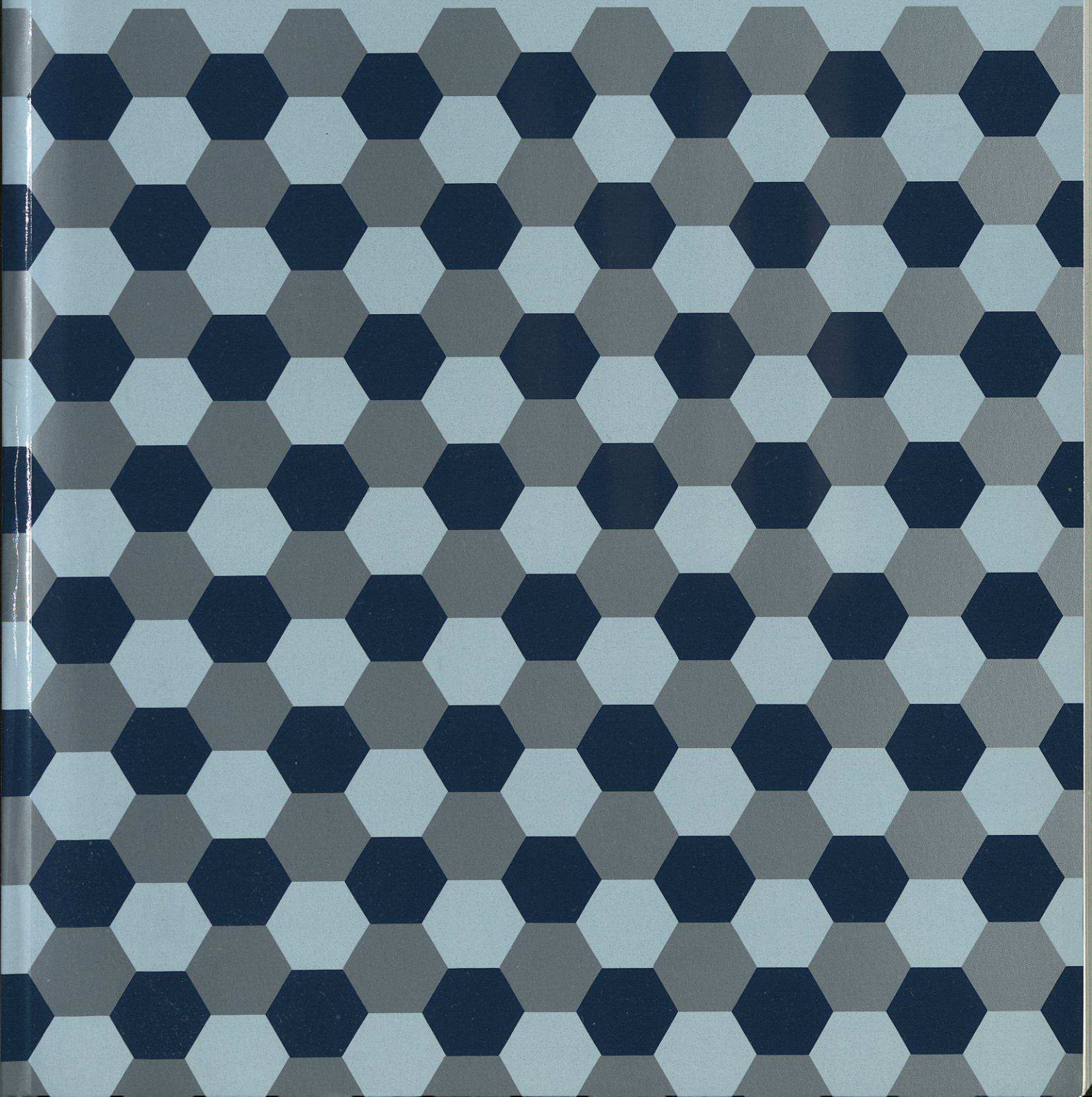
Improving Systems Development Productivity and Quality

BUTLER COX



BUTLER COX

Improving Systems Development Productivity and Quality

A Special Report of Best Practice in Europe

Butler Cox plc

LONDON
AMSTERDAM MUNICH PARIS

Published by Butler Cox plc
Butler Cox House
12 Bloomsbury Square
London WC1A 2LL
England

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

BUTLER COX

Improving Systems Development Productivity and Quality

A Special Report of Best Practice in Europe

Contents

Contents (continued)

Chapter 1

Improving systems development productivity and quality

Productivity and quality are major concerns for every business involved in the development and maintenance of substantial computer systems. It is not that current performance is necessarily poor, but rather that any and every improvement has a potentially significant benefit.

This report has been prepared by Butler Cox with the aim of providing systems development managers with practical advice, based upon proven best practice, on how to improve their departments' productivity and quality.

The detailed analysis and recommendations are based on Butler Cox's own research and consultancy experiences. In particular, we have drawn extensively on the research carried out for the Productivity Enhancement Programme (PEP). This is a continuous programme, with an international membership, an integral part of which is a research database of over 600 projects.

DEVELOPING SYSTEMS

Until recently, information systems were merely another company resource. Today, they have become the environment in which business is transacted. For more and more companies in the future, they will become the means by which business processes, markets, and competition are redefined.

Already as much as half of all capital expenditure is going into information technology. Until recently, a half of the typical business budget for information technology and services was spent on equipment. Now, that proportion is falling, and its place as the major source of spending is being taken by software and the staff to develop and maintain it.

Although it is true that a growing proportion of this expenditure is on ready-written software that can be bought off the shelf, the fact is that, in virtually every major company, that proportion is still outstripped by home-grown software.

Home-grown software, written to meet the needs of an individual business, will be with us for years to come. Specialist analyst and programming staff continue to develop new software for new applications, to modify and improve ('maintain') existing application software, and, often, to tailor ready-written software to link to existing (or new) home-grown software.

The work is demanding, time-consuming, and expensive. Strategically important systems often need to be developed in a hurry in order to respond to a brief window of competitive opportunity or to satisfy legislative or regulatory requirements. Unfortunately, pressures to reduce timescales tend to reduce productivity and quality as well, turning advantage into

disadvantage. Within the systems development community, delays, overruns, and quality shortfalls are commonplace. Any and every means of improving this position has a potentially significant benefit.

The systems development environment is a complex one. While experience and skills have grown over the years, and new tools and techniques have been introduced to speed up the development process, the demands for computer systems have become even greater. Speed of reaction to business pressures and opportunities is at a premium; top-class development skills have remained in limited supply; the number and variety of different computing equipment has expanded, each type with its own programming and operating requirements; the range of existing systems, each system making its own demands for maintenance, has grown over the years.

It is against this background of complexity and rapid change that managers are seeking an answer to a common question — how to improve two things: productivity and quality.

PRODUCTIVITY AND QUALITY

In systems development, productivity measures the work rate at which the process is carried out — the output achieved for a given input. Where output is measured in lines of code, productivity measures the internal efficiency of systems development; where it is measured in terms of the functionality of the delivered system, productivity measures the external efficiency of systems development.

Quality measures fitness for purpose of the delivered product — in terms both of technical factors, such as the reliability and maintainability of the software, and of user-perceived factors, such as ease of use and correctness.

Improving productivity and quality in systems development means making changes to one or more of the facets of the task. What makes this apparently simple challenge so complicated is, first, the large number of variables that interact within the systems development environment, and second, the difficulty of measuring productivity and quality in the first place, and hence, assessing the effect of any change.

Consider the number of variables that interact within the systems development environment. They include the precision of the initial specification of requirements, the skills of the development staff, how the staff fit together into a team, the techniques and tools that are available for them to use, the time available to do the job, testing and quality control factors, and many others. To understand these variables better and how they interrelate, it helps to put them into groups. It is better still if these groups correspond to areas that managers in systems development can do something about.

Four groups that are relevant are those of organisation, staffing, techniques and methods, and development tools.

Organisation is to do with the way the systems department is structured: how it fits into the systems function, how it relates

to its customers (the users), and what its internal sub-divisions and reporting lines are. The issue of staffing has to do with recruitment, training, development, motivation, and performance measurement and reward. Techniques, such as structured programming and data analysis, are the procedures on which systems development is based; methods specify formally how to carry out the activities embodied in the techniques (most methods are proprietary products, such as SSADM and Method/1). Finally, tools automate the activities within a development method (programmer workbenches, screen painters, and report writers are three examples of tools).

PURPOSE AND STRUCTURE OF REPORT

The purpose of this report is to provide systems development managers with practical advice, based upon proven best practice, on how to improve the systems development department's productivity and quality. The report should prove valuable to other managers and technicians within the department — indeed, to anyone concerned about improving the department's working efficiency and the value of its end products. Although the report is aimed principally at medium-sized and large businesses spending substantial sums on commercial information systems, its findings are likely to prove beneficial to managers of smaller companies as well.

The scope of the report is set firmly in the four areas outlined above: organisation, staffing, techniques and methods, and development tools. It is in these four areas that Butler Cox has most recently focused its systems development data collection research and analysis.

The report is structured into five main chapters. Chapter 2 is about organisation, and it deals with four main issues. The first is how, in a decentralised group, to divide responsibilities between the central systems function, and the functions that are devolved to the operating business units — in other words, how to balance centralisation with decentralisation. The second issue is how to achieve structural simplicity, with clear accountability and the minimum number of management layers. The two other organisational issues that deserve special attention and that are covered in this chapter are those of testing and maintenance.

Chapter 3 is about staff and motivation. Because the single largest cost element in most systems development departments is staff, improving the productivity of development staff is a critical matter. Motivation is an important element in this, and one in which several improvement opportunities are open to managers. These opportunities include broadening the scope of the work, job rotation, flexible career structuring, goal-setting and feedback, performance-related pay, and job fitting. These are opportunities that apply at the level of the individual. There are further opportunities at the level of the team, to do with team size and composition, and the role and style of the team leader. The motivation of staff engaged in maintenance is a topic that deserves attention in its own right, and one that is explored in this chapter.

Chapter 4 is on the subject of techniques and methods. Many companies have introduced one or other of a wide range of

techniques and methods that are available on the market. The benefits, however, are not always clear. Only in three cases have we found convincing evidence of consistent success with techniques and methods — when they are used to help formalise software testing, to help control maintenance, and to establish a quality-management programme.

Chapter 5 is about using contemporary tools such as fourth-generation languages like Mantis, CASE tools, and re-engineering tools. Despite the enormous benefits in productivity and quality that are widely claimed by the suppliers of tools, the results that are achieved in practice are much more ambiguous. CASE tools, for instance, are failing to deliver reduced development time, or fewer errors, or even increased reliability — although it is possible that they do increase productivity over the whole life of a system. Maintenance tools, too, have yet to make much impact. Because they are specialised, tools are relatively inflexible. They need to be selected and matched to the application environment with great care. Introducing them requires sensitivity and careful planning, which is best undertaken by implementing a pilot project.

Chapter 6 deals with the measurement of productivity and quality. The earlier chapters are about the means of achieving an end — the improvement of productivity and quality. Yet there is little point in striving for this end if improvements cannot be measured. The measurement of productivity and quality is elusive, however, which goes a long way towards explaining why few companies have adopted such measurement on a continuing basis. In this chapter, we explain the essentials of measurement, and provide some advice on implementing a continuous measurement programme.

The report concludes with a brief chapter on the next steps to take — an action checklist for managers.

RESEARCH SOURCES

This report is based on recent research that we confidently believe to be the most up to date and comprehensive in Europe. The principal source is Butler Cox's productivity and quality enhancement programme, PEP. Membership includes some of Europe's largest, and acknowledged to be leading, computer systems users. They represent all the main industry sectors: financial services, manufacturing, utilities, energy, retailing and distribution, process and chemicals, and local and central government.

The core component of PEP research is the projects database, now amongst the largest of its kind in the world, with over 600 projects. At the time of undertaking the research that forms the basis of this report, details of some 400 projects were recorded on the database, each with an average of 41,000 lines of source code. All the project information is collected by all of the PEP members, under Butler Cox's direction.

The PEP database provides an unmatched source of comparative, across-industry, practical data. Analysing the database allows us to compare such diverse factors as productivity by type of tool, error rate according to the technique in use, and delivery rate as a function of staffing level.

Chapter 2

Putting the right organisation in place

There is no 'right' organisation structure for systems development that is universally applicable. The one that suits a particular systems development department will depend on the characteristics of the parent business, the technological environment, and the systems environment. As the relative importance of these factors changes over time, the organisation of systems development will need to be modified.

Most substantial businesses today are organised as a group of operating units that are to some extent decentralised. In groups like this, it is commonplace for the systems function to be decentralised as well, at least in part. Balancing the activities of the central function with those at the operating level is the first organisational priority. Clarifying the responsibilities of users is the other side of the same coin. For systems development, the next priority is to ensure that its structure is kept simple, with the minimum number of management layers. Two other organisational issues within systems development that deserve special attention are those of testing and maintenance.

These topics are addressed in turn in this chapter. The chapter finishes with a section concerned with the identification and resolution of common organisational problems.

BALANCING CENTRALISATION WITH DECENTRALISATION

Today, most businesses are managed in a more decentralised way than was the case 10 years ago. There has been a recent similar trend towards decentralising the systems function. It is now increasingly commonplace to find responsibility for developing and implementing systems devolved to business-unit level, and responsibility for systems policy and standards retained at the centre. Although dividing responsibilities is never easy, ground rules are available that managers can use to their advantage.

THE TREND TO SYSTEMS DEVOLUTION

Faced in recent years by mounting competitive pressures and rapid market changes, more and more large businesses have chosen to decentralise. To manage the resulting decentralised group structure, head offices allow individual business units a degree of autonomy while remaining involved in the business units' strategy planning, in approving their plans and capital spending, and in overseeing their financial performance.

Decentralisation is widely held to deliver worthwhile benefits. Breaking up an organisation into smaller business units and delegating authority and responsibility to the business-unit managers brings management closer to the customer, helps to

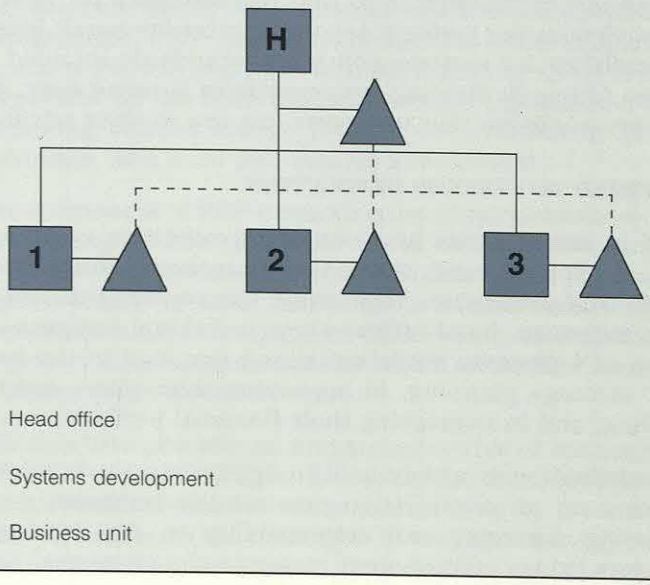
improve operational flexibility and responsiveness, and encourages innovation and specialisation. Decentralisation, it is also claimed, helps to sharpen awareness of market and competitive trends, because decision-making managers are brought closer to the action.

Devolving the systems function is just as much in evidence as decentralising the business itself. Typically centralised until the late 1970s and early 1980s, systems functions have since followed the path to devolution. Not to be confused with merely distributing computer systems physically to divisions and departments, devolution implies decentralising management authority as well (see Figure 2.1). Today, it is increasingly commonplace for business units themselves to buy and operate computers, and to develop and maintain the systems that run on them. The trend has been encouraged by advances in computing technology. The economies of scale that used to favour centralised installations disappeared years ago, with the advent of department-supporting minicomputers and personal micro-computers.

Devolving responsibility for information systems has been encouraged by most business-unit managers. They have claimed lots of benefits. Reduced costs, closer control over priorities, systems better tailored to business needs, and relief from dependence on the central function with its order backlog and ageing core systems are some that are often quoted.

As well as benefits, however, there are some very real risks. At business-unit level, there is the danger of systems staff sacrificing quality by cutting corners to meet delivery pressure from local managers. Retaining skilled systems staff can be a problem when the first allegiance of the staff is to their profession, rather than to the business. At corporate level, there is the risk of systems

Figure 2.1 Today, the systems function in most large groups is partly distributed



being expensively and unnecessarily replicated between different business units with common needs. Worse still, different business units may build their own incompatible 'islands of automation', compounding the difficulty of linking up electronically in the future.

The common result of these opposing pressures is systems functions that are divided, in part devolved to business-unit level and in part remaining centralised at head office. That means a matrix organisation and, for the centralised part, a new hybrid role. Most head office top managers now accept this as inevitable, yet the question remains of how to divide responsibilities between the centre and the business units.

The typical range of systems development responsibilities is summarised in Figure 2.2. The figure shows the responsibilities under four headings: delivering head office services, establishing the technical infrastructure, developing staff, and developing business-unit systems.

The first two of these, concerning head office services and the technical infrastructure, are responsibilities that will often apply groupwide. (The technical infrastructure consists of the corporate communications network and the standards that govern the interworking of systems across the group.) Normally, these responsibilities should be discharged from the centre. The second two headings in Figure 2.2, developing staff and developing

Figure 2.2 Systems development responsibilities may be grouped under four headings

Delivering head office services
- Providing systems for head office.
- Making central bureau services available.
- Watching trends in information technology.
- Providing group-wide development support.
- Monitoring competitors' use of systems.

Establishing the technical infrastructure
- Developing the infrastructure.
- Defining standards and interfaces.
- Defining policies and methods.
- Quality assurance.

Developing staff
- Building management awareness of information technology.
- Promoting and catalysing the use of information technology.
- Recruiting and developing systems development staff.
- Training staff in the use of systems.

Developing business-unit systems
- Budgeting and planning systems.
- Designing and implementing systems in accordance with policy and standards.
- Providing education and support for end users.
- Maintaining systems.
- Buying software.

business-unit systems, should normally be discharged at the level of the business units themselves.

This allocation of responsibilities, combining centralisation with decentralisation, can be no more than a general guideline, of course. How, in practice, to divide responsibilities between centre and business units will vary widely according to specific group circumstances. One important consideration will be that of the general management style of the group. Other things being equal, the style of the group as a whole should be followed by that of the systems function: the more decentralised the group, the more decentralised information systems should be.

In practice, however, there are powerful reasons for adopting a more centralised style of systems development management than is suggested by the group-management style. One is to take advantage of economies of scale in equipment purchasing. Another reason is *the trend to integrate systems,* which involves the design of corporate databases to support a variety of business and executive-support applications. One systems manager described to us the importance of allowing for future integration. He explained, ''We had nine different computer suppliers, 12 different operating systems, and 16 different programming languages. We had taken ourselves up a cul-de-sac. Computing had become the fiefdom of departmental barons. There was information everywhere but no-one from other departments could access it.''

A third reason for adopting a management style that is offset to the centre is the need to preserve flexibility of choice in the group's future organisation. Groups with synergistic divisions, such as multiple retailers, often find a need for cooperation and shared approaches to business ventures, such as the use of common credit-card systems. This type of group often restructures in such a way that, for instance, all manufacturing functions or all marketing functions are put under one line manager. Autonomous systems development, with mutually incompatible hardware or software, hampers the process of re-aligning the business units.

CLARIFYING THE RESPONSIBILITY OF USERS

As well as balancing centralised and decentralised responsibilities, it is important to strike a balance in the allocation of responsibilities between systems development specialists and their end-user customers. In most businesses, this is a contentious issue.

It is often suggested that users are not yet ready or willing to take on much responsibility for systems development. Sometimes, a 'damage limitation' approach is pursued. However, unless the alignment of responsibilities between specialists and users is clearly defined and agreed, the systems development function will be unable to support the business community as it should. Users will then continue to build their own systems, often undocumented and probably unmaintainable.

The devolution of control to users is, in fact, already happening. It began with the proliferation of personal computers, and continued with the move towards departmental computing in

the mid-1980s. This devolution must be managed in such a way that order is maintained without initiative being stifled. A useful way of formalising responsibilities is by reference to three levels of application that are found in most companies. These can be described as core systems, non-core systems, and personal systems.

Core systems

These are the applications that are essential to the day-to-day operation of the business. In general, they maintain and update the common corporate databases, and often provide a base for subsequent applications to use. Clearly, if these systems, which exploit database technology and commonly process high volumes of transactions, are to be developed efficiently, skilled technicians will be needed. The systems they design must be built in accordance with central policy guidelines to ensure that a coherent software infrastructure is maintained.

Senior management should take the lead in deciding what systems should be developed, and also in managing their development and implementation. These managers should be able to see the relationship between computer systems and business goals, and to work out how a computer application could effectively automate a particular business function. Responsibility for innovation becomes shared with senior managers, and no longer the sole preserve of the systems department.

Non-core systems

These are the systems that are used by a business unit, or a department within a business unit. Their purpose is to achieve the unique objectives of that business unit, and they do not normally affect the day-to-day operations of other business units. It is appropriate, therefore, that business-unit managers have control over what systems are implemented, but because it is possible that the data and programs created will be shared by other departments in the future, the systems should conform to the company-wide policies and guidelines laid down by central systems development management.

Development of these applications is frequently undertaken by the users themselves, who should be encouraged to experiment with different designs and to explore the possible applications of the newer technologies, such as end-user computing and office automation. Three separate studies conducted by the Rand Corporation in 1988 confirm the wisdom of this approach. All these studies concentrated on the effective introduction of end-user computing, and found that success in this area is closely related to the amount of control exercised by users. This is particularly significant for business-critical systems, for which speed of development and close fit to requirements, rather than technical efficiency, are paramount. The role of the systems specialist in these developments is to provide education, support, and guidance.

Personal systems

These are not application systems in the usual sense of the word, but a variety of tools and techniques that enable users to set up their own systems. They include the microcomputer-based

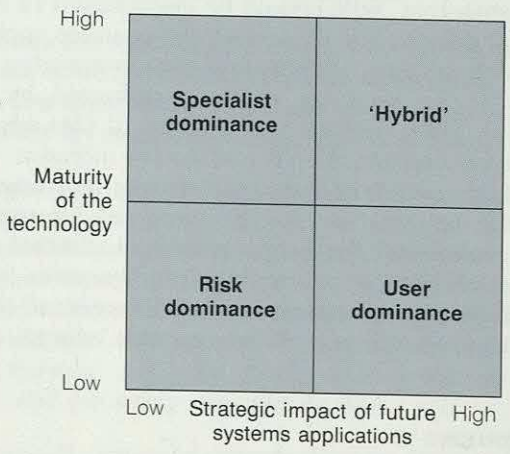
systems developed using spreadsheets, word processors, database managers, and so on. These systems are firmly in the control of the users, and the role of the systems specialist is limited to providing them with company-approved packages and training in their use.

These systems, however, frequently become the business-unit (or non-core) systems of tomorrow, with subsequent access to the corporate databases being requested to enable users to manipulate the information locally. It is important, therefore, that these users should adhere to conventions established by the systems development department for the company as a whole. Some organisations make it a rule that users can choose which suppliers of personal computer they use, but only standard products will be connected to the corporate network.

With the division of responsibilities described above, the burden of providing computer systems for the entire business is removed from the systems development department. Most business-unit projects can be developed by users, while the systems department concentrates on those that are shared by several business units. Responsibility and accountability for performance is shifted to the users for all types of systems development. This places the onus on business management to become educated in the use, control, and delivery of computer systems within their organisation, and to take time to understand the true scope of a systems project and to devote adequate resources to its completion.

The result of a recent study is depicted in Figure 2.3. It shows a matrix that can be used to define the respective responsibilities of users and systems development departments. The matrix takes into account the strategic importance to the group of future systems developments and the maturity of the technology required for these applications (not the stage of assimilation reached by the particular organisation):

Figure 2.3 The responsibilities of the systems department and users may be allocated with reference to a matrix



(Source: Complex organizations and the information systems function. Oxford Institute of Information Management, 1987.)

— If the strategic impact of applications is assessed as 'low', and the technology required as 'mature', considerations of operational efficiency are paramount and specialists should be given responsibility for them, although a user manager will be ultimately accountable. Such applications might include support systems, like payroll and general ledger.

— If the strategic impact is low, but new technology is required, the technical risk is high and the potential benefit to the business very limited. The application should probably not be developed.

— The combination of high perceived strategic impact and relatively mature technology means that the users need to be in real control of the systems strategy (the 'what'), while specialists control the 'how' of systems development. These could be core or non-core applications.

— Applications that have a high strategic impact and use new, immature technology should be entirely within the users' control, with 'an unabashed concentration on effectiveness'. These are the non-core and personal systems developed using end-user computing and office technology.

GROUPING SYSTEMS DEVELOPMENT RESPONSIBILITIES BY FUNCTION

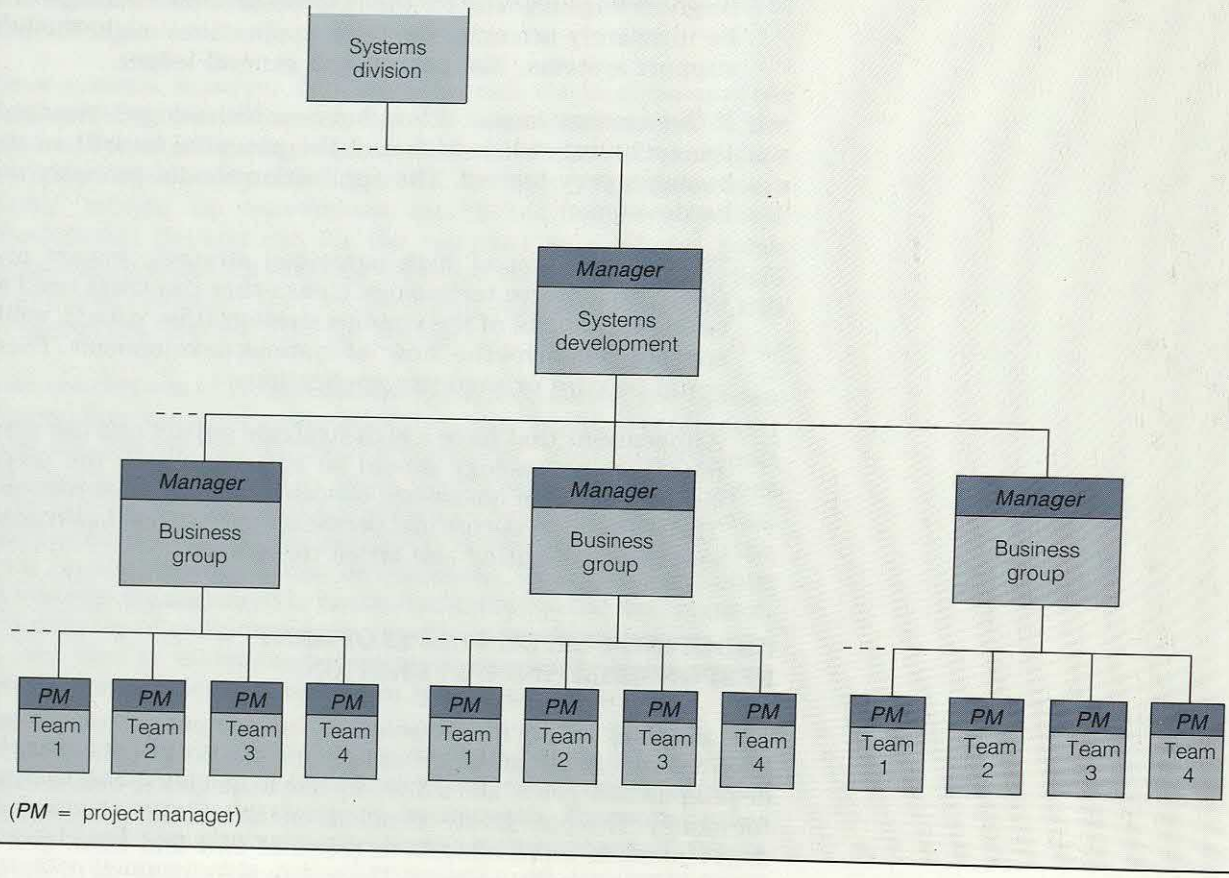
The growing reliance of businesses on computer systems has resulted in a much greater emphasis on the cost of the systems development service, and a drive by line managers to obtain value for money. This puts greater demands on the systems development department to meet budgets in terms of cost and time, and to produce high-quality systems. The key to meeting these demands is to allow greater autonomy to the people who are providing a service to line management. This involves 'flattening' the management structure, and within the simplified structure, organising each group to fulfil particular functions.

REDUCING THE NUMBER OF MANAGEMENT LAYERS

It is not uncommon for systems development departments to introduce more and more layers of management — sometimes in the belief that this creates a career structure. In fact, career advancement is a management issue that should be handled independently of the structure of the systems development function. Moreover, systems development departments that have simplified their structures have frequently improved their productivity.

Figure 2.4, overleaf, shows how such a simplified structure might work. Staff are divided into several business groups. With a centralised management style, these will be located at head office. With a devolved management style, they may be physically dispersed. Each business group contains up to 50 staff, depending on the development workload. (Fifty is about the maximum number of development staff for a business group; beyond this, staff begin to lose a sense of identity with the group.) Within each business group, staff are allocated to work on projects under a

Figure 2.4 An efficient organisation structure for systems development has no more than three reporting layers



project manager, depending on their skills, availability, preferences, and so on. Each team, wherever possible, should be kept to a maximum of six (see Chapter 3).

Responsibility for developing systems within the policy and strategy guidelines laid down by the central systems department should be devolved to the business-group managers. The role of a business-group manager is to liaise with line managers and to agree on the scope and type of each systems development service needed by the business unit. In the past, the systems development department has been a monopoly supplier of services handed out to users. This is now changing, both because users are taking control of some of their own systems developments, and also because competitive pressures on users are encouraging them to look for alternative suppliers.

As a result, the systems development department now has to adopt a more marketing-oriented approach to increase its credibility with its users, and to retain its status as the main supplier of development services. The role of the business-group manager is therefore a difficult one. It requires a person able to deal effectively with senior business managers, knowledgeable enough about technical matters to be able to guide approaches to development, and a diplomatic yet forceful personality. The advantage to line managers is that they have a single point of contact for all systems development ideas and problems.

The situation depicted in Figure 2.4 represents the optimum structure. Factors such as the overall size of the systems development function, or geographical dispersion, may suggest a need for additional management layers. Systems development managers wanting to improve the efficiency of their departments should aim for a structure that is as flat as possible.

An example of a company that has successfully done exactly this is British Airways, an international airline. It has deployed 750 development staff into business groups of around 50 staff, each one divided into four to six project teams. As a result, motivation and productivity have increased enormously. While this re-organisation has been based on lateral rather than vertical expansion, the control span of each layer of management — that is, the number of staff under each manager's direct control — has been kept manageable by delegating more authority and responsibility to business-group and team-level managers.

ORGANISING CENTRAL SYSTEMS DEVELOPMENT

Within the simplified structure described above, each business group must be organised to fulfil its responsibilities in the best possible way. In the central systems department, the emphasis should be on three main functional groups: infrastructure planning, development support, and quality assurance.

Infrastructure planning

The central systems planning group is responsible for planning the company's technical infrastructure, which is essential to ensure that future systems can be integrated. This includes defining the operating systems, languages, database-management system, data dictionary, communications protocols, and user-interface standards that will be used throughout the company. The planning group should ensure that core applications comply with the components of the infrastructure to form a flexible basis for developing non-core applications. Wherever possible, non-core applications should also comply, although a non-core application that does not conform to the standards, yet provides a good business solution, is preferable to one that conforms but is inferior in business terms.

Development support

Many organisations have found it useful to establish a separate team of systems professionals who support development teams in the use of different tools and techniques. The responsibilities of this development-support group include the provision of training in the use of modern development tools, project-management techniques, CASE tools, and so on. They are known by several different titles including the systems research group, the advanced technology group, and the development centre. The aim of the group is to concentrate specialist expertise into a 'research and development' type of role, in which the team members are not distracted by development work. Generally, these are small teams, and they provide a useful way of concentrating specialist skills — though care must be taken to ensure that they stay in close touch with their own customers.

Quality assurance

For many organisations, the quality-assurance group is a recent addition to the systems development department. With

responsibilities for systems development being increasingly devolved to business-unit level, the role of a quality-assurance team is a vital part of ensuring company-wide compliance with central systems development policy. The responsibilities of this group are, first, to initiate or develop standards, procedures, systems development and project-management methodologies, and management practices. The second responsibility is to ensure that compliance audits are carried out, by reviewing all major projects within the systems development organisation at prescribed intervals.

It is not the responsibility of the quality-management group to carry out quality-control checks itself, nor, indeed, to ensure that the guidelines are being followed; on the contrary, the group should arrange for as much of the responsibility as possible to be devolved to project managers and their teams. The department's drive for better-quality systems must centre on making individual development staff responsible for producing quality output. For this approach to be successful, everyone in the systems department should be committed to it, and take responsibility for the quality of his or her own contribution to systems development.

ORGANISING DEVOLVED SYSTEMS DEVELOPMENT

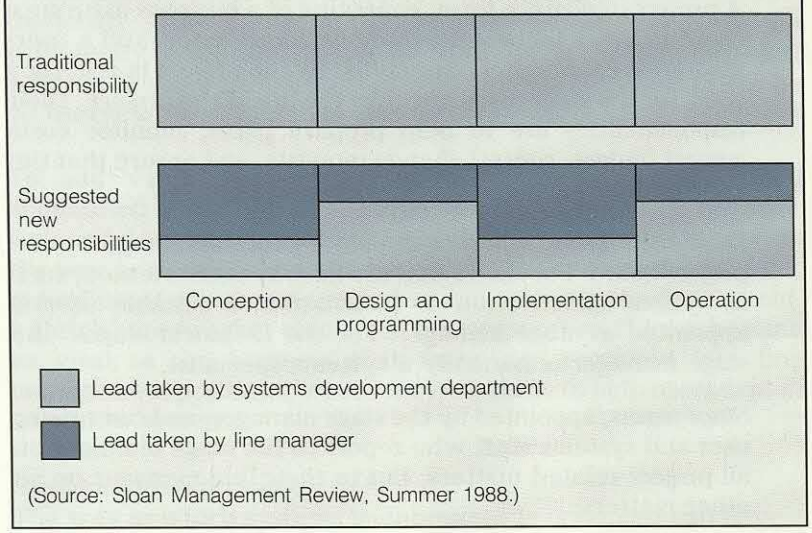
Within devolved systems development functions at business-unit level, the emphasis should be on a further three functional groups: systems development, education and user support, and systems maintenance.

Systems development

The systems development group is responsible for the detailed design, programming, testing, and implementation of all core systems and for ensuring that the systems conform with the central systems policy and standards. In the area of testing, the organisation of development teams can have a particularly strong influence on effectiveness. This subject is therefore considered in more detail later in this chapter. The systems development group is also responsible for those business-unit systems that need to be developed with traditional third-generation technology, for which specialist skills are needed. Such projects are, however, initiated by line managers, who are also the best people to manage them, because they are committed to the time and cost schedules and can mobilise user staff during implementation.

Jack Rockart, director of the Center for Information Systems Research at the Sloan School of Management, believes that, because of the business-critical nature of many applications being developed today, line managers should take the lead in both the conception and implementation stages. He suggests that, because it is not usually possible to cost-justify competitive-advantage applications, and because implementation usually provokes significant organisational changes, the systems development manager can no longer be responsible for driving these systems forward. His view of how responsibilities should be allocated between line and systems managers is depicted in Figure 2.5. Some organisations report active participation by line managers in significant development projects already, and it is certainly a trend that is set to continue. Most systems development departments,

Figure 2.5 The line manager should take on more responsibility for systems development



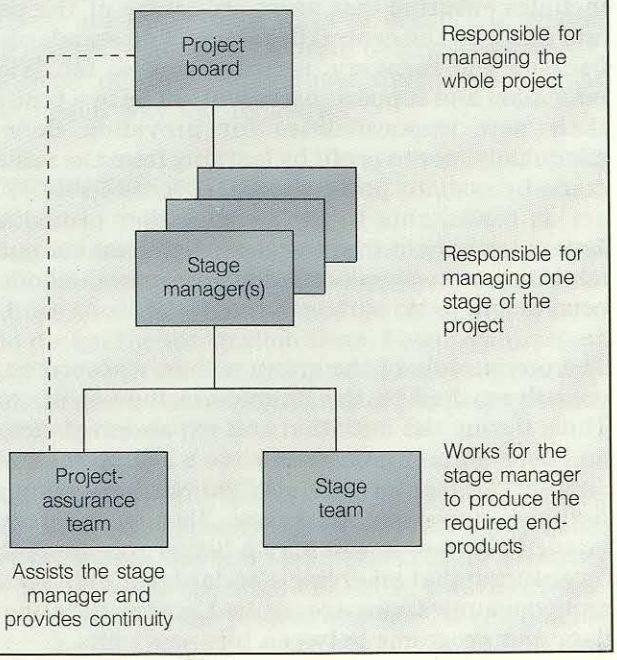
(Source: Sloan Management Review, Summer 1988.)

however, have little idea of how to involve line managers in this process.

