Allocating resources to the maintenance function

Because operational systems deteriorate with age, they need to be audited to determine whether and when to replace them. The auditing procedure should be formalised, and it should be aligned with the procedure for evaluating systems for new applications. The tasks of defining costs and setting priorities will be much easier if there is some routine maintenance-rating process established within the company so that systems development departments allocate a proportion of their capacity exclusively to maintenance.

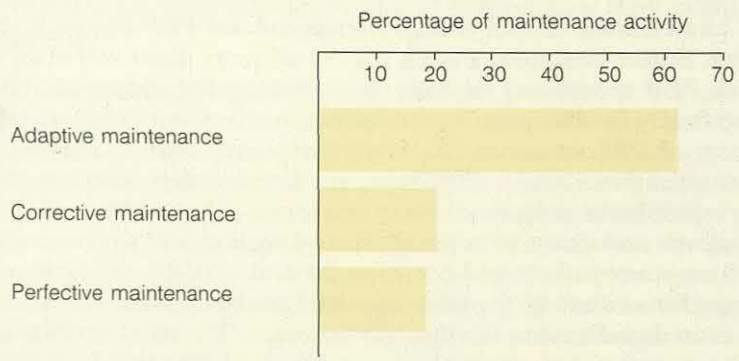
FORMALISE THE MAINTAIN-OR-REPLACE DECISION

Whether to continue maintaining a system or to replace it with a new one is a critical management decision. Choosing the moment to initiate replacement is best done by subjecting every operating system to a formal and regular review.

After a new system has become operational, there is a continuing need to maintain it. A system that takes perhaps a year to develop may have an operational life of five years or more, and more effort is likely to go into its maintenance than into its original development. Most maintenance effort is adaptive. In our survey, adaptive maintenance accounted for 62 per cent of maintenance effort compared with 20 per cent for corrective maintenance and 18 per cent for perfective maintenance (see Figure 2.1).

After a new system has become operational, there is a continuing need to maintain it

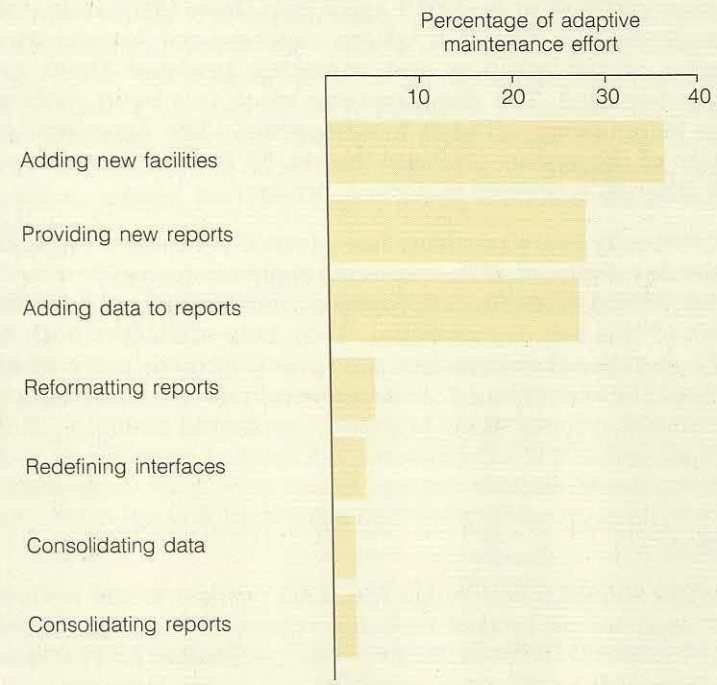
Figure 2.1 Adaptive maintenance is the main activity



(Source: Butler Cox survey of PEP sponsors)

Adaptive maintenance results in significant changes to a system in terms of both structure and coding. An indication of the nature and extent of these changes is given in Figure 2.2 which categorises the goals of our respondents in their adaptive maintenance efforts. Many of these changes are enhancements in the sense of adding new facilities, providing new

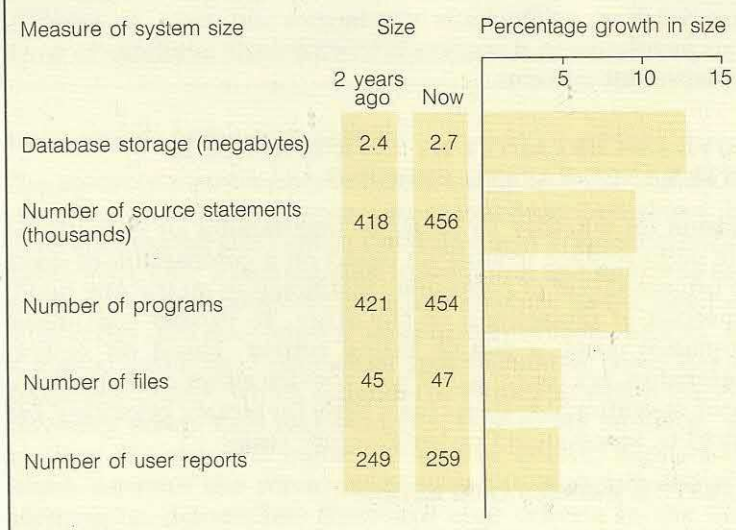
Figure 2.2 Most adaptive maintenance is to add new features



(Source: Butler Cox survey of PEP sponsors)

reports, and adding data to reports, and, as such, they extend what went before. It is therefore no surprise that systems get larger as a result of maintenance. This growth is illustrated in Figure 2.3. It compares five features of a system as they were at the time of our survey, and as they were two years earlier. All five features have grown in the period; the number of source statements has increased by 9 per cent, for instance, and the number of programs by 8 per cent.

Figure 2.3 Maintained systems grow in size over time



(Source: Butler Cox survey of PEP sponsors)

Continuous modification can leave a system in a less stable state than before. Each time the system is modified, it becomes potentially more difficult to modify it again next time. Ultimately, this process leads to a situation where maintenance becomes too expensive or too complex and operating response times are severely degraded. Too many systems reach this point without anyone being aware of what has happened. The deteriorating condition of the system can, and should, be monitored and controlled through a process of formal review.

Today, virtually every company has a formal procedure for justifying the development of new systems applications, yet few have a regular, formal procedure for auditing their operational systems. Reviews of this sort are essential. They help managers both to identify operational systems that are approaching the point when they should be redeveloped, and to re-evaluate the contribution of operational systems to the business. Conducted annually, they present an opportunity to re-assess the costs of a system as well as its benefits. While this concept is not new, it elevates maintenance to its appropriate place as a significant management consideration.

Few organisations have formal procedures for auditing their operational systems

The review should follow much the same process as the review for new applications. Indeed, we believe that cost/benefit analyses should be undertaken for existing operational systems and for new applications at the same time, using the same evaluation process. If the information required to justify (or rejustify) existing systems in this way is not readily available, it indicates a need for better control and monitoring.

Operational systems should be reviewed in the same way as new applications

A good illustration of this process in operation is provided by a manufacturing company whose systems development department has a development staff of about 60. The department has been through a two-year period of strategy formulation and review, while new systems development work has remained frozen. Now, the company is beginning to see the benefits of a change in direction. All user requests to the systems department that exceed one week of effort have first to be authorised by a steering committee. Requests for maintenance work and new applications are examined on exactly the same basis, and resources are allocated in the same way, on the basis of priorities and costs. As a result, systems development resources are being made available to work on replacement systems.

