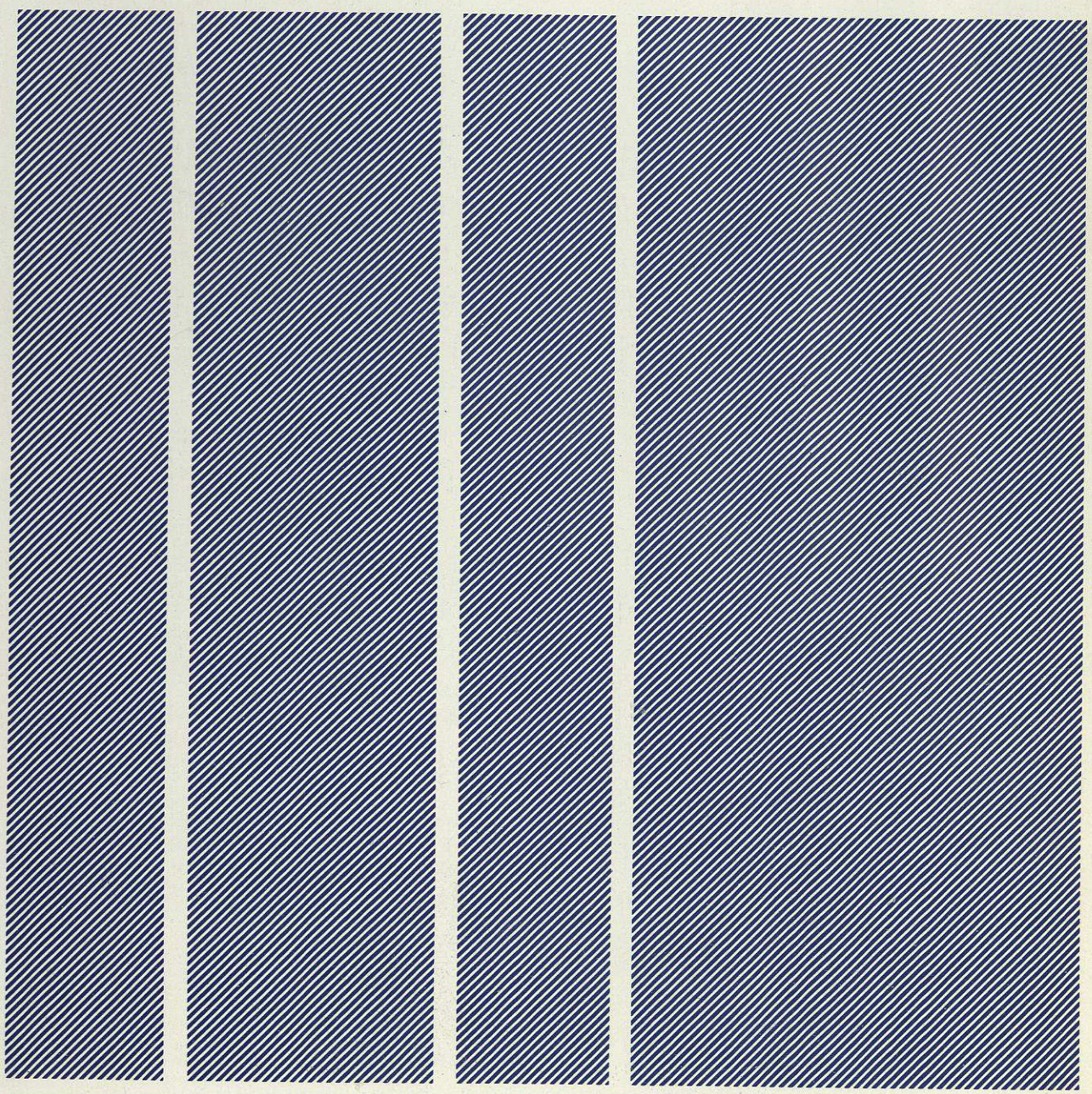
Report Series
No. 47

The Effective Use of System Building Tools

May 1985



The Butler Cox Foundation

THE BUTLER COX FOUNDATION

REPORT SERIES NO. 47

THE EFFECTIVE USE OF SYSTEM BUILDING TOOLS

ISSUED MAY 1985

Research Method

The research for this report was carried out in late 1984 and early 1985 by *David Flint*, a senior consultant with Butler Cox who has a long-standing interest in tools and methods for computer systems development.