One approach that has been successfully adopted by a large European retailer is illustrated in Figure 2.6. It has the following elements:

— *A project board*, consisting of a senior systems representative, a senior user, and a business representative. The responsibilities of the board are to authorise, review, and sign

Figure 2.6 A European retailer has created a framework for user-led projects



off each 'stage' of the project. This includes appointing the stage managers, approving all plans, and appointing the project-assurance team.

- *A project-assurance team*, consisting of a business-assurance coordinator, a technical-assurance coordinator, and a user-assurance coordinator. Appointed by the project board, they work for the stage manager(s) for the life of the project. Their responsibilities are to help prepare plans, monitor costs against budget, control change requests, and ensure that the appropriate development standards are applied.

- *Stage managers*, who are appointed for each stage by the project board. For stages that are heavily user-oriented, such as system specification or installation, a suitable user is appointed as stage manager. For the technical stages, the stage manager is normally a systems specialist.

- *Stage teams*, appointed by the stage manager, and comprising user and systems staff who report to the stage manager on all project-related matters, but to their line manager on all other matters.

The retail company that has adopted this project framework has noted several benefits from working in this way. Stage managers have been actively involved in ensuring that the user community is sufficiently committed to undertake a project. For their part, users have been prepared to make a much greater commitment of time and effort, and have assumed responsibility for ensuring that the systems provide all the appropriate facilities. As a result, better relationships and understanding have developed between business staff and systems staff.

Education and user support

The most crucial role of the education and user-support group is to educate and train users in all aspects of developing computer systems, from selection through to implementation. Its role includes ensuring that users are aware of the policy guidelines laid down by the central department on standards and protocols, back-up and recovery, security, and so on. Without this vital education and support, users will not be in a position to carry out their new responsibilities for providing their own systems adequately, nor to profit by learning from the mistakes previously made by systems professionals. A second role of the group is to act as consultants to the users, either providing support and assistance to help them acquire their own computer systems, or advising on the appointment of competent outside consultants or contractors to do so.

The precise role of the group will vary according to the stage of growth reached by the business in the use of each technology. Thus, during the initiation and expansion stages, the education and user-support group will have a limited role, generating ideas and enthusiasm for new applications, providing education, and perhaps, supplying packages. During the formalisation and maturity stages, it will play a bigger role, imposing some order by ensuring that emerging standards for data security, integrity, and communications are applied, and facilitating the sharing of data and programs between business units.

The key to the success of this group is the personality of the user-support staff. The more successful user-support services tend to be staffed by user-sympathetic and solution-oriented people, rather than by those who are more interested in technical details. Obviously, the user-oriented support person needs to be sufficiently technically competent to advise on the right technical solution or software package as well, but the emphasis has to be on business fit rather than technical elegance.

To add value, user-support personnel must be very well acquainted with the business area. We have found that the most successful user-support groups are those that are distributed to the user area, rather than located within the systems department, regardless of whether management control is devolved. At Ahold, a Dutch supermarket chain, the user-support staff have become so vital to the business that many are recruited into line management positions, where they continue to help users exploit computer systems.

Systems maintenance

The way in which systems maintenance is organised can have a very marked effect on motivation and staff morale. This is a topic that deserves special attention, so it is explored in greater detail both in this chapter (beginning on page 22) and in Chapter 3 (see page 52). First, however, we consider the organisation of testing.

ORGANISING TESTING

For the purpose of organising project teams, most systems departments distinguish between two kinds of testing: module and integration testing, which is concerned with testing the behaviour of the components of a system, and system testing, which is concerned with the functionality of a system as a whole. Module and integration testing is normally carried out within the project team, with some differences in the ways in which responsibilities are assigned. There are three main ways of organising system testing — by the project team, by a separate testing department, and by a joint team of users, operations staff, and systems developers — and each way has its merits.

MODULE AND INTEGRATION TESTING

Knowledge of the detailed system and program designs is required to develop module and integration tests. Generally, the specifiers of program modules should design the module tests, and system designers should design the integration tests. Few organisations distinguish clearly between module and integration tests, possibly because both activities are the responsibility of the project team, and do not involve users.

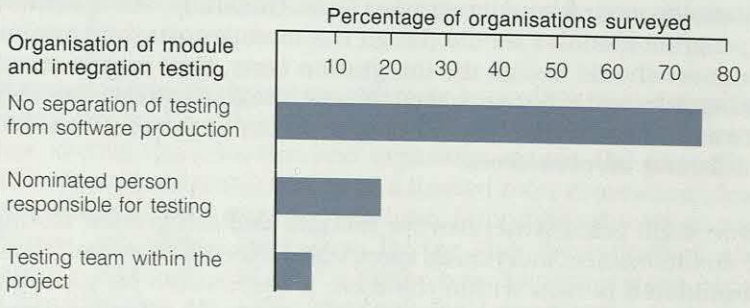
Three main team structures for module and integration testing are commonplace: individuals specify and execute their own tests; a nominated person within the team is responsible for ensuring that all tests are carried out to specified standards; a distinct team, working under the control of the project manager, is responsible for testing.

The first of these is probably the most popular. In three-quarters of the companies we have surveyed, there was no attempt to separate testing from production within the development team (see Figure 2.7). The main problem with allowing individual programmers to test their own work is the inconsistency in quality that usually results. Some programmers are undoubtedly good at testing their own modules; others, possibly because of inexperience or lack of training, perform virtually no systematic testing. Since a poorly tested module in a critical part of a system can cause considerable delays and expense during system testing, uncontrolled individual module testing is not cost-effective.

Module testing is difficult to do well. It can be very tedious for a programmer to check that each line of code and all true and false results of decision statements have been tested by a sample of test cases. It is equally, if not more, difficult for a programmer who did not write the code to carry out these tests. It is for this reason that module testing tends to be done by the programmer who wrote the code, and it is probably not effective in terms of cost or staff morale to introduce an independent module-testing team. However, the use of dynamic-analysis tools (which are discussed in Chapter 5) can remove most of the tedium from module testing, and also provide management with a printed record of the extent of the tests. At the module level, it therefore seems practical to leave the responsibility for testing with the programmer, but to provide the tools that make the job easier and that give management greater project control.

A single team member, or on larger projects, a small team, should be responsible for ensuring that module and integration testing is carried out to specified standards, even if the actual testing is carried out by the programmers themselves. At least one of the designers of the system should be part of this team. Allocating specific responsibility for testing within a project team in this way should not impose additional costs on a project. In fact, the total development cost should fall, since more reliable modules are likely to be produced, leading to a reduction in the cost of rework as a result of errors discovered during system testing and live operation.

Figure 2.7 Most organisations make no attempt to separate module and integration testing from software production



(Source: Butler Cox survey of PEP members)

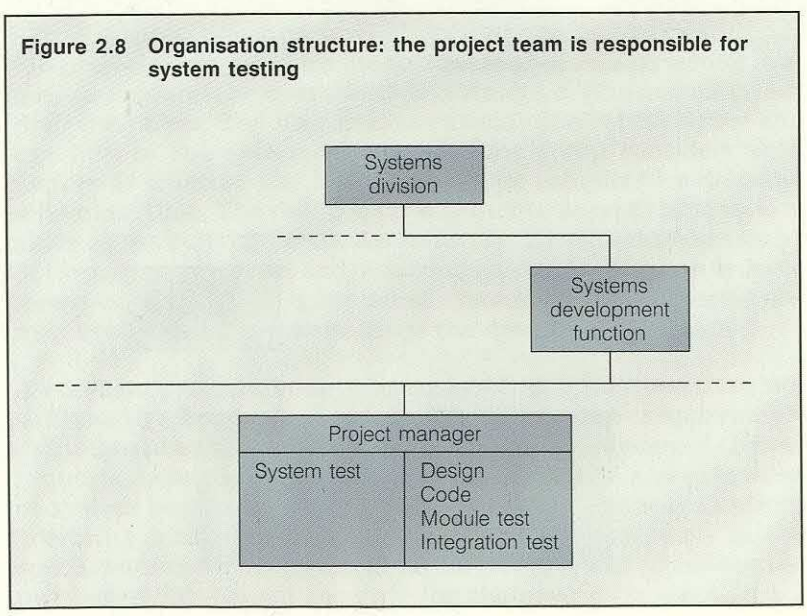
SYSTEM TESTING BY THE PROJECT TEAM

For system testing, the most common organisation structure is for the project team to take responsibility. This structure, illustrated in Figure 2.8, has been adopted by about half of the companies we surveyed. System testing is entirely under the control of the project manager, and each project team defines its own approach to system testing. Some project teams may set up a small system-testing team; others may assign the responsibility for system testing to an analyst or a designer. Some companies have a policy on how projects should structure their teams; others allow each team to define its own structure. Many companies recognise that there are benefits to be gained from separating testing from development, and set up testing teams within the project team. A few ensure that the two activities remain separate by allocating a different computer for testing.

Occasionally, the quality-assurance group is used as an independent authority to carry out random tests on the software during the main-build phase. The quality-assurance group can play a major role in defining and monitoring how software should be tested, but it is unlikely to have the resources to become closely involved in the design of all the systems under development. As a testing technique, random tests are unlikely to provide a useful measurement of each system's quality, and as a means of finding errors, they should not be used as an alternative to a properly defined series of tests.

The main benefits of placing full control of system testing with the project team are reduced costs and ease of management. In the short term, it is cheaper to allow each project team to have full control over its own testing than to incur the additional costs of a separate group of people, who have to understand the users' requirements and liaise with the project team. From the manager's point of view, assigning total responsibility for testing to the project team relieves him of the need to devote any effort to consideration of system testing.

Figure 2.8 Organisation structure: the project team is responsible for system testing



The main disadvantages are also related to cost and management. If a company has several similar projects under development at any one time, it should be possible to reduce costs by developing a common testing environment, or by purchasing a set of software testing tools that can be used on all projects. Making system testing the responsibility of individual project teams also has the disadvantage that systems development management has no independent measures of the quality of a system. Whether this is a problem will depend on how reliable the system is required to be, and how skilled in system testing the members of the project team are.

This structure for system testing can be cost-effective in a company that develops relatively small applications, with a requirement for average reliability. Companies choosing to adopt this structure should ensure that one person within the project team is given specific responsibility for system testing, and that this person has expertise in the design of system tests.

SYSTEM TESTING BY A SEPARATE DEPARTMENT

In this organisation structure, illustrated in Figure 2.9, a separate department carries out the system tests on most of the systems developed by project teams. Three of the companies in our survey had a separate system-testing department. These same three also had the most fully developed procedures for testing, and collected statistics on the effectiveness and cost of their testing. One additional company, however, had recently disbanded its system-testing department because it found that the project teams became careless in their own testing, relying on the system-testing team to find errors. The system-testing team then blamed the developers for delivering poor-quality work. The general lack of respect between the two groups led to an overall reduction in quality and productivity.

The very different experience of the International Stock Exchange is summarised in Figure 2.10. It created a separate testing group

Figure 2.9 Organisation structure: testing is the responsibility of a separate department

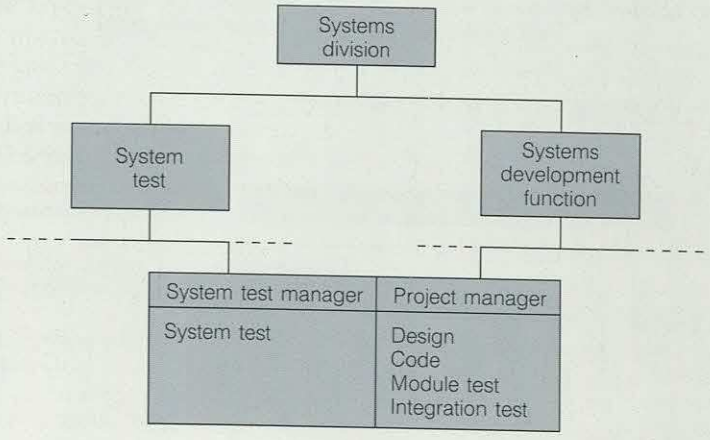


Figure 2.10 The International Stock Exchange set up a separate testing group and achieved highly reliable systems

In the period leading up to the deregulation of the UK financial services section in October 1986, the International Stock Exchange was involved in the development of some large systems that were highly visible to the public, and were essential to the future operation of the Stock Exchange. The systems department decided to set up a separate system testing group for the specific purpose of minimising the risk of implementing systems that might fail. Apart from some well publicised problems in the first hours of operational use, the systems have performed with a very high degree of reliability, and the investment in setting up a system testing group was considered to be justified.

Some of the factors considered in setting up a system testing group were:

- *Independence:* The testing group must be able to retain an objective view of the development, and should not be subject to pressure to cut short testing to bring the project in on time. The group should, however, act as advisers to the project manager, and should not have the final say on when a project is complete.
- *Terms of reference:* Terms of reference must limit the scope of the testing, because there can be a tendency for testing to expand to fill the time available.
- *Managerial support:* Senior systems management support is essential to resist pressures that may arise from the development team to limit system testing. To gain this support, management must be supplied with information on the progress of testing.
- *Marketing:* The role of the testing group should be marketed internally. The Stock Exchange produced a brochure describing the facilities offered by the group.
- *Cost:* An independent testing group is expensive. About 5 per cent of the Stock Exchange's systems development staff were in the testing group. The group also needed its own computer systems for building test environments.

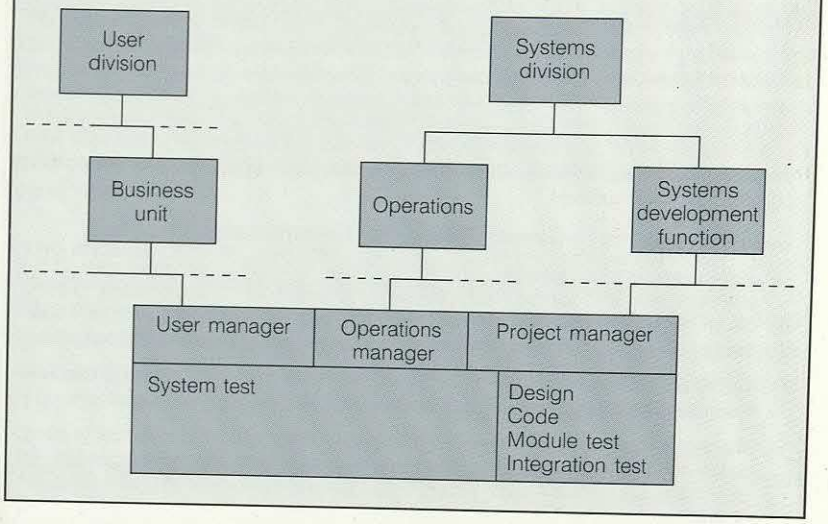
in preparation for testing the systems that were being developed for the deregulation of the UK financial services sector in October 1986. It was an expensive investment, but in this case, it did result in the development of very reliable systems. The benefits of a system-testing department are that staff develop expertise in testing techniques and that investments can be made in test tools, simulators, and databases, which may be difficult to justify on a project-by-project basis.

SYSTEM TESTING BY A JOINT TEAM

In this organisation structure, illustrated overleaf in Figure 2.11, the project team provides the technical expertise in testing, but user groups and the operations department define and carry out their own tests. The user groups examine the functionality and usability of the system. The operations group considers such factors as whether the batch run can be completed within the scheduled time. The decision on whether to accept the system is made on the basis of these measurements. This structure has many of the same advantages and disadvantages as the first one. It does, however, allow at least one set of system tests to be carried out by a group that is separate from the development team.

A common problem in developing systems is that users are not sufficiently involved, particularly during the requirements definition phase. This can lead to an excessive number of change requests. Involving the users in the specification and execution of system tests encourages them to examine the specifications critically, which should help to ensure that any faults in the specifications are corrected at an early stage, and to reduce the number of subsequent requests for changes.

Figure 2.11 **Organisation structure: the project team shares responsibility for system testing with other groups**



There is no need for a system tester to know how to design or program software, but testing requires particular skills for which training and experience are necessary. User groups should therefore include at least one specialist testing adviser, not necessarily full-time, if they are to carry out effective tests.

ORGANISING MAINTENANCE

Maintenance accounts for a huge and growing proportion of programming and analysis effort. In some installations, it can be as high as 65 per cent. The efficient organisation of this type of work requires a different approach from that of systems development. The focus of the systems maintenance group is to support the daily requirements of the existing business, with responsiveness and service delivery as foremost considerations.

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 is often better amongst staff who work in a separate maintenance function. Arranging for some or all of the maintenance workload to be undertaken outside the systems development department also has some merit.

THE SCOPE OF MAINTENANCE

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.

There are, in fact, three broad categories of maintenance, which are summarised in Figure 2.12. The first is *corrective*

Figure 2.12 **There are three broad categories of maintenance**

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

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 one of our own surveys, 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.

THE MERITS OF A SEPARATE MAINTENANCE DEPARTMENT

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.

Our own survey revealed that staff morale and motivation were significantly higher when maintenance was set up as a separate function. This view is supported by Joseph Izzo, from a California-based management group that specialises in improving company efficiency. He suggests that systems departments have two missions. The first is to maintain today's systems and to provide as fast a service to the users as possible. This, he has found, is seldom achieved. The second mission is to work on tomorrow's systems. However, when the schedule on a 'today' project slips, people are inevitably taken away from a 'tomorrow' project. The most efficient way to organise systems maintenance work, according to Izzo, is to organise it as a separate group, and to concentrate on measures to improve service levels.

The first step is to set up two teams — one for product support, and one to deal with 'intermediate' requests. The product-support team deals with requests likely to take less than 160 hours of effort. A separate project-based group deals with 'intermediate'

requests — those estimated to take between one month and one year of effort. Requests that are estimated to take more than one year of effort are deemed to be development rather than maintenance projects.

The product support group is staffed by senior people who know how to work with users. They deal with maintenance requests as they arise. No priorities are set, but requests must be authorised by a line manager. In companies that have installed such a group, turnaround is significantly improved, and the systems department's credibility is improved in the user community. The key to success, Izzo found, is to appoint a service-oriented manager to run the group. Contrary to normal expectations, he finds that after about a year, staff actually want to join the team, because its members are regarded as 'heroes' by the user community. Businesses that have set up such groups confirm this experience.

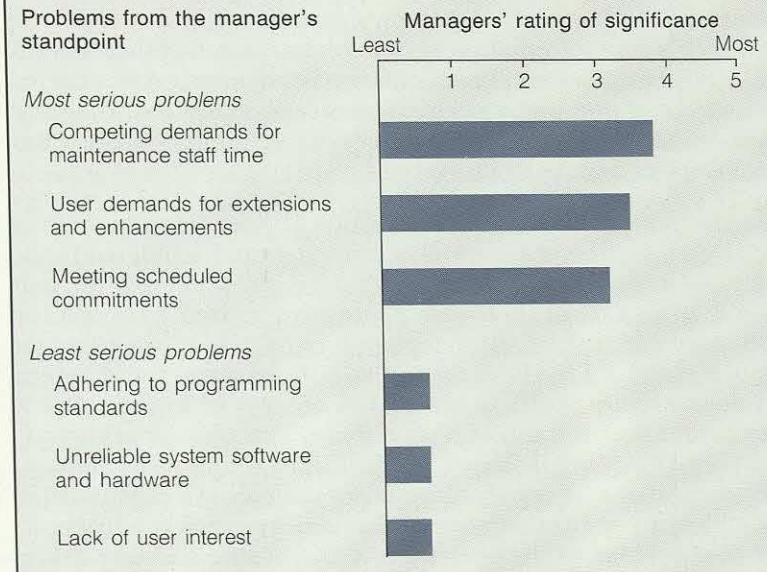
The intermediate group is run by a project manager, and the work is costed and scheduled as for any new development work. These projects are all authorised and priorities are set by senior line managers. Because the team is allowed to concentrate on one job, the typical pattern in installations organising maintenance in this way is to meet 80 to 90 per cent of the scheduled deadlines.

There are two significant points about this way of organising maintenance work. The first is that the maintenance group should be seen as an important part of the systems department. This means it should be led by a high-profile manager, and should be staffed by service-oriented personnel. The second is that line managers should take responsibility for the maintenance function — requesting, authorising, and setting priorities for the work. Motivation and productivity will both improve as a result.

No similar characteristics were evident in our own, somewhat smaller survey. Of the companies 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 2.13). 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 2.13 Competing demands for maintenance staff time is the most significant problem



(Source: Butler Cox survey of PEP members)

THE MERITS OF 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.

One source is that of systems users themselves. Advances in fourth-generation languages are making this 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.

An alternative is to contract maintenance work to a third party. This 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.

The FI Group, a major systems and software house based in the United Kingdom, is a good example of a contractor who undertakes third-party software maintenance work. Maintenance is contracted out to third parties for numerous reasons. 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 a local government authority 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 concentrated on developing replacement systems.

While the possibility of contracting out at least some part of a company's maintenance work is becoming more feasible and can, clearly, be a highly successful alternative, most companies 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.

IDENTIFYING AND RESOLVING COMMON ORGANISATIONAL PROBLEMS

The following problems were frequently mentioned by systems managers during our research. The suggested solutions are summarised in Figure 2.14. They fall naturally into two groups: those that have an impact on the systems department's effectiveness, and those that affect its efficiency.

PROBLEMS HAVING AN IMPACT ON EFFICIENCY

Four common problems are low productivity, rising development costs, high staff turnover, and poor-quality systems development by users.

Low productivity: Organisational changes can help with productivity. They include using smaller teams (no more than six), introducing more flexible jobs (both discussed in Chapter 3), and shedding some layers of management.

Rising development costs: These are frequently caused by poor management of the product-definition and construction stages. Appointing users to manage projects will usually result in a better definition of the project objectives, tighter control over project enhancements, and better marshalling of user-department resources during implementation. The result should be better control over costs.

High staff turnover/lack of a suitable career structure: There are many reasons for high staff turnover that are not within the scope of an organisational change to cure. A surprisingly consistent body of research, however, has identified lack of job interest and lack of a suitable career structure as prime causes of discontent. (Providing a flexible career structure is discussed in Chapter 3.)

Figure 2.14 Common organisational problems can be resolved with reference to the approach suggested in this report

Organisational problems		Suggested actions
Problems that affect the systems department's effectiveness	Failed attempts to decentralise	— Train staff fully for their new roles prior to decentralisation — Manage careers centrally to retain broad opportunities — Introduce job rotation to prevent isolation
	Low 'presence' in the business	— Appoint business manager(s) to foster relationships with senior line managers — Ensure that organisational management style aligns with group management style — Devolve some user-support staff to business areas
	Priorities set too low	— Ensure that organisational management style aligns with group management style
	No overall systems planning	— Clarify the respective responsibilities of user and systems staff — Introduce a planning group to develop and enforce a coherent software infrastructure
	User confusion over communications	— Re-align systems development to form business groups and channel all user communications through the business-group manager
Problems that affect the systems department's efficiency	Low productivity	— Use smaller teams (no more than six people) — Widen the scope of jobs — Shed some management layers
	Rising development costs	— Appoint users to manage projects
	High staff turnover/lack of suitable career structure	— Widen the scope of jobs — Introduce and make explicit a lateral career structure — Shed some management layers
	Development of poor-quality systems by users	— Set up an education and user-support group staffed with business-oriented personnel

It is interesting to note that the highest turnover rates usually occur amongst the newest recruits. These people have often been recruited from a university background in which they enjoyed considerable autonomy and status. Fitting into a structure with a steep reporting hierarchy can be daunting and demoralising. In organisations with fewer layers and flatter structures, turnover is usually considerably lower.

Development of poor-quality systems by users: Users should be encouraged to experiment with new technology in order to learn how to apply it appropriately to their business area. As we have seen, it is a mistake to try to control their efforts too soon. Often, the reluctance of users to consult systems staff is a legacy of poor previous service, lack of interpersonal skills in systems staff, and a poor appreciation of real business problems by the systems department. The best way to foster a better working partnership is

to set up an education and user-support group, to be located in the business area and staffed by user-sympathetic personnel. Their role will be to encourage and guide the users' efforts. This type of role is best performed by people with a bias towards business rather than technical solutions.

PROBLEMS HAVING AN IMPACT ON EFFECTIVENESS

Five common problems concern decentralisation, low presence, priorities set too low down, absence of infrastructure planning, and confusion over communications.

Failed attempts to decentralise: A common reason for the failure of attempts to move centralised staff into a business area is lack of preparation. There are usually cultural barriers to break down for both systems staff and business-unit staff. One company observed that "when you put a user together with a systems designer, what you get at first is nothing like either of them had in mind . . . then they work on it". Often, the systems staff are cut off from their colleagues and unable to integrate successfully with their new business partners. There are two important prerequisites before this type of re-organisation can take place. One is to train staff fully for their new roles, prior to dispersing them. The other is to manage careers, so that staff in small, decentralised units are given the same opportunities to move into different posts as their 'centralised' colleagues. Frequent job rotation can also prevent feelings of isolation.

Low 'presence' in the business: It is quite possible for a business to be more critically dependent on systems than line management recognises. If a particular systems department believes this to be the case, low presence is clearly a problem. It can often mean that business management is devolved, while systems development remains centralised and is thus seen to be remote and irrelevant to the business. In these circumstances, there is usually a wide cultural gap to overcome as well, and raising the profile of the department will inevitably be a slow process.

The most successful way to increase the presence of the department is to concentrate on a growth area such as sales, marketing, or production, and to appoint a business manager to foster a more positive relationship with line management. To build on the relationship and to ensure that systems are produced that the department actually wants, some user-support staff could subsequently be devolved to the business area.

Priorities set too low down: This is really a variation of the previous problem, and usually implies that line management does not recognise the value of computer systems to its business area. Priorities, however, should not be decided by the systems department. It is senior line management's responsibility to decide how much money to invest in systems, and what the business priorities are for development. This can be achieved only by senior systems management fostering a partnership with senior business executives, and encouraging them to agree on a systems strategy and priorities. This is more easily accomplished if the management style of systems development is closely aligned with the group management style.

Absence of infrastructure planning: This is a common problem in organisations that have neither laid down clear policies and guidelines governing the respective responsibilities of users and systems staff, nor defined a common systems architecture within which coherent planning can take place. The first priority is to establish the principle that users should decide what systems are developed, and that systems staff should provide the standards required to enable applications to be shared by business units if needed. A systems-planning group can then be created to develop and enforce the standards necessary to safeguard flexibility, compatibility, and consistency in systems development, through a common software infrastructure.

User confusion over communications: The proliferation of various 'information centres' and user-support groups, as well as multiple development centres, can be confusing for user departments. To ensure that the most appropriate service is always offered, it is essential to provide the user with a single point of communication. This should be the business-group manager.

We have seen, in this chapter, that while there is no direct correlation between productivity levels and the structure of the systems department, there are, nevertheless, many things of an organisational nature that systems managers can do to provide a better service to the business. Organisational design is not, however, just about structural form. The behaviour of individuals in an organisation, and therefore their performance, is influenced by a variety of factors, which are discussed in detail in the next chapter.

Chapter 3

Improving staff motivation

Because the largest single cost element in most systems development departments is staff, improving the productivity of development staff is a critical matter. Motivation itself is an important element in this. Happily, the nature of the work itself presents no obstacle: systems development work is intrinsically highly motivating. Having said that, it is commonplace for systems development staff to expect more satisfaction from their work than they actually get. In other words, the reality of motivation falls short of potential.

To bridge the gap, several options are open to managers. Chief amongst these are broadening the scope of the work, job rotation, flexible career structuring, goal-setting and feedback, performance-related pay, and job fitting. These are opportunities that apply at the level of the individual. Further opportunities for improving motivation exist at the level of the team. They are to do with team size and composition, and with the role and style of the team leader.

A closely related topic, but one which is best handled in its own right, is that of the motivation of staff engaged in maintenance — given the sheer volume of maintenance that occurs in the typical systems department today.

THE HIGH MOTIVATING POTENTIAL OF SYSTEMS DEVELOPMENT WORK

Systems development work can be highly motivating, particularly where an element of staff management is involved, but the motivating potential varies widely from job to job and between different companies.

The results of a survey carried out in the United States have been used to calculate a measure called the Motivating Potential Score (MPS) for a range of occupations. The MPS derives from the Job-Diagnostic Survey technique originally developed by two American researchers, J Richard Hackman and Greg R Oldham. According to Hackman and Oldham, the motivating potential of a job is derived from five key measurable job dimensions: skill variety, task identity, task significance, personal responsibility, and work feedback. An equally weighted combination of the first three dimensions is used to provide a measure of the perceived importance of the job.

Skill variety is the extent to which the job calls for different skills and talents. Task identity measures the completeness or wholeness of the work involved in the job. Task significance is to do with the job's impact on other people. The fourth dimension measures the job holder's perception of personal responsibility for the work in terms of freedom, independence, and discretion

in determining job procedures. The fifth dimension, work feedback, is concerned with the job holder's knowledge of the outcome or effectiveness of the work. Both the extent and the timeliness of feedback are important.

Each of the dimensions is rated on a scale of 1 (low) to 7 (high), and the MPS is defined as the product of the perceived importance of the job (an equally weighted combination of the first three dimensions), the personal responsibility of the job holder for the work done, and work feedback. MPS measures can therefore range from 1 to 343.

Figure 3.1 shows a sample list of occupations, together with the MPS for each one. The MPS of 154 for data processing professionals places the occupation at about the same level as managerial and other professions, and well ahead of other occupations in terms of motivating potential.

THE MOTIVATING POTENTIAL OF DIFFERENT SYSTEMS DEVELOPMENT JOBS

This high overall score for the motivating potential of data processing work does, however, hide wide variations between individual jobs within any one organisation, and across organisations. Figure 3.2, overleaf, shows the MPSs for five systems development jobs and the average for all systems development jobs, and compares them with the MPSs of two further categories of job — other professional staff and other managers. Of the five systems development jobs, data processing management has the highest MPS, at 199, and maintenance the lowest, at 106. Programming scores 137, while analysts and analyst/programmers are virtually the same at 154 and 152 respectively. Of the dimensions that make up the overall MPS for each of the five data processing jobs, work feedback scores lowest in all cases except one.

In our own survey of motivating factors, carried out amongst 600 data processing professionals in seven businesses, we undertook

Figure 3.1 The motivating potential of data processing is quite high

Job category	MPS*
Other managers	156
DP professionals	154
Other professionals	154
Service	152
Sales	146
Construction	141
Machine trades	136
Bench work	110
Clerical	106
Processing	105

*MPS: Motivating Potential Score, a measure of the motivating potential of jobs. The higher the score, the more motivating the job.

(Source: Couger, J D and Zawacki, R A. *Motivating and managing computer personnel.* Chichester: Wiley, 1980.)

Figure 3.2 There are wide variations in the motivating potential of jobs within systems development

Job dimension	Systems development jobs						Other jobs	
	Analysts	Analyst/ Pro-grammers	Pro-grammers	Main-tenance [1]	Managers	All staff	Pro-fessionals	Mana-gers
Skill variety [2]	5.55	5.45	5.23	4.80	6.16	5.41	5.36	5.57
Task identity [2]	5.37	5.29	5.00	4.30	5.80	5.21	5.06	4.72
Task significance [2]	5.75	5.72	5.46	5.40	6.30	5.61	5.62	5.81
Responsibility for work done	5.31	5.49	5.13	4.70	6.10	5.29	5.35	5.73
Knowledge of outcome of work (feedback)	5.20	5.05	5.10	4.30	5.25	5.13	5.08	5.15
Motivating Potential Score (MPS)	154	152	137	106	199	154	154	156

Notes: 1 Data relates to staff who spend more than 80 per cent of their time on maintenance work.
2 The average of the rating for each of these dimensions forms the rating for the importance of the job.
3 MPS is calculated by multiplying the average rating of the first three dimensions by the rating of the last two dimensions.

(Source: Couger, J D and Zawacki, R A. *Motivating and managing computer personnel.* Chichester: Wiley, 1980, and Couger, J D and Colter, M A. *Improved productivity through motivation.* Prentice Hall, 1985.)

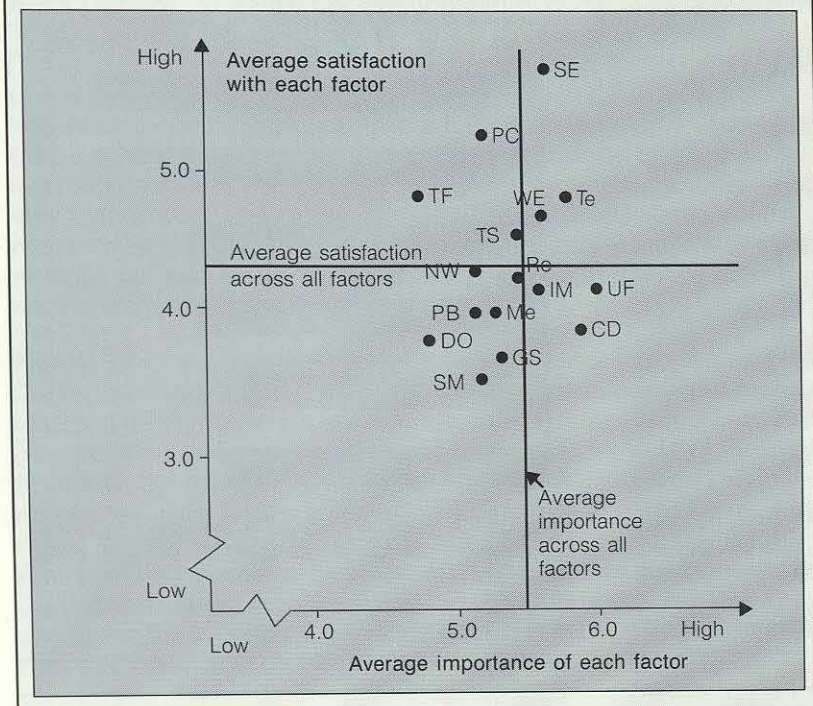
a similar investigation to the one summarised in Figure 3.2. Our survey respondents also quantified their responses using a seven-point scale, which we were able to reconcile with the points-scoring method used in the American surveys. We refer to the measures of motivation derived from our own survey as Job Motivation Scores (JMSs) to distinguish them from the MPSs.

In our survey, we asked for two sets of responses to the job-diagnostic survey questions. One set measured the importance that respondents attributed to the dimensions in affecting their ability to work well; the other set measured the level of satisfaction that they attributed to each job dimension in the context of their working environment. There was little correlation between the two. Most of the dimensions were rated as being quite important, scoring between five and six (out of seven). Satisfaction with these factors was, in general, not rated quite as highly (about four to five), and there was a wide spread between the different factors. Certain divisions stood out as being relatively important but satisfaction was relatively low — technology, career development, user factors, and, to a lesser extent, relationships with immediate managers. On the other hand, team factors and personal circumstances scored relatively high on satisfaction but lower on importance. These satisfaction/importance ratings are shown in Figure 3.3.

We found both similarities and differences between the JMS and MPS results. There was considerable agreement between the surveys over the large difference in the motivating potential of jobs within data processing. Figure 3.4, on page 34, shows the JMSs of the six jobs that we measured. Both results suggest that the motivating potential of jobs rises through the ranks from programmers' jobs to systems development managers' jobs.