REMOVE UNCERTAINTY BY INTRODUCING A MAINTENANCE-RATING PROCESS

A decision on whether to continue maintaining an operational system or to replace it must be based on a comparison of costs — the projected cost of continuing maintenance on the one hand, and the cost of replacing it on the other. To predict continuing maintenance costs, a simple rating system, based on system characteristics, is a useful aid: it may provide either a comparative rating of operational systems (as a basis for setting priorities, for instance) or assessments on an absolute scale.

A rating system is a useful aid to predicting maintenance costs

COMPARATIVE MAINTENANCE RATING

A comparative rating might be produced on the basis of a 'maintenance profile' of the software, developed from a set of

criteria relating to such features as system age (maintenance gets harder as systems get older), system size (the larger the system, the more costly it is to maintain), and complexity. A fuller list of such features, and of the criteria relating to them, is shown in Figure 2.4. The maintenance rating of a system can be assessed by allocating, for each of these features, a score of, say, between one and four. Because the relative importance of each feature will vary depending on an organisation's circumstances, it makes sense to weight each one (again using scores of one to four, for instance).

Figure 2.4 Characteristics to consider when preparing a maintenance rating

Characteristic	Comments
System age	Maintenance usually gets harder as a system ages.
System size	Can be measured in effective lines of code (ELOC) or number of function points.
Complexity	Can be derived by dividing ELOC by the number of programs or, more accurately, by counting the number of function points or using a code analyser.
Development language	Maintenance is usually easier with higher-level languages.
Development methods and tools	Systems developed using structured methods are more stable, less error-prone, and easier to maintain.
System quality	Can be assessed in terms of the number of errors reported over time and the number of change requests submitted (although this measure can be misleading).
Type of change	Whether imposed from outside the business (such as a change in regulations) or from within the business.
Change controls	Systems lacking adequate controls are higher-risk, so harder to maintain.
Operational environment	Operational demands exert time pressures that make maintenance harder (and increase the likelihood of control procedures being circumvented).
Staffing	Includes technical expertise and specific knowledge of the system.

ABSOLUTE MAINTENANCE RATING

The absolute maintenance rating is a slightly more sophisticated version of the simple comparative rating described above. In the United Kingdom, the Central Computer and Telecommunications Agency (CCTA), which supplies information and advice to central government departments on the planning and use of information technology, has developed a 'system maintenance profile' which is a good example. Criteria are grouped under three headings: *adequacy to user*, which assesses the extent to which the system currently meets user requirements; *risk to the business*, which assesses the risk and impact of system failure; *support effort*, which assesses the resources required to maintain the system adequately. Altogether, there are nine criteria in the CCTA's system maintenance profile, and a total of 16 measures (between one and three measures for each criterion), as shown in

Figure 2.5. Each measure delivers a score. The scores are totalled. Systems scoring 100 or more are candidates for renewal.

Figure 2.5 The CCTA's 'system maintenance profile'

Category	Criteria	Measures
Adequacy to user	Desirable changes	— (Man-days per annum on desirable changes/thousand lines of code) + 1. — Degree to which desirable changes are being blocked (1 = not at all, 5 = completely).
	Changes backlog	— (Estimated man-days to clear backlog of changes/ thousand lines of code) + 1. — Degree to which system is failing to meet requirements (1 = fully, 5 = marginally).
Risk to business	Staffing	— Degree to which staffing is a problem (1 = none, 5 = major).
	Change control	— Quality of documentation (1 = excellent, 5 = non-existent). — Change control procedures (1 = good, 5 = non-existent).
	Errors	— Testing procedures (1 = good, 5 = non-existent). — (Errors per annum/thousand lines of code) + 1.
	Impact of errors	— Rating of effect of errors on the business (1 = nil, 5 = significant).
	State of code	— System age (1 = 1 to 7 years, 2 = 8 to 14 years, 3 = >14). — Structure (1 = good, 5 = bad). — Program size (thousand lines of code/number of programs),
Support effort	Staffing	— (Maintenance effort per annum/ thousand lines of code) + 1.
	Mandatory changes	— (Annual effort on mandatory changes/thousand lines of code) + 1. — Reduction in mandatory changes if system redesigned (1 = nil, 5 = substantial).

(Source: Managing Software Maintenance, CCTA, October 1987)

DEFINE THE SHARE OF RESOURCES TO BE ALLOCATED TO MAINTENANCE

Maintenance-rating procedures of the kind described above help to establish the costs of and priorities for redeveloping existing operational systems. Prolonging a system's life means bearing heavier maintenance costs but, at the same time, reducing the workload of the systems development function, thereby freeing more capacity for developing new applications.

This raises the question of whether the systems department should allocate a fixed proportion of its development resources to maintenance and, if so, how much. We believe that allocating a fixed share of capacity to maintenance is a sensible approach. The

proportion should be kept under review, however, and it will need to be changed from time to time.

Limiting maintenance capacity as a matter of policy is, in fact, commonplace among PEP sponsors. The purpose is usually to avoid maintenance work continually displacing new development work. This limit is sometimes expressed as a proportion of the budget, and sometimes in terms of the type of maintenance work that is accepted. The former usually works better, particularly when the procedure for assessing maintenance is built into that for assessing new applications, along the lines discussed above.

Maintenance work should not be allowed to displace new development work

An example of how this policy can work in practice is the experience of one of the United Kingdom's public utilities. Two years ago, it limited the proportion of the systems department's budget to be devoted to maintenance work to 40 per cent. This limit was introduced to help overcome conflicting demands for new applications. The policy worked well, but the limit has recently had to be increased to 50 per cent, and resolving conflicting demands for maintenance is now a more serious problem than it is for new applications.

This example confirms that formal monitoring of the maintenance environment is required to implement such a policy successfully, because the pressure of competing demands for the limited resources will increase. Restrictions on maintenance, however rational, will often be seen by users as leading to the provision of an inadequate service. However, if the limit is imposed as part of an overall strategy to manage the applications portfolio, a proper justification can be made in terms of contribution to the business.

Chapter 3

Organising and staffing maintenance work

The organisation of maintenance work in project teams or in a separate function appears to have little bearing on either the demand for, or the performance of, that work. Morale, however, does seem to be better amongst staff who work in a separate maintenance function.

Improving staff motivation is a critical issue. Contrary to conventional wisdom, maintenance *is* intrinsically motivating (as we reported in PEP Paper 7), regardless of how it is organised. Effective maintenance does, however, require careful staff selection and training and, what is most important, management recognition. The alternative of arranging for some or all of the maintenance workload to be undertaken outside the systems development department also has some merit and is an increasingly feasible option.