The work for this report built on previous Butler Cox research, notably that for Foundation Report No. 36 (Cost-Effective Systems Development and Maintenance). The research was extended by a literature search, and the published material that significantly influenced the development of this report (including that referred to explicitly) is listed in the Bibliography at the end of the report.

The experience of Foundation members and other organisations in using system building tools was researched by means of focus groups, individual interviews and analysis of published case histories. Information was collected on more than 70 installations in France, the Netherlands, Sweden, the United Kingdom and the United States which have, collectively, hundreds of man-years' experience of using several dozen different system building tools.

Using information from about 30 installations, we were able to calculate some representative productivity figures for some of the more commonly used tools. We were also able to form views about the applicability of different types of tools, the best ways of using them and the consequences of using them (or, in some cases, of not using them) both for users and for information systems departments. The overall picture was sufficiently clear to enable some simple models to be built to demonstrate the financial implications of using system building tools.

Research Conclusion

The research showed that the use of advanced system building tools, such as Mapper, Focus, Linc or Gener/ol, could enable systems to be developed (and maintained) between four and 20 times as fast as is generally possible with Cobol and PL/1. The research also identified the most common mistakes that are made in the application of system building tools. In particular, we found that the effective use of system building tools requires a fundamental change in the development approach.

The report is therefore structured as a methodology for selecting and using system building tools. By following this methodology, Foundation members can avoid the mistakes and maximise the benefits that are obtainable from system building tools.

The main findings of the research are highlighted in the report synopsis.

Additional report copies

Normally members receive three copies of each report as it is published. Additional copies of this or any previous report (except those that have been superseded) may be purchased from Butler Cox.

Photoset and printed in England by Flexiprint Ltd., Worthing, Sussex

THE BUTLER COX FOUNDATION

REPORT SERIES NO. 47

THE EFFECTIVE USE OF SYSTEM BUILDING TOOLS

CONTENTS

THE EFFECTIVE USE OF SYSTEM BUILDING TOOLS

REPORT SYNOPSIS

The construction of application systems is a central and crucial activity for every information systems department. In the past this activity has produced severe problems for systems managers and, on occasion, for the whole organisation. For example: inadequate computer systems delayed Morgan Stanley's entry into the gold market for several years; the computer systems at the British Driver and Vehicle Licensing Centre at Swansea have been much criticised; and in May 1983, the Stockholm Stock Exchange was totally shut down for about ten days by an overload in its electronic system for registering trades.

Probably every reader of this report will remember systems that were very late, over-budget or that failed to meet their objectives to such an extent that they had to be abandoned.

Major disasters have become less common as the information technology industry has matured and as proper management disciplines have come into use. In addition, some information systems departments have introduced formal quality control and structured analysis and design methods. These typically have produced some increase in productivity together with a more manageable development process. But the main benefit arising from these methods has been a greater degree of confidence in the system specification and the avoidance of major errors in specification and design.

Despite these improvements, system construction remains a slow and expensive process in most large organisations, and the delay between a user request and the first live use of a system may still be several years. The resulting impatience of users has to some extent been contained by personal computing and in-house timesharing (or information centres). Nevertheless, most organisations still have some form of development backlog. Worse still, many users have needs that they never put forward because they believe that the information systems department will be unable to meet them promptly and economically.

Since the beginnings of commercial computing, system developers have sought aids and tools that would speed-up and simplify the system development process. Indeed, a Cobol compiler is just such a tool. More recently, many suppliers have promoted modern system building tools as the means of increasing development productivity. There are now hundreds of such tools, each claiming to solve the users' problems. A representative sample of these tools is listed in Figure S.1. (Sometimes these tools are known collectively as 'fourth-generation languages', but we do not use that term in this report because we believe it is misleading.)

Figure S.1 Representative sample of system building tools

Tool	*Supplier*
ADS	Cullinet
ADF	IBM
All	Microdata
Applications Master	ICL
Focus	Information Builders
Gener/ol	Pansophic
Linc	Burroughs
Maestro	Philips
Mantis	Cincom
Mapper	Sperry
Natural	Software AG
Nomad	NCSS
Profs	Burroughs
Ramis	Mathematica
TIS	Cincom
UFO	Oxford Software

Many Foundation members now have substantial experience of using modern system building tools. Whilst some members are enthusiastic users, others have rejected their use, or even abandoned them after building several systems with them. The most common objections to using system building tools are:

—The need for excessive amounts of computer power.