The dimensions rated as most important in our survey were task identity, skill variety, and responsibility for the work done. Feedback about the results of the work done was rated as less

Figure 3.3 There is little correlation between the average ratings of importance and satisfaction with respect to the factors affecting development productivity



Key to factors:

CD Career-development opportunities	Me Methods	SM Senior management relationships
DO Departmental organisation	NW Nature of work	TF Team factors
GS Goal setting	PB Pay and benefits	Te Technology
IM Immediate manager relationship	PC Personal/family circumstances	TS Training and skills
	Re Recognition	UF User factors
	SE Security of employment	WE Work environment

(Source: Butler Cox survey of PEP members)

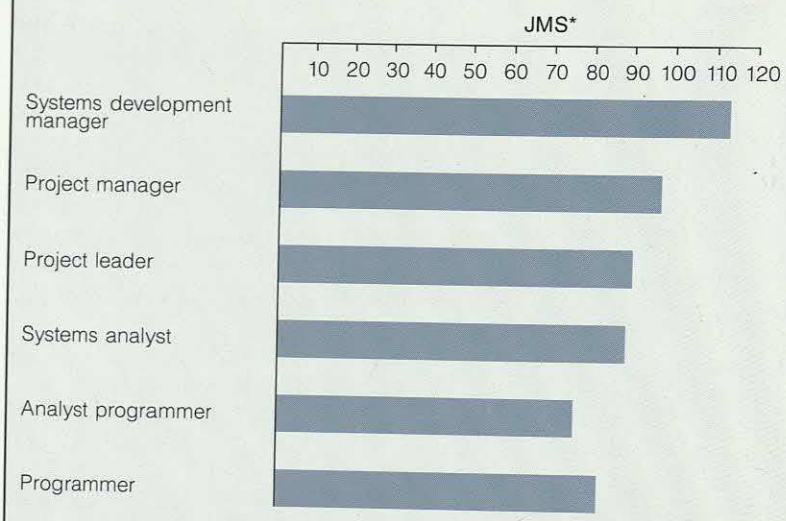
important, and the level of satisfaction with the feedback received was lower than the level of satisfaction with the other dimensions. We also asked about the importance of, and satisfaction with, feedback from the respondents' immediate managers. Here, importance was rated much higher than satisfaction than for any of the five job-motivation dimensions. This emphasises that systems development managers should be paying much greater attention to providing feedback about an individual's performance.

THE MOTIVATING POTENTIAL OF SPECIFIC WORK FACTORS

Our survey also revealed that two further factors — support responsibility and work variety — have a marked effect on the motivating potential of a job.

Support responsibility: Responsibility for directly supporting the user community is a positive motivating factor in systems development work, as is responsibility for directly supporting an aspect of the hardware or software. These findings are apparent both from Figure 3.5, overleaf, and from a comparison of the motivating scores returned by the different businesses represented

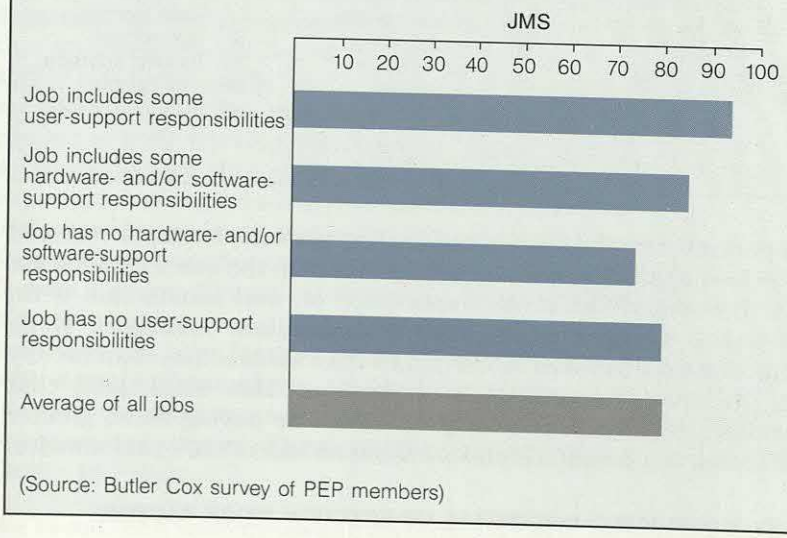
Figure 3.4 The motivating potential of jobs rises through the ranks, from programmer to systems development manager



*JMS is the product of three ratings, each of which is in the range 1 (low) to high (7). The three are importance of the job, responsibility for the work done, and knowledge of the outcome of the work.

(Source: Butler Cox survey of PEP members)

Figure 3.5 Job Motivation Scores (JMSs) vary according to the support responsibilities of the job



(Source: Butler Cox survey of PEP members)

in our survey. Figure 3.5 shows that, when user or technical support is included, a job has a greater motivating potential than when it is excluded.

One of the businesses in our survey reported a significantly higher JMS score for its systems development staff than the other six. We believe that this is due, in part, to the job-enlargement policy that this company has adopted. Its systems development staff are encouraged to become experts not only in systems development project work, but also in defined areas of software, hardware,

and user support. The consequence of the job-enlargement policy is to increase skill variety, task significance, and personal responsibility.

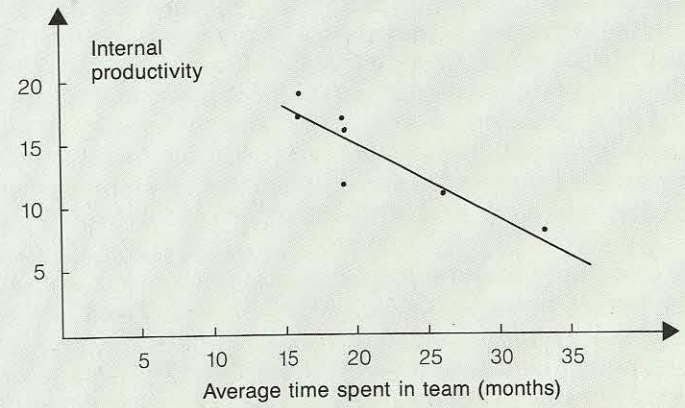
Work variety: After a time, any job can become mundane when it lacks variety. Greater work variety is a positive motivator. Apart from career development, which by nature introduces individuals to a changing pattern of work and responsibility, the most obvious way of introducing variety is through job rotation. Some businesses take a planned approach to job rotation precisely because of the benefits it can deliver. One, for instance, moves programmers into new teams every two to two-and-a-half years, and systems analysts every three to three-and-a-half years. It further increases work variety by providing its staff with opportunities to develop productivity aids.

Figure 3.6 compares the typical internal productivity for each of the businesses in our survey with the average time spent in project teams. The figure suggests that there is a relationship between productivity and the time spent in project teams — with internal productivity decreasing as the average time increases. This does not necessarily imply a causal relationship — both parameters could be influenced by project size, for example. The implied relationship is, however, consistent with the fact that projects of short duration are more manageable than long ones, and that they are better for avoiding the troughs in enthusiasm, drive, and vision that are often the consequence of prolonged project work.

IMPROVING THE MOTIVATION OF INDIVIDUALS

Although it is commonplace for systems development staff to expect more satisfaction from their work than they actually get, there are steps that companies can take to improve the situation. Motivating staff involves equipping them for the new roles that are emerging from the re-alignment of the systems function to the business, and taking positive measures to maximise the contribution of each individual. Six specific actions can be taken

Figure 3.6 Internal productivity* decreases as the average time spent in a project team increases



*See Chapter 6 — Productivity Index
(Source: Butler Cox survey of PEP members)

by systems development managers to ensure that each of their staff is making the greatest possible contribution — broadening the scope of jobs, introducing job rotation and flexible career structures, improving goal-setting and feedback, rewarding achievement with performance-related pay, and fitting jobs to people.

BROADENING THE SCOPE OF JOBS

The role of the systems development professional is becoming more diverse. To develop the types of systems that are being used to support business activities directly, it will be critical for systems staff to have some knowledge of the business. The greater involvement of users in the development process, using modern development tools, will require systems staff to have people-oriented skills. Jobs that are usually regarded as more 'technical', such as systems maintenance, are often performed far more successfully by service-oriented people. These trends point to the need for staff who are able to operate far more flexibly than has been the case in the past. The role of the education and user-support specialist, for example, requires technical programming skills, business knowledge, analytical ability, and interpersonal skills. These can be acquired only by enabling as many people as possible to operate in wider roles.

In response to these pressures, there is an increasing trend to move away from the traditional role of programmer, analyst, or business analyst, towards a more 'hybrid' role, such as an analyst programmer who uses modern development tools. There are two major advantages to widening the scope of systems development jobs. The first is that it creates a more flexible workforce, who are able to undertake a wider variety of work in response to changes in demand. The second is that the individual gains greater job satisfaction and is likely to be more productive as a consequence.

INTRODUCING JOB ROTATION

Moving staff between jobs is a useful way of broadening the skills of the individual, increasing job interest, and improving motivation and productivity. Philips, an electronics multinational company based in the Netherlands, provides positive encouragement for job rotation. Below management grades, staff are expected to spend no more than two or three years in the same place; at management level, this is extended to four years. The philosophy is one of encouraging change, fresh insight, and creativity, while trying to minimise 'ownership' of systems. Turnover of systems staff at Philips has been very low, at about 2 per cent a year.

SAAB, a Swedish car and aerospace manufacturer, does not expect systems people to stay in one job for more than a year, and finds that moving people around encourages them to have a more flexible outlook and gain a wider appreciation of the business. Frequently, these moves involve a transfer to a business-support group from a central development team, and occasionally, systems staff will move into line-management positions in the business area.

INTRODUCING A FLEXIBLE CAREER STRUCTURE

Managing careers is not high on the systems manager's list of priorities. This is indicated by the results of one of our own surveys of systems development staff (see Figure 3.7). It contrasts the views of systems development managers and their staff about the staff factors that they thought most important. Managers rated training and skills as most important, whereas the staff themselves rated career development, which included acquiring new skills and opportunities for promotion and advancement, uppermost.

Two further important issues make career planning a matter of urgent management attention. One is that providing practical career advancement for dispersed systems staff is one of the critical features of successful devolution to business units. The other is that a lack of suitable career options is one of the main reasons for staff attrition, according to a recent survey (see Figure 3.8, overleaf).

To provide a flexible career structure, systems development managers must recognise the wider roles that are emerging for the systems department, and provide more scope for 'lateral' development. An alternative to the traditional vertical career path, in which the main route to promotion is through the programmer/analyst/project leader path, is shown in Figure 3.9, on page 39. The main advantages of such lateral development paths are described in the following paragraphs.

Figure 3.7 There are marked discrepancies between the views of managers and their staff on the importance of various people-related factors to productivity

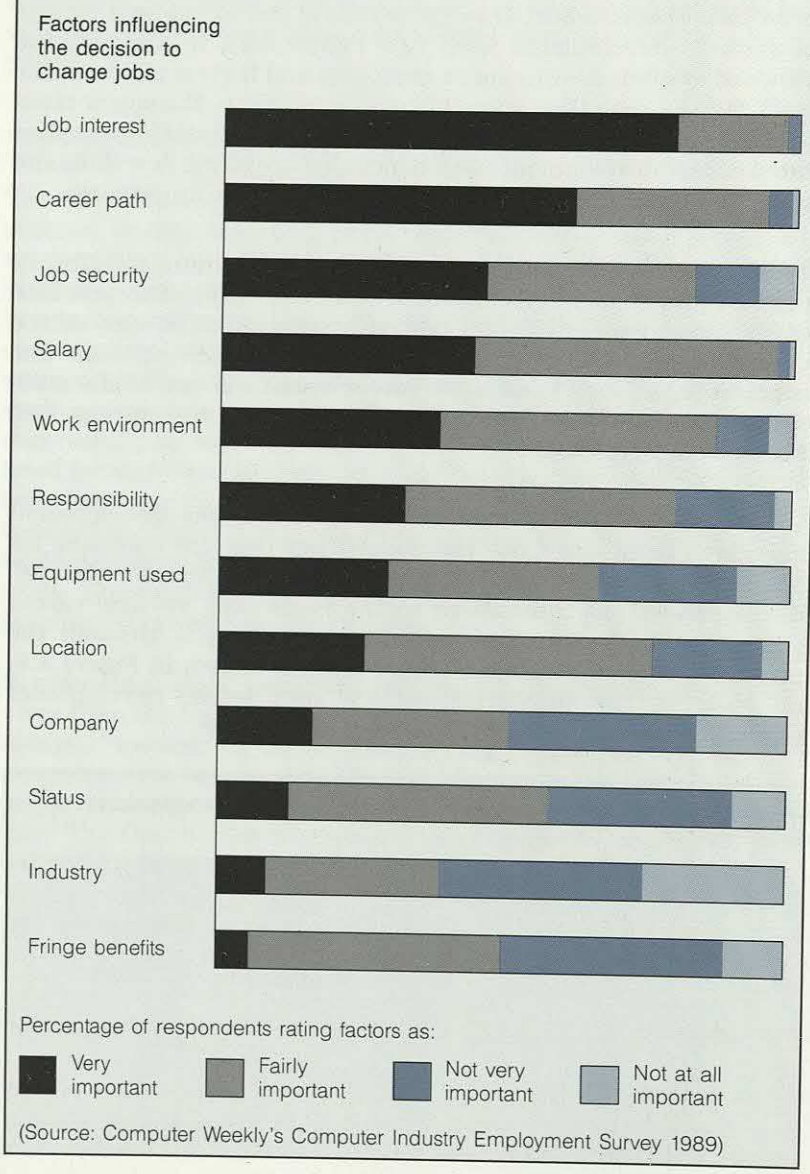
Based on frequency of mention by systems development managers in a telephone survey in which they were asked about the human factors that are important in achieving systems development productivity. For comparison, the importance rankings given by systems development staff in response to the questionnaires are also shown.

People or people-related factor affecting productivity	% of systems development managers mentioning the factor	Rank order given by development staff
Training and skills	85	8
Motivation	50	*
Project management/leadership	45	6
Support by technology	35	3
Recognition, achievement, personal worth	35	7
Office environment	30	5
Job factors	30	12
Team factors	30	15
Career development	20	2
User factors	15	1
Methods	15	10
Pay and benefits	15	13
Organisation structure and policies	15	16
Senior management/interpersonal communications	10	14
Goal setting and achievement	—	11
Security of employment	—	4
Personal/family circumstances	—	9

*Questionnaire respondents were not asked to rank motivation as a separate factor

(Source: Butler Cox surveys of PEP members)

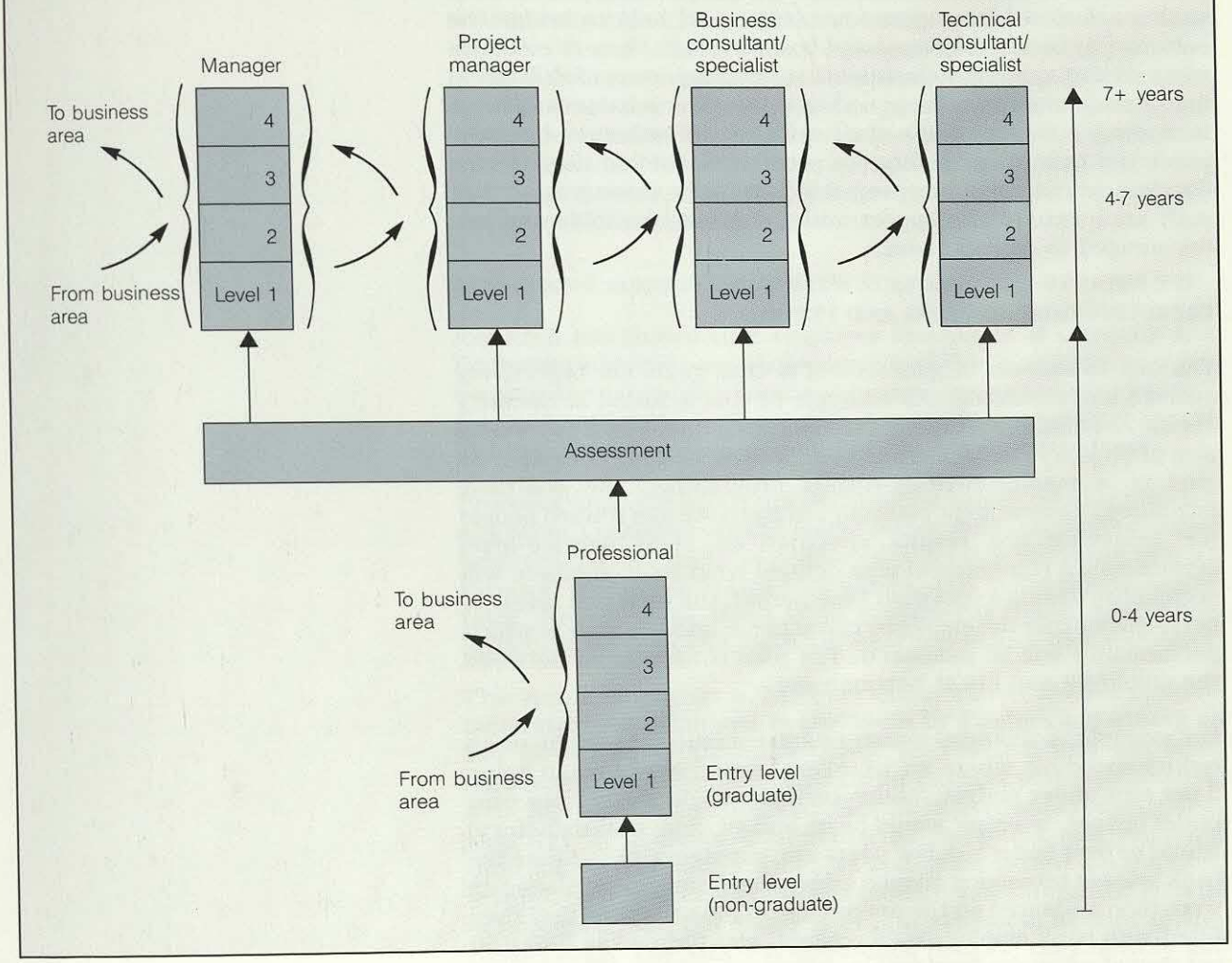
Figure 3.8 Job interest and career paths are rated the most important considerations by systems staff in changing jobs



(Source: Computer Weekly's Computer Industry Employment Survey 1989)

Alternative, but equal, career paths are provided for technical and non-technical staff. One result of the traditional career pattern is that programmers are moved into analyst/programmer and user-support roles regardless of whether they have the ability to deal with system users. Business and interpersonal skills are subordinated to technical skills, yet these are of equal importance to the systems department that is re-aligning itself to work more closely with its business partners. The key is to provide a structured framework of suitable career opportunities for everyone, recognising the potential value of both technical and non-technical skills. In most businesses, this will also require a change in the pattern of recruitment to test for the appropriate personality traits that will allow recruits to operate successfully in broader, business-oriented roles. In this structure, promotion to a senior level is possible for both technical and non-technical staff, without either having to move into a management post.

Figure 3.9 Lateral career-development paths are an alternative to vertical career progression



Line and project-management paths are explicitly provided. A major disadvantage of the typical promotion path, based on technical performance, is that it leads both to over-promoted technicians, who are unable to function adequately as managers, and to unfulfilled managers, whose real talent may be hidden behind average technical performance. In both cases, valuable expertise is misdirected, and inefficiency results. People with limited management ability, who may be excellent technicians, should be identified early – that is, within the first four years of their career – so that they can be provided with an equally satisfying non-management career. Likewise, people with management potential can be trained for the role early.

Lateral movements are planned and encouraged, both between major career paths, and to and from business areas. In this model, all staff spend up to four years gaining a wide knowledge of the profession. Lateral movements between different roles (maybe in different business groups) are encouraged, and all junior systems staff are seconded to business areas as a necessary part of learning the job. After four to seven years, the individual builds on basic skills and moves into a career path, with lateral movement still

possible between paths and to business areas. After seven years, an individual usually finds it extremely difficult to move across paths. Lateral movements into the business provide systems staff with much-needed business knowledge, and help to bridge the cultural gap between systems and business staff. Current evidence suggests that systems departments are net importers of skills from line-management functions; unless this inflow is balanced, there is a danger that systems staff will be demotivated by their perceived lack of suitability for promotion outside the systems department. Careers, however, still have to be managed, so that staff are aware of the opportunities that are available and are encouraged to exploit them.

IMPROVING GOAL-SETTING AND FEEDBACK

The key to success in goal-setting is that goals are objectively defined and measured. An example of this is found at Security Pacific Automation Company, the California-based data processing arm of Security Pacific Corporation, a bank holding company. As part of a management-by-results programme, the company introduced 'commitment planning', to motivate and reward people for achieving the results specified in their service-level agreements. A commitment plan defined what each employee will accomplish during a specified time period, the different levels of performance that the employee can achieve, and the ways in which performance will be measured. The plan is negotiated between the employee and his or her manager.

For example, a financial-management commitment might be to reduce spending, where an 'excellent' rating would mean being 5 per cent under budget, 'above average' would mean 3 per cent under budget, 'average' would be on budget, and 'unsatisfactory' would be over budget. A few years ago, management felt that the bank was not promoting enough employees from within. Managers were then measured on the percentage of job vacancies that they filled with bank employees. 'Excellent' was defined as filling 90 per cent of vacancies from within, 'above average' was 85 per cent, and so on.

IBM in Australia has staff-turnover objectives written into the performance objectives of every line manager from the chief executive down. In 1988, IBM Australia's actual rate of staff turnover was 8.9 per cent and its objective for 1989 was 6 per cent. Nothing could be more objective and measurable than that.

We have seen that jobs enabling the individual to obtain feedback from their work naturally and quickly are intrinsically more motivating than jobs in which feedback is delayed. The nature of most systems work is such that a system designer, for example, may not know for several months whether the design of a system is good or bad. Systems managers therefore need to find alternative ways of providing systematic and timely performance feedback to their staff. The easiest option is to link the feedback process to the annual appraisal scheme, as most organisations already have these schemes in place. However, at all but the most senior level, annual appraisals are probably not frequent enough. The objective should be to provide continuous feedback on performance and achievement.

The most satisfactory results, however, are achieved by moving the process of goal-setting and feedback outside the appraisal system altogether. One company, characterised by a high productivity rating, prepares work-assignment briefings to cover the next 10 to 20 days of work for programmers, and 30 to 40 days of work for systems analysts. Each work assignment is formally appraised upon completion, and the appraisal is sent to the human-resources manager. This work-assignment and appraisal procedure takes place outside the six-monthly and annual formal appraisals, which are concerned with training requirements, salary reviews, and career development.

REWARDING ACHIEVEMENT WITH PERFORMANCE-RELATED PAY

Research has shown that employee incentives, if carefully and fairly administered, can play a significant role in motivating staff, because they serve as a means of recognising and rewarding staff for work well done. If they are paid in a timely manner, they will also reinforce the goal-setting procedure discussed above.

Probably more job offers are declined for salary reasons than for any other. Companies in the public sector, with less flexible salary schemes, have usually experienced a greatly increased rate of turnover when their salaries fall significantly below private-sector rates. Nevertheless, there is no evidence that high pay, while attracting recruits, can motivate staff and reduce turnover rates.

The status of pay as a 'hygiene' factor rather than a positive motivator was examined in the 1960s by Frederick Herzberg at the University of Utah. Certainly, no research that has been conducted since has been able to prove otherwise. Cor Alberts, a divisional director from CAP Gemini in the Netherlands, put it this way at a recent conference on recruiting and retaining information technology staff: ''IT staff want to develop and they want to have new challenges and to learn new things. The growth is important and the salary is only a yardstick, at least in the Netherlands. The salary is questioned because they need to get enough in comparison to other people in the IT profession, or in the company itself.''

Professor Robert Zawacki, a human-resources consultant, has explained that the 'money' issue is not how much systems staff earn, but is concerned more with *equity vis-à-vis their perceived reference group* (our italics). In other words, salary is the device whereby employees measure the comparative value that different employers put on their skills. But Professor Zawacki has added: ''The foundation is the money, and the job is the home you put on that foundation, but once the foundation is solid, they [systems staff] want something else — meaningful work.'' The message is very clear — it is essential to pay market rates, but when staff have achieved parity with, or even an advantage over, their reference group, salary alone does not motivate them.

Bonus schemes have been used for years as a productivity incentive for blue-collar workers. There is now increasing evidence that performance-related pay is beginning to be used as a means of attracting senior managers in industries where competition for good people is fierce, and can now account for as much as 20 per cent of total remuneration. Where it is applied more widely,

however, performance-related pay does reduce staff-turnover rates. There are three basic types — share options, results-related bonuses (often based on profitability), and individual merit pay.

Share options: Share-based schemes (which are usually based on an option to purchase shares in the future at a predetermined price) are not normally directly performance-related because share values are subject to all kinds of market pressures. These types of schemes are not, of course, available to public-sector organisations, and neither are they under the direct control of the systems department. Nevertheless, where share-option schemes do exist, as many employees as possible should be encouraged to join because they tend to generate loyalty to the company.

Results-related bonus: These can be organised at group (for example, project-team), department, or company level. At the project-team level, performance/delivery objectives are set at the beginning of the project, and bonuses are paid at the end, to an agreed formula, if the objectives are met. Departmental and company-level bonuses are similar in concept, but are usually based on criteria such as profitability. They are typically awarded separately from normal salary reviews, depend on how well the company performs, and are paid annually. Such schemes are not common in non-profit-making organisations, where it may be more difficult to set performance objectives.

Individual merit pay: Merit pay is an individual award, paid to an agreed formula, for meeting pre-agreed standards of performance. It is highly motivating because it is directly related to individual performance. While it can be divisive, and it can be demotivating for the poor performer, it works well for the majority of employees.

Finally, companies have found four lessons to be of the greatest importance to the success of a performance-related pay scheme. The incentives should be paid in a timely manner, and be linked to short-term goals; the performance payment should be kept separate from normal salary payments; payments should not be awarded as a matter of course, but instead should be related to measurable performance objectives and not awarded for average results; the goals set for performance should be mutually agreed and realistic.

FITTING JOBS TO PEOPLE

In times of increasing staff shortages, greater flexibility can be obtained by fitting jobs to people rather than vice versa. This is the approach taken by Morgan Guaranty Trust Company of New York, an international bank with systems staff in Europe and America. Whenever a member of staff moves, it is seen as a chance to restructure a job, to take account of the new staff member's strengths and weaknesses. This does not mean a major re-organisation every time someone leaves. It is merely an adjustment to suit a particular situation, which frees Morgan Guaranty from the usually unsatisfactory attempts to recruit staff who match a rigid job specification.

OPTIMISING TEAM SIZE AND COMPOSITION

Because team working is commonplace in systems development, it is important to understand the factors that affect team

productivity. Team size and team composition are two of the most significant.

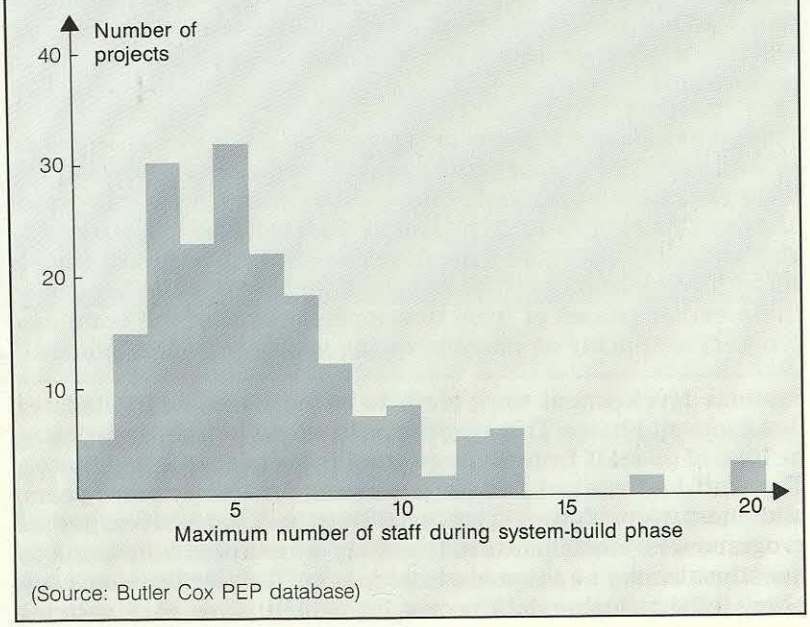
THE BENEFITS OF SMALL TEAMS

Most companies undertaking large systems development projects now usually break the project into a series of smaller, self-contained ones. One company we talked to estimated that one of its current projects would require between 100 and 140 work-years of effort. The project could be phased, but the first phase could not be reduced to fewer than 80 or 90 man-years. Furthermore, this phase had to be completed within nine months. To avoid the difficulties of managing such a large project, the company chose to split the project into separate sub-projects, each one to be undertaken by teams of no more than eight people. Another company has, in the past, used teams of up to 40 staff — but is seeking ways of avoiding this in future. It has learnt that large teams lead to problems defining and allocating responsibilities and accountability, identifying 'whole' or 'complete' pieces of work, communicating between team members, and staff involvement.

These companies have recognised the benefits of using small teams. This may explain why, in one of our surveys, the importance of team size was ranked very low (77th out of the 84 factors we assessed). Companies with small project teams no longer perceive team size as an important issue. Figure 3.10 shows the maximum number of staff used at any one time in the projects recorded in the PEP database. The most prominent peak is five. Seventy-five per cent of projects have a peak staffing of 12 or less.

Our view is that, whenever practical, systems development project teams should be limited to just five or six people. This view aligns with the research of Dr R Meredith Belbin of the Industrial

Figure 3.10 Seventy-five per cent of projects have peak staffing of 12 or fewer



(Source: Butler Cox PEP database)

Training Research Unit (formerly part of University College, London). He found that a team of four was the minimum necessary to accommodate the essential team roles effectively. Teams of six were found to be best in terms of their stability and endurance, and their ability to allow either for some overlap in team roles, or for one or two individuals to concentrate on single roles.

THE EFFECT ON PRODUCTIVITY OF TEAM COMPOSITION

Although much systems development work can be accomplished by individuals, there are times when genuine team working is needed in every project, such as during the design phase. Team composition and ensuring that the roles of the individual are clearly defined then become crucial. Grouping individuals into teams helps to ensure that everyone is committed to, and working towards, achieving the overall objective of developing a successful system.

There is a considerable body of material about the number of identifiable roles in a team. A case in point is the work carried out by Dr Belbin, mentioned above. His research led him to identify eight team roles, each of which he believed to be essential to the success of the team (see Figure 3.11). This analysis is based on the assumption that there is little or no ambiguity in role definition — something that becomes increasingly difficult to achieve as the complexity of tasks to be undertaken by a team increases. In practice, however, Belbin found that one individual can perform more than one role. According to this research, therefore, the number of individuals in a team need not be as many as eight.

Individuals who are brought together in a systems development team do not immediately form a closely-knit unit. Teams go through their own stages of development — known as orientation, internal problem solving, growth and productivity, and evaluation and control — as we illustrate in Figure 3.12, on page 46. (The stages are, of course, quite distinct from the development phases of the project that the team is working on.) Team performance is heavily influenced by the team-working stage of development that has been reached. Each stage in the team-development process is characterised by different behaviour and team performance.

Team development is likely to stagnate at the internal problem-solving stage, preventing performance from progressing to the high point associated with strong cohesion and alignment of individual and team goals. Moreover, changes in team composition, structure, and leadership can cause team development to revert to an earlier stage. Team leaders need to recognise and reduce the impact of these earlier phases of team development so that the team can progress as quickly as possible to the most productive phases.

Systems development work tends to be more routine in the later development phases. This assertion is based on a study undertaken in 1986 of 68 staff from 20 large-sized firms in the United States. The staff had worked in data processing for five or more years, and most were systems analysts who had worked earlier as programmers. Participants in the study were asked to respond to questions aimed at assessing how routine the work was at each phase of development. The results, which show that systems

Figure 3.11 Eight roles are essential for a successful team

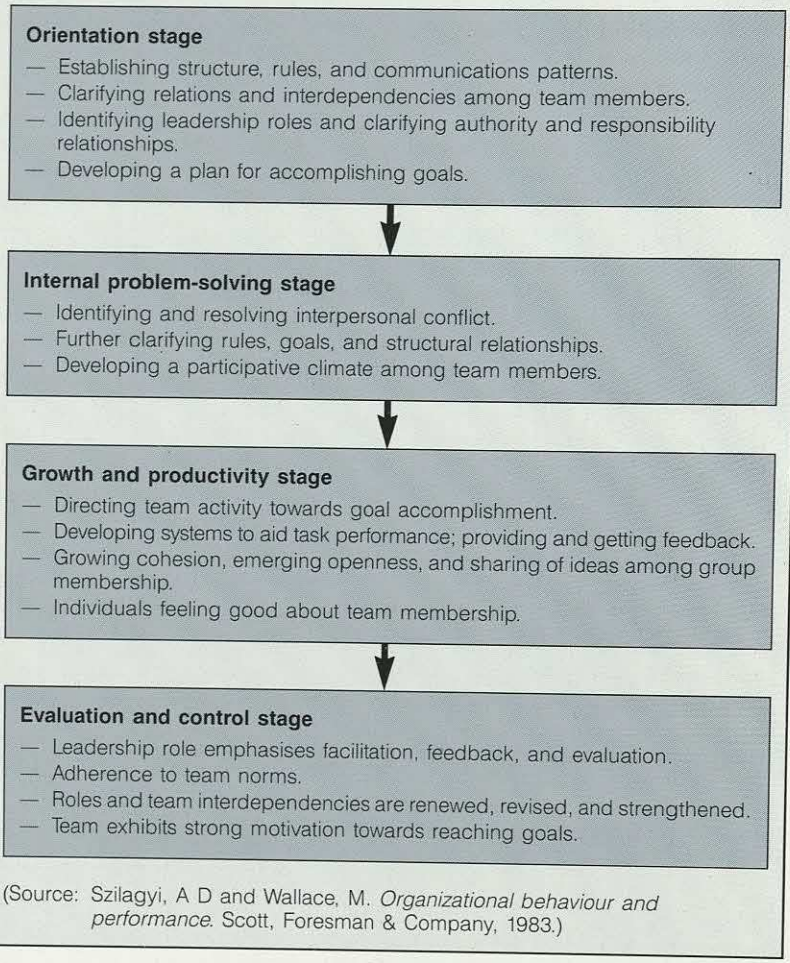
Role	Typical characteristics	Positive qualities	Allowable weaknesses
Company worker: turning concepts and plans into practical working procedures; carrying out agreed plans automatically and efficiently.	Conservative, dutiful, predictable.	Organising ability, practical common sense, hard-working, self-discipline.	Lack of flexibility, unresponsiveness to unproven ideas.
Chairman: controlling the way in which a team moves towards the group objectives by making best use of team resources; recognising where team's strengths and weaknesses are; ensuring best use of members' potentials.	Calm, self-confident, controlled.	A capacity for treating and welcoming all potential contributors on their merits and without prejudice. A strong sense of objectives.	No more than ordinary in terms of intellect or creative ability.
Shaper: shaping the way team effort is applied; directing attention generally to the setting of objectives and priorities; seeking to impose some shape or pattern on group discussion and on outcome of group activities.	Highly strung, outgoing, dynamic.	Drive and a readiness to challenge inertia, ineffectiveness, complacency, or self-deception.	Proneness to provocation, irritation, and impatience.
Plant: advancing new ideas and strategies with special attention to major issues; looking for possible breaks in approach to the problems with which group is confronted.	Individualistic, serious-minded, unorthodox.	Genius, imagination, intellect, and knowledge.	Up in the clouds, inclined to disregard practical details or protocol.
Resource investigator: exploring and reporting on ideas, developments, and resources outside the group; creating external contacts that may be useful to the team and conducting any subsequent negotiations.	Extroverted, enthusiastic, curious, communicative.	A capacity for contacting people and exploring anything new. An ability to respond to challenge.	Liable to lose interest once the initial fascination has passed.
Monitor-evaluator: analysing problem; evaluating ideas and suggestions so that team is better placed to take balanced decisions.	Sober, unemotional, prudent.	Judgement, discretion, hard-headedness	Lacks inspiration or the ability to motivate others.
Team-worker: supporting members in their strengths; underpinning members in their short-comings; improving communications between members, and fostering team spirit generally.	Socially orientated, rather mild, sensitive.	An ability to respond to people and to situations, and to promote team spirit.	Indecisiveness at moments of crisis.
Completer-finisher: ensuring team is protected as far as possible from mistakes of both omission and commission; actively searching for aspects or work that need a more than usual degree of attention; maintaining a sense of urgency within the team.	Painstaking, orderly, conscientious, anxious.	A capacity for follow-through. Perfectionism.	A tendency to worry about small things. A reluctance to 'let go'.

(Source: Belbin, R M. *Management teams — why they succeed or fail.* Heinemann, 1981.)

development work becomes more routine as the phases progress, are shown in Figure 3.13, overleaf.

Teams consisting of people with similar personalities tend to work best on simple routine tasks. Such teams encourage cooperation and communication. Thus, teams made up of people with similar personalities will be more appropriate during the later

Figure 3.12 Teams go through four stages of development



(Source: Szilagyi, A D and Wallace, M. *Organizational behaviour and performance*. Scott, Foresman & Company, 1983.)