Effective maintenance requires careful staff selection and training, and management commitment

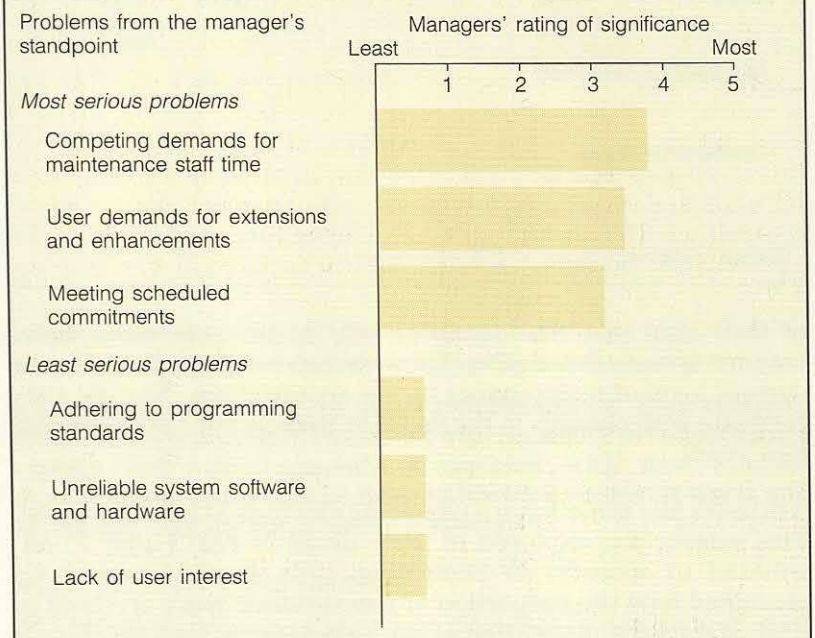
CONSIDER ESTABLISHING A SEPARATE MAINTENANCE ORGANISATION

The relative merits of different ways of organising maintenance within a systems department have been debated for years, but a survey of maintenance organisation in 130 businesses in the United States, undertaken in 1987, identified some common characteristics. The businesses in which maintenance was organised in project teams were smaller than the sample average. In these businesses, although the maintenance backlog was shorter than average, the software was more difficult to maintain, the problem of managing maintenance seemed more severe, and the maintenance staff were less positive than average about their work. In contrast, where maintenance was undertaken as a separate activity, the businesses were larger than the sample average, the maintenance backlog was longer than average, and the software under maintenance was older, but management and staff problems seemed less severe than average.

No similar characteristics were evident in our own, much smaller survey. Of the organisations we surveyed, maintenance was undertaken by project-team staff in 15, and by a separate maintenance function in eight. We detected no significant differences between the two forms of organisation in terms of staff experience, the pressure of conflicting demands for staff time, staff turnover, communications with users, or documentation problems. The amount of corrective maintenance as a proportion of the whole was about the same in both forms of organisation. Size was not a factor as it was in the US survey. We found no evidence to support the view that separate maintenance functions are more likely to be the norm in larger businesses. In fact, our evidence suggested that higher levels of maintenance (above 40 per cent of the total development effort) are associated with project teams.

Our survey showed that, from the managers' standpoint, the most significant problem was competing demands for maintenance staff time, and the least significant was a lack of user interest (see Figure 3.1). There was no evidence to suggest that the way in which maintenance was organised made any difference to these perceptions. On the other hand, both staff morale and motivation were higher when maintenance was organised in a separate function rather than in project teams.

Figure 3.1 Competing demands for maintenance staff time is the most significant problem



(Source: Butler Cox survey of PEP sponsors)

MOTIVATE, TRAIN, AND MANAGE MAINTENANCE STAFF

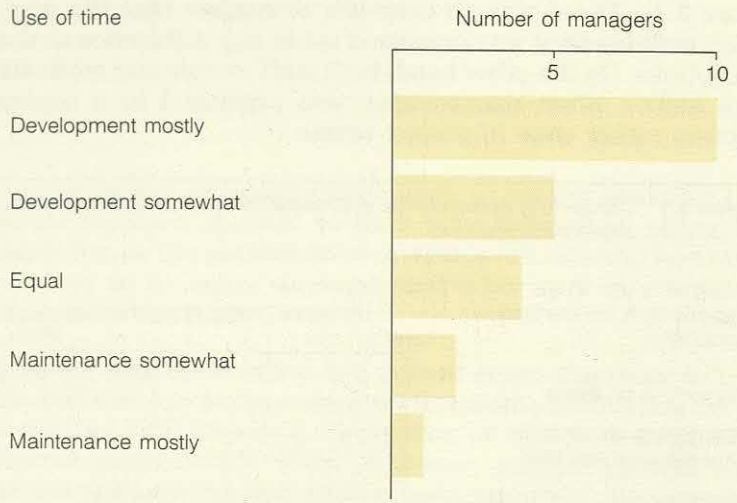
Maintenance can be more motivating than new systems development

The common perception is that maintenance work is less motivating than new systems development. We believe this to be a misconception. Maintenance can be intrinsically more, not less, motivating. To change perceptions demands careful staff selection and training — and, most important of all, a change in management attitudes.

THE QUESTION OF MOTIVATION

Software maintenance has long been considered less important than new systems development. It is often an afterthought in system design, and is perceived as demanding limited skill and enjoying little prestige and attention. As a result, programmers have tended to avoid maintenance work, preferring instead to work on new systems development assignments. This perception was confirmed by our own survey, which showed that managers attach more importance to systems development work than they do to maintenance (see Figure 3.2 overleaf). Seventy per cent of managers rated systems development as being more demanding

Figure 3.2 Seventy per cent of managers rated systems development as being more demanding of their time than maintenance



(Source: Butler Cox survey of PEP sponsors)

of their time than maintenance; only 14 per cent rated maintenance as more demanding. These ratings bore no relation to the current level of maintenance in the organisations. Nor did they correlate with changes in the levels of maintenance over the past two years.

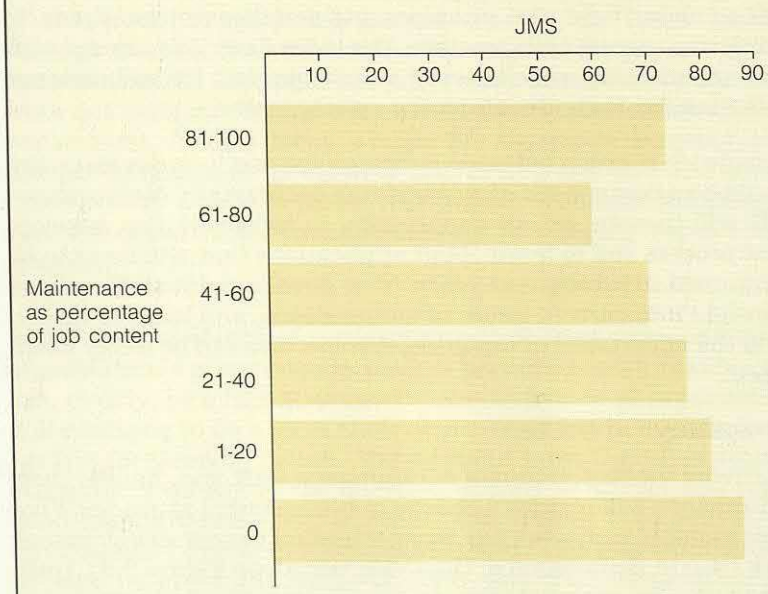
Attitudes like these have a damaging effect on staff motivation. This subject was explored in some detail in PEP Paper 7, distributed to sponsors in September 1988. In that paper, we examined how the proportion of maintenance work involved in a job affects the motivation of the person doing that job. This is illustrated in Figure 3.3, which compares jobs on a scale called the Job Motivating Score (JMS) scale. The pattern is one of falling job motivation as the proportion of maintenance work increases, except for those staff who are involved full-time, or almost full-time, in maintenance.

Software maintenance work can provide high levels of job satisfaction

High levels of job satisfaction can, however, be obtained from working in a maintenance environment. Software maintenance work is intrinsically highly motivating because it is challenging, it offers great variety, and the results are highly visible.