—Lack of recovery features.

—Lack of security and integrity features.

—Inability to cope with complex applications.

As a consequence, many organisations restrict the use of system building tools to a narrow range of applications — often to applications provided through an information centre.

Despite the fact that some organisations find great difficulties in using system building tools, others have achieved remarkable successes. We believe that most information systems departments can obtain major benefits by introducing or extending the use of system building tools. In this report we explain how organisations can improve their effectiveness by using these tools.

Our research has shown that introducing and making effective use of system building tools requires management commitment and a disciplined approach that has six main stages:

—Understand the value of system building tools.

—Determine the objectives for using system building tools.

—Select the appropriate system building tools.

—Define the necessary changes in the development environment.

—Build the pilot application.

—Extend the use of the system building tools.

Each of these stages is the subject of one chapter of this report.

CHAPTER 1

UNDERSTAND THE VALUE OF SYSTEM BUILDING TOOLS

System building tools can provide considerable benefits both to the organisation and to the information systems department. These benefits are widely misunderstood because it is believed that they apply only to the technical process of systems development. The improvement in development productivity is important and is often much greater than is generally recognised by systems staff. But improved development productivity produces benefits for the whole organisation, not only for the information systems department. These benefits are critical in planning for the future of information systems.

With the right system building tools, systems will be better, will cost less, will be delivered sooner, and will be easier and less expensive to maintain and enhance. As a result, the information systems department can build many more systems, cut costs, and become more responsive to business needs. In this chapter we explain how these benefits can follow from the use of system building tools.

The benefits are of differing significance to different organisations, and are provided to differing extents (and sometimes not at all) by the various tools currently available. Thus, different organisations need to set different objectives for using system building tools. In Chapter 2 we explain how to identify the objectives that are appropriate. Subsequent chapters deal with selecting and installing system building tools in order to realise the potential benefits.

IMPACT ON DEVELOPMENT PRODUCTIVITY

Improved development productivity is the key factor in transforming the technology of system building tools into benefits for the organisation. Lines-of-code per man-day has been widely used in the past as a measure of development productivity but, for reasons given more fully in Appendix 1, we recommend that development productivity should be measured in function points delivered per development man-day.

The function point concept was devised by Allen Albrecht of IBM as a measure of system size. Function points delivered per man-day is a good measure of development productivity because:

- Function points measure function delivered, not program size.
- Function points are independent of the language and machine used.
- The function point concept is independent of organisational structures.

As an example, a system with 1,000 function points is equivalent to between 50,000 and 100,000 lines of Cobol or PL/1. Our research has shown that, for a range of organisations and over several years, the productivity for development in PL/1 and Cobol ranges between 0.03 and 1.5 function points per man-day. The mean value during the past six or seven years has been about 0.2 function points per man-day (see Appendix 3), but for the purposes of this report we have assumed that the normal productivity today in Cobol or PL/1 is 0.35 function points per man-day. On this basis, a system comprising 30,000 lines of PL/1 would require about six man-years of effort.

When advanced system building tools are used the productivity is usually between 1.5 and 7.5 function points per man-day. For example:

- A United Kingdom manufacturer has increased development productivity by a factor of seven by using Focus.
- A Dutch insurance company has measured a seven-fold increase in productivity through the use of Mapper.
- A French oil company estimates that Ramis has provided a five-fold productivity improvement.

Much higher productivity can be obtained in special cases or if prototyping is used. But, in general, advanced system building tools enable development productivity to be four to 20 times as great as with Cobol and PL/1.

BENEFITS FOR INDIVIDUAL SYSTEMS

For individual systems, system building tools can reduce the development cost, decrease the development time, improve the quality and make the development process more manageable.

System development costs will be reduced

The most advanced system building tools, used most effectively, allow systems to be built with much less labour and computer power than is required when using conventional languages such as Cobol or PL/1. In many quite ordinary organisations — local authorities, banks and manufacturers — savings of more than 75 per cent have been achieved. In a few exceptional cases the savings have been very much larger.

In the Santa Fe Railway, for instance, a team of four people using Mapper built in 17 months a system that included, as just one of its functions, a railway yard system. Working in parallel, a team of up to 20 programmers built an equivalent system using IBM Assembler. This system took two years to develop, and was subsequently replaced by the more comprehensive Mapper system.

In the Corporation of Lloyds, the introduction of Focus enabled the information systems department to quote an acceptable price for a long-desired system for which previous quotations (by the department and by external suppliers) had been unacceptable.