Figure 3.13 Earlier phases of systems development are less routine than later phases

Development phase	Extent of routine tasks (on a scale of 1 to 5)
Planning	2.20
Problem analysis	2.49
Development/testing	2.87
Implementation	3.25
Evaluation	2.96
Operation/maintenance	3.64

(Source: White, K B and Leifer, R. *Information systems development success: perspectives from project team participants*. MIS Quarterly, September 1986.)

development phases, when the extent to which work is routine is greatest. By contrast, teams made up of unlike individuals work better during the earlier project phases when the amount of routine work is smaller. Such teams are good for problem-solving tasks, and for tasks involving complex decision-making because the team members stimulate each other, producing a higher level of performance and quality. Teams made up of unlike individuals can, however, create a great deal of conflict. On the other hand, teams of similar people encourage conformity, which can lead to unproductive activity if the team norms (for work output, quality, working practices, and so on) are not consistent with team goals.

From the above, it is clear that the formation of a balanced team requires that account be taken of considerably more than the technical expertise of individual team members. Those responsible for forming teams have to be concerned with the personalities of the members, and to be aware of the need to change the team composition as a development project progresses. In future, the composition of project teams may need to become more fluid, with individuals being assigned to them from time to time and on a part-time basis, so changing the composition of the team in terms of personality as well as skill. The need to do this becomes increasingly important as the size of project teams decreases, as

it will do with the use of contemporary systems development methods.

Cohesion and self selection are further considerations in team formation. Cohesion describes the extent to which team members are able to form a closely knit working unit. Productivity increases with increasing cohesion, mainly because cohesive teams are better at conforming to team norms, provided the norms are aligned with team goals. Cohesion decreases as team size increases. It also decreases as intrateam competition increases (although it increases with growing interteam competition).

Self selection, whereby team membership is decided by the team members themselves, is an approach to team formation that can be successful. In the publication, *Peopleware: productive projects and teams*, T DeMarco and T Lister report on how one company advertises new projects on the noticeboard and invites staff to form themselves into teams to bid for the work. The potential teams are assessed in terms of their suitability to the work, how well the individuals complement each others' skills, and the likely disruption to other work. Cohesion among the members of teams formed in this way was usually high.

ENCOURAGING THE RIGHT STYLE OF LEADERSHIP

The definition and measurement of leadership remain something of a mystery. In addition to acting as a driving force, an important role of team leaders is to influence, assist, and motivate team members in their work. The team leader's role in systems development is somewhat clearer, however. Alongside the directing and guiding role is a facilitating one. In discharging this role, one of the prime measures of the leader's effectiveness is the ability to resolve conflicts amongst the team members.

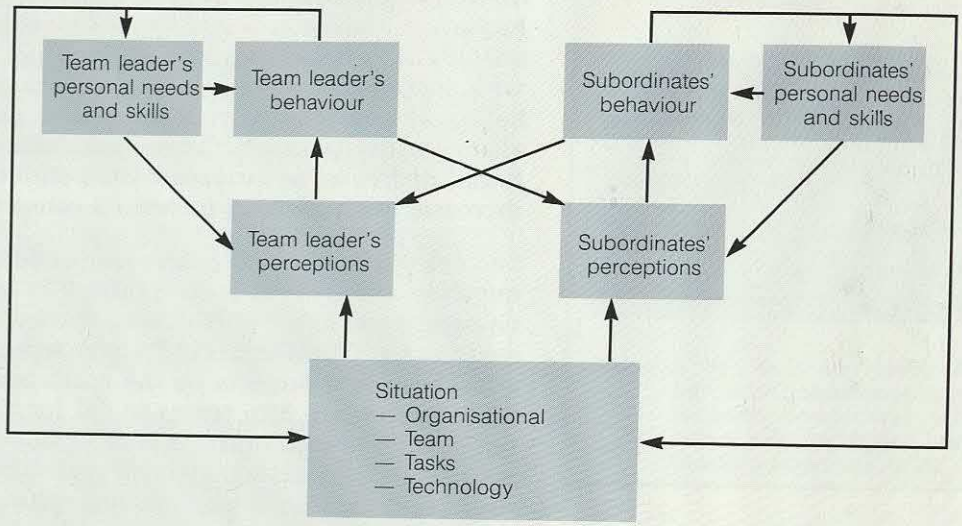
CHARACTERISTICS OF LEADERSHIP

Leadership is difficult to define. A composite view of the characteristics of team leadership is shown in Figure 3.14, overleaf. This illustrates that the team leader is only one influence on an individual's behaviour. The team leader's behaviour is likewise influenced by many factors, including that of the individuals in the team. Team leaders therefore need to take account of the factors that may be influencing individual performance and act either to change the factors that are causing unproductive behaviour or to increase the strength of other influences that promote productive behaviour. Rather than acting as a driving force, the primary role of team leaders is to direct, influence, assist, and motivate team members in their work.

If leadership is difficult to define, it is equally hard to measure. To date, no attempt has been entirely satisfactory. In the 1940s and 1950s, measuring leadership traits was fashionable. The idea was to identify features of successful leaders in terms of physical characteristics, social background, intelligence, personality, and task-related and social characteristics. There proved to be little correlation between these characteristics and a person's effectiveness as a leader, but they did point to the importance of certain characteristics of leaders — alertness, self-confidence, personal integrity, initiative, self-assurance, dominance, their need

Figure 3.14 The team leader is only one of several influences on the behaviour of subordinates

Note that people's behaviour is not directly influenced by their situation — only by their perception of it. Their own personality and skills, as well as their perception of the work situation, are all influenced by their previous work experience both within the team and organisation and from previous employment.



The arrows represent influences and the directions in which they apply

for achievement and responsibility, their high task orientation, their active participation in various activities, their personal-interaction strengths, and their willingness to cooperate with others.

In the 1950s, behavioural theories were concerned with leaders' actions. The theories concentrated on two basic leadership styles — task orientation and employee orientation. Research at that time concluded that behaviour alone was an insufficient explanation of leadership in practice, and that other 'situational' factors needed to be taken into account.

By the late 1960s, situational theories were in vogue. These theories were concerned with results and indicated that leaders' effectiveness depended on their ability, first, to diagnose a problem, and then to change either the various situational factors or to adopt an appropriate leadership style.

THE ROLE AND STYLE OF THE TEAM LEADER

Although the definition and the measurement of leadership remain something of a mystery, the team leader's role in systems development is clearer. Alongside the role of getting the job done is the facilitating role — it is oriented primarily towards helping individual team members to increase personal reward and satisfaction by aligning individual goals with team goals. Four key behaviour patterns persist, regardless of the style that a particular leader adopts to suit changing circumstances. The four behaviour patterns are known as participative, supportive, goal-oriented, and organisational.

Participative behaviour stresses sharing information, consulting team members, and using their ideas and suggestions in decision-

making. Supportive behaviour emphasises concern for the welfare and well being of team members, and the creation of a friendly and pleasing environment. Goal-oriented behaviour is concerned with setting challenging goals, expecting team members to perform at the optimum level, and continually seeking improvements in performance. Organisational behaviour includes planning, organising, controlling, and coordinating individuals' activities. Planning is also concerned with minimising ambiguity in role definitions, and minimising role conflict — both problems that decrease with smaller teams. Different team members will respond in different ways to the behaviour patterns of the team leader. An effective team leader will therefore need to adjust his behaviour to suit specific situations and individuals.

The importance of leadership abilities and styles is certainly clear to the development staff we have surveyed. Overall, they ranked the leadership abilities and style of their immediate manager as sixth out of 84 factors. In general, the level of satisfaction with the immediate manager's leadership is lower than the level of importance that it is accorded by development staff — to a greater extent than for most other factors studied in our survey. There are, however, considerable differences from company to company.

A preliminary analysis of the data suggests that at least half of the companies we surveyed need to pay attention to this area. The problem appears to lie mainly in the fact that staff do not feel that they are being given the opportunity to participate sufficiently in their immediate manager's decision-making. Some companies are already well aware of the need to do this, however. One, noted for its high systems development productivity, emphasised staff participation in a recent recruitment campaign. This campaign was based on a survey of existing staff, who identified participation as a consistent and necessary theme in their working environment.

Another important characteristic of team leadership is the flexibility to adapt leadership style to suit the circumstance of the moment. Flexibility becomes increasingly important when project and team requirements change from phase to phase of systems development. Flexibility is required for several reasons. One is to handle the changing nature of work in the different phases. Another is to handle the development in team working that takes place between initial orientation and final evaluation. Other requirements of leadership flexibility are to handle different situations and individual team members, and to handle different types of conflict, which is discussed further in the next section.

THE STRENGTHS REQUIRED OF A TEAM LEADER

To substantiate their position, leaders need a portfolio of strengths, called an influence base. The relative importance of these influences will determine a leader's effectiveness in getting his group to perform well, and in resolving conflicts. According to some researchers, there are nine separate sources of influence that can be distinguished within a leader's influence base.

According to a survey by H J Thamhain and D L Wilemon, *Conflict management in project life cycles*, reported in the Sloan Management Review, project managers consider the top three influences to be expertise, authority, and work challenge (see

Figure 3.15). In the research programme of which the survey was a part, researchers looked at the effect of each influence on two measures of leaders' effectiveness — project performance and conflict resolution. They found that the more that project managers used expertise and work challenge to influence team members, the better their overall performance and the greater their ability to resolve project-related conflict (see Figure 3.16). Although authority is perceived by project managers to be important (they rank it second to expertise), their superiors believe

Figure 3.15 Nine sources of influence can be distinguished within a leader's influence base

Source of influence (in rank order)	Characteristics of project manager's ability to influence and gain support
Expertise	Perceived as possessing special knowledge or expertise considered important.
Authority	Perceived as having the power to issue orders.
Work challenge	Perceived as being able to provide work of a kind that will particularly provide personal enjoyment; oriented toward the intrinsic motivation of personnel.
Friendship	Personally attractive to the individual.
Future work assignment	Perceived as being capable of influencing future work assignments.
Fund allocation	Perceived as being capable of directly dispensing funds.
Promotion	Perceived as being capable of indirectly dispensing valued organisational rewards.
Salary	Perceived as being capable of directly dispensing monetary rewards.
Penalty	Perceived as being capable of directly or indirectly dispensing penalties to be avoided.

(Source: Thamhain, H J and Wilemon, D L. *Conflict management in project life cycles.* Sloan Management Review, vol. 6, no. 3, Spring 1975.)

Figure 3.16 The more project managers use expertise and work challenge to influence team members, the better their overall performance and the greater their ability to resolve conflicts

Source of influence	Correlation* between source of influence and effectiveness	
	Conflict resolution	Project performance
Expertise	+0.25	+0.40
Authority	−0.25	−0.25
Work challenge	+0.20	+0.30
Friendship	+0.10	+0.05
Future work assignments	+0.10	—
Promotion	+0.10	—
Fund allocation	—	+0.15
Salary	—	−0.25
Penalty	+0.35	+0.30

*Kendall rank correlation coefficients, which can range from −1 to +1. Positive correlations indicate that the source of influence has a positive effect on effectiveness.

(Source: Thamhain, H J and Wilemon, D L. *Conflict management in project life cycles.* Sloan Management Review, vol. 6, no. 3, Spring 1975.)

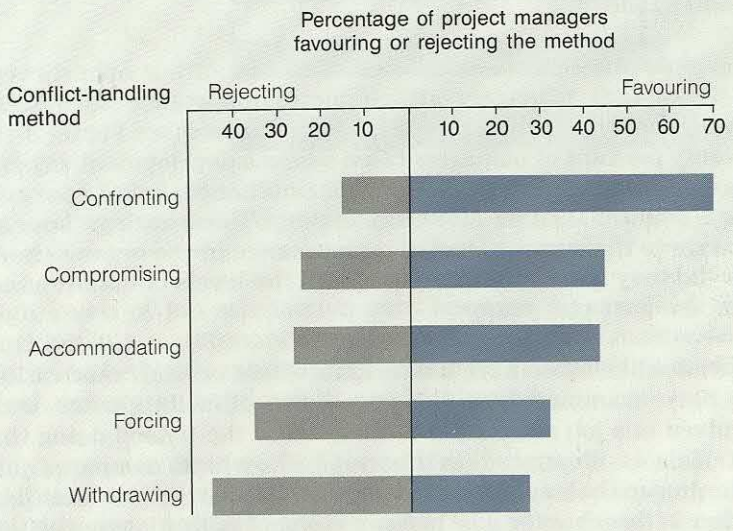
that its use leads to lower effectiveness ratings in terms of both project performance and conflict resolution.

The ability to resolve any conflicts that arise is one of the prime measures of a team leader's effectiveness. Because teams are composed of people with different personalities, some conflict is virtually inevitable. Up to a point, conflict can be beneficial because it can help to introduce ideas that lead to better decision-making, but conflict is destructive if it erodes team effort and spirit, if it results in poor decision-making, or if it introduces lengthy delays resulting from matters of insignificance. The degree of conflict between team members therefore has to be managed.

There are five basic ways of managing conflict. One is by confrontation, in which the disputing parties solve their differences by focusing on issues, looking at alternative approaches, and selecting the best one. The second is by compromise: searching for a solution that brings some degree of satisfaction to all. The third is by accommodation, in which the parties seek areas of agreement and pay less attention to areas of difference. The fourth is called forcing, which involves the group's adopting the viewpoint of one party at the expense of another. Finally, there is withdrawal, in which the group retreats from the area of conflict.

Figure 3.17 shows how Thamhain and Wilemon's research found conflict-handling methods to be favoured or rejected by the project managers they surveyed. Confrontation was favoured by the greatest number and rejected by the fewest. Withdrawal was least popular. Project managers who emphasised expertise and work challenge as their most important influences were most likely to resolve conflicts by confrontation, and to avoid withdrawal. Those favouring withdrawal (and compromise) tended to use friendship as their most influential means of managing conflict.

Figure 3.17 Most project managers resolve conflic by confrontation



(Source: Thamhain, H J and Wilemon, D L. *Conflict management in project life cycles.* Sloan Management Review, vol. 6, no. 3, Spring 1975.)

Thamhain and Wilemon also found that project managers who emphasised expertise had to deal with increased conflict on technical issues. They concluded that project managers were more concerned about the outcome of a conflict and its impact on project performance than they were about the intensity of conflict.

The implication for systems development is that team leaders should be selected on the basis not only of their experience and technical expertise but also their ability to resolve conflicts among team members.

MOTIVATING MAINTENANCE STAFF

Software maintenance has long been generally considered as less important than new systems development. It is often an afterthought in systems design, and is perceived as demanding limited skill and enjoying little prestige and attention. Yet the sheer volume of maintenance work — constituting 50 per cent or more of the systems development workload — demands that managers pay particular attention to improving staff motivation in this area. They may do so in two ways: by avoiding attitudes that are damaging, and by emphasising staff selection and training.

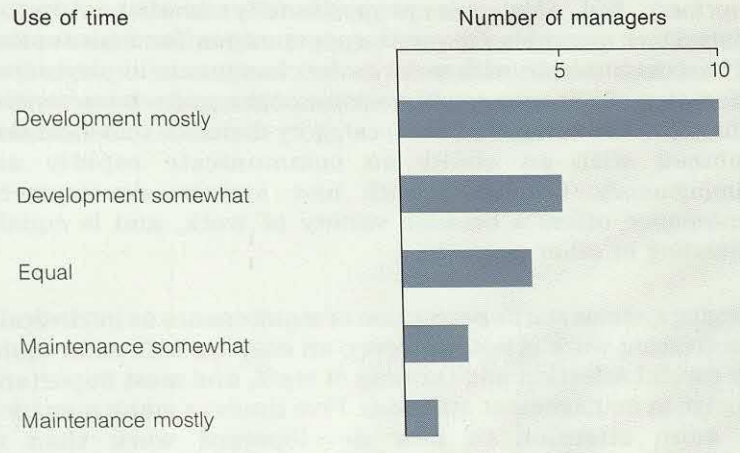
ATTITUDES TO MAINTENANCE

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. Yet programmers tend to avoid maintenance work, preferring instead to work on new systems development assignments. One of the reasons for this is the ambivalence of many managers towards maintenance.

This perception has been confirmed by one of our own surveys. It shows that managers attach more importance to systems development work than they do to maintenance (see Figure 3.18). Seventy per cent of managers rated systems development as being more demanding 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.

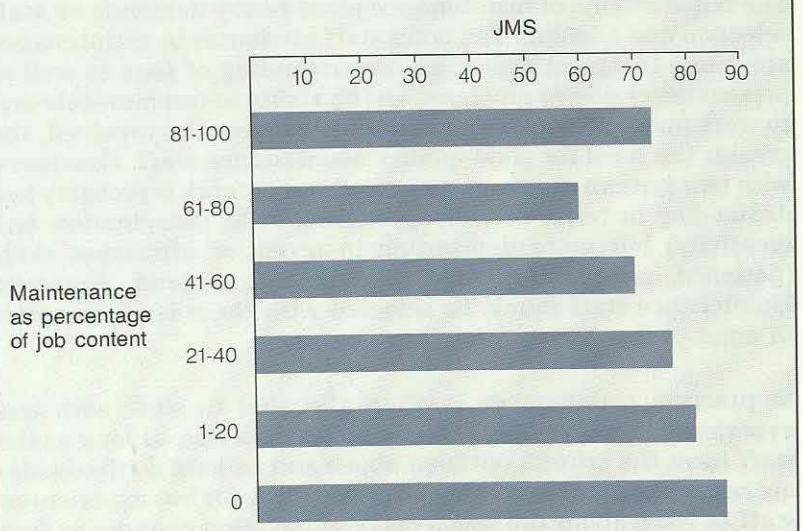
Attitudes like these have a damaging effect on staff motivation. We have examined how the proportion of maintenance work involved in a job affects the motivation of the person doing that job. This is illustrated in Figure 3.19, which compares jobs according to their Job Motivating Score (JMS), which we described earlier in this chapter. 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.

Figure 3.18 Seventy per cent of managers rated systems development as being more demanding of their time than maintenance



(Source: Butler Cox survey of PEP members)

Figure 3.19 Job Motivation Scores (JMSs) vary according to the amount of maintenance work performed



(Source: Butler Cox survey of PEP members)

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

This applies to all three categories of maintenance — corrective, adaptive, and perfective (see page 22). Corrective maintenance is often very frustrating, because of the absence of complete documentation and the difficulties of recreating 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 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.

Changing systems staff's perception of maintenance as intrinsically unmotivating work is not, however, an easy task. It will require very careful selection and training of staff, and most important, a change in management attitudes. Five times as many managers pay more attention to new development work than to maintenance, than vice versa. Until managers see maintenance as an important strategic issue, problems of low staff morale are certain to persist.

SELECTING AND TRAINING MAINTENANCE STAFF

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 systems 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 as important for maintenance staff as 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.

Chapter 4

Using techniques and methods

Most systems development managers would like the systems development process to become more manageable and less dependent on the skills of individual analysts and programmers — experts who are in short supply and expensive to train. In other words, they would like a well defined, systematic procedure or set of processes for developing systems.

With this in mind, many companies have introduced one or other of a wide range of techniques and methods that are available on the market. (Techniques, such as data analysis, are rigorous procedures on which systems development is based. Methods are ways of implementing the ideas embodied in these techniques.)

The benefits are not clear-cut. Sometimes, they manifest themselves in the form of lower staff skills than would otherwise be the case; at other times, in the form of improved ease of use of the end product. Only in three cases have we found convincing evidence of consistent success in applying techniques and methods: when they are used to help formalise software testing, to help control maintenance, and to help establish a quality-management programme.

FORMALISING SOFTWARE TESTING

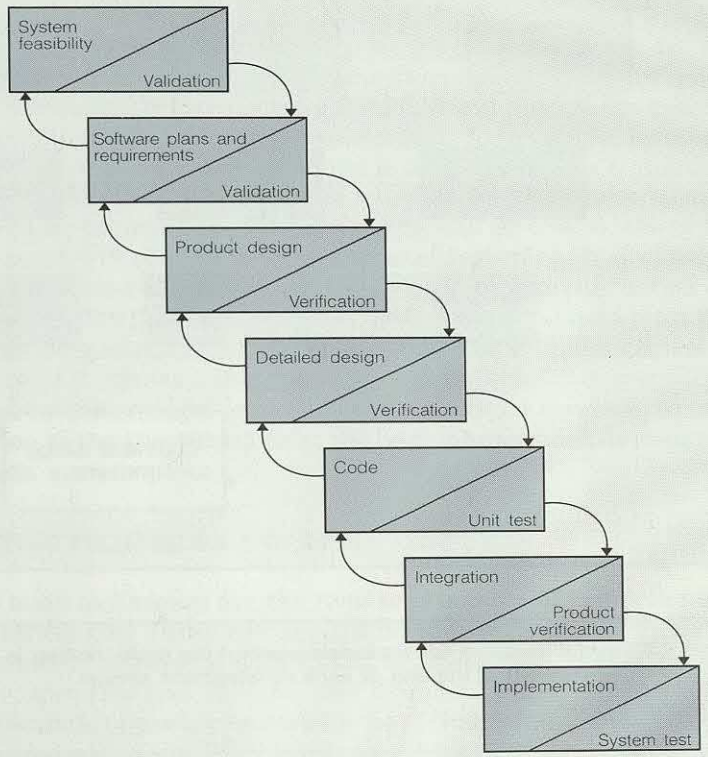
Software testing is a fruitful area for improving both productivity and quality. The first step is to make sure that testing takes place throughout the development cycle, rather than merely at the end. This is called whole-cycle testing, and several testing aids and test-data preparation aids are available.

WHOLE-CYCLE TESTING

There are well established and widely used life-cycle models for the software-development process, the best known of which, first presented in 1970, is the so-called 'waterfall model', illustrated in Figure 4.1. The main feature of this model is that development proceeds through a series of well defined phases. In an ideal development, each phase is verified and proved error-free before the developers proceed to the next. In practice, some iteration is required when errors introduced in one phase are not detected until a later phase. This iteration process is represented in Figure 4.1 by the upward-pointing arrows.

The shortcomings of the approach implied by the waterfall model have become apparent in recent years. The most significant are that testing is viewed as a secondary activity, added on to the end of each phase, and that system testing is not planned until the final development phase.

Figure 4.1 In the traditional 'waterfall' model of the software-development life cycle, testing is viewed as a secondary activity



An alternative approach is illustrated in Figure 4.2, overleaf. It shows development occurring in three main, parallel streams of activities. In each development stream, the first objective is to produce specifications. The second is to specify what to test. The third is to develop the test environment. Only then are the components assembled ready for testing. Testing is thus carried out at the end of each development stream, and measures different aspects of the development in each stream, as described in Figure 4.3, also overleaf. The primary focus is on the testing activity rather than on the production activity, and the outcome of each stream of activity is both a product and a measurement of its quality.

The benefits of this modified approach are four-fold:

- By developing test specifications and the test environment concurrently with lower-level specifications or program code, the overall development time is shortened.
- Developing a test specification can highlight deficiencies in the requirements, design, or module specifications; it therefore provides a valuable opportunity to review the specifications.
- Management's attention is focused at an early stage on defining the important features of the system.
- Developing the tests is a separate activity from producing the design or program. It is much easier for people to define

Figure 4.2 In the modified software-development life cycle, the emphasis is on testing rather than production

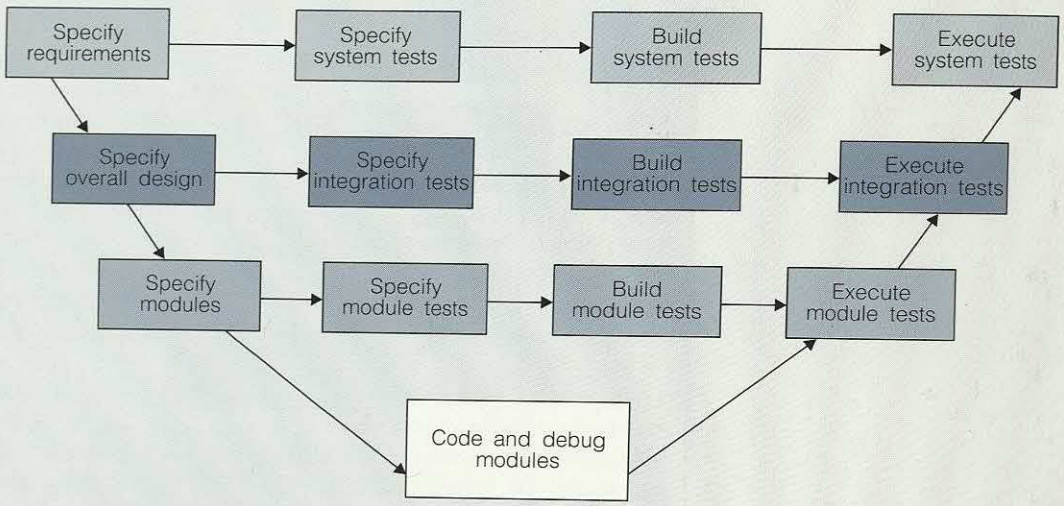


Figure 4.3 In the modified software-development life cycle, testing is carried out at the end of each development stream

Module tests

A program module is the smallest testable component of a system. Its specification comprises a definition of its input data, its output data, and the processes for transforming one into the other. The purposes of module testing are:

- To verify that the module conforms to specified standards.
- To verify that measures of the module's characteristics, such as complexity, are within specified ranges.
- To verify that the module performs its specified functions when executed with a representative sample of input data.
- To verify that each line of code and each of the possible branches have been successfully executed at least once.

Integration tests

Integration tests are designed to measure the behaviour of combinations of modules. They are of two types:

- Verifying the consistency of data definitions that are passed between the modules. This applies both to data that is passed directly, and to data that is passed via a database or shared memory.
- Verifying that all calling paths through the combinations of modules are exercised.

System tests

System tests are designed to measure the behaviour of the total system. This includes tests for some or all of the following features:

- The functionality required by the users.
- The ability to start the system.
- The ability to change the hardware configuration of the system. This particularly applies where there are back-up processors or peripherals that can be substituted in various combinations in the event of failures.
- The ability to restart the system and to recover lost transactions following a failure.
- Performance characteristics, such as response times, delays, and throughput.
- The behaviour of the system when loaded to the limits of its resources.
- The ability to prevent unauthorised users from gaining access to the system.

objective tests of a product that they have not built, and the test cases developed under these circumstances are likely to be a better sample.

The most critical question to be decided is what to test. Management should clearly define the measurements of quality that it requires, before tests are specified and the test environment is created. If ease of use is a requirement, for example, tests could be designed to measure how long it takes to input a transaction, how quickly the system can be learnt, and how many mistakes are made; knowing that these aspects will be tested, the system designers will concentrate on the user interface. If accuracy of data is stated as an important requirement, the activity of specifying the tests will highlight whether all the data must have a high degree of accuracy, or whether some is less critical. As Figure 4.2 shows, the decision about what to test — the requirements, design, or module-test specifications — can be taken as soon as the specifications at the beginning of each development stream are complete.

TESTING TECHNIQUES AND AIDS

The main *techniques* are the review processes of inspections and walkthroughs. These are applicable to testing (or verifying) the documents associated with software production (that is, program code, specifications, designs, user manuals, and so on). Inspections and walkthroughs are less widely used than testing tools, although our analysis of the PEP database shows that the benefits to be gained are quite substantial. Formal *methods* for software development are widely used, but although testing is a component of them, it is poorly described, and is not based on the concept that testing and production are activities to be carried out concurrently. Software testing methods are, however, beginning to be produced, and significant developments can be expected in the future.

No testing aid will, of course, reduce the intellectual effort involved in designing the test environment and selecting test data. The use of testing aids will, in itself, do nothing to improve the quality of testing. Nor will any single aid to testing cover all aspects of the process.

The quality of every deliverable produced during the development of a system, including requirements specifications, designs, code, and test specifications, should be analysed as part of the normal development process. Various techniques and tools are available for this purpose. The two most commonly used techniques are inspections and walkthroughs.

In addition, two types of tools — static analysers and dynamic analysers — can be used to analyse the quality of the code itself. These tools are discussed in detail in Chapter 5.

The use of these techniques and tools is essential for almost all applications with high-reliability requirements, such as those where human life depends on the successful operation of the software. These applications should therefore be written only in a language that can be analysed by these tools.

Inspections: The inspection technique was first developed by Michael Fagan while he was working at IBM in the early 1970s. An inspection is carried out by a team, typically of four people, whose roles are precisely specified. A key to successful inspections is that the team must identify errors only; it must not be side-tracked into discussions of solutions, or alternative design strategies. It is important that the results of the inspection are recorded, and that all errors are corrected by the original designer or programmer.

Inspections are time-consuming (typically, between 4 and 8 per cent of total development effort), and need to be scheduled in the project plans. The total time (including preparation time) for an inspection of a design that produces 1,000 lines of code is 10 to 20 man-hours, and for an inspection of the 1,000 lines of code produced from this design, 20 to 60 man-hours.

In carrying out inspections, each person needs to have a clear understanding of his individual role, and of the purpose of the inspection procedure. The techniques are not easy to learn, and an organisation that intends to introduce inspections should train its staff on formal courses.

Studies of the effectiveness of performing inspections on source code indicate that an inspection typically detects up to 60 per cent of the errors in the code. The reduction in development cost, after allowing for the additional cost of the inspections, is estimated at 10 per cent. This is consistent with the results reported earlier in this chapter; organisations using inspections or walkthroughs had an internal productivity level about 15 per cent higher than organisations that did not use them. If the subsequent maintenance phase is also included, the savings may be considerably larger.

Walkthroughs: These are a less formal type of inspection and may have few, if any, of the formal characteristics of inspections. There are very few rules on how to carry out a walkthrough. At a minimum, it involves one person checking another's work. Because of the lack of formality, walkthroughs are cheaper to carry out than inspections. They are almost certainly less effective, although quantitative data is lacking.

There are many tools to help and support the testing process. These range from tools to help in the process of testing, such as a test harness, to tools that help in the management of the test. These and other testing tools are discussed in Chapter 5.

Testing covers both the development process (the process of producing the first version of the system), and maintenance (the upgrade and enhancement of the system through the remainder of its useful life). It is to the last of these areas that we now turn our attention. 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.

CONTROLLING MAINTENANCE

As well as improving software testing, techniques and methods can help to improve both productivity and quality in software maintenance. Organising maintenance, and the motivation of maintenance staff, are topics that are discussed in Chapters 2 and 3 respectively. Here, we are concerned with gaining control over the maintenance process.

The first step is to formalise the process of deciding whether a maintenance request justifies a system-replacement decision. From a procedural standpoint, the way to do this is by adopting a maintenance-rating method. Next comes the question of what proportion of total resources to allocate to maintenance.

Managing the maintenance process should itself be formalised by breaking the process down into a series of steps. These steps need to be carefully coordinated, not merely monitored individually.

FORMALISING THE MAINTAIN-OR-REPLACE DECISION

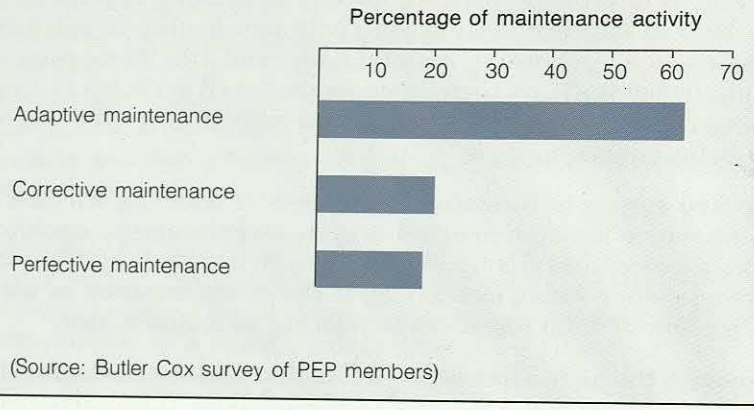
Whether to continue maintaining a system or to replace it with a new one is a critical management decision because operational systems deteriorate with age. There is a need to audit the system to determine whether and when to replace it. The auditing procedure should be formalised, and it should be aligned 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. Thus, 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 one of our surveys, 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 4.4, overleaf). 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 4.5, also overleaf, which categorises the goals of our survey respondents in their adaptive maintenance efforts. Many of these changes are enhancements in the sense of adding new facilities, providing new 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 4.6, on page 63. 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.

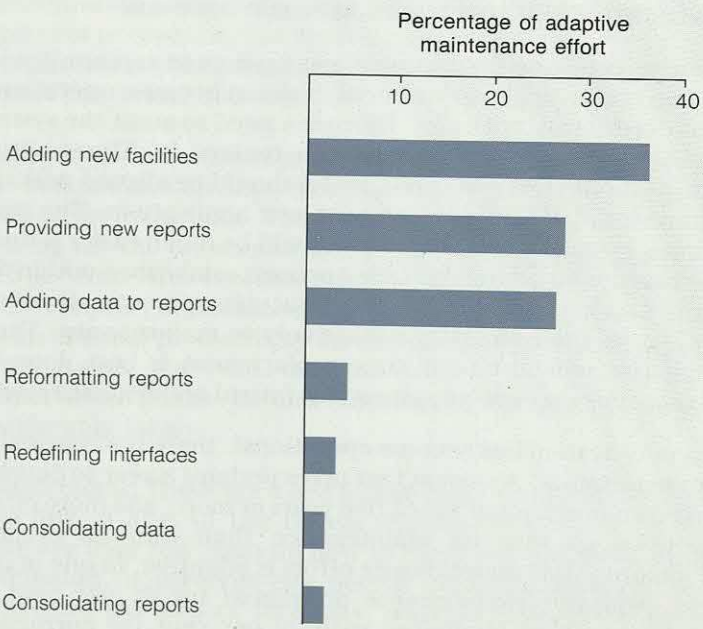
Continuous modification can leave a system in a less stable state than before. Each time the system is modified, it becomes

Figure 4.4 Adaptive maintenance is the main activity



(Source: Butler Cox survey of PEP members)

Figure 4.5 Most adaptive maintenance is to add new features

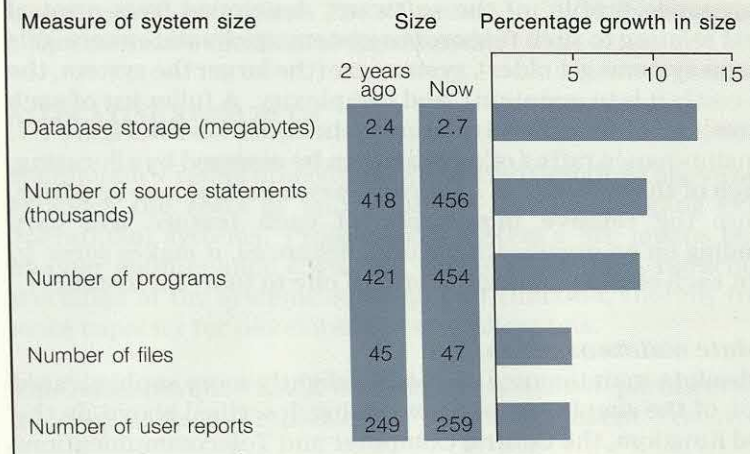


(Source: Butler Cox survey of PEP members)

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

Figure 4.6 Maintained systems grow in size over time



(Source: Butler Cox survey of PEP members)

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.

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.

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.

MAINTENANCE RATING

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.

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 4.7. 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).

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

Figure 4.7 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.

maintenance profile, and a total of 16 measures (between one and three measures for each criterion), as shown in Figure 4.8. Each measure delivers a score. The scores are totalled. Systems scoring 100 or more are candidates for renewal.

ALLOCATING RESOURCES

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

Figure 4.8 There are nine criteria in 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). — Quality of documentation (1 = excellent, 5 = non-existent).
	Change control	— Change control procedures (1 = good, 5 = non-existent). — Testing procedures (1 = good, 5 = non-existent).
	Errors	— (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)

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 the companies we surveyed. 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.

An example of how this policy can work in practice is the experience of a public utility. A few 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.

MANAGING THE MAINTENANCE PROCESS

For maintenance work to be effective, it is essential 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 six further steps: 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 4.9.