This applies to all three categories of maintenance described on page 1 — corrective, adaptive, and perfective. Corrective maintenance is often extremely complex and demanding, because of the absence of complete documentation and the difficulties of re-creating error conditions. Often, it has to be completed in a matter of hours. Adaptive maintenance is similar to new development in terms of its phases, but the emphasis is different. Analysis is the dominant phase in adaptive maintenance. The remaining phases of design, implementation, testing, and system release/integration are no less important, but they are proportionally smaller. Adaptive maintenance provides frequent opportunities for maintenance staff to communicate with users as the changes are implemented. Perfective maintenance combines some of the

Figure 3.3 Job Motivation Scores (JMSs) vary according to the amount of maintenance work performed



(Source: Butler Cox survey of PEP sponsors)

main characteristics of the other two categories. Each category demands technical skill, combined with an ability to communicate rapidly and unambiguously. Compared with new systems development, maintenance offers a broader variety of work, and is equally demanding in other respects.

STAFF SELECTION AND TRAINING

Staff who are best at maintenance are different from those who are best at developing new systems

The requirements of maintenance place heavy demands on staff selection and training. The main staff attributes in maintenance are sound technical ability, an understanding of past as well as present development practices, and an ability to communicate and to work under pressure. The shorter the timescales involved, the greater the need for good-quality maintenance staff. Compared with new systems development, maintenance work is probably less demanding in terms of conceptualising skills (imagination and creativity) but more demanding in terms of affiliation skills (patience, adaptability, and willingness to lend support). Maintenance staff should be selected with these characteristics in mind.

In practice, maintenance is often allocated to staff with less experience than average. There is no harm in this, as long as the staff meet the criteria outlined above and as long as timescales are not critical. It does provide an opportunity for less experienced staff to learn about the problems of application changes at first hand — experience that they can put to good use in development projects by encouraging designers to think about the implications for maintenance.

In contrast to conventional wisdom, maintenance demands more staff training than does new system development, particularly when the maintenance staff are relatively inexperienced. In terms of technical and problem-solving skills and training, there is little

difference between the requirements of maintenance and new development, but two further considerations point to a difference in training requirements. The first is the need for maintenance staff to understand past practices and development methods, in addition to current best practice. The second consideration is that of communication, which is even more important for maintenance staff than for their development counterparts.

Periodic job rotation between maintenance and new development should be a component of any training programme. Maintenance staff will thereby get an opportunity to influence the development process and to learn about applications that will need to be maintained in subsequent years. New development staff will get an insight into current issues of maintenance, and learn to understand the importance of designing systems that can be easily maintained.

MANAGEMENT ATTITUDES

Improving the motivation of maintenance staff and, hence, their performance will require a change in management attitudes. Five times as many managers pay more attention to new development work than to maintenance, than vice versa (see Figure 3.2). Until managers see maintenance as an important strategic issue, problems of low staff morale are certain to persist.

Managers must see maintenance as an important strategic issue

CONSIDER ARRANGING FOR OUTSIDE MAINTENANCE

An alternative to maintaining systems within the systems development department is to arrange for some or all of the work to be undertaken outside the department, either by the system users themselves, or by a third-party contractor.

MAINTENANCE BY USERS

Advances in fourth-generation languages are making user-maintenance an increasingly practical proposition. It is now commonplace for businesses to provide users with query languages through which they can derive data and generate their own reports. It is a small step beyond this to provide tools sufficiently powerful to enable users to add functionality to a system — in other words, to undertake their own adaptive maintenance. This is discussed further in Chapter 5.

Maintenance by users is an increasingly practical proposition

MAINTENANCE BY THIRD PARTIES

Contracting maintenance work to a third party offers three benefits: it releases systems development department resources for other work; it overcomes the 'technology gap' problem, when the system being maintained is based on technology that is no longer current; it introduces a formal contractual relationship between users and maintainers.

Third-party maintenance is a feasible option

The FI Group, a major systems and software house, is a good example of a contractor who undertakes third-party software maintenance work. One assignment involved a leading building society that was obliged to modify its mortgage-administration system and contracted the work out so that it could, itself, concentrate on new development work. In the four-year period to April 1988, the project team assigned to the work had made 600

separate changes. The team, which was drawn from a larger pool of staff, all of whom were familiar with this kind of work, varied in size between three and five according to the nature and priority of the work. Another assignment was for an Inner London borough council that contracted to maintain its payroll system because the IBM CICS and Assembler skills demanded by the work were not available within the council's own information systems department. A third client, a major life assurance company, contracted to maintain its existing unit-linked and non-unit-linked systems over a two- to three-year period, while the information systems department concentrates on developing replacement systems.

Management must provide an environment that supports maintenance staff

While the possibility of contracting out at least some part of an organisation's maintenance work is becoming more feasible and can, clearly, be a highly successful alternative, most organisations will continue to do a lot of their own maintenance work in-house for the foreseeable future. Management must therefore turn its attention seriously to the question of how to attract and retain good maintenance staff. In short, the answer is to provide an environment that actively supports them. This may be achieved, in part, by providing methods, tools, and training programmes, but changing the technology alone is not enough. An organisation must create an environment in which maintenance is perceived to be as important to the operation of the business as any other function.

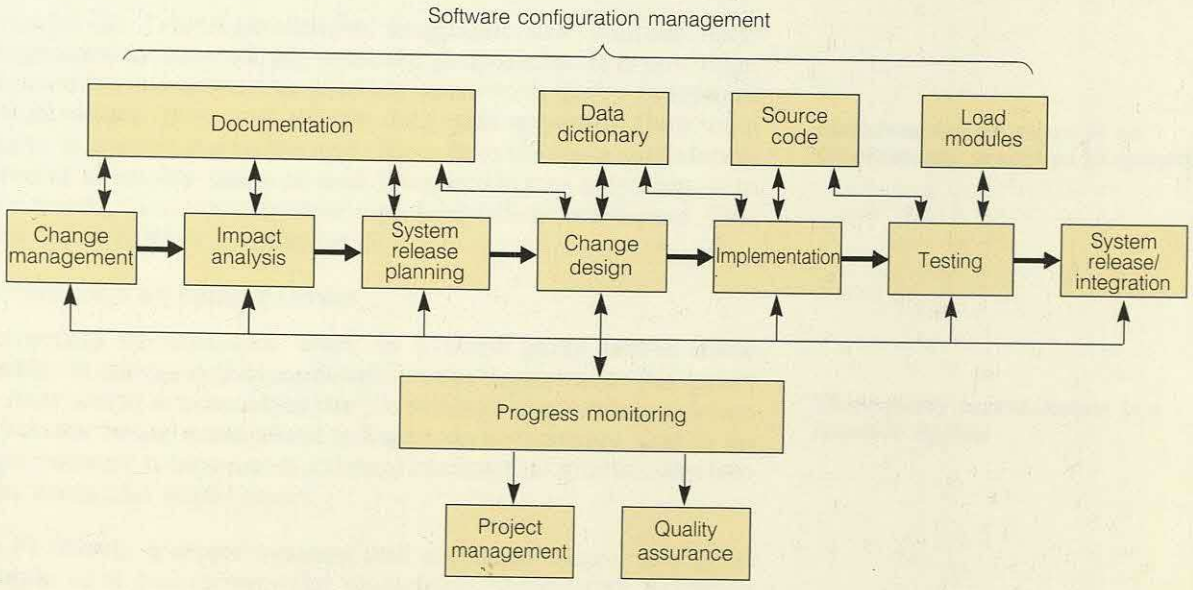
Chapter 4

Controlling the maintenance process

For maintenance work to be effective, it is vital to control the input to the process — the procedure by which change requests are notified and managed in the first place. This procedure of change management is the first of several steps in the maintenance process. Change management is followed by impact analysis, system release planning, change design, implementation, testing, and system release/integration. These steps, which occur sequentially, are supported by a further activity that continues concurrently — progress monitoring. The whole process is illustrated in Figure 4.1.