Development times will be decreased

Because less effort is needed to complete a system, it can be developed faster. In the Leamington Spa Building Society a team of three people using Linc developed and installed a system equivalent to about 70,000 lines of Cobol in just five months. (This system was subsequently acquired by Burroughs who have now sold systems based on it to other building societies.)

Systems quality can be improved

The high productivity obtainable through the use of system building tools makes it possible to build prototypes before implementing the complete system. Prototypes generally provide a better means of establishing the user requirements for systems than do conventional analysis and design, and this leads to better designs and higher-quality systems. The immediate benefit is a better solution to the business problem. Other benefits include more user involvement and less enhancement in the earlier years of the project. (The value of prototyping is discussed at greater length in Chapter 4, and the procedures for using it are outlined in Chapter 6.)

The development process will be more manageable

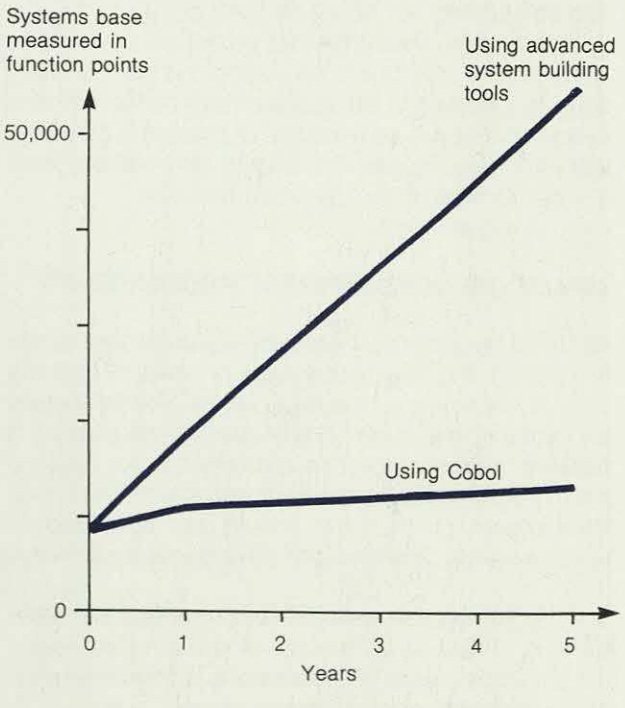
The reduced effort and time required by system building tools make the development process inherently easier to manage. In addition, the use of these tools provides a better base for estimates, since prototyping provides a more accurate specification and also postpones the point at which it must be finalised. The reduced overall duration of the development process makes planning easier, whilst the shorter period between finalising the specification and delivering the system allows less time for changes in external circumstances to dictate changes in the system.

Prototyping also allows final estimates of development and execution costs to be deferred until after the prototype is complete, at which point more will be known about the users' needs and the computer resource requirements.

BENEFITS FOR THE INFORMATION SYSTEMS DEPARTMENT

The development improvements brought about by system building tools also provide benefits for the information systems department. Because the tools reduce both the effort and the elapsed time needed to build (or enhance) an application, the department will be able to meet users' requests more quickly and the user will be able to pay for more enhancements and systems. Thus, at one extreme, system building tools allow more systems to be developed for the same cost. Alternatively, the tools enable the same number of systems to be developed at a reduced cost. In reality, most organisations will make a trade-off between these two options.

Figure 1.1 Growth in applications systems base due to the use of system building tools



Developing more systems

If the number of development staff is held constant then the use of system building tools allows the total number of systems developed to be increased dramatically. Figure 1.1 shows for a hypothetical information systems department the total systems inventory (measured in function points) and its projected growth over a five-year period.

We have assumed that in the base year (Year 0) the department spends 36 per cent of its budget on systems development, 12 per cent on maintenance and enhancements and 52 per cent on running installed systems. If the department continues to develop systems in Cobol it would add 10 per cent to its systems base in the first year and slightly smaller increments in subsequent years as extra resources have to be found for maintenance. (We have also assumed that all systems are replaced after five years of service.)

Figure 1.1 also shows the growth in the base of installed systems if the department builds all its new systems with advanced system building tools. Over five years, nearly four times as many systems are developed — an impressive increase compared with the continued use of Cobol or PL/1.

The cost of running the greater number of systems made possible by system building tools will, of course, increase the total cost of running the department.