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

Change management

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

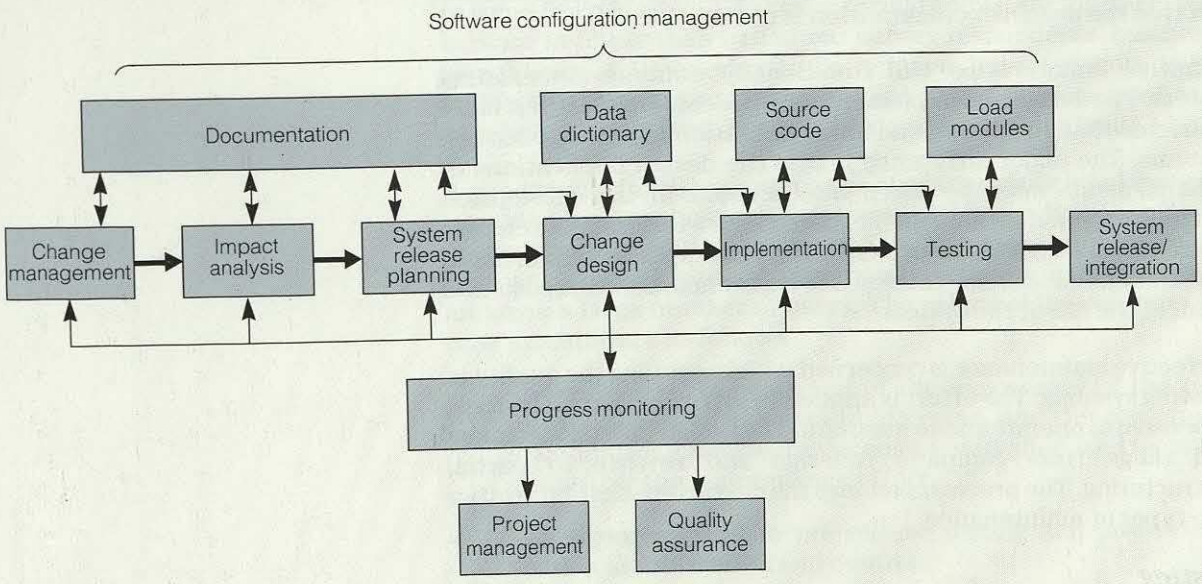
formal change-request document that forms the basis of a contract between the user and the maintainer.

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 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.

Figure 4.9 The formalised maintenance process consists of seven steps



(Source: Butler Cox)

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 organisations, reflecting a difference in the extent to which formal maintenance contracting is established.

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. 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. Walkthroughs 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 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.

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

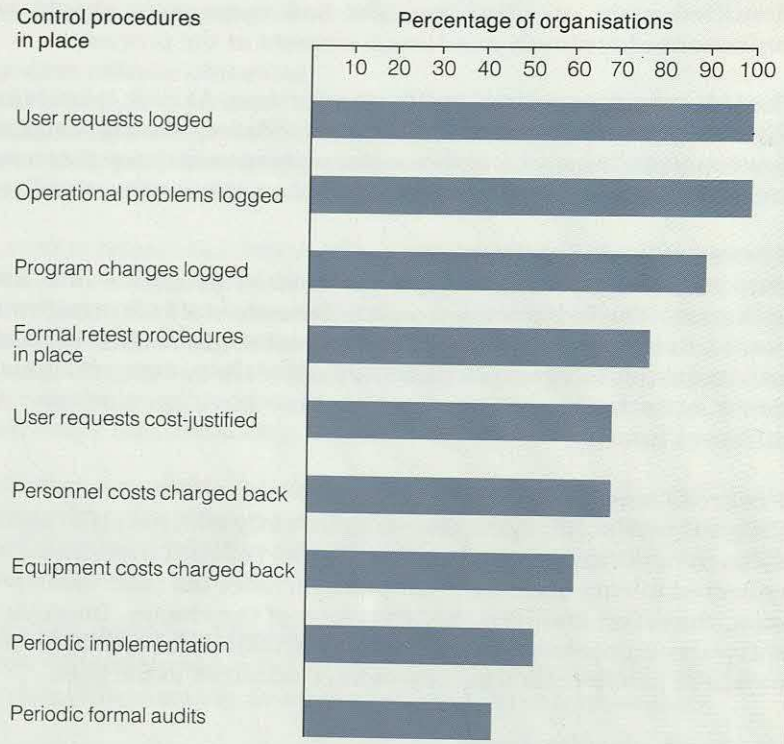
PROGRESS COORDINATION

Most companies claim to have a clearly defined procedure in place that corresponds to change management. Certainly, every respondent in our survey examining this issue recorded all user requests and operational problems, but our respondents admitted to some failings as well. Periodic formal audits, for instance, were in place in fewer than half of our survey respondents' businesses (see Figure 4.10 overleaf). To achieve improvements in the maintenance environment, the steps in the process need to be carefully coordinated, not simply monitored individually.

A good model for the maintenance of a large application system is provided by Peterborough Software (UK) Ltd, a British software house (see Figure 4.11 on page 71). The model is particularly relevant to multisite, multiversion software implementation amongst a large number of users. The principal lessons from this case example 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

Figure 4.10 Most surveyed organisations have formal control procedures in place



(Source: Butler Cox survey of PEP members)

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, 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. At the same time, there are several automated change- and configuration-control packages currently being introduced to the market that could help organisations to reduce administrative overheads and increase their control over system changes.

IMPROVING QUALITY

Quality-control procedures should be carried out at intermediate stages of the development in addition to those at the end of the development cycle. They should focus on four key quality characteristics, whose emphasis differs depending on the nature of the application under development.

Within this general framework, our project-database analysis points so far to only one technique that is clearly beneficial in terms of improved quality: inspections or walkthroughs. Other methods and techniques have a role to play, but we have found no evidence that they improve quality and productivity consistently. Probably the

Figure 4.11 Peterborough Software (UK) Ltd provides a good model for the maintenance of large application systems

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.

most fruitful way to improve quality across the board is by introducing a quality culture throughout systems development.

QUALITY-CONTROL PROCEDURES

In the past, systems quality-control procedures focused on the product by checking that a completed computer system met the original specification. The techniques used include system and acceptance testing, and a post-implementation review. Few, if any, quality checks were carried out at intermediate stages of the development process.

This approach to systems quality assurance is concerned only with checking that the final system meets the original requirements, and not with the overall process by which the product is developed. The result is often that the delivered system meets neither the users' requirements nor their expectations, in terms of functionality, operational performance, usability, development cost, and delivery date. The defects discovered when the final system is inspected are often caused by mistakes made at the early stages of the development process — the requirements-definition stage, for example.

To overcome the shortcomings of this approach to assuring systems quality, many systems development departments have been encouraged, by the availability of methods and tools, to concentrate on improving the effectiveness of the development process. These methods and tools make it possible to enforce a standard approach

to development and make it easier to check the quality of the software at various stages of its development. The stages of the cycle required to complete a project, from initiation to completion, are precisely defined, as are the deliverables to be produced at each stage. The deliverables can then be checked before development staff proceed to the next stage. In this way, defects can be detected earlier and corrected before the software is delivered to the users.

The role of many systems quality-assurance departments today is to define the development process that will be used and to carry out the quality-control checks at the end of each development stage. The development procedures, and the procedures for carrying out the checks, are usually defined in great detail and enshrined in the 'systems development standards manual'. The quality-assurance staff themselves are perceived as 'policemen', whose main role is to ensure that the procedures are followed and that those who break the rules are identified.

This is often an inadequate approach to improving the quality of systems, as we point out in Chapter 2 on pages 13 and 14. Sometimes, the existence of such quality-assurance departments hinders, rather than helps, the development of new systems. All that is achieved is the creation of an additional layer of bureaucracy concerned with enforcing standards, ensuring that rigid procedures are followed, and insisting that lengthy checklists are completed.

The difficulty arises because traditional systems quality-assurance concepts are based on too narrow a definition of systems quality. The procedures typically used are concerned with ensuring that the final system has the specified *functionality*. This is insufficient to ensure that the system meets the users' *real* needs. Other equally important aspects of quality, such as the quality of the user interface, the operational performance of the system, the ease with which the system can be modified to meet changing business requirements, and the quality of the documentation, are largely ignored by conventional approaches to systems quality assurance.

High-quality systems are ones that conform to users' expectations, in terms of its functionality, operational performance, ease of use, and documentation. Two features of this definition are particularly important: the emphasis on users' expectations, and the fact that quality is not limited to the software itself. It is the entire package — the software, documentation, manuals, training, and user-support — that determines the users' satisfaction and thus their perception of the quality of the software. This definition does, of course, encompass traditional concepts of systems quality — in particular, the need to produce software to budget with the minimum number of errors. However, the emphasis on the way the software is constructed is recognition of the fact that quality in software is also a matter of how easy it is to modify and extend systems to meet changing business requirements, and how well systems can meet performance criteria.

QUALITY CHARACTERISTICS

Today, most systems quality-assurance procedures are designed to ensure that the functionality provided by applications software meets the users' requirements. However, even where the quality of the system is checked at intermediate stages in the development

life cycle to ensure that the finished product does meet the functional requirements, it may still be regarded as being of poor quality by the user community. This is because the quality-assurance procedures do not take account of the users' needs in other areas — operational performance, ease of use, and the ease with which the system can be modified are obvious examples. Figure 4.12 describes how one business with a conventional quality-assurance function has realised the need to take a broader view of systems quality.

Four characteristics are particularly important: functional requirements, operational performance, technical features, and ease of use. By defining and meeting quality objectives specified in terms of these characteristics, it is possible to build application systems that the user community regards as high-quality. Although the functional requirements of a system are generally defined in great detail, the other three characteristics are often ignored in systems specifications. These characteristics are usually determined by ad hoc decisions made at the analysis and programming stages.

Functional requirements

The functional requirements define what the application system has to do, down to the level of describing the data to be entered, the

Figure 4.12 Conventional quality assurance is not sufficient to guarantee the quality of systems

Insurance company

This organisation has a large systems department, with more than 100 development staff. The quality-assurance function resides within the systems department and is staffed by three people — a manager and two project managers. There is a well established quality-assurance culture, based on a comprehensive code of practice that covers methods, techniques, and the use of tools. The code of practice was developed four years ago and is updated annually.

At the start of a project, user expectations are documented on a project-authorisation form. This includes a quality plan, although in most cases, the plan indicates only that the code of practice will be followed. Systems development staff believe that a more detailed statement of quality objectives is desirable because it would enhance the value of post-implementation reviews.

The quality-assurance staff are invited by a user or a departmental manager to review application systems at regular intervals. An initial review can take several weeks, and the actions agreed are followed up later. Because of the number of development staff, the quality-assurance staff are very busy. However, they usually carry out reviews when a problem is detected rather than at predetermined stages in the development life cycle, even though they realise that scheduled reviews are a better means of detecting problems earlier in the development process. External consultants have also been employed, with considerable success, to review particular projects, both from a business and a technical viewpoint.

The quality-assurance manager considers that quality assurance based solely on controlling and reviewing the development process is insufficient to provide the quality required — in particular, for the business aspects of a system. The existing procedures mean that insufficient attention is paid at the beginning of a project to issues such as the feasibility of changing work practices in user departments. Many of the quality-control reviews carried out at present are concerned with technical issues — for example, program walkthroughs require up to 25 per cent of the programming effort. To progress beyond this to a wider quality-management programme, senior management must lend their support to giving quality assurance greater prominence throughout the organisation. This support is now being sought.

The quality-assurance manager told us that his aim is to make users responsible for quality in systems development projects by providing them with a code of practice and making them accountable for the business success of the projects. He believes that, when the wider quality-management programme is in place, his department will need fewer staff because the quality-assurance process will be an integral part of the whole organisation.

rules for deciding whether to accept or reject the data, and the processing to be performed once the data has been accepted. Most systems specifications contain adequate functional requirements, and it is relatively straightforward to assess the quality of a system in terms of how well it meets these.

Operational performance

The operational-performance characteristics of a system define the expected performance in terms of response times (for online systems), and the elapsed time required to perform specific processing loads for batch systems. If these characteristics are defined at the outset, the quality of the final system can be assessed against them. However, the objectives should be set bearing in mind special factors that will degrade performance, such as peak processing loads or changes in workload.

Technical features

The technical features of a system relate to the way the software itself is constructed. The technical quality of a system can be specified in terms of its mean time between failures, the ease with which it can be maintained and extended, how easy it is to change the basic system by parameters specified at run time, for example, and how easy it is to re-use parts of the software in other applications. Checklists should be constructed for each of these characteristics, and used to assess the technical quality of the software. Figure 4.13 shows a sample checklist for assessing how easy it will be to extend a particular application.

Ease of use

The increasing use of PC-based software packages by the user community has raised users' expectations considerably about ease of use. Despite this, mainstream applications are still developed that users find boring, tedious, or difficult to use. A poor or inadequate user interface can mean that a system is regarded as being of poor

Figure 4.13 Technical quality: a checklist for assessing how easy it will be to extend a system

This checklist can be used to assess how easy it will be to modify or extend a system's existing computational and/or data-storage limits (field sizes, record length, file sizes, and so on).

System characteristics indicating that modifications or extensions will be easy

- The system allows key parameters to be modified at run time. It should also validate the run-time entries to ensure they are within allowable boundaries.
- The documentation adequately describes what constraints of the system may be altered and how to do it.
- The system specifically tests for each code that can be input to the system, so that any code not explicitly recognised by the system is rejected.
- There are enough fields of an adequate size to allow for reasonable growth.

System characteristics indicating that modifications or extensions will be difficult

- Parameters are coded into the program logic.
- Files are sequential or index-sequential.
- Low-level protocols are used for network communications.
- Incompatibilities between system modules have been resolved by linking them via specially written programs.

(Adapted from: *"The Quest for Quality"*, published in *Datamation*, March 1, 1985.)

quality even though it meets all of the functional requirements, has high operational performance, and is technically sound.

The quality of a system therefore depends also on its user-interface characteristics. For example, the quality of the user interface might be specified in general terms as one that provides clear, unambiguous messages for users, that requires the minimum number of keystrokes to be used, that provides a rapid response time, and that has simple, unambiguous error-recovery procedures. These general terms can then be defined in more detail. Clear messages for users might be defined in terms of clear command prompts, and the existence of a help facility, a tutorial mode, a terse mode, audio responses, and pointers to the most likely next activity. Specifying the user-interface characteristics in these terms will allow the quality of the user interface to be defined and assessed.

QUALITY PROFILE OF DIFFERENT APPLICATIONS

The full requirements for an application system can be defined in terms of the four types of system characteristics described above, and the extent to which these requirements are met provides an indication of the quality of the system. It is important to remember, however, that users' perceptions of quality are determined largely by their expectations. Different types of application are used by different types of user with different expectations. The implication is that the relative emphasis given to each of the four types of systems characteristics will vary according to the type of application. For some types of application, its quality will be judged largely on the quality of the user interface; for others, it will be judged largely on the technical quality of the software.

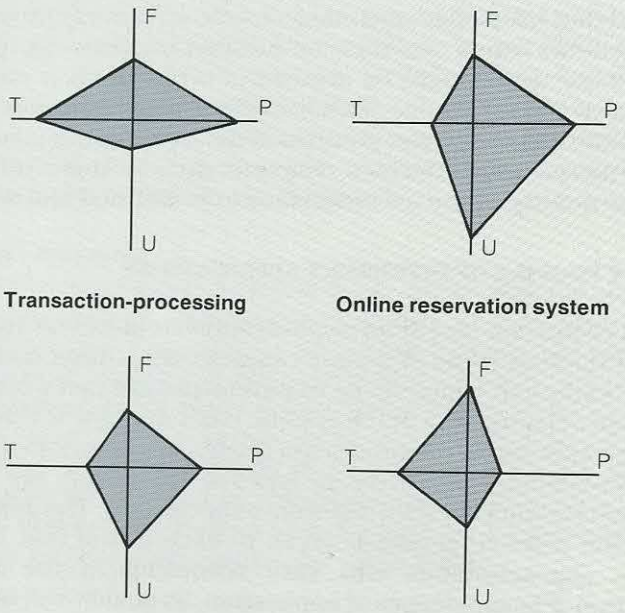
Different types of application therefore have a different 'quality profile', which can be expressed diagrammatically, as shown in Figure 4.14, overleaf. Transaction-processing applications, for example, require a high level of technical quality and high levels of operational performance, whereas the quality of an accounting package is determined much more by how well it meets the functional requirements and by the quality of its user interface.

The different quality profiles also imply that different emphases are required on checking the quality of the software product being produced and on assuring the quality of the development process itself. Ensuring that the software meets the functional requirements requires a heavy emphasis on quality-control checks as the software is developed. High technical quality and good operational performance are determined more by the quality of the development process. Figure 4.15, overleaf, shows the relative emphasis on product and process quality required for each of the four system characteristics.

In general, greater emphasis on product quality will increase the cost of developing an application because it will be necessary to carry out a greater number of, and more extensive, quality-control checks. Greater emphasis on process quality means that substantial initial effort is put into defining a formal development process and ensuring that it is followed. However, emphasising process quality will result in better-designed and more flexible software.

Figure 4.14 Different types of applications have different quality profiles

The characteristics of a system can be expressed in terms of the functional requirements, operational performance, technical quality, and ease of use. The quality of a system can be assessed in terms of how well the software matches these characteristics. The relative emphasis of each of the characteristics will be different for different types of application.



F = Functional requirements
P = Operational performance
T = Technical quality
U = Ease of use

Figure 4.15 Different system characteristics require different emphases on product and process quality

System characteristic	Relative emphasis on:	
	Product quality	Process quality
Functional requirements	★★★	★
Operational performance	★★	★★
Technical quality	★	★★★
Ease of use	★	★★★

ESTABLISHING A QUALITY-MANAGEMENT PROGRAMME

In a good quality-management programme, quality checking should take place at each stage of the development cycle. Of the many techniques and methods that are available to help with this, only walkthroughs seem to benefit quality with any degree of consistency, according to our analysis. Techniques and methods, such as structured analysis and structured design, seem to have a

generally adverse effect. The reasons behind this are not clear and this is an area that we are continuing to investigate.

A good way to improve quality is by encouraging it to become a way of life for all the staff. That requires top management attention and a good deal of time and effort.

Quality checking

Most systems development departments realise that it is not sufficient to check the quality of applications software once only, at the end of the development life cycle. Errors or mistakes discovered as a system is implemented may have been caused by an error made right at the beginning of the development process, and will be very expensive to correct because much of the work already done will have to be redone. Barry Boehm asserts in *Software Engineering Economics* (published in 1982 by Prentice Hall) that the cost of correcting an analysis error at the maintenance stage is 100 times more than the cost of detecting and correcting the error immediately. Other research indicates that the cost of correcting an error made early in the life cycle increases exponentially the longer it remains undetected.

These problems can be overcome by applying quality assurance at each stage of the development cycle. To achieve this means that the stages of the cycle must be clearly defined, so that quality checks can be carried out at the end of each stage. In this way, errors can be detected as they occur and can be corrected at minimum cost.

The deliverables at the end of each stage should be specified in detail and should reflect the *four* characteristics described earlier in this chapter. The quality of the application system being developed can then be assured by checking that the work delivered conforms to the specification. Figure 4.16 shows the deliverables that may be specified for various stages of the development cycle.

Structured techniques and formal methods designed to help at different stages of the cycle seem, according to our database analysis, to have a generally adverse effect on both productivity and quality. This is shown in Figure 4.17 and Figure 4.18, on pages 78 and 79 respectively. The exception is the walkthrough technique.

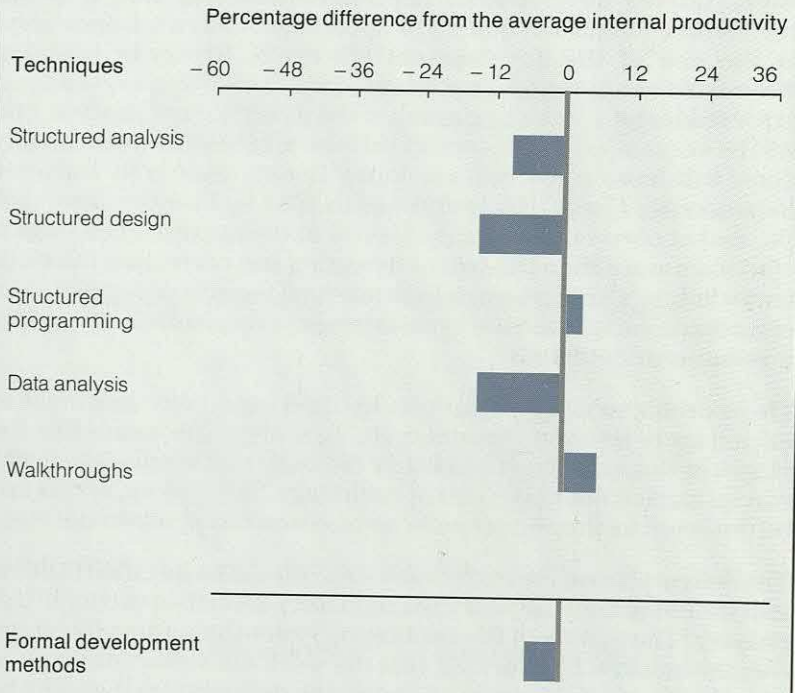
Figure 4.16 Deliverables must be specified for each stage of the development life cycle so that quality checks can be made

Development stage	Sample deliverables
Feasibility study	Project scope; analysis of current system; project plan and justification
Logical system design	System flowchart; logic charts; illustrative outputs
User procedure design	User manuals and examples
Computer procedure design	File layouts; test requirements
Program design	Structure chart; source code; object code; test results
System evaluation	Estimated vs actual (costs, timescales, effort, benefits, and so on)

(Adapted from: "Improving the Productivity of EDP Systems Development", published in *Systems Development*, September 1988.)

Figure 4.17 Most projects that use techniques and methods have lower-than-average internal productivity

Each chart shows the difference in internal productivity relative to the average level for all projects of a similar size. Thus, projects using structured analysis techniques have an average internal productivity level that is 0.3 lower than the average for similar-sized projects.



(Source: Butler Cox PEP database)

Using walkthroughs

Our project database analysis shows that the internal productivity levels of projects using formal walkthroughs are slightly higher than the average for projects of a similar size, and about 15 per cent higher than those not using formal walkthroughs.

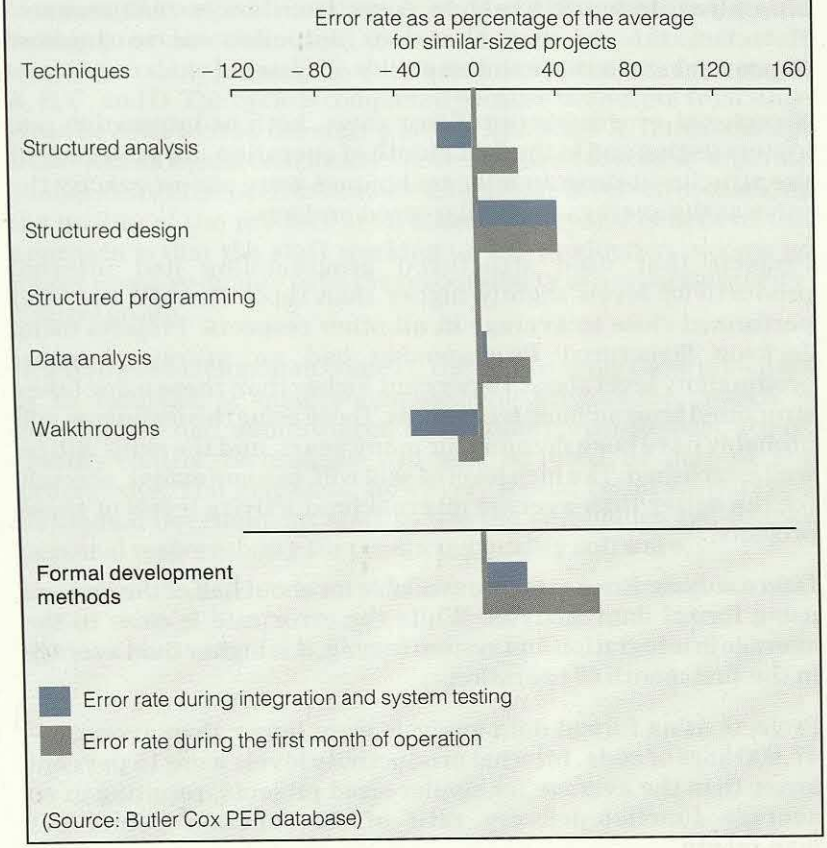
The only technique that results in consistently lower error rates is formal walkthroughs (including inspections). This is clearly shown in Figure 4.18. The error rate at integration and system testing for projects using this technique was about 35 per cent below the average for similar-sized projects, and about 10 per cent below average in the first month of operation. This is a very encouraging result because it implies that the lower error rate at integration and system testing was due to inherently higher quality, not to insufficient integration and system testing. Formal walkthroughs are used on 16 per cent of our database projects.

Using other structured techniques

The use of techniques and methods other than structured walkthroughs seems to have an adverse effect on the technical quality of projects, according to our database analysis.

Figure 4.18 Most projects that use techniques and methods have higher-than-average error rates

Each chart shows the difference in error rate relative to the average error rate for all projects of a similar size. Thus, projects using structured analysis techniques have an average error rate during systems and integration testing 15 per cent lower than the average error rate for similar-sized projects, and an average error rate during the first month of operation 20 per cent higher than the average rate for similar-sized projects.



(Source: Butler Cox PEP database)

Structured analysis: Projects using structured analysis have slightly lower-than-average internal productivity levels, and more software errors. (The sample size was small, however.) The use of fourth-generation languages is marginally lower than average, resulting in a rate of function delivery of 10 function points per man-month, about 30 per cent below average. The use of structured-analysis techniques may, of course, also contribute to a better fit of the developed systems to business needs, but the data currently stored about the projects does not allow us to confirm this.

Error rates are available for about half of the projects using structured-analysis techniques. Error rates in integration and system testing were lower than average, but higher than average in the first month of operation.

Structured design: Error rates were higher than average in integration and system testing, and in the first month of operation, both by about 40 per cent. (Error rates are available for about two-thirds of the projects using structured-design techniques.)

At 57,000 lines of code, structured-design projects were larger than the average of 45,000 lines. Internal productivity levels were 15 per cent lower than the average for projects of a similar size, and the average rate of function delivery, at eight function points per man-month, was more than 40 per cent below average. This is not surprising, because such techniques are more likely to be used for large applications developed in traditional languages. These measures do not necessarily imply that the use of structured-design techniques reduces development performance. The main benefit of structured design is likely to come from easier maintenance. However, the existing data does not allow us to measure improvements in the maintainability of systems.

Structured programming: Error rates, both at integration and system testing and in the first month of operation, for projects that use structured-programming techniques were almost exactly the same as the average for similar-sized projects.

Projects that used structured programming had internal productivity levels slightly higher than those that did not, and performed close to average in all other respects. Projects using Jackson Structured Programming had an average internal productivity level about 15 per cent higher than those using other structured programming techniques. Those using this technique will probably have been doing so for many years, and the skills will be well established. The high level of skill will, to some extent, account for the better-than-average internal productivity levels of these projects.

Data analysis: Error rates are available for about half of the projects using formal data analysis. While the error rate is close to the average in integration and system testing, it is higher than average in the first month of operation.

Projects using formal data analysis were larger than average — 47,000 lines of code. Internal productivity levels were 15 per cent lower than the average for similar-sized projects, resulting in an average function-delivery rate of 10 function points per man-month.

Creating a quality culture

The quality-improvement procedures described in this chapter imply a higher-than-usual number of quality-control checks. One way to handle this is to increase the number of staff in the quality-assurance department.

A better way is to make each member of staff in the systems department personally responsible for the quality of the work he or she produces, as we mention in Chapter 2 on page 14. The aim should be to create a 'quality culture' so that quality is 'a way of life' for all staff. Creating such a culture requires a commitment to quality from the organisation's top management. It takes a good deal of time and effort, but it produces two main benefits: the quality of the products is improved, and the cost of assuring quality is minimised, because it is not necessary to employ a vast army of quality-control inspectors.

Japanese manufacturing companies are well known for their quality cultures. Staff have the opportunity to work on different stages of the production process, experiencing all facets of the work,

and eventually, gaining knowledge of the complete process. This means that, when they are working at a particular stage, they understand the consequences of defects introduced at earlier stages. It also means that they are in a better position to make recommendations for improving the process, and are able to check the quality of the product at each stage in the process.

Suppose, for example, that the production process has five stages (A, B, C, D, and E). Staff working on stage B would be in a position to review the output from stage A; staff working on stage C would be able to review the output from both stage A and stage B; staff working on stage E would be able to review the outputs from stages A, B, C, and D. The cycle is completed because the output from stage E can be reviewed by the staff who work on stage A. If this concept is taken to its logical conclusion, there is no need to employ separate quality-control inspectors because all staff are involved in checking the quality of the product at all stages. The major benefit of this approach is that the staff working on the production process no longer perceive quality to be the responsibility of a separate quality-control group.

In a systems development context, the way to apply these principles is to establish a quality-management programme, which is coordinated and administered by a quality-management group. Quality-control techniques will still be used as part of the programme. The emphasis, however, should be on encouraging individual development staff to use the techniques and to take personal responsibility for producing quality software.

Chapter 5

Using contemporary tools

Tools automate some of the activities within a systems development method. Cobol is a tool that is well established. Contemporary tools include fourth-generation languages like Mantis, CASE tools, and re-engineering tools.

Contemporary tools are widely claimed by their suppliers to deliver enormous benefits in productivity and quality — even the imminent redundancy of the programmer. None of these claims has been achieved in full, however — often because of the absence of an effective means of measuring benefits and then taking actions to ensure that the benefits are continually attained.

Although there are benefits to be gained, managers should treat contemporary tools with caution. Fourth-generation languages are a case in point. Although some companies consistently achieve high internal productivities with them, others have been much less successful. According to our analyses, certain CASE tools are failing to deliver reduced development time, or fewer errors, or even increased reliability — though it is possible that they increase productivity over the whole life of a system. Maintenance tools have yet to make much impact, though they can prove advantageous in maintenance management and testing, and can be justified when used to maintain systems likely to continue in operation for several years.

Because they are specialised, tools are relatively inflexible. They need to be selected and matched to the application environment with great care. Introducing them requires sensitivity and careful planning, which is best undertaken by implementing a pilot project.

As well as tools for the specialist system builder, a new breed of tools designed for the business user is beginning to make its mark. The systems development department should aim both to encourage and coordinate their uptake.

FOURTH-GENERATION LANGUAGES FOR NEW SYSTEMS WORK

Our analysis of the productivity and the levels of use of fourth-generation languages reveals that most organisations could make significant improvements by further exploiting fourth-generation languages. Although fourth-generation languages account for only 10 per cent of code in the projects on our database, they provide as much as 30 per cent of delivered functionality. Some companies consistently achieve very high productivity levels in terms of delivered functionality (external productivity) through the use of these languages.

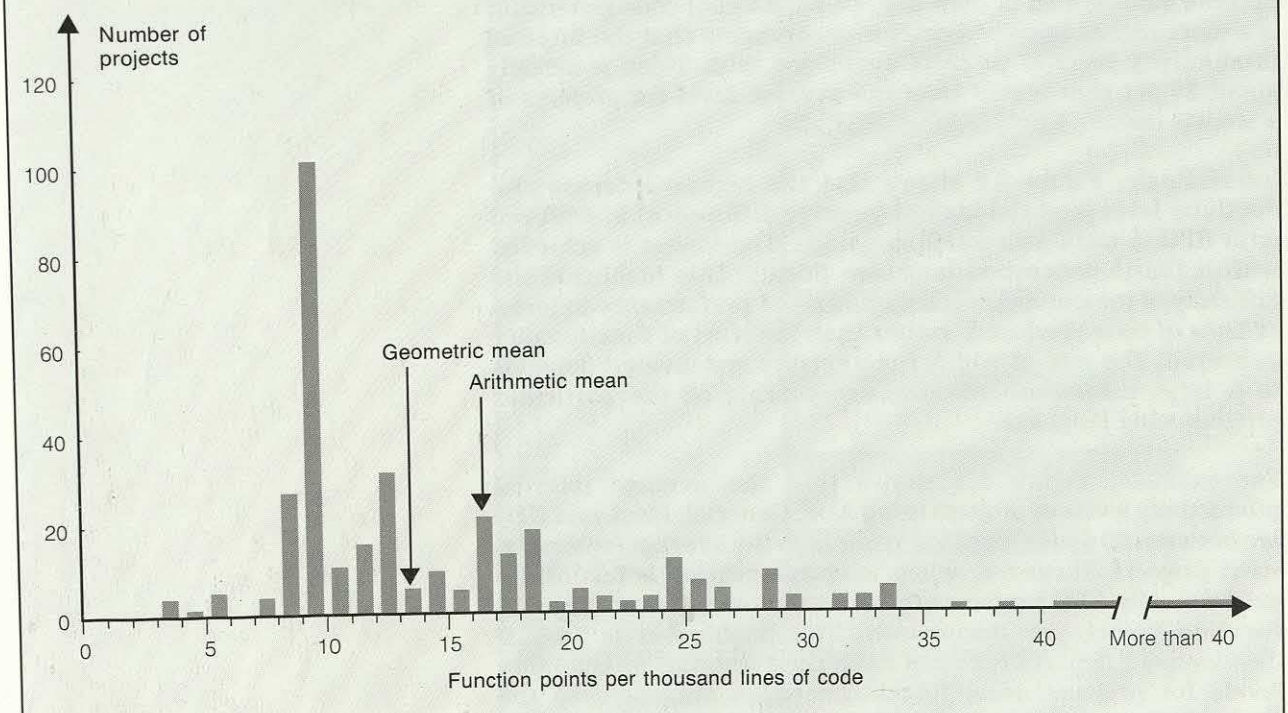
THE USE OF FOURTH-GENERATION LANGUAGES

Fourth-generation languages are syntax-based programming languages in which an application can be written. Fourth-generation languages differ from older languages, such as Cobol, in being more concise (that is, the commands are more powerful), and in not requiring the developer to have detailed knowledge of the underlying computer systems.

Fourth-generation languages are used less commonly than third-generation languages. Forty-four different fourth-generation languages were identified on projects surveyed during our research, but none of them is dominant in the way that Cobol is for third-generation languages. Natural (which comprises 3 per cent of the total code on our database of projects) is the most widely used, followed by ADF, Gener/ol, and Guest (1 per cent each of total code). However, the picture changes when the contribution of fourth-generation languages to total delivered functionality is analysed. Natural and Telon each contribute 4 per cent of the functionality of the projects, Gener/ol about 3 per cent, and SQL and ADF 2 per cent each. The differences are accounted for by the different amounts of function delivered per line of code (language gearing — see Chapter 6) of fourth-generation languages.

Figure 5.1 shows the distribution of the language gearing of the surveyed projects, which varies from four to over 60. The geometric and arithmetic means are about 14 and 17 respectively. There is a pronounced peak at around 10 function points per

Figure 5.1 The language gearing of the projects surveyed varies from four to over 60



(Source: Butler Cox PEP database)

thousand lines of code, and several other minor peaks. The principal peak is associated with projects written mainly in Cobol. The minor peaks coincide with the use of PL/1 and RPG, and the more widely used fourth-generation languages, such as Natural.

THE BENEFITS OF FOURTH-GENERATION LANGUAGES

Although fourth-generation languages help to raise function-delivery rates, the internal productivity of projects using these languages varies widely. Figure 5.2 shows the average internal productivity levels of projects developed with the leading fourth-generation languages. It also shows, for each language, the difference between the internal productivity levels and the average levels for similar-sized projects. The figure also shows equivalent data for the leading third-generation languages and for Assembler.