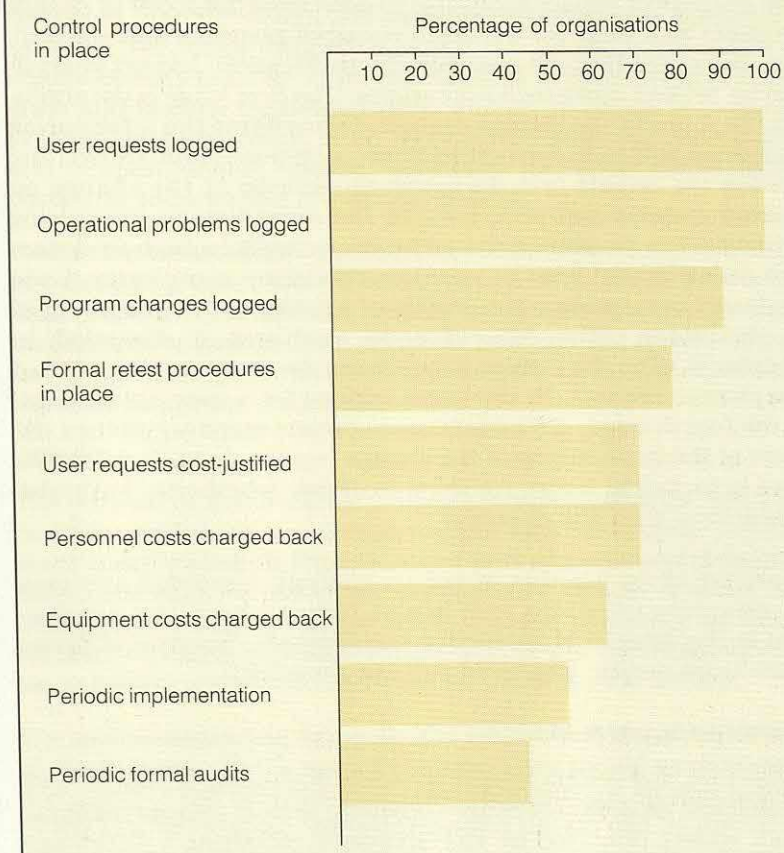
Most PEP sponsors claim to have a clearly defined procedure in place that corresponds to the first step, change management. Certainly, every respondent in our survey records all user requests and operational problems, but our respondents admitted to some failings as well. Periodic formal audits, for instance, are in place in fewer than half of our survey respondents' businesses (see Figure 4.2). In order to achieve improvements in the maintenance environment, the steps in the process need to be carefully co-ordinated, not simply monitored individually.

Figure 4.1 The formalised maintenance process



(Source: Butler Cox)

Figure 4.2 Most PEP sponsors have formal control procedures in place



(Source: Butler Cox survey of PEP sponsors)

FORMALISE THE MAINTENANCE PROCESS

To appreciate the importance of formalising the steps in the maintenance process, it helps to understand more precisely what they are.

CHANGE MANAGEMENT

A formal procedure for change management is essential

Change management is the critical first step in the maintenance process. A formal procedure for change management is essential for two reasons: it provides a common communication channel between maintenance staff, users, project managers, and operations staff, and it provides a directory of changes to the system, for status reporting, project management, auditing, and quality control. The basic tool of the change-management procedure is a formal change-request document that forms the basis of a contract between the user and the maintainer.

An important element of change management is version control (or software configuration control). It means tracking different versions of programs, releases of software, and generations of hardware, and it plays a major role in ensuring the quality of delivered systems. Version control also ensures that software is not degraded by uncontrolled or unapproved changes, and provides an essential audit facility.

IMPACT ANALYSIS

The purpose of impact analysis is to determine the scope of change requests as a basis for accurate resource planning and scheduling, and to confirm the cost/benefit justification. Impact analysis can be broken down into four stages. The first stage is determining the scope of the change request, by verifying the information contained within it, converting it into a systems requirement, and tracing the impact (via documented records) of the change on related systems and programs. In the second stage, resourcing estimates are developed, based on considerations such as system size (in estimated lines of code) and software complexity. Code analysers that measure the quality of existing code can be helpful at this stage. (The role of code analysers is discussed in Chapter 5.) The third stage is analysing the costs and benefits of the change request, in the same way as for a new application. In the fourth stage, the maintenance project manager advises the users of the implications of the change request, in business rather than in technical terms, for them to decide whether to authorise proceeding with the changes.

There are three benefits of impact analysis: improved accuracy of resourcing estimates and, hence, better scheduling; a reduction in the amount of corrective maintenance, because of fewer introduced errors; improved software quality.

SYSTEM RELEASE PLANNING

In this step, the system release schedule is planned. Although well established amongst software suppliers, system release planning is not widely practised by PEP sponsors, reflecting a difference in the extent to which formal maintenance contracting is established.

System release planning is not widely practised

A system release batches together a succession of change requests into a smaller number of discrete revisions. System releases can take place according to a timetable that is planned in advance. The timetable planning gives users the chance to set priorities for their change requests, and makes testing activities easier to schedule. The problem with system releases comes, of course, when corrective maintenance is required urgently.

Software is available to help monitor system releases. The software records the changes incorporated in, and the date of, each release, and provides information for project control, auditing, and management.

CHANGE DESIGN AND IMPLEMENTATION

The common thread in the work in these two steps is that they are undertaken to satisfy an often short-term user requirement.

Corrective maintenance, in particular, will be undertaken in a limited time and will be concerned primarily with fault repair (with little regard for careful design and integration of changes). Emergency repairs must subsequently be linked to the formal software-maintenance process and be treated as a new change request. This will ensure that the repairs are correctly implemented and that the design documentation is updated.

Adaptive maintenance will functionally enhance an existing system. The design and implementation process is similar but more restricted than the design and implementation of new application systems. The major difference is that the design implications of enhancements must be taken into account in the subsequent program and module implementation. Failure to design the change at each level can result in an increasingly complex, unreliable, and unmaintainable system. This leads to higher maintenance costs and reduces the life of the system.

Perfective maintenance is concerned with improving the quality of existing systems. The effort is applied to software that is the most expensive to operate and to maintain. The design tasks undertaken will range from complete redesign and rewrite to partial restructuring. The process combines the characteristics of the other two types of maintenance.

TESTING

The purpose of maintenance testing is to ensure that the software complies with both the change request and the original requirement specification. It forms a major part of a successful quality-assurance plan. In principle, maintenance testing is much like development testing except that, because the scope of maintenance testing is potentially narrower, fewer test cases have to be prepared, validated, and filed in the test-case library.

Maintenance test cases should be created as a result of the first stage in the impact analysis

The maintenance test cases should be created as a direct result of the first stage in the impact analysis. They should be sequenced according to the principle of incremental testing, so that defects in the change-request specification and design can be identified early on. Walk-throughs and inspections should be implemented routinely as a formal element in the process.