Figure 1.2 Total departmental costs if system building tools are used to build more systems

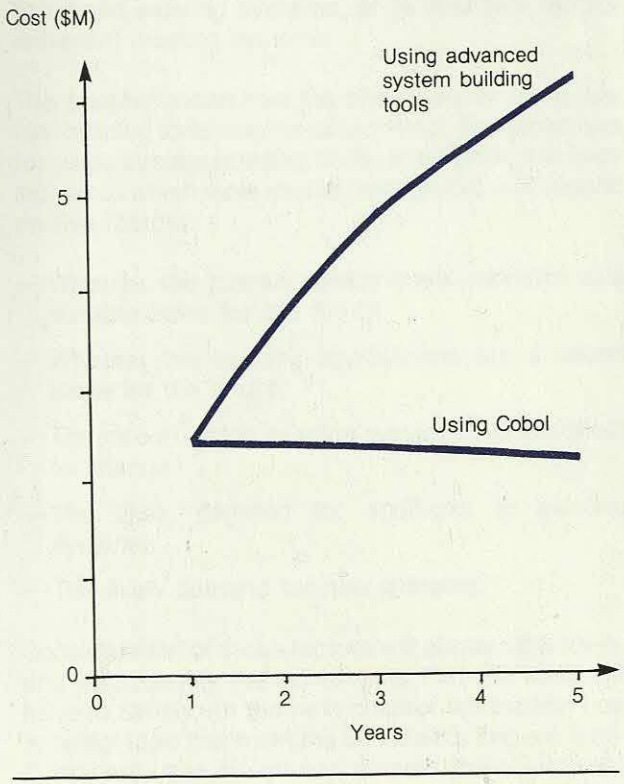


Figure 1.3 Impact of system building tools on departmental cost-effectiveness

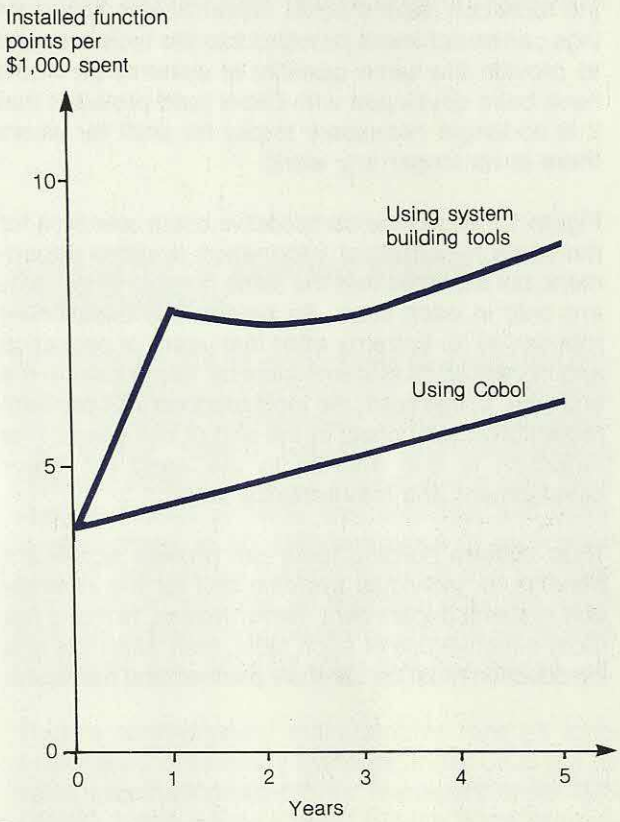


Figure 1.2 shows that the cost of the continued use of Cobol reduces slightly over the five-year period because of the improving price-performance of computer hardware. The figure also shows that, if system building tools are used for all new development, the total departmental costs increase by a factor of more than 2.5. This increase reflects the larger number of systems developed and their lower machine efficiency.

Nevertheless, the overall cost-effectiveness of the department improves because the total costs rise more slowly than the increases in the systems base. Figure 1.3 shows the improvement in departmental cost-effectiveness obtained by using system building tools, measured by the number of installed function points per $1,000 of expenditure in each year. System building tools provide this improvement because, for this hypothetical department, the savings in development and maintenance outweigh the extra operational costs. This might not be the case either if transaction volumes were especially high, or if maintenance levels were particularly low, or if systems lasted on average for more than five years.