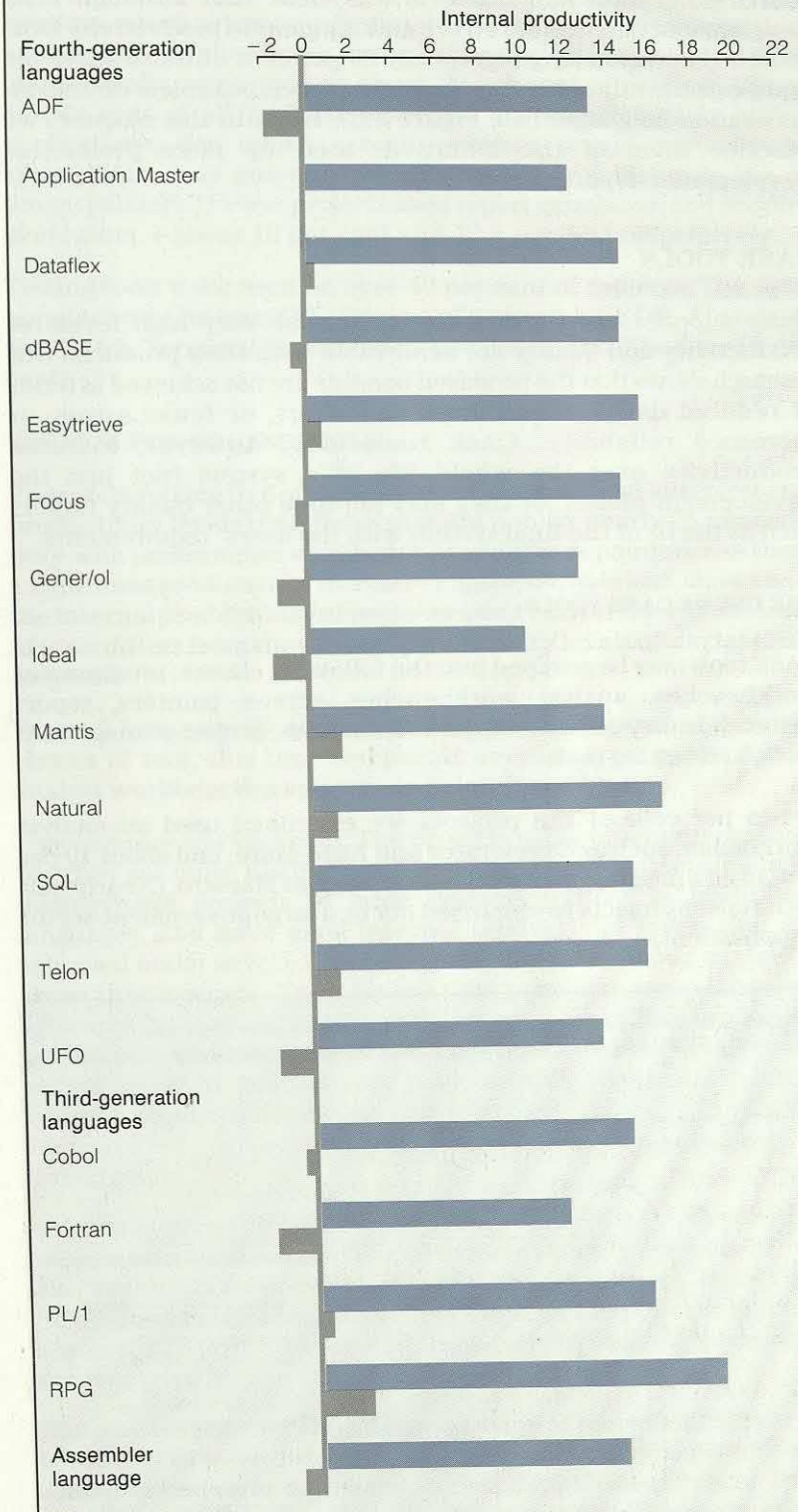
Few projects are developed just with a fourth-generation language, however. Code written in fourth-generation languages is widely scattered among other surveyed projects, and is often found in conjunction with Cobol code. The projects for which the average internal productivity levels are shown in Figure 5.2 are those with a fourth-generation language as the primary or secondary language. Because of the small number of projects using each of the different languages, the data shown in the figure can be taken only as an indicator of the performance of the fourth-generation languages.

Projects using Mantis, Natural, and Telon have average internal productivity levels that are better by over 30 per cent than those for similar-sized projects. Natural and Mantis are fairly well established languages and development staff are likely to have become quite skilled in their use. Telon, a Cobol code generator, is a more recent language, so it is encouraging that the internal productivity level of projects developed with Telon is already about 25 per cent higher than the average level for projects of a similar size.

Interestingly, Figure 5.2 shows that the highest internal productivity level (and the highest positive difference) is achieved with RPG, a third-generation language. The lowest is achieved with a fourth-generation language (Ideal). This highlights the difficulty of measuring overall development performance in terms of lines of code produced, rather than in terms of functionality delivered. The rate at which functionality is delivered depends to a large extent on the language gearing of the particular programming language.

For example, Figure 5.2 shows that the average internal productivity levels of projects using ADF, Gener/ol, Ideal, and UFO are between 25 and 60 per cent lower than the average for similar-sized projects. However, when language gearing is taken into account, only UFO projects are below the overall average of 13 function points per man-month. The implication is that if development departments can raise their internal productivity levels for projects using fourth-generation languages to the average, they will usually be able to increase their function-delivery rate to four times that of typical Cobol projects.

Figure 5.2 The internal productivity of projects using different types of programming language varies widely



(Source: Butler Cox PEP database)

When we examined the levels of external productivity, or functional delivery, of the various development departments using fourth-generation languages, it was clear that although most development departments were making gains in productivity from fourth-generation languages, only 20 per cent of those surveyed were consistently achieving the higher levels possible with fourth-generation languages (see Figure 5.3). Later in this chapter, we describe some of the initiatives used by more productive development departments.

CASE TOOLS

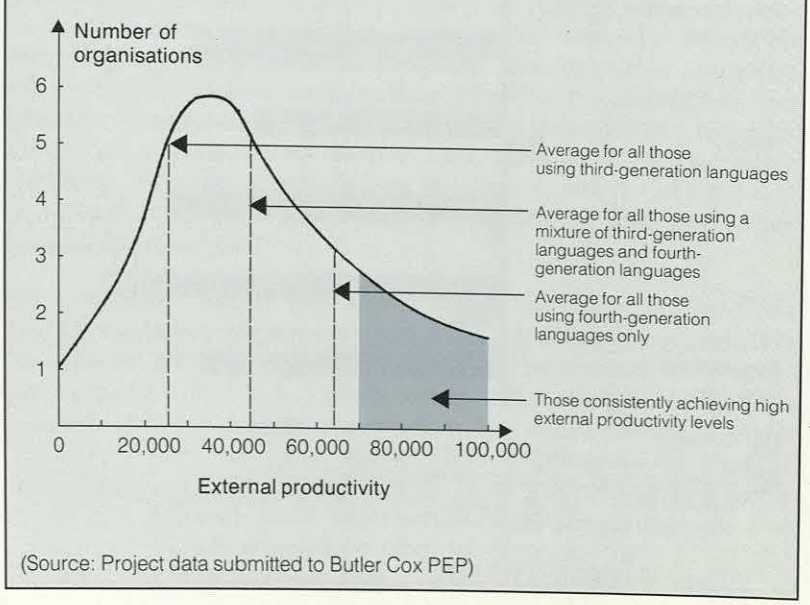
Many suppliers of CASE tools claim that very high levels of productivity and quality are achievable with their products. Our research shows that the promised benefits are not achieved in terms of reduced development times and effort, or fewer errors, or increased reliability. CASE tools may, however, increase productivity over the whole life of a system (not just the development phase), or they may improve other quality factors such as the fit of the final system with the users' requirements.

THE USE OF CASE TOOLS

CASE tools may be grouped into the following classes: programmer workbenches, analyst workbenches, screen painters, report writers, enquiry generators, data dictionaries, project-management tools, and testing tools.

Seven per cent of the projects we examined used an analyst workbench, such as Excelerator and Auto-Mate, and about 10 per cent used a programmer workbench, such as Maestro. (Nearly half of the latter projects were carried out by a large government-sector organisation.)

Figure 5.3 Only 20 per cent of the organisations surveyed are consistently achieving high external productivity



(Source: Project data submitted to Butler Cox PEP)

Data dictionaries were used on nearly 25 per cent of projects, of which more than a quarter used Datamanager.

Painters, which automatically generate code to support transaction-processing applications from screens that are designed interactively, were used on about 20 per cent of projects. (Since 85 per cent of all projects surveyed are categorised as online applications, for which screen painters would normally be appropriate, only one-out-of-four suitable projects made use of screen painters.) Fewer projects used report generators and enquiry generators — about 10 per cent and 12.5 per cent respectively.

Testing tools were used on over 40 per cent of projects. The most popular were Intertest (7 per cent of projects); CEDF, Abendaid, and Batch Terminal Simulator (5 per cent); and Xpediter (4 per cent).

THE BENEFITS OF CASE TOOLS

Projects involving five of the eight classes of tools had lower internal productivity levels than those that did not, by nearly 25 per cent. Only with programmer workbenches and screen painters was there a significant productivity increase. Figure 5.4, overleaf, shows how the internal productivity of projects using each of the eight classes of tools differs from the average productivity levels of similar-sized projects.

Figure 5.5, on page 89, shows a similar analysis for each of the eight classes of tool, this time comparing average error rates. Again, analyst workbenches and screen painters did best.

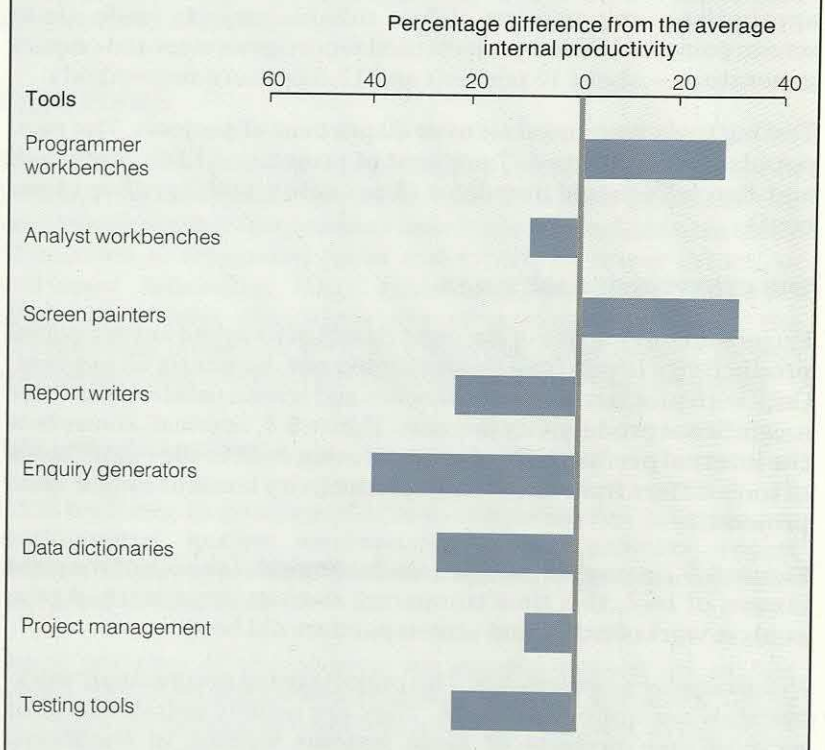
Programmer workbenches: The projects using programmer workbenches are quite distinctive. They are usually enhancement or maintenance projects of large systems written in traditional languages, and have good internal productivity levels that are achieved under severe time pressures. Their new-code content is lower than average. Their severe time pressures mean that they delivered functionality at the very low rate of four function points per man-month (the average from our research is 13). Since most enhancement or maintenance projects have significantly lower internal productivity levels, the relatively good levels of productivity of projects using programmer workbenches is encouraging.

Error rates were available for about two-thirds of the projects using programmer workbenches. The error rates were significantly higher than average both in integration and system testing (70 per cent higher), and in the first month of operation (140 per cent higher). (These high error rates are due, however, to the very high error rates reported by one organisation in the survey.)

Analyst workbenches: The small proportion of projects using analyst workbenches were distinguished by being developments of new, smaller-than-average systems — about 28,000 lines of code. They were also characterised by their higher-than-average fourth-generation-language content. The function-delivery rate for these projects, at 19 function points per man-month, was also above average. Otherwise, their performance was close to average. On average, the internal productivity levels were slightly below the average for similar-sized projects, with the smaller projects having

Figure 5.4 Most projects that use tools have lower-than-average internal productivity

This chart shows the difference in internal productivity level relative to the average for all projects of a similar size. Thus, projects using analyst workbenches have an average internal productivity that is 10 per cent lower than the average for similar-sized projects.



(Source: Butler Cox PEP database)

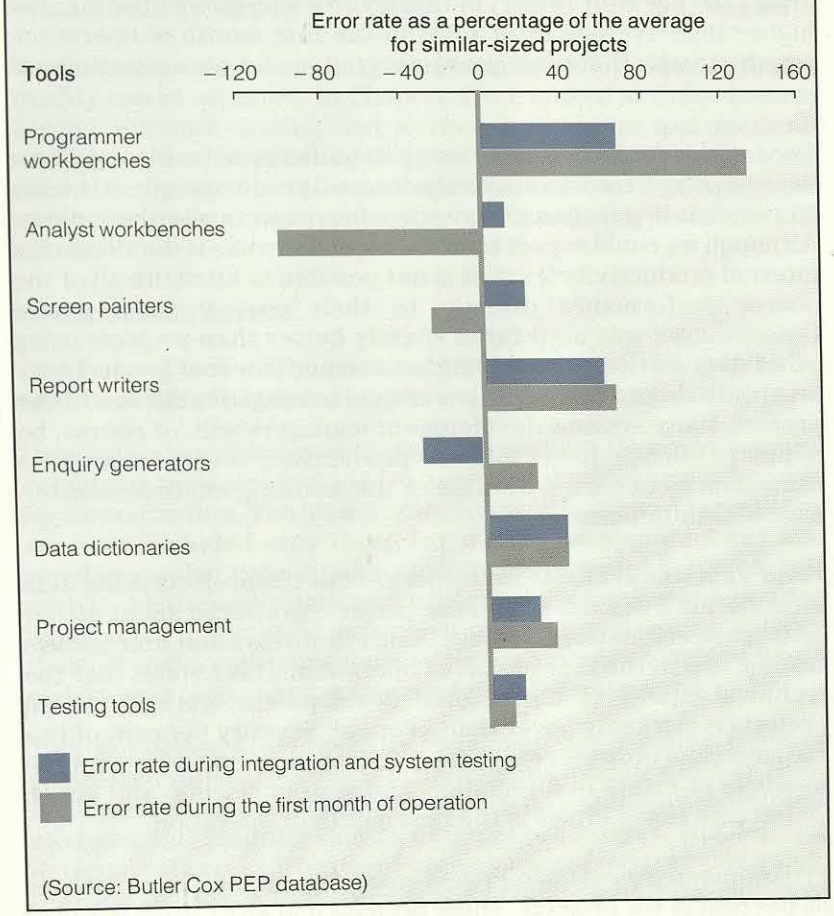
lower levels than larger ones. Like formal development methods, analyst workbenches may also help to produce systems that are a better fit with business needs, but again, the data collected does not at present enable us to measure this.

Error levels for those projects using analyst workbenches are slightly higher (10 per cent) than average in integration and system testing, but substantially lower (100 per cent) than average during the first month of operation. However, error data for the first month of operation was available only for a quarter of the projects using analyst workbenches.

Screen painters: Screen painters are usually associated with fourth-generation languages, particularly code generators such as Telon. The average size of the projects — 49,000 lines of code — is above the overall average. Internal productivity levels are nearly 25 per cent above average for the size of projects and nearly 35 per cent higher than those projects not using screen painters. This translates into at least a 25 per cent reduction in effort, and as much as 40 per cent. The high internal productivity levels and use of fourth-generation languages meant that these projects delivered about 21 function points per man-month.

Figure 5.5 Most projects that use tools have higher-than-average error rates

This chart shows the difference in error rate relative to the average error rate for all projects of a similar size. Thus, projects using analyst workbenches have an average error rate during systems and integration testing 70 per cent higher than the average error rate for similar-sized projects, and an average error rate during the first month of operation 140 per cent higher than the average rate for similar-sized projects.



(Source: Butler Cox PEP database)

Error data was available for 60 per cent of the projects using screen painters. Compared with other projects of similar size, these projects had about 20 per cent more errors in integration and system testing, but about 25 per cent fewer errors in the first month of operation.

Report writers: The projects that used report writers are larger than average — 56,000 lines of code — and have near-average language gearing. The internal productivity levels are lower than the average for the size of project, by over 10 per cent. These projects delivered functionality at a low rate of six function points per man-month, owing to the slightly higher-than-average time pressure.

Enquiry generators: The average size of the projects that used enquiry generators is close to the average. Their internal productivity levels are, however, more than 25 per cent below average for the size of project, and the function delivery rate of nine function points per man-month is nearly 40 per cent below the average.

The projects that used these types of function-generation aids had considerably more software errors than the average during the first month of operation. It is very likely, however, that the higher error rates are not directly associated with use of these aids. Error data was available for 60 per cent of projects using report writers. For their size, these projects had about 60 per cent more errors in integration and system testing and 65 per cent more in the first month of operation. Error data was available for 60 per cent of projects using enquiry generators. They had fewer-than-average errors (30 per cent fewer) in integration and system testing, but higher-than-average error rates in the first month of operation, which suggests that inadequate integration and system testing was carried out.

Data dictionaries: Projects using data dictionaries also perform below average, having an average internal productivity level nearly 25 per cent lower than the average for projects of a similar size. Although we could expect their use to lead to some reduction in the internal productivity level, it is not possible to attribute all of the poorer performance directly to their use. Projects where Datamanager was used fared slightly better than projects using other data dictionaries, having an average internal productivity level only about 10 per cent lower than average for the size of the project. Many systems development managers will, of course, be happy to tolerate lower internal productivity levels for projects using data dictionaries, because of the resulting improvements in ease of maintenance.

Error data was available for 60 per cent of the projects using data dictionaries. For their size, these projects produced about 40 per cent more errors than average, both in integration and system testing, and in the first month of operation. This implies that the technical quality of the original development work for these projects is markedly lower than average. Seventy per cent of the Datamanager projects reported error data; they had near-average numbers of errors in integration and system testing, and about 20 per cent fewer errors in the first month of operation.

Project-management tools: Project-management tools were used on 35 per cent of the projects. These projects had an average internal productivity level slightly lower than the average for projects of a similar size.

Projects using project-management tools have error rates 20 per cent above the average (for the size of project) during integration and system testing, and 25 per cent above average during the first month of operation.

Testing tools: Testing tools are used on about 40 per cent of the projects. These projects have lower internal productivity levels (by more than 10 per cent) than projects of a similar size. The main reason for using such tools, however, is to improve the technical quality of systems.

The projects using testing tools had slightly higher error rates both in integration and system testing (15 per cent higher), and in the first month of operation (10 per cent higher). Thus, although the tools may have helped to identify more errors, the reliability of the

developed applications in the first month of operation was marginally worse than average.

Measuring and analysing the benefits associated with tools, either productivity or quality, provides a means of focusing attention, and hence, effort, on the areas of greatest benefit. It is then the responsibility of the management to implement initiatives to increase the levels of benefit.

TOOLS FOR TESTING SYSTEMS

Testing is an area where improvements in both productivity and quality can be attained. In Chapter 4, we looked at the process of formal software testing and various techniques and methods underlying this. Supporting this process, and the techniques and methods, are various tools and aids that assist in conducting and managing testing, and generating and analysing test data and results.

CONDUCTING TESTING

Two types of tool — debuggers and test harnesses — are used during module testing to help in the process of debugging and testing:

Debugging is a distinct activity from testing. However, several debugging tools also include features that enable them to be used for formal testing. The list in Figure 5.6, overleaf, is not exhaustive; we have included only those for which the manufacturer also supplies another type of testing tool that can be used in conjunction with it. All the organisations we interviewed use some debugging tools, since it would be expensive to develop programs without them. If additional debugging tools are required, it would be worth considering tools that are also useful for formal testing.

Test harnesses provide an environment for running partially completed software when it is undergoing module tests, or being debugged. They provide facilities such as simulating incomplete modules, intercepting calls to external procedures, and defining external data areas. The use of such a harness could provide a more uniform approach to module testing throughout a development team.

MANAGING TESTING

Test-management tools help in the management of the tests rather than in the process of testing. They are particularly useful if there are many test cases to manage, and as a long-term investment in maintaining the test environment for regression testing.

Companies that are already controlling their software developments using tools for code management and configuration management should consider extending their scope to include testing. Companies that are not already using such tools would be well advised to consider investing in them.

TEST-DATA PREPARATION AIDS

Tools to ease the process of creating and using test data provide one or more of four functions: capture and playback of test scripts,

Figure 5.6 There is a wide range of testing tools available in the United Kingdom

The tools in this list have been selected from those that are obtainable and that are supported in the United Kingdom. Inclusion in this list does not indicate an endorsement of the product. The criteria for inclusion are that the product should be supported on Digital, IBM mainframe, or ICL computers and that Cobol or PL/1 should be supported on language-dependent products. Some static-analysis products, for example, have been excluded because they are aimed at military systems, and languages such as Coral and Ada.

	Capture/playback	Test data generation	Test database generation	Comparison	Static analysis	Dynamic analysis	Debugging	Test management
ABL Europe Ltd — TIP	✓						✓	
Advanced Programming Techniques Ltd — Oliver						✓	✓	
— Simon						✓	✓	
C A Computer Associates Ltd — CA-Datamacs/II		✓						
— CA-EZTest/CICS							✓	
— CA-Optimiser						✓		
Compuware — CICS Playback	✓			✓				
— File-aid			✓					
Digital Equipment Corporation — Dec Test Manager				✓				✓
Gerrard Software Ltd — Testgen		✓						
IPL Software Products Ltd — Softest		✓		✓				✓
John Bell Technical Systems — Pro-Quest					✓	✓		
— Testa	✓							
On-Line Software International — Datavantage		✓	✓					
— InterTest		✓					✓	
— ProEdit			✓					
— Verify	✓			✓				
Program Analysers Ltd — Testbed					✓	✓		
QA Training Ltd — Evaluator	✓			✓				✓
Rand Information Systems Ltd — Testline						✓		✓
Sterling Software — Comparex				✓				
Verilog UK Ltd — Logiscope					✓	✓		
XA Systems UK — Pathvu				✓				

test-data generation, test-database generation, and file and output comparison.

Tools providing these functions are mainly used during system testing and during the maintenance phase of a project, where 'regression' tests are carried out to check that the system's behaviour has not changed unexpectedly as a result of maintenance activity.

The main benefit provided by these tools is the automation of tasks that would be tedious and time-consuming to carry out manually. In some cases, the amount of test data required to carry out a satisfactory range of tests would be so large as to preclude a manual approach; the implication of this is that some systems cannot be adequately tested without the use of such tools. The use of these tools does not in any way, however, reduce the need for careful test design. The function of the tool is simply to automate the process of generating test data within the parameters defined by the test design.

An important consequence of automating a tedious manual task is the increase in accuracy that is achieved. If each piece of test data is designed to test a particular function, any inaccuracy in the creation of test data is likely to mean that some functions are not tested as the designer intended.

Capture and playback tools

These tools record users' inputs and system responses. The user input can then be replayed, and the system responses compared. These tools are particularly useful for testing online systems with significant amounts of data entered by users via screen-based systems. They greatly simplify the creation of test scripts, and can save the cost of employing large numbers of unskilled staff to type in the data. They have their main value during system testing, but could also be used very effectively during the module and integration testing of those parts of the system that handle the user interface. The tools run either on the host computer, or on a PC that emulates a terminal on the host computer.

The tools contain some or all of the following components:

- Capture and recording of all the user's inputs, including mouse movements, where these are used. This input is stored as a script, which can be edited if required.
- Recording of responses generated by the system.
- Editing capability on the captured input data.
- Replay of the captured (and edited) script at varying speeds.
- The ability to run multiple copies of the script or scripts on 'virtual' visual display units.
- Comparison of the system-generated responses between different runs of the script, and documentation of the results.

In using a capture/playback tool, each script must be designed to test particular features of the system. If the tool is used merely to capture a large amount of unplanned user input, very little benefit will be gained.

Capture/playback tools can facilitate tests that would otherwise be very difficult to carry out. A good example is stress testing — subjecting the system to large volumes of test data, or to high transaction rates. This particularly applies to systems, such as ticket-reservation systems, which have large numbers of user terminals. In the test environment, a large number of terminals will almost certainly not be available, and even if they were, organising large numbers of staff to simulate the expected volume of user inputs would be difficult and expensive.

These tools also have particular benefits during regression testing because they allow a script of input commands (including deliberate user errors) to be repeated precisely. To compare the results of the original test and the test of the modified system, differences have to be sought on possibly hundreds of screens. Most capture/playback tools can do this rapidly and without error. In addition, much less effort is required to carry out regression tests since the reruns of the script can be carried out in batch mode without supervision.

Test-data generation tools

These tools facilitate the automatic creation of large files of data. They are particularly useful for testing systems that process large volumes of data in sequential files. A comprehensive test of such systems generally requires each record in the test-data files to be different from all other records. To generate such files manually would be very time-consuming and prone to error. It would, of course, be possible to write a separate file-generation program for each system that is developed, but it is likely to be more cost-effective to purchase a data-generation tool if systems developed by the organisation often use sequential input data files.

Data-generation tools use at least two methods to generate the files. One is to define the file off-line using a special programming language. The other is to generate files from within the program itself, by embedding control statements in the program. These statements either generate new records, or select and modify records from existing files.

The generated files contain records in user-defined formats. The tools allow the values of fields in successive records to be generated in various ways — random numbers within a specified range, values clustered about a specified point, sequential values, dates in various formats, and so on. These features allow the test designer to include data to test for particular conditions such as data on, or either side of, boundary values, and also to generate large numbers of different records which may be used for volume testing.

Test-database generation tools

These tools provide a means of testing a system on a realistic database without risking the live database. It is not usually advisable to use the live database for system testing, but even where it is possible, it may be more convenient to use a smaller and more easily monitored subset of the live database.

Comparison tools

These tools are the only ones that operate on the outputs from the system under test. Systems that generate significant amounts of output in the form of files can benefit from the use of these tools. They can save time and improve accuracy when the tester is looking for small differences between runs of a test program, or when he is comparing expected results with actual results. They can be particularly valuable in the maintenance phase of a system's life cycle, where it is essential to verify that a correction or a change to a program does not have unexpected side effects. The tools produce printed reports that management can use as an objective measure that the system has not been degraded by the change.

File-comparison tools identify records within a file that have been inserted, deleted, or modified. The ability to make comparisons in this way is usually included as a feature of capture/playback tools. In these instances, comparisons are made of outputs sent to a display screen by the system. Some file-comparison tools are included in manufacturers' operating systems — for example, the 'Difference' command in Digital's VMS operating system.

TOOLS FOR MAINTAINING EXISTING SYSTEMS

In addition to fourth-generation languages and CASE tools for generating software for new systems development and testing tools, a range of tools is emerging designed to assist with the maintenance of existing systems. These tools fall into two categories: management tools and testing tools (covered above), and maintenance-support tools.

THE USE OF MAINTENANCE TOOLS

Although software tools theoretically help to simplify maintenance by improving the development process, they have yet to make much impact in this area. Most systems that are currently being maintained were developed using Cobol, as Figure 5.7 shows. Of 24 companies in a snap survey by Butler Cox, nine stated that over 90 per cent of their maintained code was written in Cobol, and another six reported that at least 70 per cent of their maintained code was in Cobol.

Figure 5.7 Most systems that are currently being maintained were developed using Cobol

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 members)

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 such as Mapper and Application Master.

MANAGEMENT TOOLS AND TESTING TOOLS

Although not directly concerned with the task of maintenance, maintenance-management tools and maintenance-testing tools have an important complementary role to play.

Maintenance-management tools

Management tools help to improve the planning and control of maintenance. Tools in this category are of 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 the introduction of successive versions of software. 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.

Maintenance-testing tools

Several of the testing tools described earlier are available to help with the maintenance task. They provide source and file-comparison facilities, cross-reference analysis, code analysis, and test-data operation facilities. The purpose of such tools is to provide enhanced status reporting, auditing, and quality assurance, and to improve the efficiency of the testing process. With a testing environment supported by techniques, methods, and tools, 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 that 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.)

MAINTENANCE-SUPPORT TOOLS

Maintenance-support tools are 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 relatively expensive, maintenance tools can cost less than renewing the system. They can be justified when 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 5.8.

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 to the maintenance function staff 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

Figure 5.8 There are several kinds of maintenance support 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

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.

Code analysers can also be used to measure the technical quality of a system. Their use in the area of quality measurement is discussed later in this chapter.

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 re-organisation 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 staff 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 $6 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 relatively 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 often prove 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

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 (this is a process known as forward engineering).

Today, there are several products on the market, such as Pacreverse from CGI Systems, and Revengg from Advanced Systems Technology. Although these tools hold great promise in reducing the maintenance effort, it is not clear yet how much of the 'engineering' is carried out automatically by the tool and how much requires human assistance.

Both maintenance-oriented tools and fourth-generation languages help to improve either the productivity or the quality of applications development. To measure these improvements, current levels of productivity and quality need to be known. Although most development departments have the basic information required to measure the benefits associated with any particular tool, very few actually carry out any analysis of this data. There are many reasons for this, ranging from lack of time (a false economy), to a lack of expertise. Although we have not come across a tool that helps measure productivity, there are tools that help measure the technical quality of applications. These tools can prove very effective in measuring quality and highlighting areas that need further work to improve the quality of a particular application.

TOOLS FOR MEASURING TECHNICAL QUALITY

Two types of tools are available to help analyse the quality of the code within a system. Called static analysers and dynamic analysers, they are suitable for use with commercial and scientific programming languages such as Cobol, PL1, Fortran, and C. Code written in manufacturer-specific languages, such as Tandem's TAL, or in fourth-generation languages, cannot be analysed by these tools today.

STATIC ANALYSERS

Static-analyser tools examine the structure of code without running it, and can typically find between 10 and 20 per cent of all errors in a program. They are cost-effective tools in the development of reliable systems, but do not seem to be widely used in the commercial environment. Much of their use has been in avionics and military systems. There are considerable benefits to be gained from the use of these tools in commercial applications, however, and their use should be considered carefully by all development departments.

The tools provide managers with objective measurements of characteristics that are directly related to quality. These help to identify areas of poorly structured or excessively complex code. It is advisable to redesign such code before proceeding further with testing. If part of the code is unavoidably complex (a complex logical algorithm, for example), extra attention should be paid to its module testing since it is likely to contain an above-average number of errors.

Static analysers typically assess the following characteristics of the code:

- Conformance to user-specified standards (for example, no more than a predefined number of lines in a module, or no use of 'go to' statements).
- The paths (or different sequences of instructions) through a program.
- Complexity analysis. Two of the most widely used measures of complexity are McCabe's measure, and the number of knots, explained in detail in Figure 5.9.
- Data-flow analysis, showing procedure calls, use of procedure parameters, and unreferenced or unused data items.

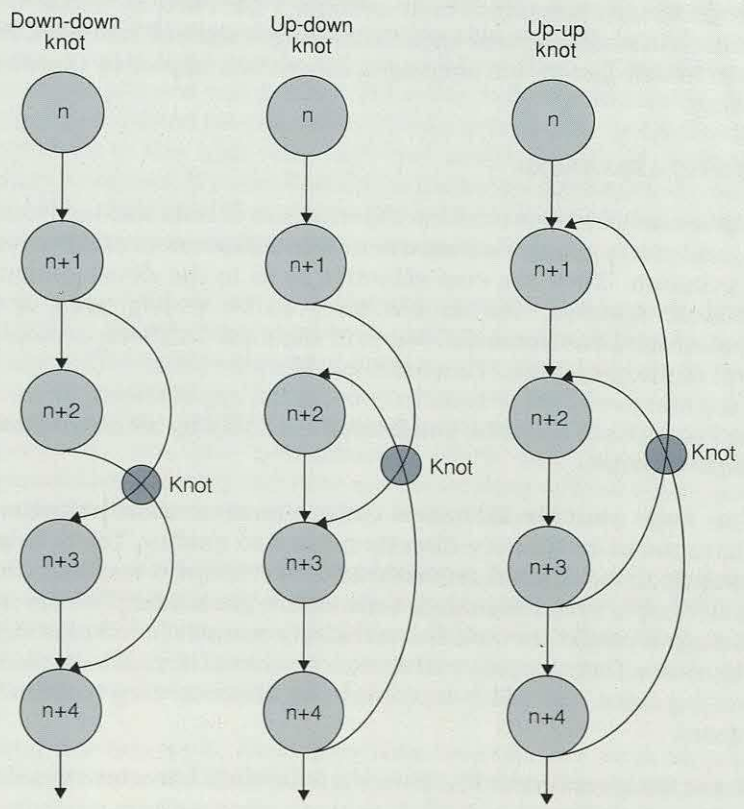
Figure 5.9 Two of the most widely used measures of the complexity of a program are McCabe's measure and the number of knots

McCabe's measure

McCabe's measure is defined as one more than the number of decision statements in a program. The metric is very simple, but experience shows a significant correlation between McCabe's measure and the number of bugs, or debugging effort applied to a program. Programs with a McCabe value in excess of 10 seem to have disproportionately more bugs than those with values of less than 10.

Knots

The purpose of looking for 'knots' is to identify unstructured code, which tends to contain more errors than properly structured code. A control-flow knot is defined as occurring when two control jumps cross, as illustrated in the diagram. Three types of knots are depicted. A down-down knot is relatively harmless, and represents an 'if ... then ... else' construct. Up-down knots are more likely to represent unstructured code, but may arise from 'do' or 'while' loop constructs. Up-up knots always represent unstructured code.



— Cross-referencing of all data items.

Some training is required in the interpretation of these measures, but carefully used, they can identify many coding errors before any attempt is made to run the code. This obviously saves effort and machine time.

DYNAMIC ANALYSERS

Dynamic-analyser tools provide objective measures of the testing procedure, commonly known as 'white-box' testing. This procedure is carried out as part of the module-testing phase, and may also be done during integration testing. The tools monitor the code while it is being executed, and produce a report at the end of the execution, giving various statistics. These statistics can be used to assess the effectiveness of the test cases.

It is essential that white-box testing is carried out, since in a typical program, over 50 per cent of the code is not directly related to end-user functionality, but to the manipulation of internal pointers, flags, and intermediate results. 'Black-box' testing, which views the system externally in terms of inputs and outputs, cannot be designed to guarantee complete coverage of this 'hidden' code.

One tool, Testbed, provides three measurements, or test-effectiveness ratios (TERs), resulting from a dynamic analysis of the code. Other dynamic-analysis tools, which are sometimes also known as coverage analysers, provide at least the first measurement. These measurements are:

— TER1 — statement coverage analysis: the percentage of the lines of code that have been exercised at least once. No operational system should be released where this measure is less than 100 per cent.

— TER2 — branch coverage analysis: the percentage of all outcomes of branch instructions that have been exercised at least once. The goal of testing should also be 100 per cent, although this is not as easy to achieve as 100 per cent on TER1.

— TER3 — path coverage analysis: there are several ways of measuring paths through a system, all of them quite complex. It is difficult, in practice, to achieve 100 per cent path coverage during testing, and except in ultra-high-reliability systems, it is probably not worth attempting it.

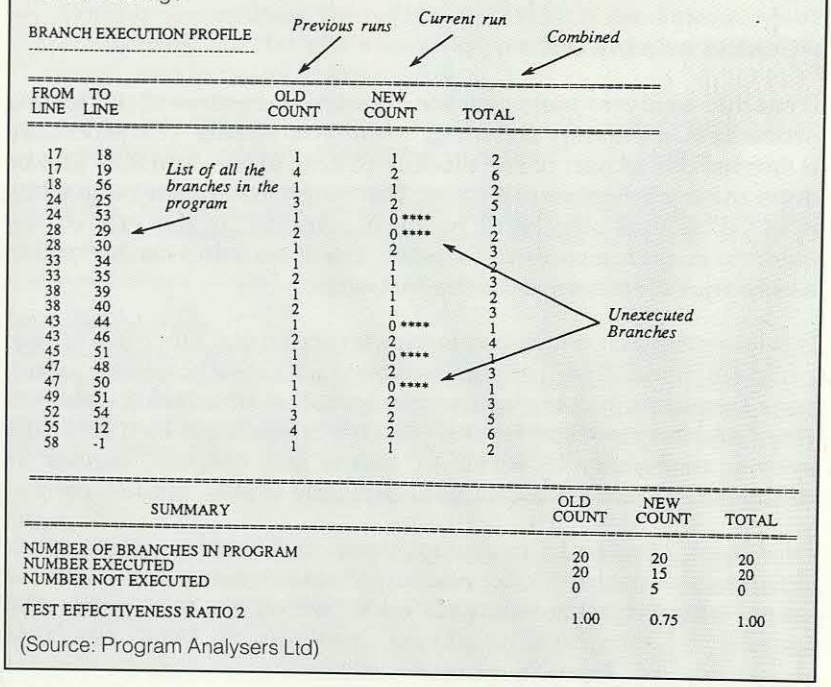
An example of output from Testbed for TER2 is shown overleaf in Figure 5.10.

An experiment carried out on a large military system in the United States showed that, when static analysis was combined with dynamic analysis, 70 per cent of all errors in the system were discovered. The remaining errors were caused largely by errors in the specifications or misunderstandings of the written requirements.

Project managers will find the use of this type of tool particularly helpful because of their ability to provide reports containing objective measures of progress, such as *"tests covering 78 per cent of statements and 64 per cent of branches have been successfully*

Figure 5.10 The testing tool, Testbed, provides three measurements resulting from a dynamic analysis of the code

The diagram is an example of output from Testbed for the second of the measurements, or test effectiveness ratios (TERs), that it provides. TER2 is an analysis of branch coverage.



(Source: Program Analysers Ltd)

completed''. This gives project managers much better control over a project than having to rely on a programmer's typical estimate that *''testing is 95 per cent complete''*.