The test-case library itself builds up over time. At first, it contains only the test cases prepared for and validated during original development. It grows as test cases for successive maintenance tests are added to it. A file of this sort is called a regression testing file. A few tools are available from suppliers (such as IBM and Digital) to help with regression testing. Although limited in what they can do (in terms of features such as automatic revalidation, for instance), they are able to provide administrative support.

SYSTEM RELEASE/INTEGRATION

This step consists of releasing the revised programs into live operation. The implications for maintenance staff are significant because it is their responsibility to ensure that any revised versions are completely integrated with other parts of the system, which may never have been revised or which may have been revised at different times.

PROGRESS MONITORING

Progress monitoring takes place concurrently with the other seven steps in the maintenance process. The sort of data that should be collected during progress monitoring includes the time taken per step, the effort involved, and the scope of the change. Collecting and filing data of this sort so that performance can be monitored, both over time and between systems, is consistent with the general PEP philosophy. Improving software maintenance

productivity is difficult if there is no record of where problems and successes have occurred in the past.

COORDINATE THE STEPS IN THE MAINTENANCE PROCESS

It is important to consider how the steps in the process fit together

There is no panacea for solving the problems of maintenance. It is essential, however, to consider not only how each individual step in the process works, but also how the various steps fit together.

Peterborough Software (UK) Ltd, a software house based in the United Kingdom, provides an example of how companies can successfully coordinate the steps in the software maintenance process. The problems that it faces are unusually demanding. The company maintains a range of payroll software packages. The packages run on a variety of computers, under the control of different operating systems, both within the United Kingdom and overseas. Altogether, Peterborough Software has 250 customers. The software coding differs from country to country, to take account of local statutory regulations, such as taxation. Thus, several releases of the same package are current at a time, and all have to be supported in the field. The regulations change frequently and without much warning, and maintenance changes therefore have to be implemented swiftly and accurately. The difficulties faced by Peterborough Software are further compounded when customers create nonstandard versions of the software by failing to apply maintenance modifications that are issued to them, or applying them in the wrong sequence.

How does Peterborough Software arrange its maintenance procedures against this background of complexity? The answer lies in disciplined adherence to procedural steps similar to the ones we have described here, and in the use of a computer-based program monitoring system known as the Problem Monitoring System (PMS).

The maintenance procedure is carried out by two divisions within Peterborough Software. One is the Customer Support Division, which effectively looks after change management, impact analysis, and system release planning. The other is the Development Division, which is responsible for coding, testing, and quality assurance.

Change requests received by the Customer Support Division come from three sources. The first is customers, whose requests take the form of enhancements (called facility requests), queries, and error reports. The second source is impending legislative changes. The third is the market. To survive, Peterborough Software has to compete by offering products that are constantly being improved. Maintenance arising from customers is both adaptive and corrective in nature; from the other two sources, it is mostly adaptive and perfective.

Customers are the most important source of change requests — the Customer Support Division receives up to 400 telephone enquiries a day, for instance. Enquiries are routed to application-support groups organised by software product and by the kind of equipment it runs on. Within the application-support groups, consultants familiar with the way the software can be used, and

with the way it works, form the first line of response. They are able to resolve most of the enquiries on the spot, but 20 per cent have to be passed to the Development Division for resolution. It is here that the PMS comes into its own. It logs problem reports at every stage of response and resolution, using customer references and event codes. When a coding change is made, for instance, the programmer records the details on the PMS. These are immediately available to others, so duplication is avoided. The PMS helps to coordinate adaptive and corrective maintenance work. It monitors maintenance progress, and produces management statistics.

The Development Division is organised into groups that specialise in analysis, coding, and quality assurance. Tested software is batched for release. Different forms of release reflect the level of support that Peterborough Software provides. For instance, versions for release which are necessitated by government legislation get full support. Any earlier versions still left in the field beyond a certain date no longer enjoy full support.

The Peterborough Software example is a model for the maintenance of any large application system, but particularly for multisite, multiversion implementation with large numbers of users. The principal lessons are as follows:

- Recognition of the cost and of the importance of the post-release phases of the system life cycle, and the consequent planning (for example, replacement, migration, and technical design) for the maintenance effort.
- The rigour applied to pre-release testing and post-release version identification and control.
- The formal contractual basis that clearly specifies the responsibilities of supplier and customer.
- Recognition of the relative importance of problems that occur in practice at the operational level (including those deriving from imperfect documentation or training), and at the code maintenance level, and of the need to provide adequate support staff at both levels.

A coordinated programme will become more critical as the complexity of systems increases

A coordinated programme, effective across the whole maintenance process, and designed to control changes to the system, will become more and more critical as the complexity of systems increases. Formal procedures are essential to ensure that software is not degraded and to provide an audit facility. The experience of Peterborough Software may serve as a model. At the same time, there are several automated change- and configuration-control packages currently being introduced to the market that could help PEP sponsors to reduce administrative overheads and increase their control over system changes.

Chapter 5

Using methods and tools

The right methods and tools can help to reduce the maintenance workload significantly. Enforcing standards and using modern development methods are both ways of improving the quality of new systems, so that maintenance becomes easier when the systems are operational. Standards and methods encourage the development of new systems with maintenance in mind. Other contemporary tools are designed to be directly beneficial to the maintenance process. They include management tools, testing tools, and maintenance-support tools. The growing availability of modern development methods opens up yet another avenue for reducing the maintenance workload on the systems department — developing systems to be maintained, at least in part, by the users themselves.

DEVELOP NEW SYSTEMS WITH MAINTENANCE IN MIND

PEP sponsors are well aware of the need to develop new systems with maintenance in mind. They are doing this in three ways. The first is by involving staff who have already worked on system maintenance and who are therefore able to bring the benefit of their experience to bear. (We touched on this on page 13.) The second is by enforcing development standards, and the third is by using modern development methods.

ENFORCE DEVELOPMENT STANDARDS

Ease of maintenance is a feature of systems quality. No system can be described as high-quality if it is hard to maintain. Ease of maintenance must be built in at the design stage; it cannot be ensured by the functional specification. Enforcing design standards encourages the development of high-quality systems.

Ease of maintenance must be built in at the design stage

Designing for maintainability requires an understanding of the problems caused by system complexity and structure. Computer systems are inherently complex, but improving their maintainability means making them as simple as possible. This can be achieved in three ways: by *partitioning* the system into a small number of identifiable modules, each meeting a specific purpose; by arranging for the modules to be as *independent* as possible (in other words, minimising their interdependence); by *interconnecting* the modules through a single command-and-control structure.

Partitioning and interconnection are essentially aids to comprehension; module independence is the key to flexibility. A system consisting of simple, well-defined modules that interconnect loosely with each other is relatively easy to change by decoupling from the structure the modules that will remain the same, adjusting the structure itself, and replacing other modules with new or modified versions that are as simple as those that

went before. Distinguishing the modules from the structure has the advantage of localising any program errors that may come to light.

USE MODERN DEVELOPMENT METHODS AND TECHNIQUES

Using modern methods and techniques to develop new systems is widely acknowledged as beneficial to maintenance, because of the improvement in quality that results. This general recognition was confirmed by those of our interviewees who had first-hand experience of using modern methods and techniques over a period of several years.