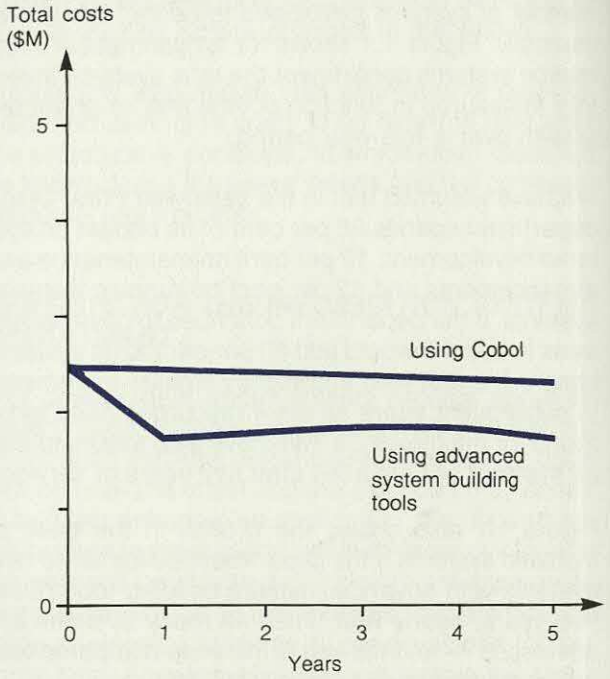
Reducing information systems costs

In view of the rapid cost increases shown in Figure 1.2 it may seem strange to suggest that system building tools can reduce costs. Nevertheless, cost savings can be achieved, provided that the tools are used to provide the same quantity of systems as would have been developed with Cobol (and provided that it is no longer necessary to pay for staff for whom there is no longer any work).

Figure 1.4 shows the comparative costs over time for the same hypothetical information systems department, but assumes that the same number of systems are built in each case. As before, the department rebuilds all its systems after five years of operation and increases its systems base by 10 per cent in the first year. In this case, the tools produce a 24 per cent reduction in total costs at the end of five years. This reduction is due entirely to the need for fewer development and maintenance staff.

Thus, system building tools can provide significant benefits for individual systems and for the information systems department. Nevertheless, to make the most effective use of such tools, their selection and introduction must be carefully planned and managed.

Figure 1.4 Total departmental costs if system building tools are used to build the same number of systems



CHAPTER 2

DETERMINE THE OBJECTIVES FOR USING SYSTEM BUILDING TOOLS

System building tools are often sold, and sometimes even evaluated, in terms of their technical characteristics rather than the benefits they can provide. In order to ensure that the tools chosen will provide the greatest benefits, each organisation must first establish its objectives for using them.

Different tools are designed for use with different hardware and software environments and for developing different types of application. Thus a tool that is appropriate for developing a management information system may not be suitable for reducing the cost of mainstream transaction processing systems. It is therefore essential to determine the systems environments in which the tools will be used and the types of applications that will be developed with them.

System building tools also provide the option of rebuilding existing systems at an acceptable cost. Another key objective to be determined therefore, is whether a tool will be used to build only new systems, to rebuild existing systems, or to maintain (and/or enhance) existing systems.

This chapter shows how the objectives for using system building tools may be determined. The objectives for using system building tools, and hence the decision as to which tools should be selected, will depend on five factors:

—Whether the current systems environment is a suitable basis for the future.

—Whether the existing applications are a sound basis for the future.

—The rate at which existing systems are expected to change.

—The likely demand for additions to existing systems.

—The likely demand for new systems.

Consideration of these factors will clarify objectives, and also identify the constraints that the tools will have to satisfy. (In the next chapter we explain how to select tools that meet the constraints and are likely, if they are used effectively, to meet the objectives.)

SUITABILITY OF EXISTING SYSTEMS ENVIRONMENTS

We define a systems environment as a specific combination of hardware architecture, operating system, file or database management system, teleprocessing monitor (where relevant) and development process. Most substantial organisations now have several distinct systems environments, and some have as many as 20. Different types of application system (operational support systems and management support systems, for example) often are run in different systems environments, and are sometimes developed with different tools and procedures.

Thus a confectionery manufacturer runs its long-established order-entry systems under CICS but its sales information system for managers under IMS DB/DC. There is also a VSPC service in the information centre.

There may also be different software environments for batch and online systems. For example, several public utilities process batch systems on ICL equipment using the DME operating systems, but use VME for their online systems.