SELECTING TOOLS FROM THE TOOL SET

Tools should not be selected for use on an application in isolation. Instead, they should be selected in the context of the wider development environment in which they are to be used, and in the light of the development application they are aimed at.

Serious problems with tools are commonplace, frequently as a result of the selected tool's inability to develop the required applications fully. Problems of this sort are rarely attributable to a shortcoming in the tool itself, but instead to a mistake in selection. If their potential is to be fully realised, tools should be chosen and integrated into the development environment with due regard to the relationships between the application in question, the development approaches available, and the systems development techniques and methods that can be used. Failure to observe the critical nature of these relationships will produce a less-than-adequate application, and loss of confidence in the tools.

THE NEED FOR A TOOL-SELECTION PROCEDURE

Contemporary tools cannot be used for all types of applications, and the range of applications is increasing. It follows that a range of tools is required, each matched to the characteristics of one or more

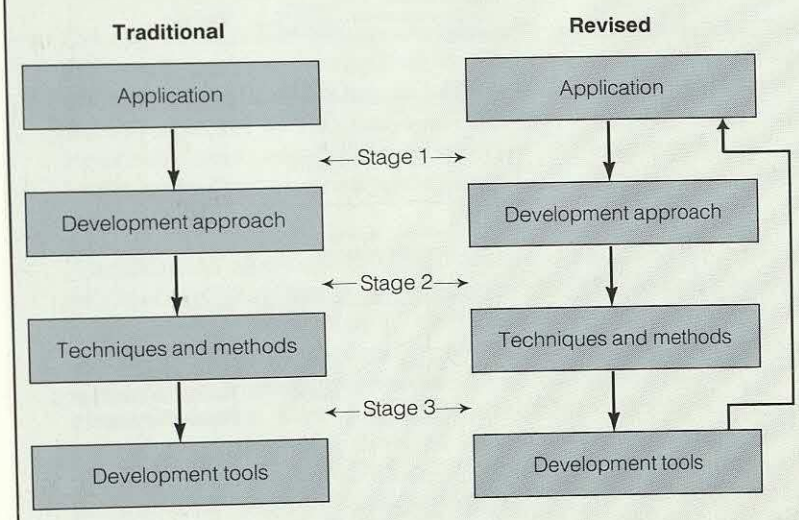
applications, if the full potential of the tool is to be realised for a particular development project.

The need to select the right tool for an application rarely arose with traditional tools such as third-generation languages because they could be used to develop most types of application. The traditional procedure for selecting an appropriate third-generation language is illustrated in the first column of Figure 5.11.

An analogy can be made here with house building. Using the more traditional tools, such as third-generation tools, was equivalent to building a house as a traditional craftsman would do, designing and building each component from basic materials. Using modern tools is equivalent to building a house by using prefabricated components, such as windows, doors, and wall panels, as the basic elements. As with modern house-building techniques, modern tools certainly enable the final product to be built much more quickly, but unless the right set of components is selected, it will not be possible to build the application according to the original design.

Most organisations are using the procedure that was developed for third-generation languages to choose which modern tools to use for an application. In our analogy, this is comparable to selecting the prefabricated components without considering what type of

Figure 5.11 The revised procedure for selecting development tools ensures that the capabilities of the tools match the characteristics of the application



Stage 1
The type of application determines which development approach should be adopted

Stage 2
The development approach is supported by various techniques and development methods

Stage 3
The techniques and methods are supported by various development tools

Stage 1
The type of application determines which development approach should be adopted

Stage 2
The development approach is supported by various techniques and development methods

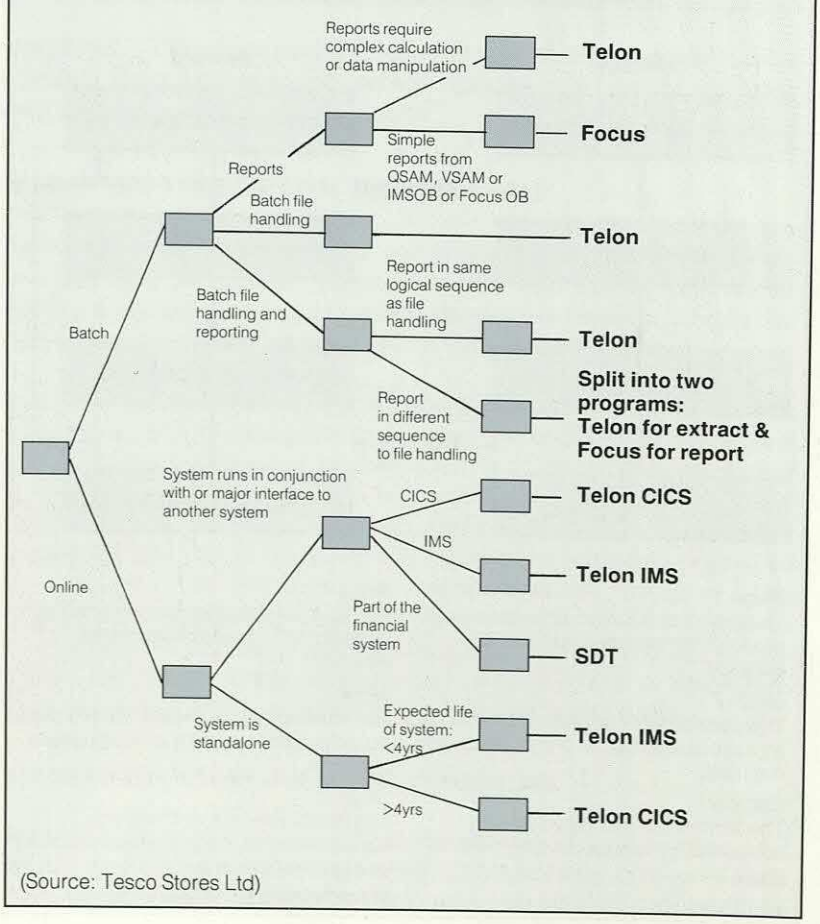
Stage 3
The techniques and methods are supported by various development tools, whose capabilities match the characteristics of the application

building is to be constructed — an apartment, a house, an office block, or a hospital.

A proper selection procedure should ensure that the contemporary tool not only supports the systems development techniques and methods, but also that it is able to develop the required application. This formal procedure, which is contrasted with the traditional procedure in Figure 5.11, should be adopted for selecting all tools. If it is implemented correctly and updated regularly, it will deliver an effective match between the development environment and the application. This should ensure that all projects developed with these tools are successfully completed.

The procedure adopted by Tesco Stores Ltd, a supermarket chain, is a case in point (see Figure 5.12). Tesco uses three main development tools — Telon, Focus, and Cobol. Cobol is used mainly for the maintenance of existing applications. SDT, a fourth-generation language from McCormack and Dodge, is used for financial systems. The company has clear guidelines for deciding which development tool should be selected for a particular application. These guidelines, accompanied by detailed instructions, are issued to developers in a document entitled, *The Development Language Selection Criteria*. This document gives the reasons for selecting a particular language, and two diagrams, one for new applications

Figure 5.12 Tesco has a procedure for selecting appropriate development tools for each new application



(Source: Tesco Stores Ltd)

and one for maintenance, indicating which development tools will be appropriate for applications with certain characteristics.

DEFINING THE ELEMENTS OF THE DEVELOPMENT ENVIRONMENT

Before the procedure for selecting the application-development environment can be implemented, the elements need to be defined and the definitions documented so that everyone involved in the selection procedure has the same basic understanding of the application-development environment. This document is used whenever a development project, either maintenance or new, is started. The elements that need to be defined can be grouped into four categories:

- *Development approaches:* The complete cycle, phases, and activities of the development of an application. All the major development approaches used within the organisation should be briefly described, with an explanation of the objectives and the actions required at each phase. It should be clear to the reader how the phases flow from one to another.

- *Systems development techniques and methods:* The techniques are the procedures on which systems development methods are based. All the systems development techniques and methods available within the organisation should be briefly defined, and associated with the development approaches that they support. Several development approaches may be supported by a single technique or method, and several techniques and methods may support one phase of a development approach.

- *Development tools:* The tools, typically computer-based, that automate parts of or support the development methods and techniques. All tools should be identified and described in a similar manner to the systems development techniques and methods, and associated with the various methods and techniques that they support. Again, there may be multiple associations.

- *Application characteristics:* To ensure that the tool that is selected will facilitate the efficient and successful development of the application, the nature of the applications developed by an organisation needs to be clearly understood. The nature of an application can be defined in terms of a set of characteristics. Each application can be described in these terms, and hence, be defined in a consistent manner. Most applications can be defined for this purpose in terms of between 10 and 20 characteristics, relating primarily to the development approaches and the tools currently used by a particular organisation. Examples of the kinds of characteristics of an application that will determine whether or not it is a suitable candidate for a particular development approach are listed in Figure 5.13, overleaf. Examples of the kinds of application characteristics that will determine whether a particular development tool is appropriate are listed in Figure 5.14, also overleaf. These lists can be amended and supplemented to suit an individual organisation.

  The characteristics must be clearly defined so that they will be consistently interpreted by different readers. They should not

Figure 5.13 Every application has characteristics that will determine the suitability of using particular development approaches

Scope of the impact	A measure of the impact of the application throughout the organisation. This could range from company-wide applications to personal systems.
Clarity of the definition of users' requirements	This could range from well defined and easy to understand, to poorly defined and difficult to understand.
Urgency	A measure of the urgency of the development of the application, and of the deadline for installing it.
Number of locations	The number of geographical locations or sites.
Complexity	A measure of how difficult the application will be to develop, in view of its complexity.
Security	The level of security that the application must provide for access to the application itself and to the data.
Audit requirements	The level of audit that the application must provide. This could range from none, to very high (for financial systems).

Figure 5.14 Every application has characteristics that will determine the suitability of using particular development tools

Application type	Definitions should reflect the type of application rather than the business area — for instance, transaction processing rather than financial systems.
Level of integration	A measure of the level of integration expected between this application and other application types, databases, and machine environments. This could range from none, to numerous and very complex.
Performance requirements	A measure of the required performance of the application. Some may require instant response times; for others, response times will be less critical.
Type of development	An indication of whether the application is a modification, an addition, or an enhancement.
Level of portability	A measure of the portability of the developed application. This could be across different machine configurations, or across the machines of different manufacturers.
Likelihood of enhancements	The expected time from first installation to the first major enhancement.
Volume of data	An estimate of the total volume of data.
Security	The level of security that the application must provide for access to the application itself and to the data.
Complexity	A measure of how difficult the application will be to develop, in view of its complexity.
Size	An estimate of the total size of the application.
Expected life	An estimate of the life of the application.
Interface with end user	The degree of familiarity that the users will have with the system.
Flexibility	A measure of the likely extent and frequency of change.

be too detailed or too technical, because they need to be kept to a manageable number. The lists should be amended as the development environment evolves. They provide the basis for the preparation of the selection tables described in the next section.

The level of definition and the number of definitions in each category will vary from one organisation to another. The definitions should be reviewed on a regular basis to take into account the evolutionary changes in the development environment, and the introduction of new types of applications, development approaches, and systems development techniques, methods, or tools.

PREPARING SELECTION TABLES

Two tables need to be prepared to serve as the basis for matching the development approach and the tools with the application. Their structure and the kinds of information they should contain are described in this section. Once prepared, the tables can be used for any development project. They are based on the lists of characteristics described above, with input from experts in the areas of development concerned. They will, of course, need to be updated periodically to reflect changes in the development environment.

Both tables are organised in a grid format and used in a similar manner. The first is used to select the development approach for a specific application. The second is used to ensure that the tools selected will enable the required application to be developed.

An example of part of the table for selecting tools is shown in Figure 5.15. The application characteristics are listed on the left-hand side of the table; the types of tools are listed across the top. The application characteristics should be listed, as far as possible, in order of importance. A maximum score should be assigned to each of the application characteristics to indicate their relative importance — say, 20 for the most important characteristic, and five

Figure 5.15 Use of the development-tool selection table ensures that appropriate development tools are used for each application

Application characteristics	Development tools available		
	Cobol	Focus	Telon
Expected life (20)			
Less than one year	10	15	20
Between one and three years	7	10	20
Over three years	5	15	20
Performance requirements (20)			
Very high (interactive)	20	15	15
High (time-critical)	15	15	15
Medium	10	20	20
Low (not time-critical)	10	20	20
Volume of data (15)			
Less than 5 megabytes	10	15	15
More than 5 megabytes	12	5	13

for the least important. Each application characteristic is broken down into a range of options, each of which receives a score. In Figure 5.15, for example, the 'expected life' of the application is considered one of the most significant characteristics and is given a maximum score of 20. This is broken down into three options — 'less than one year', 'between one and three years', and 'over three years'.

The numeric value entered onto the grid is an indication of the ability of the tool to develop an application that supports that option. If a particular development tool can fully support that option, it receives the maximum score for that characteristic. If it provides adequate support, it receives a lower score. If it provides no support, it scores zero. If, for example, the expected life of the application being considered is over three years, the application needs to be developed bearing in mind the continuing support that the tool will be able to provide, and the ease of maintenance of the application over the longer term. The capability of each tool to develop such a system is considered in turn. Cobol scores five as it is not the strategic development tool for this organisation, and it produces applications that are not the easiest to maintain. Focus scores 15; although it is not the strategic development tool either, it does produce applications that are easy to maintain. Telon is the strategic development tool and produces applications that are easy to maintain; it scores the maximum of 20.

The procedure for compiling the selection tables is summarised on the left-hand side of Figure 5.16. This part of the procedure is carried out only once, before the process is initiated. Once the selection tables have been finalised, they should be tried out on several recently completed developments. This should reveal any errors in the selection tables, and also demonstrate how well the developments were supported by the development approach, and systems development techniques, methods, and tools chosen. Modifications to the selection tables should be made as and when appropriate.

PREPARING DOCUMENTATION

The elements of the selection procedure should be fully documented, and regularly updated. The documentation should consist of:

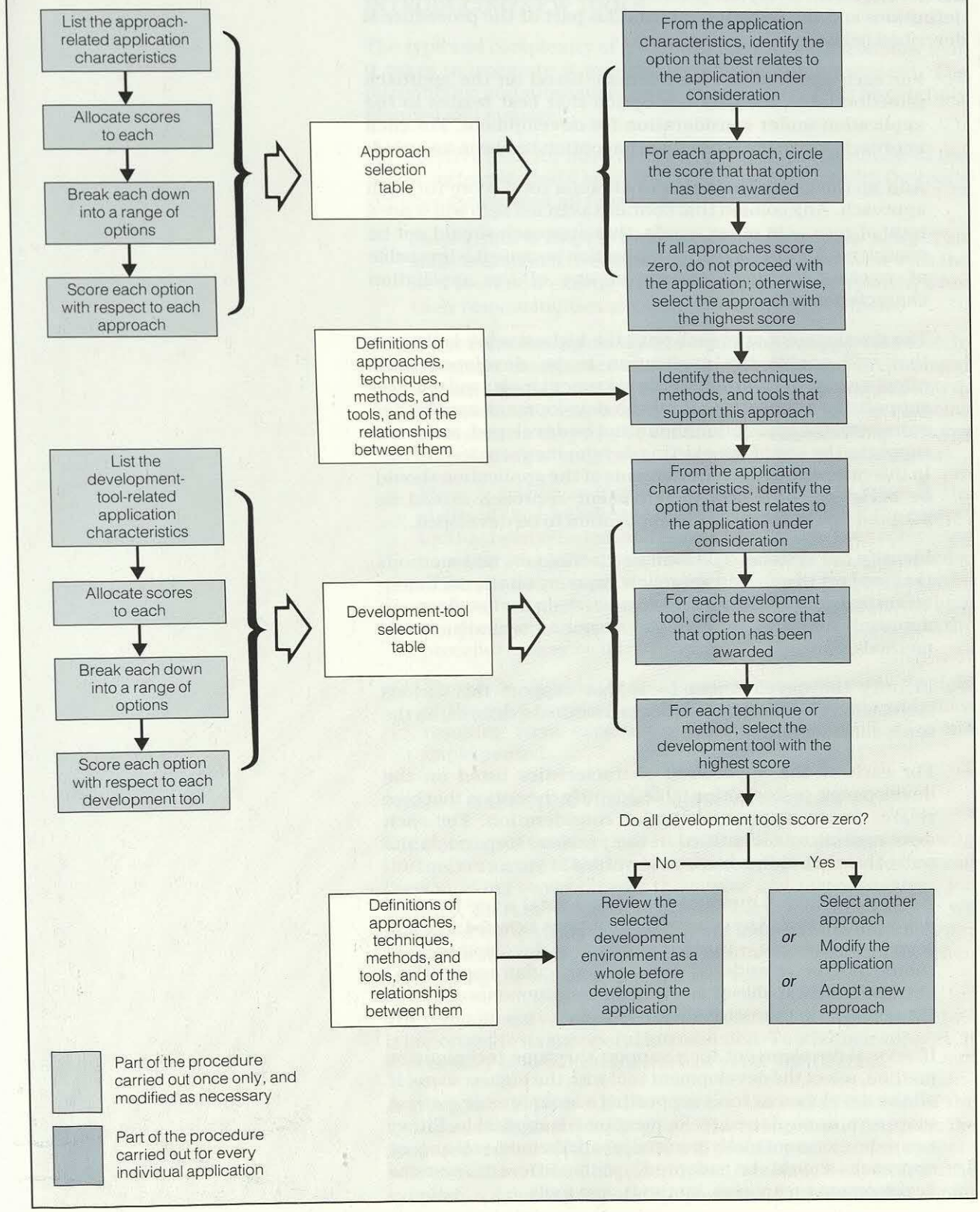
- The definitions of, and relationships between, the various development approaches, and systems development techniques, methods, and tools.
- The approach and development-tool selection tables, and the instructions for their use.

All the comments and decisions made during the process should also be documented. If a development should subsequently fail, the appropriate part of the definitions or selection tables can be amended by referring to the documentation. In this way, mistakes will not be repeated.

MAKING THE PROCEDURE PART OF THE DEVELOPMENT PROCESS

The remainder of the selection procedure, illustrated on the right-hand side of Figure 5.16, should be carried out at the beginning of

Figure 5.16 Elements of the development environment should be selected according to a formal procedure



each development project and is best done at a meeting attended by one or two users, the internal technical experts, and several of the systems development department project managers, all of whom will contribute from their experience, and one of whom will manage the development. Everyone present should be acquainted with the definitions and the selection tables. This part of the procedure is described below:

- For each application characteristic listed on the approach selection table, identify the option that best relates to the application under consideration for development. For each approach, circle the score that that option has been awarded.

- Add up the circled numbers to obtain a total score for each approach. Any column that contains a circled zero will score a total of zero — in other words, that approach should not be considered for this particular application because it is incapable of meeting the requirements of one of the application characteristics.

- The development approach with the highest score is the one that will enable the application to be developed most effectively, providing that the development tools available also support the application. If all the development approaches score zero, the application should not be developed, as it is not supported by any of the existing development environments. In this case, either the requirements of the application should be reviewed, or a new development approach should be adopted that will enable the application to be developed.

- Identify the systems development techniques and methods that support the chosen approach, drawing on the documentation that defines the relationships between the development approach and the systems development techniques and methods.

- Identify the development tools that support the various systems development techniques and methods, drawing on the same documentation.

- For each of the application characteristics listed on the development-tool selection table, identify the option that best relates to the application under consideration. For each development tool identified in the previous step, circle the score that that option has been awarded.

- Add up the circled numbers to obtain a total score for each development tool. Any column that contains a circled zero will score a total of zero — in other words, that development tool should not be considered for this particular application, because it is incapable of meeting the requirements of one of the application characteristics.

- If several development tools support the same technique or method, select the development tool with the highest score. If all the development tools supporting a technique or method score zero, none of the development tools is applicable. Either a new development tool is required, or a different development approach should be adopted, with different systems development techniques, methods, and tools.

— Review the selected development environment as a whole, checking that the various development tools that need to interface with each other are compatible.

INTRODUCING NEW TOOLS

The type and complexity of a tool will have a bearing on how long it takes to integrate it into the development environment. The introduction and subsequent integration of a tool must be carefully planned and managed.

An effective plan for introducing a new tool should consist of the following four stages (it helps to stay in close contact with the tool's supplier throughout all four stages):

— *Stage 1: Marketing the implementation plan internally.* This stage is designed to ensure that all the staff involved with the new tool know exactly what the implementation plan is, what their responsibilities are, and how it will affect them.

— *Stage 2: Initiating changes to exploit and support the tool.* In this stage, changes are made to the development environment so that the tool can be optimally supported and exploited. These changes may be of a 'one-off' nature, such as reducing team sizes, or they may be continuing changes, such as defining and creating a 'cook book' (described later in this chapter).

— *Stage 3: Implementing a pilot application.* In Stage 3, the ability of the tool is tested on a pilot application. If the correct tool has been selected, and Stages 1 and 2 have been completed correctly, the pilot application will succeed. As a consequence of the pilot, areas may well be identified where changes can be made to make better use of the tool — for example, areas where the documentation may be reduced, or where changes in the procedures may be introduced.

— *Stage 4: Modifying the development environment in the light of the pilot application.* In Stage 4, any recommendations resulting from experience with the pilot application are implemented.

STAGE 1: MARKETING THE IMPLEMENTATION PLAN

The objective of this stage is to ensure that all the staff involved with the tool are aware of the implementation plan and of its effects on their working environment, their roles and responsibilities, job security, market value, and so on. Earlier in this report, we identified the staffing factors that are important to productivity; managers should pay particular attention to these factors when considering the introduction of a new tool.

The commitment of senior systems department managers to the tool is important to its successful introduction. To win their support, it is necessary to demonstrate the tool's cost justification.

A group should be created to 'market' the tool to the rest of the systems development department. This group should comprise the proposed technical expert (who will probably be one of the people who carried out the initial pilot project), a technical expert from the supplier, a sales representative from the supplier, and a senior

project manager from within the systems department. This group will be responsible for introducing the tool to development staff. All staff involved with the tool — developers, managers, and user managers — should attend a one-day presentation. Half the day should be spent introducing the tool, and half should be spent discussing the plan for introducing it into the organisation. This should encourage a positive attitude to the tool and its use.

The supplier should take the lead in the first half-day session, providing general background information on the tool, showing how it will be used in the proposed environment, describing the types of applications it will be used for, and providing details of the benefits that it can provide. In-house members of the group will participate in a supporting role.

In the second half-day session, the in-house members of the group will take the lead, clearly defining each stage of the plan and indicating the proposed timescales. All questions should be answered either during, or shortly after, the meeting. If outstanding issues are left unresolved, they may become stumbling blocks at a later stage. The pilot application should be described, the expected timescales should be made clear, and the members of the development team who will work on it should be announced.

STAGE 2: INITIATING CHANGES TO EXPLOIT AND SUPPORT THE TOOLS

To maximise the benefits obtained from using tools, various changes may need to be made to the development department. These changes could affect any aspect of the development department, from the computer hardware to the roles of the staff, and to a large extent, are dependent on the type of tool. For instance, with CASE tools, supporting the development method is critical; with fourth-generation languages, limiting team size or prototyping may be critical. Certain changes are applicable, regardless of the type of tool, because they help to improve understanding about the capabilities of a tool. Two that have proven very effective in practice are introducing the role of technical expert, and defining and implementing a 'cook book' and a 'tool-limitations list'.

Introducing the role of technical expert

Whenever any new tool is introduced, it is good practice to designate a technical expert as a focal point for all enquiries. The trials carried out during the selection process should provide several of the development staff with a reasonable knowledge of the tool. One of them would be the obvious choice to become the technical expert for the chosen tool. The technical expert will also be responsible for keeping up to date on the latest enhancements to the tool and for resolving any problems that arise.

The selection of the technical expert can be difficult, as the expert needs both good communications skills and the ability to learn detailed technical information. One company that has followed this practice described this person as a 'human catalyst' — someone who is convinced of something and can then encourage its use throughout the department, with ease. A further factor is that of personal credibility. The technical expert should have a well established record within the department — bringing in a new person sometimes serves to alienate existing staff.

Implementing a 'cook book' and a 'tool-limitations list'
A 'cook book' can be useful to help resolve problems that arise in using a tool. Figure 5.17 is an extract from one company's cook book, specifying how a screen-based system should be developed with Focus. Normally, compilation of the cook book is the responsibility of the technical expert.

A similar aid is the 'tool-limitations list'. This contains detailed information on the limitations of the various tools currently being used and is particularly helpful when deciding which tool to use for a particular application. An example of part of a tool-limitations list used by a retail company is shown in Figure 5.18, overleaf. Again, responsibility for compiling the tool-limitations list normally lies with the technical expert.

STAGE 3: IMPLEMENTING A PILOT APPLICATION

Before a tool is made available for general use, one or more pilot applications should be developed. The experience gained will be used to refine the use of the tool and the development methods. If the selection procedure has been followed correctly, and if appropriate changes have been made in the development department to support the tool, no major problems should arise with the pilot application. It will simply confirm that the tool can develop

Figure 5.17 A cook book advises users and developers on the use of a fourth-generation tool

Specifying a screen-based system

Do:	***Keep things simple*** Do you need flashy formatting? The more colours, special formatting, and highlighting you use, the more complicated the code becomes. Determine the functions of each screen as you would for a third-generation language — the more the functions are broken down, the simpler the coding.
Do:	***Be precise*** Document the validation required behind each screen, for each field which requires validation. Be clear on screen processing — do not be afraid to use program design language to define the program flow for the screen sequence in pseudo-English — this is as important as it is with third-generation-language specifications.
Do:	***Be careful with PFKEYS*** When using PFKEYS to navigate through a system, use the default keys when possible. If other keys are required for special functions, use PF5, 6, 7, 8, 9, 10, 11.
Do:	***Use painter*** When designing the screens, use Focus painter. This helps you design, and saves the programmer time. You will know that the screen can be used in Focus.
Do:	***Issue your own information messages*** When the user presses an invalid PFKEY, or invalid data is entered, issue meaningful error messages. When an action has been taken (for example, job submitted or record deleted), issue a confirmation message.
Consider:	***Response times*** A fourth-generation language may be quicker to code, but will be slower than a third-generation language to respond. Is this critical to your system?

Figure 5.18 The tool-limitations list specifies the limitations of a particular tool

Limitations of Focus from an organisation's tool-limitations list.*
Focus cannot update any file except a Focus or a VSAM file. These files cannot be read by any other language except Focus or Cobol programs making use of Focus Host Language Interface. However, Focus files can easily be created from QSAM or VSAM files, or DL/1 databases. Similarly, QSAM files can easily be created from Focus files.
Without central database control for simultaneous users, only one user can update a file at a time. Theoretically, a maximum of 128 simultaneous users is possible, but Information Builders indicates that about 20 is a more realistic limit. The operational range is between 5 and 20 users; typically, 15 users are supported.
Focus has no facility for automatic forward recovery (which is available in IMS). It is possible to code your own back-up logging and recovery routines in Focus.
Alternatively, frequent back-up copies of files can be taken. In the event of an irretrievable corruption of the database, the back-up copy would be restored and the user would have to re-enter his updates from the time the copy was taken.
No audit trail is provided for external files, except for IMS trace (which can be very large). Limited audit information is available for Focus file modification.
The 'non-procedural' nature of the Focus language makes complex processing difficult to achieve. Cobol subroutines should be used for complex logic and calculations whenever necessary.
The 3800 (laser) printer format character sets (that is, boxes and lines) are not available using Focus.
* These are the limitations of the version of Focus that one organisation has experienced in its particular environment. Information Builders informs us that the current version of Focus overcomes most of these limitations.

the required applications, and increase the confidence of the development department in its ability to do so.

Because the pilot application is an important step in gaining acceptance of the tool, it should be:

- A real business application — that is, an application required by users — but not one that is critical to the success of the business. It is advisable to become reasonably experienced with a tool before using it to develop critical business applications.
- Typical of the type of application for which the tool was selected.
- Small — that is, an application that can be developed fully in two to six months. If it takes much longer than this to produce results, developers will lose sight of the overall development life cycle and the impact of the tool.
- As far as possible, in the normal development environment.

Extra effort will, of course, be required to monitor the project, to collect detailed information about its progress, and to document any difficulties that were experienced. This effort should not, however, be taken into account in measuring the performance of the tool as it will not be incurred in a normal project. The tool supplier should also be involved in the first pilot application. This may be expensive, but in the majority of cases, it is very productive. One company that failed to do this eventually had to abandon the

first project that a fourth-generation language was used for. At the outset, the development department made an inaccurate estimate of the machine resources that would be required by the tool. It lost control of the application, as users demanded more and more functionality at the prototyping stage, and it failed to delegate responsibility to the user department, where it would have been appropriate to do so. All these factors contributed to the failure of the project, and all could have been avoided.

On completion of the pilot application, the whole project should be assessed to identify any changes that might enhance the use of the tool. The information gathered can also be used to produce guidelines for estimating the cost and effort likely to be involved in future development projects.

STAGE 4: MODIFYING THE DEVELOPMENT ENVIRONMENT

The development environment may need to be changed in some manner to facilitate the introduction of improvements identified as a consequence of the pilot application. For example, it may be necessary to modify standards, to reduce the level of documentation, or to increase the level of processing capacity. Such changes should be assessed, and if required, implemented. However, the temptation to make continual changes should be resisted. We recommend that suggested changes be fully documented and reviewed at regular intervals — say quarterly — to decide whether they are applicable, and to assess the costs and implications of implementing them.

With all the administrative and organisational changes implemented, the organisation is now in a position to use its set of tools to best advantage. The only outstanding problem that it might now face is knowing which of the tools available for use is the most appropriate for a particular application. Ensuring that the most appropriate tool is used for a particular application is a far more complex task than choosing a third-generation programming language to use on a project, as we explained earlier in this chapter. If the procedure for selecting tools from the tool set is updated to include the new tool, this problem should be avoided.

USER TOOLS

So far in this chapter, our interest has been in tools for specialists in systems development departments. In most companies today, however, some applications development is being carried out outside the development department. Systems departments may or may not be aware of this work, and may or may not be supporting it. As a consequence, there is no consensus on either the role of users in applications development, or on the scope of end-user computing.

In the last two years of the 1980s, there were marked improvements in end-user computing tools. With the growth in the use of PCs, a wide range of user-oriented tools became available, with improved user interfaces, automatic validation of information, and powerful commands making them easier to understand and use.

Further advances are certain to make user tools easier to use and more business-oriented. More and more users will therefore be

able to play a constructive role in ensuring that the organisation's computing resources are used for the maximum benefit of the business. The resources of the systems department are, however, limited, and therefore need to be allocated carefully to ensure that they are used to the greatest possible effect.

Systems departments should start by categorising the different types of user so that each category can be provided with the level of support, guidance, education, and tools that will enable business users to make the most effective use of the computing resources available to them. Without such a categorisation, it will be difficult to allocate resources in the most effective way and to plan for the growth of end-user computing.

The systems department should then set guidelines for different types of application, and encourage users to seek the development department's 'seal of approval' for each application. Encouraging users to have their developments approved will prevent the proliferation of poorly documented applications.

CATEGORISING USERS

There are four main ways in which systems departments can classify different types of business user: by their role, by the type of data they access, by their department, or by their need for or use of applications and tools. The last of these ways is the best. It gives rise to five categories of user:

- Category 1: Potential users, who at present do not use any computer-based applications.
- Category 2: Those who have a need to use or who use only applications and packages that have been written for a specific task that requires them to input data — for example, an accounts application.
- Category 3: Those who have a need to use, or who do use, enquiry and analysis tools to access databases and analyse the data.
- Category 4: Those who have a need to develop or who use end-user tools to develop small applications, primarily for personal use.
- Category 5: Those who have a need to develop, or who do develop, applications that may be used by many other users.

Each category of user is, in effect, an expansion of the one prior to it. Users tend to move through Categories 1 to 5 when first introduced to end-user computing, and regress through the categories as they move into the higher managerial roles. Each category refers to the use (actual or potential) made of end-user tools rather than to the type of tool used. Therefore, someone using a spreadsheet simply to add up a list of figures would be in Category 2, a user loading data into a spreadsheet from a database and analysing it would be in Category 3, and a user writing macros and developing a spreadsheet for a specified task would be in Category 4 or 5.

Staff can be assigned to the appropriate category by means of a simple questionnaire that assesses their use of tools as well as

their needs. There will, of course, need to be some mechanism for re-assessing at regular intervals the category to which an individual is assigned, because neither his needs nor the technologies used will be static. Once staff have been categorised in this way, the appropriate level of support and resources can be allocated in the most effective manner. Figure 5.19 suggests how the various types of support and resources — tools, training, help, guidance, and so on — might be allocated. In this figure, the tools shown in the cells on the first row have been classified as follows:

— *Fixed-processing tools*: These are the applications and packages used to support the daily work of the users. Most of the applications will have been developed in-house or bought as packages. All of these tools carry out a fixed processing task on specified information.

— *Flexible processing tools*: These are the packages, such as spreadsheets and financial modelling packages, that allow users to process data in a predefined manner.

— *Data-access tools*: These enable data to be accessed and retrieved from centralised or corporate databases. They generally permit 'read only' access and the data is transferred to a local machine if it is to be amended or modified. These tools use a simple programming language or a pseudo-English language syntax.

— *Report-generation tools:* These generally enable reports to be generated from a local or centralised database. Again, they tend to use a simple programming language or a natural-language syntax.

— *Fourth-generation tools*: These are used to develop applications (sometimes with the cooperation of the development

Figure 5.19 Allocating appropriate resources to the different types of end user will encourage growth in, and improve, end-user computing

	Category of user (Relationship with development department)				
Type of support needed	1. Potential user (None)	2. Current user (Weak)	3.Data-access user (Medium)	4. Personal developer user (Strong)	5. User developer (Very strong)
Tools	—	Fixed/flexible processing tools	PLUS Data-access tools	PLUS Report-generation tools	PLUS Fourth-generation tools
Machine (access)	—	Dumb terminal/PC	PC	Powerful workstation	Powerful workstation
Training	IT awareness	Use of tools	PLUS Basic data processing	PLUS Basic development	PLUS Best practice for data processing
Help (you telephone us)	—	Permanently staffed help desk	Permanently staffed help desk	Telephone	Telephone or face-to-face
Guidance (we advise you)	—	Hardly at all	Very little	Hand-holding	Hand-holding and directing

(Source: Butler Cox Foundation)

department and sometimes without) that tend to be run on PCs or intelligent workstations.

By way of illustration, an individual classified in Category 2 would normally be provided, as a minimum, with access to either a dumb terminal or a PC. A dumb terminal would be adequate for someone who required access only to fixed-processing tools — that is, applications that were already fully developed, and that only required data to be input. A PC, however, would be needed by someone who required access to the more advanced flexible-processing tools such as spreadsheets. Such staff would usually need to attend a standard training course on the use of the tools. Support would be provided via a permanently staffed help desk because this type of user typically requires immediate assistance. There would be little need for any further guidance other than that provided by the training course.

ISSUING GUIDELINES FOR DIFFERENT TYPES OF APPLICATION

Guidelines for end-user applications should be defined to avoid constraining users. Applications should be classified by size, the number of users, the type of data they access, and so on. The classification can also serve to determine the level of inspection required to attain the systems department's 'seal of approval', discussed below.

An example of a matrix that can be used to classify end-user applications and to define the guidelines for their development is shown in Figure 5.20. In this example (which is based on work

Figure 5.20 A matrix can be used to classify applications and define the guidelines for their development

	Data attributes	Application attributes	Project attributes	Associated guidelines
Class A (simple spreadsheet or database query)	• Personal • Non-strategic • Low-volume • Independent	• Personal • Standalone • Low complexity	• One to five workdays • No formal project management warranted	• Obtain authorisation • Use password • Back up data • Use common sense • Document as appropriate • Label the application and output reports
Class B (spreadsheet used on regular basis or database reporting program used by more than one person)	• Departmental • High-volume • Used by other programs	• Corporate • Used by more than one person	• Six to 20 workdays • Some project approval/project management warranted	Class A guidelines + • Do recommended control analysis • Document • Get 'seal of approval' for system security and so on
Class C (micro-based DBMS application, or complex spreadsheet, or simple spreadsheet used for critical decision-support)	• Strategic or sensitive • Used to update corporate database	• Complex • Uses non-recommended technology	• 21 to 40 workdays • Formal project approval/project management warranted • More than 40 days — system development standards apply	Class B guidelines + • Do compulsory control analysis • Do feasibility and cost/benefit analysis • Get agreement from development department

(Source: Based on an example from the Software Management Institute)

done at the Software Management Institute), all end-user applications are classified into one of three classes, according to their attributes. An application is always allocated to the highest possible class. If, for instance, it had data and application attributes in Class A, and project attributes in Class C, it would be considered as a Class C application. Examples of the types of applications that might fall into each category are included in the matrix.