Four development methods and techniques are particularly significant in reducing maintenance costs

Of the wide spectrum of development methods and techniques, four are particularly significant in reducing maintenance costs: prototyping, to define and test requirements; data dictionaries and data analysis techniques; methods for structured design and programming; re-usable modules, a technique that consists of skeletons of proven logic, which minimise the amount of new code to be maintained. Because they improve the development process, all four help to make maintenance easier, but those most widely used by the respondents in our survey are methods for structured design and programming. Nearly 80 per cent of respondents claimed to have used structured programming, and 57 per cent to have used structured-design methods. Data dictionaries and data analysis techniques were used by 55 per cent of respondents, and prototyping by 10 per cent. No significant use of re-usable modules was reported.

All four of the development methods and techniques recommended above should be supported by contemporary software tools. Computer-aided software engineering (CASE) tools and fourth-generation languages support the first three methods to deliver systems that are more robust, more stable and of higher quality. Software-configuration management tools help to control reusable modules that can be used not only in development but in maintenance as well.

The experience of the systems development department within a public sector organisation in the United Kingdom illustrates the benefit to maintenance of using a fourth-generation language. Cobol has been displaced as the principal language for new systems development by Gener/ol, a fourth-generation language. As a result, the demand for experienced staff in systems development is much reduced, to provide the same level of maintenance as before, and it has therefore been possible to raise the proportion of effort devoted to new systems. In addition, further reductions in the need for maintenance are expected in future, because of the improved quality of new systems developed using Gener/ol.

PLAN TO EXPLOIT CONTEMPORARY TOOLS DESIGNED TO HELP WITH MAINTENANCE

Although software tools of the sort we have just mentioned help to simplify maintenance by improving the development process, these tools have yet to make much impact on maintenance. Most systems that are currently being maintained were developed using Cobol, as Figure 5.1 overleaf shows. Nine respondents stated that over 90 per cent of their maintained code was written in Cobol

Figure 5.1 Development language of systems in maintenance

Proportion of maintained code written in a particular language	Number of respondents maintaining code in:*			
	Cobol	PL/1	Assembler	Other
90-100%	9	2	0	3
20-89%	8	1	1	10
Under 20%	1	0	7	5

*The entries in the table record the number of respondents who have that proportion of code written in the designated language.

(Source: Butler Cox survey of PEP sponsors)

and another six reported that at least 70 per cent of their maintained code was in Cobol.

Languages other than Cobol are becoming more common in maintenance work, however, and are already more widespread than either PL/1 or Assembler. These other languages comprise various fourth-generation languages — for example, Mapper and Application Master.

Three categories of tool provide direct assistance in maintenance

In addition to contemporary tools for generating quality software in new systems development, a range of tools is emerging designed to provide direct assistance in maintenance. These tools fall into three categories: management tools, testing tools, and maintenance-support tools.

USE MANAGEMENT TOOLS TO ENHANCE PLANNING AND CONTROL

Management tools help to improve the planning and control of maintenance. Tools in this category come in two types. One type aims to help with the job of estimating (which should take place during impact analysis). The other type helps to control how successive versions of software are progressively introduced. Estimating tools tend to be linked to proprietary systems development methods, which limits their use in a maintenance environment. Whatever the tool, the ability to calibrate estimating models to the characteristics of the maintenance environment is essential.

A range of configuration and change-management tools is available to control the change process in maintenance work. These tools ensure that successive versions of software are progressively introduced into a production environment under controlled conditions. They also have the ability to generate management and audit reports. Several of them can also be applied to the development environment and can then be used to progress software into the production and maintenance phases.

USE TESTING TOOLS

A variety of testing tools are available which provide source and file-comparison facilities, cross-reference analysis, data standardisation, and code analysis. The use of such tools provides enhanced status reporting, auditing, and quality assurance. With an

automated testing environment, test data and information is easier to maintain and the testing process is simpler to administer.

The use of knowledge-based techniques is likely to have an impact on testing tools — for instance, by using rules to define additional test cases. Some interesting tools are also being developed which incorporate the use of hypertext, which acts as a navigational aid for searching through program structures. (Hypertext allows 'chunks' of text to be related to each other so that the user can decide which relationships to pursue and when to pursue them.)

USE MAINTENANCE-SUPPORT TOOLS

Maintenance-support tools are having the biggest impact on the maintenance process

This third category of tool, designed to help the maintenance process, is having the biggest impact on the maintenance process. Maintenance tools are aimed at the impact analysis and design steps of maintenance. They provide a powerful means of analysis and design, and are valuable where large amounts of existing code have to be examined or modified, especially where the code itself has been subject to previous modification. Although expensive, maintenance tools can cost less than renewing the system. They can be justified where the maintained system is likely to continue in operation for several years. Three kinds of maintenance support tool are currently available: code analysers, restructuring tools, and re-engineering tools. Some of the better-known examples are identified in Figure 5.2, along with a selection of software-configuration management products.

Figure 5.2 Maintenance tools

Category	Tool	Supplier
Code analysis	Pathvu Cobol/SF Recoder Software Testbed Flowtec Via/Insight	Peat Marwick McLintock IBM Corp Language Technology LDRA Ltd Maintec SA Viasoft Inc
Restructuring	Structured Retrofit Cobol/SF Recoder Superstructure Astec PM/SS	Peat Marwick McLintock IBM Corp Language Technology Group Operations Inc Maintec SA Adpac Corp
Re-engineering	PSL/PSA Bachman	Meta Systems Ltd (Keith London Associates) Bachman Associates
Change control	Change Man CCC PVCS ADR/Librarian	Optima Software Inc Softool Corp (K3 Software Services Ltd) Polytron Corp ADR Inc

Code analysers

Code analysers report on the degree to which programs (in the main, Cobol) are syntactically correct, and they indicate the complexity of the existing code. So-called static code analysers report, in addition, on departures from programming standards. Dynamic code analysers report on the results of a test run; they may, for instance, report the number of untested lines of code.

The experience of the systems development department of a Belgian utility highlights the risk of failing to exploit the benefits of code-analysing tools. To meet one of its application requirements, the department selected a packaged software product. At first sight, it seemed to fit the need closely — it was designed to a similar specification — but experience showed that this first impression was false. The package has had to be extensively modified to cope with increased data-storage and transaction volumes, which has led to significant changes to its internal structure. During the space of just one year, the maintenance effort has reached half of the original estimate of developing the complete system from scratch. Code-analysis tools could have helped to clarify the suitability of the design in the first place, and to estimate overall life-cycle costs and resourcing requirements more accurately.

The aspirations of a large agricultural merchant provide a further illustration of the potential of code analysers. The systems department has had to face a problem that is not uncommon — that of losing many experienced staff in a short space of time, following an organisational change. Having no alternative but to assign staff to maintenance who had little or no direct knowledge of the systems, the department turned to a code analyser (in this case, Via/Insight). Although it is still too early to assess the impact of this code analyser, the department is expecting to obtain three important benefits: transfer of knowledge to the maintenance staff about the application of the systems, at the code level; improved code reliability; reduced maintenance turnaround time as a result of better productivity.

Restructuring tools

Restructuring tools transform unstructured code into new, functionally equivalent code that is restructured in accordance with top-down principles, and is fully documented. The steps in the restructuring process are the following: analysis (in much the same way as with a code analyser), code reorganisation and redesign (done manually with all but the most sophisticated restructuring tools), code generation from the revised program design, and verification.