The guidelines associated with that class of application are then applied, to ensure that the user is not unnecessarily restricted. For the development of a simple spreadsheet, for example, categorised as Class A, the following guidelines would apply:

- Obtain appropriate authorisation to develop the application. Professionals often have implicit authorisation by virtue of their job level; clerical staff may have to request it from a supervisor.
- Use passwords to restrict access to the application.
- Always back up both the data and the application.
- Document the application and procedures for using it.
- Label the application and any output it produces as 'Class A'.

For Class B and Class C applications, the guidelines would become progressively more stringent because the scope of such applications is wider and the risks are therefore greater. Classifying end-user applications in this way will ensure that they are evaluated prior to development, that appropriate development guidelines are followed, and that future users are aware of the standards to which each application was developed. In some organisations, however, it will not be practical to classify all applications, and the guidelines should be aimed at the riskier Class B and Class C applications.

ENCOURAGING USERS TO SEEK THE SYSTEMS DEPARTMENT'S 'SEAL OF APPROVAL'

Systems departments should encourage users to regard the concept of the 'seal of approval' as the equivalent of the acceptance testing they carry out on applications developed by systems staff. In providing its approval, the systems department should be looking for good documentation, comprehensive testing, consistent use of data, and so on. The systems department will also have the opportunity to add security, backup, or systems features that the user may not have considered. Once the applications have been approved, any subsequent maintenance and enhancements can be carried out in a controlled manner either by users or by the development department.

Clearly, not all end-user applications will require the same level of inspection. Indeed, some will need none at all. If users are required to submit major applications for inspection, however, and if the process is conducted effectively, the end-user development environment can be effectively managed.

Chapter 6

Measuring productivity and quality

The preceding four main chapters of this report have been concerned with the opportunities available to managers to make both productivity and quality improvements in four main areas of systems development: departmental organisation, staffing, the use of techniques and methods, and the adoption of contemporary tools. Changes in those areas can have a powerful influence on productivity and quality — yet the scale of the influence will remain indeterminate if productivity and quality cannot themselves be measured.

Measuring project productivity entails comparing output with input. The task is complicated by the need to account for the relationship between effort, the size of the project, and its duration. Fortunately, the three parameters of effort, project size, and project duration can be linked, enabling project productivities to be compared across different working environments and different companies.

Measuring quality similarities and differences between projects and companies is more difficult, however. The quality characteristics can be placed in four important categories; technical, ease of use, operational, and functional fit. The four entail as many as 11 different measures, whose significance varies widely between different companies. Although most companies' interest is focused on just two or three aspects of quality, such as maintainability, flexibility, and reliability, very few actually measure any form of quality. The last of these, reliability (in the form of software error rates), is tracked on a project-by-project basis by PEP.

The measurement of productivity and quality form the two main sections of this chapter. The chapter concludes with a section on implementing a measurement programme, in which three main requirements are stressed: collecting information early, avoiding misinterpreting the measurements, and providing measures at the right level.

MEASURING PRODUCTIVITY

Productivity, which measures the work rate of the systems development activity, compares the output achieved for a given input. Assessing productivity is difficult because it is not obvious how to measure outputs and inputs, and the effects of time compression and size are highly distorting.

Three project parameters can be related, however, through a formula known as the software equation. They are manpower effort, project size, and project duration (elapsed time). The software equation yields a productivity measure called the

Productivity Index, PI. The same three parameters yield a further index called the Manpower Buildup Index, MBI.

MEASURING OUTPUT AND INPUT

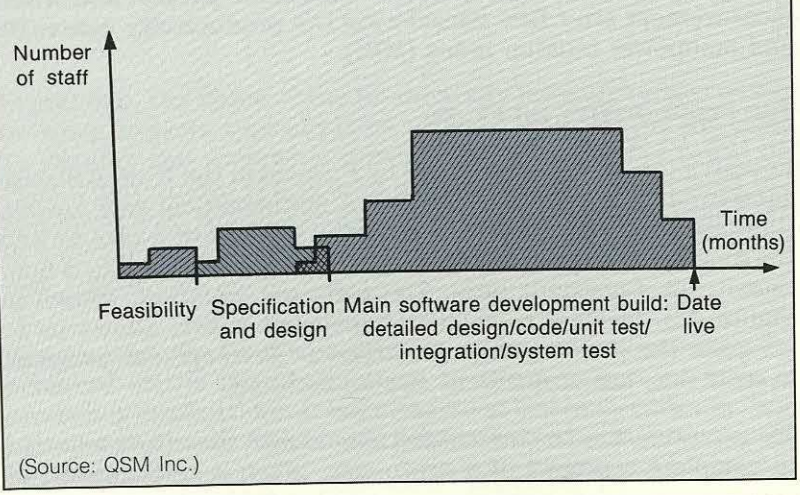
Take input first. The key is to keep the assessment simple. The straightforward way is to calculate it in terms of total man-months of effort by using the staffing profile for each stage of the project. The project manager can usually sketch this out in five to ten minutes. There is no need to go into detail in distinguishing between productive and non-productive contributions, because broad-brush figures are adequate for most projects. 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 is readily available.

It is possible to 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. Cost accounting data is often recorded to satisfy predefined criteria — for example, everyone records a 7.5-hour day, five days a week of productive work. The reality may differ considerably. To analyse this data usually takes much longer than asking the project managers to sketch the staffing profile.

As an instance of the broad-brush numbers that can be collected, Figure 6.1 sets out the staffing profile constructed over the key development stages of *feasibility, specification and design, and*

Figure 6.1 System development stages: staffing profile

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



(Source: QSM Inc.)

main build. Total staff used each month are shown without detailing who was on holiday, absent, or engaged in non-productive activities. Notice that some of the development stages overlap. The extent of this overlap needs to be assessed (but only approximately) because separate numbers are required for the effort used in each phase.

Next, consider the measurement of output. Counting the number of *effective lines of code* (ELOC) is one of the easiest ways of measuring the output (end product). Automating the collection of the ELOC statistics through a program that scans the appropriate libraries to count lines of code is relatively straightforward. The following rules apply:

- Lines are indicated by delimiters.
- Only executable lines are counted, not expansions.
- Comments are not counted.
- Delivered lines only are counted (those eventually discarded are ignored).
- New or amended lines only are counted, not unchanged lines.
- Data definitions are counted once only.

Another measure of the end product 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 using a spreadsheet to sum the values can be helpful.

Symons of Nolan Norton has proposed a modified form of function-point counting, referred to as Mark II. Symons has attempted to address some of the problems associated with the Albrecht approach and to make the method more suitable for modern systems. To date, Mark II is little used outside the UK government and the Albrecht method remains the *de facto* standard with the resultant greater availability of knowledge and supporting material.

We return to the question of function points and their measurement after first considering the productivity index (PI) and manpower buildup index (MBI).

THE PRODUCTIVITY INDEX (PI)

Research (originally undertaken by Putnam in the United States) shows that it is possible to derive a mathematical relationship (Putnam's software equation) between the size of a project and the time and effort needed to complete it. The relationship is non-linear, not a constant ratio. The software equation, shown in Figure 6.2, yields a productivity measure which allows us to compare the productivities of different development projects, even if they are of different size or duration.

The equation may be re-arranged to calculate the effort required to complete a project of a given size, given a certain level of

Figure 6.2 Productivity Index (PI)

The PI is a measure of a project team's efficiency. It is derived from an empirical formula ('software equation'), which defines a parameter called PM, the Productivity Measure:

$$PM = \frac{Size}{(Effort/B)^{1/3} \times (Time)^{4/3}}$$

Where:

- Size is the number of source statements.
- Effort is in man-years.
- Time is the duration of the main-build stage in years.
- B is a staff skills factor that takes account of the point in the systems life cycle at which peak manning occurs. It varies with project size, from 0.16 for small projects of around 5,000 lines of code, to 0.39 for projects exceeding 70,000 lines of code.

The PI is derived from the PM, using the following conversion table:

PM	PI	PM	PI
754	1	17,711	14
987	2	21,892	15
1,220	3	28,657	16
1,597	4	35,422	17
1,974	5	46,368	18
2,584	6	57,314	19
3,194	7	75,025	20
4,181	8	92,736	21
5,168	9	121,393	22
6,765	10	150,050	23
8,362	11	196,418	24
10,946	12	242,786	25
13,530	13		

(Source: QSM Inc)

productivity. This equation shows that effort required for a project depends on its duration (time) as well as its size, and on the productivity measure that applies in the particular development environment.

The productivity 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.

In practice, the value of the productivity measure ranges widely between projects, typically from around 3,000 to 50,000 and more. To simplify this somewhat unwieldy range, a number called the Productivity Index (PI) is used instead. The two are related in a non-linear way, so that a range of productivity measures from 3,000 to 240,000 is converted to PI values ranging from about 7 to 25 (see Figure 6.2).

In summary, the PI of a project is a measure of the productivity achieved at the main-build stage by the development team in producing applications. It is a measure of internal efficiency, and not of the value or functionality delivered to the business by the application.

The average PI of the 400-or-so projects on the PEP database in 1989 was about 15. PI values below 15 imply lower-than-average productivity; above 15, they imply higher-than-average productivity.

It is important to note that, because of its non-linear nature, small changes in PI value imply big shifts in team performance. Consider, for example, a typical project of 40,000 lines of code, with the main-build stage taking 10 months. At a PI of 15, the effort works out to be 60 man-months. At a PI of 14, the project takes a month longer and the manpower effort rises by 30 per cent. At a PI of 16, the project takes a month less and manpower effort drops by about 30 per cent. Thus, a one point movement in PI from around the average of 15 represents a productivity change of about 30 per cent.

THE MANPOWER BUILDUP INDEX (MBI)

The software equation takes account of the effects of compressing or extending the project timescale. When the timescale is compressed, the total manpower effort is increased substantially. This happens because the timescale is often compressed by carrying out, concurrently, tasks that would usually be done sequentially. In turn, this means that more staff are working on the project at any one time, which means that there are more paths of communication between team members, more opportunities for errors to arise and to remain undetected, and a greater management overhead.

The effect of time compression (and expansion) is represented by a measure called the Manpower Buildup Index (MBI). As with the PI, the MBI is expressed as a simple integer value, or level, ranging between one and six (see Figure 6.3). Level 1 represents a slow staff buildup. Projects with an MBI of one take the longest, but require the least effort. Usually, low MBI values are associated with projects that are subject to staffing constraints. Level 6 represents the opposite end of the spectrum — the 'throw people at it' approach. On projects of this type, many tasks are carried out concurrently, with virtually no constraints on money or the number of staff. For a given size and PI, projects with an MBI of six usually take the shortest time to develop, but require the most manpower effort.

In general, MBI values from one to three indicate below-average rates of manpower buildup; values of between four and six indicate above-average rates.

Consider, again, a typical project of 40,000 lines of code and a PI of 15. An MBI value of three leads to a main-build duration of 10 months and effort of 60 man-months. Reducing the MBI to one means extending the duration to 12 months, but effort falls to only 25 man-months. Raising the MBI to five saves time by reducing the duration to eight months, but the effort nearly doubles to 115 man-months.

Low MBI values reduce project manpower effort and increase project duration. The disadvantage of projects with low MBIs is that the extended timescales mean that there is a greater chance

Figure 6.3 Manpower Buildup Index (MBI)

The MBI is a measure of manpower buildup. It is derived from an empirical formula defining a parameter called MM, the Manpower Buildup Measure:

$$MM = \frac{Effort}{B \times Time^3}$$

Where:

- Effort is in man-years.
- Time is the duration of the main-build stage in years.
- B is the same staff skills factor as for the PI.

The MBI is derived from the MM, using the following conversion table:

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

(Source: QSM Inc)

of the requirements changing before a project is completed, and that it is often more difficult to keep the project team constant and motivated.

The MBI measure can be used by systems development managers, when they are planning projects, to assess whether a project can realistically be completed in a given time. High MBI values identified at the planning stage point to potential problems and high risks. A few systems development departments can consistently achieve above-average PIs under considerable time pressures, but they are a small minority.

FUNCTION POINTS AND LANGUAGE GEARING

The PI measures the efficiency of a project team. It is an important measure for systems development managers interested in assessing the internal efficiency of their departments. A second, equally important, measure is departmental *effectiveness*, which is concerned with the *functionality* delivered to the business, per unit of effort.

Internal efficiency is analogous to the fitness of a cyclist, which determines the effort that is put into pushing the pedals. What really matters, however, is the distance the cycle travels for the effort that is put in, and this is determined by the gears on the cycle. The cyclist may not be at peak fitness, but a high gear will enable him to travel, say, 10 times the distance for a given effort. High-level languages are analogous to high gears; the higher the *language gearing*, the fewer the number of lines of code that will be required to produce a given level of system functionality.

This does not mean, however, that programming languages with the highest language gearing should always be used. Just as trying to cycle uphill in an inappropriate high gear will result in significantly slower speed or not being able to peddle at all, so

a failure to match the language to the application can result in significantly more effort being used.

The best known unit of measure of system functionality for commercial data processing systems is the function point, the measure of end-product value we referred to on page 122. If function points are not counted directly, it is possible to estimate delivered functionality by multiplying the number of thousands of lines of code by the appropriate language gearing, using the values shown in Figure 6.4. This table must be used with caution as the multiplication can result in very inaccurate estimates. The average number of source lines of code needed to generate one function point works out to be 70 for all the projects in the PEP database, but the range varies widely from as few as 10 or less (high language gearing) up to 200 or more (low language gearing). We recommend that both lines of code and function points are counted and that the table is used only to identify potential inaccuracies in these counts.

Knowing the effort required to develop a project and the language gearings for the programming languages used, it is possible to calculate the *functional delivery rate*, expressed as function points per man-month. The functional delivery rate can be calculated

Figure 6.4 Language gearing

The list shows the language gearing, expressed in terms of the number of function points per thousand lines of code, for the high-level languages used in PEP projects.

Language	Language gearing	Language	Language gearing
Acumen	35	Guest	35
ADF	50	Ideal	35
ADS/Online	50	Keyplus	25
Algol	10	Lotus	100
APL	35	M204	35
Application Factory	50	Magna8	35
Application Master	35	Mantis	70
APS	60	Mapper	18
Artemis	33	Mark IV	25
Ask	30	MFS	25
Basic (Compiled)	13	Natural	18
C	8	Nomad	25
CA-Earl	35	Pascal	11
CBAS	13	PL/1	13
CLI	25	PLDS	30
Clipper	25	PPL	25
Cobol	10	QMF	70
CSP	35	Quickbuild	35
Culprit	65	Quiz	70
Data	25	Rally	35
Dataflex	25	Ramis	25
Datatrieve	50	Rapidgen	35
dBASE	30	RDB	25
DCL	6	RPG	17
DDL	35	SAS	30
EAL	35	SIR	35
Easytrieve	65	SQL	70
Enform	50	Sybol	14
FCS	25	Telon	70
Filetab	17	TIG	10
FMS	20	Transact	35
Focus	25	UFO	30
Fortran	10	Whip	10
Gener/ol	70	Wizard	35

(Source: Software Productivity Research, Inc)

for the main-build and for other phases of the life cycle. The PEP database provides average functional delivery rates, but it should be remembered that this rate is influenced by the size of the system, its duration, and the internal efficiency of the team. The functional delivery rate alone can be misleading to systems managers.

MEASURING QUALITY

Many systems departments have initiated software quality-assurance programmes to increase the effectiveness of applications development. The majority of systems departments find, however, that it is very difficult to direct such programmes and to justify their cost when there is no quantitative evidence of their benefits. The missing component of many software quality-assurance programmes is software quality measurement.

Software quality measurement enables the qualities of applications, such as reliability, ease of use, maintainability, and so on, to be quantified in useful and consistent terms. Properly implemented, a measurement programme will help the systems department to specify and produce applications of the quality that users require, to identify where improvements might be made to the development process, and to justify the costs of a software quality-assurance programme. In effect, software quality measurement provides essential management information to the systems department.

In practice, however, software quality measurement has met with mixed success. In many systems departments, there is considerable resistance to the concept, generally based on misunderstandings about its purpose, its cost, and the level of effort required to introduce it. In others, where quality measurement programmes have been implemented, only limited benefits have been gained, because the scope of the programmes has been too narrow. While there is a wealth of material available on the subject of software quality measurement, much of it is of a very academic nature, not well suited to the commercial environment, and much of it is applicable only to particular aspects of applications development.

What is required is a practical, consistent, and comprehensive approach to measuring software quality, ensuring that both users and developers are satisfied with the applications that are delivered. To appreciate how to put such a programme in place, it helps first to understand the characteristics of quality, then the quality measures that support those characteristics, and finally how to assign priorities to the measures.

QUALITY CHARACTERISTICS

Today, most systems quality-assurance procedures are designed to ensure that the functionality provided by applications software meets the users' requirements. However, even where the quality of the system is checked at intermediate stages of the development cycle to ensure that the finished product does meet the functional requirements, it may still be regarded as being of poor quality by the user community. This is because the quality-assurance procedures do not take account of the users' needs in other areas — operational performance, ease of use, and the ease with which the system can be modified are obvious examples.

Analyses of users' expectations for applications software have been carried out by Barry W Boehm and his colleagues. In their early work, *Characteristics of Software Quality* (published by Oxford: North Holland in 1978), they identified a large number of software characteristics that contribute to users' overall perceptions of software quality.

Four of these characteristics are particularly important, as we mentioned in Chapter 4: functional requirements, operational performance, technical features, and ease of use (see Figure 6.5). By defining and meeting quality objectives specified in terms of these characteristics, it is possible to build application systems that the user community regards as high-quality. Although the functional requirements of a system are generally defined in great detail, the other three characteristics are often ignored in systems specifications. These characteristics are usually determined by ad hoc decisions made at the analysis and programming stages.

QUALITY MEASURES

Different software quality characteristics have varying degrees of measurability. Ease of use, for example, can be assessed by the user only in subjective terms, and in any case, will be defined differently from application to application. Constraints such as these should be considered when selecting the measures to be used.

Several research projects into quality assurance have produced lists of quality measures. One of the most widely accepted among software quality-assurance experts, and the one that we recommend to PEP members, is that developed originally in the United States for the Rome Air Development Center, and known as the RADC approach. Although the RADC approach was originally defined for military applications, it has been applied successfully to the development of commercial computing applications.

The RADC approach defines a set of 11 user-oriented quality factors — reliability, flexibility, maintainability, re-usability, correctness, testability, efficiency, usability, integrity (which actually refers to security), interoperability, and portability — which extend throughout the software life cycle. It is important to define each characteristic fully to avoid confusing different factors.

Figure 6.5 Four software characteristics are important in defining and meeting quality objectives

Functional requirements	Define what the application system has to do, down to the level of describing the data to be entered, the rules for accepting/rejecting the data, and the processing of accepted data.
Operational performance	Defines performance in terms of response time and elapsed time (for batch systems).
Technical features	Define meantime between failures, ease of maintenance, ease of parametric change, and ease of re-use of software elements.
Ease of use	Defines user interface in terms such as number of keystrokes, error recovery procedures, help facilities, and message clarity.

Figure 6.6 shows how the 11 RADC quality factors match the four quality characteristics referred to above. The 11 quality factors were originally defined to help predict the quality of a final application as it is being developed. We have devised an appropriate user-oriented measure for each of the factors so that the quality of *existing* applications can also be assessed.

These measures are summarised in Figure 6.7, overleaf. It lists the basic data items that need to be captured from the application in order to produce the measures. The final 'calculation' column gives the formula for calculating each quality measure from the basic data items. The calculations ensure that the measures are normalised so that they can be used to compare the quality of different applications, where practical.

ASSIGNING PRIORITIES TO QUALITY MEASURES

When developing a new application, it is not always possible to meet all of the quality requirements desired by all groups of users. There are two main reasons for this. First, a high level of quality in one of the 11 RADC quality factors may imply a low level of quality in one of the other factors. For example, a high level of portability will usually imply a low level of efficiency, and *vice versa*. Second, the project manager will often have to make trade-offs between the time, cost, and quality of the application.

The implication is that quality should not be specified at a higher level than the application warrants. For example, the maintainability requirements of the application can be reduced if the lifetime of the application is known to be short.

The conflicts in quality priorities occur because of the conflicting requirements of the main groups who get involved in the use of an application — the application's users, their managers, the development managers, the maintenance and support teams, and computer operations staff. Figure 6.8, on page 131, shows which of the 11 quality factors are of most interest to each of these groups. Because of the complexity of these conflicting interests, most systems departments will need to select just two or three of the quality factors that they need to control during development. The cost of controlling more qualities than this becomes prohibitive.

Usually, the most important quality factors to concentrate on are the three that will increase user satisfaction through reduced costs and better service — maintainability, flexibility, and reliability.

Figure 6.6 The 11 RADC quality factors can be categorised in terms of the four quality characteristics defined in PEP Paper 9

Quality characteristics			
Functional	Technical	Operational	Ease of use
Integrity Interoperability Portability	Correctness Re-usability Maintainability Flexibility Testability	Efficiency Reliability	Usability

Figure 6.7 Most qualities of existing applications can be quantified directly or indirectly

Quality factor	Measure	Basic data items	Calculation
Reliability	Mean time to fail (MTTF)	Hours of use (H) Number of failures (N_1)	MTTF = H ÷ Ni
Flexibility	Effort to implement a change in requirements (F)	Developer hours per change (He) Function points or lines of code changed (Sc)	F = He ÷ Sc
Maintainability	Effort required to diagnose and respond to a fault (M)	Number of faults fixed (N_2) Developer hours for fixing faults (Hf)	M = Hf ÷ N_2
Re-usability	Proportion of application consisting of re-used code (R)	Size of re-used code (Sr) Application size (lines of code or function points) (S)	R = Sr ÷ S
Correctness	Conformance to requirements (C)	Number of defects (D) Application size (lines of code or function points) (S)	C = D ÷ S
Testability	Effort required to test the application fully (Ef)	Number of test cases (T) Application size (lines of code or function points) (S)	Ef = T ÷ S
Efficiency	Online efficiency (E)	Number of transactions (Tr) Computer resources used: — CPU time (R_1) — Disc transfers (R_2) — Network traffic (R_3) — Etc	 $E_1 = R_1 \div Tr$ $E_2 = R_2 \div Tr$ $E_3 = R_3 \div Tr$
Usability	Ease of use (Ea)	Number of unfounded fault reports (N_3) Hours of use (H) Number of users (U) Application size (lines of code or function points) (S)	Ea = $N_3 \div (H \times U \times S)$
Integrity	Access-control quality	Subjective ratings for: — User-access control — Database-access control — Memory protection — Recording and reporting access violations — Immediate notification of access violations	None (use the subjective ratings)
Interoperability	Effort required to link the application to another (EI)	Effort in hours to link applications (HI) Application size (lines of code or function points) (S)	EI = HI ÷ S
Portability	Effort required to transfer the application to another environment (Ee)	Effort in hours (Hc) Application size (lines of code or function points) (S)	Ee = Hc ÷ S

The choice will, however, depend on the nature of the system in question. Typical quality requirements for specific types of application are:

- Systems with a long life: maintainability, flexibility, and portability.
- Publicly accessed systems: usability, integrity, and reliability.
- Systems that can cause damage to property or lives if they go wrong: reliability, correctness, and testability.
- Systems that use advanced technology: portability.

Figure 6.8 Different groups of users have different quality needs

Quality factor	Group of users				
	Application users	User management	Development management	Maintenance and support	Operations
Reliability		✓			✓
Efficiency					✓
Usability	✓	✓			
Integrity		✓			✓
Correctness	✓	✓			
Interoperability			✓	✓	
Maintainability			✓	✓	
Flexibility		✓	✓	✓	
Portability				✓	
Testability			✓	✓	
Re-usability			✓		

A valuable technique for minimising the cost of providing high (user perceived) quality is to develop an operational-use profile. Such a profile shows the expected level of use of each of the functions of the application. Suppose, for example, that an application has two main functions, one of which will be used for 90 per cent of the time and the other for 10 per cent of the time. In this situation, it is obviously better to concentrate on improving the quality of the most frequently used part of the application rather than to spread the quality-assurance effort evenly over both of the application's functions.

IMPLEMENTING A MEASUREMENT PROGRAMME

Implementing a measurement programme and taking action on the basis of the results can boost internal systems productivity by as much as 20 per cent a year. Organisations can achieve one-off benefits soon after the introduction of a measurement programme, but sustained improvements will not be realised unless the measurement and improvement programme remains in place for several years — that is, where organisations take a strong initiative to manage productivity and quality, and are able to justify expenditure on improving the development process.

Implementing a measurement programme, like implementing an application, requires careful planning. It is important to start collecting information early; there is no merit in waiting until the development environment is established and stable. It is also important to ensure that measurements are not misinterpreted, and that they are provided at a level appropriate for the person who will be using them.

COLLECTING INFORMATION EARLY

By collecting measurement data as soon as possible, systems managers have a basis on which to plan future developments.

Waiting until the development environment is stable is not justified; if software measurement is to be of value in the long term, it must be implemented in such a way that it is still useful in a changing environment.

Collecting the data required to measure productivity and software quality properly need not be an onerous task — useful benefits can be gained by spending the equivalent of 1 or 2 per cent of development effort on gathering measurement data. In many systems departments, much of the data will already be available in a machine-readable form from systems used to manage development resources, change requests, and correct software faults.

AVOIDING MISINTERPRETING THE MEASUREMENTS

It is important that all staff with access to the measurements understand how to interpret them correctly, and how objectives for improvement are set on the basis of this interpretation. In one company, the development manager was particularly interested in the improvements in productivity that had resulted from the introduction of a fourth-generation language. He examined the internal productivity measures, which showed a decline owing to the learning curve, and initiated an internal enquiry. If he had looked at the external productivity measures, he would have seen that the fourth-generation language had actually improved the function delivery rate by over 50 per cent. As a direct result of misinterpreting the measurements, he wasted a lot of money and time examining a problem that did not exist.

Ensuring that staff are aware of the measurements and the objectives may require an awareness campaign or a training programme. Hewlett-Packard, which introduced a company-wide software metrics programme, set up a training course that gives hands-on experience to those who will be involved in the collection and use of metrics. They believe that this training greatly contributes to the success of the programme. Cranfield IT Institute (part of the Butler Cox Group) also provides several educational courses in this area of measuring the performance of the systems development department.

Staff should understand that measurements will not be used to judge the performance of individuals and to reward those who do well, although some organisations pay a bonus related to the performance of the whole department.

PROVIDING MEASURES AT THE RIGHT LEVEL

Information about the development department will typically be collected at a very low or detailed level, processed, analysed, and combined to produce measures at progressively higher levels. At each level, the measures provide different benefits to different staff. Development staff, for instance, will be interested in the number of errors and function points produced. The business manager, on the other hand, will be more interested in percentage increases or decreases in productivity and quality, and time and cost savings.

No single set of management measures will satisfy all managers, nor should a single set of measures be imposed across the board. Different companies should take into account the needs of each manager and select an appropriate set of measures. At least three points of view should be considered — those of the development team, those of the departmental managers, and those of the business customer.

These measures can be presented very effectively in a variety of ways. Several of the more popular presentations of measures are:

- *Trend line charts* show the variation in a software measure (such as reliability) over time. They are useful for determining if there is a trend or pattern in the occurrence of any measure.
- *Histograms* show frequency of data by various categories and classifications. They are used, for example, to show the distribution of productivity over a range of projects.
- *Pareto diagrams* are a particular type of histogram that can be used to show various measures by type and frequency.
- *Scatter diagrams* show the existence (or lack) of a relationship between two factors. If a straight line is apparent in the plot, there is likely to be a relationship between the factors.
- *Control charts* show a software measurement plotted over time within statistical control limits. If the plotted line exceeds either of the limits, there is a strong possibility that something is going wrong with the development process. A control chart of error rates helps to determine if a process is either 'in control' (with only random errors occurring) or 'out of control' (with errors occurring more generally).

Chapter 7

Conclusion

In this report, we have presented the independent analysis of over 400 projects, and the experience of over 100 of Europe's largest organisations that are involved in the development of computer-based applications.

We have shown that the search for improvements in the productivity and quality of the development department needs to be a continuous one, and that it is a complex task. An organisation is not 'productive' or 'unproductive'; it lies somewhere on a continuous and moving path between the best and the worst.

Every organisation can improve the productivity and quality of its development activities. What the research has shown is that there is no single answer, and that, indeed, obsessive pursuit of a single solution can lead to an overall downturn in productivity or quality. For example, introducing new tools into a badly structured environment will not give an uplift in performance. Moreover, the step usually leads to a subsequent rejection of the tools while the organisation moves on to seek the next instant solution, overlooking the real and often multiple constraints on greater productivity.

Amongst the many possible routes to better performance, this report has identified the following:

ORGANISATIONAL ACTIONS

- In a divisionalised group, dividing responsibilities between central and devolved parts of the overall systems development function, offsetting the division more towards the centre than is implied by the general management style of the group as a whole.
- Clarifying the responsibilities of users by formalising responsibilities with reference to three levels of application: core systems, non-core systems, and personal systems.
- Flattening the management structure of systems development and organising groups to fulfil particular functions within the simplified structure.
- Identifying a member of each project team to take responsibility for ensuring that module and integration testing is carried out to preset standards.
- Weighing the relative merits of different approaches to system testing — testing by the project team, by a separate department, and by a joint team — and adopting that best suited to the circumstances.

— Organising maintenance as a separate department under a qualified manager, thereby helping to improve staff morale and motivation.

STAFF MOTIVATIONAL ACTIONS

— Boosting staff motivation in several ways: broadening the scope of jobs, introducing job rotation, providing a flexible career structure, improving goal-setting and feedback, rewarding achievement with performance-related pay, and paying attention to fitting jobs to people.

— Limiting project-team sizes wherever possible to six staff at most, and breaking down large projects into self-contained sub-projects of smaller size. (Despite evidence that productivity decreases with project size, there are clear benefits to be gained — such as reduced development time, reduced staff turnover, fewer requirements changes, and less risk of overrunning.)

— Taking account of individuals' personalities as well as technical skills when forming project teams. (Teams of people with similar personalities are most appropriate for the later development phases; with dissimilar personalities, for the earlier phases.)

— Selecting project-team leaders on the basis not of their technical skills, but more of their strengths in participation, flexibility, and ability to manage conflict.

TECHNIQUES AND METHODS

— Replacing the long-established waterfall model of the systems development process by a different model that emphasises testing rather than production.

— Formalising software testing. Making full use of walkthroughs and inspections. Using test-management aids and test-data preparation aids.

— Formalising the decision on whether to maintain or replace existing systems. (To help with this, perhaps introducing a maintenance-rating system.)

— Allocating a set proportion of resources to maintenance, to avoid maintenance work continually displacing new development work.

— Introducing a seven-step maintenance process: change management, impact analysis, system-release planning, change design, implementation, testing, and system release/integration.

— Establishing a quality-management programme. Helping to improve quality across the board by introducing a quality culture throughout the development department.

TOOLS

— Introducing or extending the use of fourth-generation languages. (Although they do little, if anything, by themselves

to increase internal productivity, they can do a lot to boost external effectiveness by increasing the rate of functional delivery. Adapting methods, techniques, and management style to exploit the fourth-generation languages will improve internal productivity.)

- Avoiding reliance exclusively or excessively on CASE. (The promised benefits are not yet being achieved either in terms of reduced development time and effort, or fewer errors, or increased reliability.)
- Using maintenance-support tools, which support the impact-analysis and change-design steps of maintenance. (They can be justified when the maintained system is likely to continue in operation for several more years.)
- Using static- and dynamic-analysis tools to help assess the quality of code within a system.
- Avoiding the selection of system development tools from the tool set in isolation. (Instead, they should be selected in the context of the wider development environment in which they are to be used, and in the light of a specific development application.)
- Recognising productivity and quality problems arising from the misuse of techniques and tools. (Paying particular attention to technique and tool selection, and to staff training.)
- Adopting a four-stage plan to smooth the introduction of new development tools: internal marketing, environmental adjustments, pilot application, and environmental modification.
- Encouraging the use of non-specialist development tools by the user community, but coordinating their uptake through a process of categorising users, allocating tools to categories, and promoting the idea of the system department's 'seal of approval'.

What is also clear is that for the opportunities for improvement to be clearly identified and for improvement to be systematically pursued, a better understanding of both the present and the changed situation needs to be established. To this end, the research has shown that there is scope for measuring and monitoring productivity and quality in every organisation.

A measure need not be precise, or be on an absolute scale, to be of value: it simply needs to give a reliable basis for comparison and to show the relative levels of improvement in a consistent manner. The research has shown that this can clearly be done today.

PRODUCTIVITY AND QUALITY MEASUREMENT ACTIONS

- Putting in place a productivity-measurement programme. (The easiest way to do this is to measure effort, project duration, and project size. Measuring project size in lines of code can yield an internal efficiency rating in the form of a

Productivity Index (PI). Measuring it in terms of delivered functionality yields a broader rating. Both form the basis of a common yardstick.)

- Avoiding rapid rates of manpower buildup (MBI) unless business pressures or legislation dictates otherwise — in which case the effect needs to be recognised. (High MBIs mean increasing the manpower effort while shortening the timescale of a project, but more than 50 per cent of development effort can be squandered by adding staff to a project to reduce elapsed time.)

- Implementing a quality-measurement programme, taking care to identify the most appropriate quality measures.

No organisation, and no manager, is alone in being concerned about development productivity, and rightly so. In terms of the potential effects on today's business, such concern is fully justified. It is not just a matter of the high investment involved in computer systems. Rather, it is the fact that the health of most businesses now rests on the timely delivery of effective and reliable systems. Fortunately, as this report has demonstrated, there are many practical routes to improvement.

Butler Cox

Butler Cox is an independent, international consulting company specialising in areas relating to information technology.

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.

Butler Cox provides a range of consulting services both to organisations that are major users of information technology and to suppliers of information technology products.

Consulting for Users
Supporting clients in establishing the right opportunities for the use of information technology, selecting appropriate equipment and software, and managing its introduction and development.

Consulting for Suppliers
Supporting major information technology and telecommunications suppliers in assessing opportunities, formulating market strategies, and completing acquisitions and mergers.

Education
The Cranfield IT Institute, now a wholly owned subsidiary of the Butler Cox Group, educates systems specialists, IT managers, line managers, and professionals to understand more fully how to apply and use today's technology.

Foundation
The Foundation is a service for senior managers responsible for information management in major enterprises. It provides insights and guidance to help them to manage information systems and technology more effectively for the benefit of their organisations.

The Foundation carries out a programme of syndicated research that focuses on the business implications of information systems, and on the management of the information systems function, rather than on the technology itself. It distributes a range of publications to its members that includes research reports, management summaries, directors' briefings, and position papers. It also arranges events at which members can meet and exchange views, such as conferences, management briefings, research reviews, study tours, and specialist forums.

Recent Foundation publications and events have covered such topics as:

- Assessing the value of information technology investments.
- Managing multivendor environments.
- Staffing the systems function.
- Pan-European communications: threats and opportunities.
- Systems security.
- Emerging technologies.
- Managing information systems in a decentralised business.

PEP
The Productivity Enhancement Programme (PEP) is a participative service whose goal is to improve productivity in application systems development. It provides practical help to systems development managers and identifies the specific problems that prevent them from using their development resources effectively. At the same time, the programme keeps the managers abreast of the latest thinking and experience of experts and practitioners in the field.

The programme consists of individual guidance for each member in the form of a productivity assessment, and publications and forum meetings that are available and open to all members.

Recent PEP publications and events have covered such topics as:

- The influence on productivity and quality of staff personality and team working.
- Managing software maintenance.
- Quality assurance in systems development.
- Making effective use of modern development tools.
- Organising the systems development department.
- Trends in systems development among PEP members.
- Software testing.
- Software quality measurement.

Butler Cox plc
Butler Cox House, 12 Bloomsbury Square,
London WC1A 2LL, England
☎ (071) 831 0101, Telex 8813717 BUTCOX G
Fax (071) 831 6250

Cranfield IT Institute Limited
Fairways, Pitfield, Kiln Farm, Milton Keynes,
Buckinghamshire MK11 3LG, England
☎ (0908) 569333, Fax (0908) 569807

Belgium and the Netherlands
Butler Cox Benelux bv
Prins Hendriklaan 52,
1075 BE Amsterdam, The Netherlands
☎ (020) 75 51 11, Fax (020) 75 53 31

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, Germany
☎ (089) 5 23 40 01, Fax (089) 5 23 35 15

Australia and New Zealand
Mr J Cooper
Butler Cox Foundation
Level 10, 70 Pitt Street, Sydney, NSW 2000, Australia
☎ (02) 223 6922, Fax (02) 223 6997

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