The experience of a major oil company illustrates the use of a restructuring tool. All of the commercial systems (over 1,200 programs) were written in a programming language no longer in common use. The level of expertise needed to use the language was substantial and required very skilled maintenance staff. This language was very difficult to use and required an extensive amount of training. New programmers would serve an apprenticeship with the senior staff to learn the language and it could be as long as two years before programmers would be allowed to work unsupervised with the language.

In 1985, the company planned to rewrite all of the applications written in this language. It estimated that this would cost about six dollars per line and that it would take 10 calendar years to complete all the work, at a total cost in excess of $15 million. Management approval to proceed was granted. However, before going ahead, the company evaluated the possibility of restructuring its systems as an alternative to the high-risk, high-cost rewrite strategy, using a restructuring tool. It chose Recoder as the tool

and submitted a new plan, which indicated that all the code could be restructured and the existing systems re-engineered in two years.

These tasks were, in fact, completed in less than two years; after 14 months, 850 programs had been restructured. A billing system of over 500 programs was completed at an average of one to two hours per program, and the total cost was eight cents per line. On another system, one of the company's restructuring goals was to improve its run-time performance. With the improvements implemented, the daily run-time was cut by three to four hours, and the annual production cost savings were $170,000.

Restructuring tools are expensive, however. Prices range from $60,000 for Adpac's PM/SS product, to more than $100,000 for IBM's Cobol/SF. Despite the suppliers' claims of productivity gains as high as 60 per cent in subsequent maintenance activities, restructuring tools are often hard to justify.

Re-engineering tools

Re-engineering tools go one step beyond restructuring tools. They have the ability to form an entirely new design from existing code. They work first by translating existing Cobol code back to a design-level representation (this is a process known as reverse engineering), then by working forward from that point to create entirely new, restructured code. Currently, there are only two re-engineering products of significance. One is Bachman Associates' reverse-engineering tool, which is available only in the United States. The other is Meta Systems' PSL/PSA, which is marketed in Europe through Keith London Associates of the United Kingdom.

These products are at the trial stage and provide facilities in a limited software environment. Nevertheless, they are pointers for the way in which CASE tools for maintenance environments will develop.

LIMIT THE RANGE OF METHODS AND TOOLS USED DURING DEVELOPMENT

Systems departments should adopt only a limited number of new methods and tools

For most systems development departments, contemporary methods and tools represent a significant departure from what has gone before. They carry with them an overhead burden in terms of learning, standards, and previous practice. It is for this reason that the average systems development department should be careful to limit the number of new methods and tools that it adopts. By taking on more than a few, a department risks so diluting its expertise in any one of them that productivity becomes lower, not higher, than before.

The experience of a leading multinational oil company illustrates the point. Systems development and support is devolved to business-unit level. A central systems group provides highly specialised skills, and recommends methods and standards to be adopted by the business units. The central systems department has no authority, however, to impose its recommendations. Not surprisingly, the business units, being driven by expediency, have tended to go their own way. Now, the group finds that a growing number of languages and methods are in place, constraining the

opportunities to benefit from exchanging resources and expertise across the business units.

DEVELOP SYSTEMS THAT CAN BE MAINTAINED BY USERS

Recent advances in fourth-generation languages now make it possible for users to become directly involved in maintaining their own systems. It is common for organisations to provide users with query languages through which they can derive data and generate reports. It is a small logical step from this to the provision of languages, such as Mapper and Gener/ol, that are powerful enough to allow users to add functionality to systems. A systems development department based in the Netherlands has begun to do this with considerable success. It began some four years ago by introducing the idea of direct user-maintenance on an internal invoicing system common to several independent business units within the parent group. Because it was recognised at the outset that the system specification would differ between the business units, the core of the system was designed in such a way that business-specific enhancements and changes could be generated by the users themselves.

Users can become directly involved in maintaining their own systems

Code-generating software was written by the systems department (using Natural and Adabas), enabling users to specify their own data requirements and functions without recourse to the systems staff themselves. The software was designed to take account of the fact that users have no specialist systems knowledge or skills.

The result has been to reduce the amount of maintenance carried out by the systems department, from an estimated five to six man-years each year, to a quarter of this. Encouraged by its success, the department is now planning to adopt the principle of user-maintained systems as a central element of its future policy. It has taken the decision to produce a number of general-purpose modules that will allow users to specify some of the interfaces between different application areas.

Advances in maintenance-support tools have lagged behind advances in development tools. This situation is set to change in the immediate future. We predict that the current generation of maintenance-support tools will provide significant benefits for those organisations that have to maintain existing Cobol programs.

Chapter 6

Reaping the benefits of improved software maintenance

Software maintenance consumes significant systems development resources, and improvements to maintenance will result in increased productivity and value for money for PEP sponsors. We have made suggestions for improvements in a number of areas, summarised in Figure 6.1.

Figure 6.1 Action checklist

Formalise the maintain-or-replace decision

Conduct annual reviews to re-assess the costs and benefits of maintained systems.

Remove uncertainty by instituting a maintenance-rating process.

Define the share of resources to be allocated to maintenance as part of the overall strategy to manage the applications portfolio.

Motivate, train, and manage maintenance staff

Organise staff to optimise job satisfaction, perhaps by establishing a separate maintenance organisation.

Select maintenance staff with suitable personal characteristics (patience, adaptability, and willingness to lend support).

Train staff in the technical and communication aspects of the work. Make job rotation a component of any training programme.

Utilise third-party services to provide specialist skills and to relieve internal staff for new developments.

Formalise the maintenance process

Introduce a consolidated programme of action, ranging through change management, impact analysis, system release planning, change design and implementation, testing, and system release/integration.

Adopt a formal procedure for change management, followed by impact analysis, to evaluate maintenance requirements.

Improve the development environment

Develop new systems with maintenance in mind. Involve staff with maintenance experience in the development process, adopt firm standards, and use modern development tools.

Draw on the four most useful methods and techniques — prototyping, data dictionaries and data analysis, structured design, and re-usable models — all of which are supported by contemporary software tools.

Limit the range of development tools to constrain maintenance skill requirements.

Develop systems that can be maintained by users.

Exploit contemporary tools designed to help with maintenance:

- Use management tools such as configuration and change control software to improve planning and control of maintenance.
- Use testing tools to reduce the complex administrative tasks associated with test monitoring, auditing, and quality assurance.
- Use maintenance-support tools to help with the maintenance-rating process, and to prolong the life of existing systems.

These improvements will be achieved, however, only if the role of maintenance is properly understood. The purpose of maintenance is to ensure that software continues to serve business goals, and it must be managed accordingly. Maintenance is a

process with its own rules and techniques; there are well-established maintenance procedures, underpinned by automated tools, designed to reduce the complexity of the maintenance task, but they will provide the promised benefits only if they are integrated with control procedures and used to support a clearly defined approach to maintenance that is based on a clear view of the maintainability of a system and of the options available for maintaining it.

Choosing an approach to the management of maintenance is not an easy task. Some systems will be quite stable while others will be complex and volatile. There is no one way of dealing with maintenance that will suit all organisations, but the procedures described here are the foundation of a management approach that is widely applicable, irrespective of specific implementation issues.

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.

P E P

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.
- Managing Contemporary System Development Methods.
- Influence on Productivity of Staff Personality and Team Working.
- Managing Software Maintenance.

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 791c,
1064 EB Amsterdam
☎ (020) 139955, Fax (020) 131157

France
Butler Cox SARL
Tour Akzo, 164 Rue Ambroise Croizat,
93204 St Denis-Cédex 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 and 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