If the existing systems environment is an unsuitable basis for future developments, especially those that provide business advantages, it may be desirable to move the systems to a different environment. It may also be necessary to redevelop the systems.

An organisation therefore might decide to select system building tools for use in its existing systems environments, or it might replace (or supplement) them with new environments that make the best use of modern system building tools. To determine whether to continue using an existing environment, two questions must be answered:

—Can the existing systems environment support the general direction for future systems?

—What are the constraints on choosing a new systems environment?

Can the systems environment support future systems?

Amongst the developments that are affecting the nature of systems in many organisations are:

- The growth in end-user computing.
- The growth in online managerial access to operational data.
- The integration of systems across business functions.
- The growth in the use of systems by customers and suppliers.
- The integration of systems at the point of use.
- The growth of inter-business data communications.

The relative importance of these developments varies between organisations but, in our view, the majority of large organisations will feel the impact of most of them within the next five years. Many have already done so.

Some of these developments can provide advantages to the business — for example, by locking customers into the product ordering and delivery systems. It is therefore critically important for the information systems department to be able to support these developments.

In our view the growth of user access and manipulation of data is the most important single trend in data processing. In most organisations this is mainly confined to distinct information centre environments, based on copies of live databases and constrained by resource problems. By 1990, in many organisations, these constraints will have ceased to be tolerable. It will therefore be necessary for the same systems environment to support both operational systems and user access and manipulation.

The importance of decision-support databases (that is, systems optimised for flexibility and access rather than machine efficiency) is already clear. And, in some organisations, these databases are already being used both for transaction processing and management information systems. We expect this trend to continue as machine efficiency is progressively traded for human effectiveness, specifically the effectiveness of users and application developers. Decision-support databases will increasingly replace older databases as the basis for operational systems in the same way that those databases have replaced serial files.

The systems environment may either assist or impede the changes required in the nature of information systems. For example:

- Managerial access to operational data will be eased if there is a suitable query facility, if data structures correspond to managers' views of the business, if there is an online data dictionary and if there is a facility to transfer data to a suitable personal computer or workstation.
- Inter-business data communications will be easier to introduce if data complies with the relevant industry-specific standards and if the telecommunications software supports the appropriate protocols.

What are the constraints on choosing a new systems environment?

The constraints on the choice of a new systems environment may be due either to external policy or to the need for compatibility with existing systems. Moreover, an orderly move to a new systems environment will usually require some applications to be rebuilt in the new environment. Not only does this requirement increase the cost of installing a new environment, it may also require a temporary ban on system enhancements. Because these issues relate directly to applications we discuss them in the next section (on page 7).

External policy constraints

Some information systems departments are free, in consultation with their immediate customers, to make their own decisions on systems environments. Others are subject to external controls. In the public sector such restrictions usually originate with an elected council or politically appointed board or with the government. The restrictions are usually imposed in an attempt to stimulate or support indigenous products and suppliers.

In the private sector the constraints are more often due either to a central policy imposed by a corporate head office or to the deep scepticism felt by many senior managers of the ability of anyone except IBM to build computers or manage 'convergence'.

It is possible, if the arguments are really good, to get such policy constraints changed, but this is always a slow and laborious process. In general, it is probably best to accept the constraints unless it becomes clear, after attempting to choose suitable system building tools, that there is no suitable tool for the prescribed systems environment.

Compatibility with existing systems

Although the new systems environment may be greatly superior to the existing one, the present databases and systems will continue to be used for some time (possibly for several years). During the transition period there will be a need for some compatibility between the old and new environments, specifically in the areas of data access and/or terminal communications.

If the existing systems are large and monolithic it may be necessary to phase the transfer to the new environment, but impossible to transfer the old database in parallel. In this situation, the new environment will have to allow systems to access the old database. At present this usually means using the same computer, and this could restrict the choice of a new systems environment, which in turn could impose unacceptable restrictions on the choice of system building tools.

Ideally, the same terminal should be able to access applications running in both the old and new environments. Thus, the old and new environments should allow similar screen formats and function-key conventions to be used.

SOUNDNESS OF EXISTING APPLICATIONS

If the existing applications are seriously unsound it may be necessary to rebuild existing systems to new designs. If systems have to be rebuilt, it will usually be sensible to do so in a new systems environment in order to obtain the benefits of increased productivity and greater flexibility.

The key questions to be considered are:

—Are the existing systems fit for their purpose?

—Can you afford the costs of rebuilding systems?

—Can you justify the time required to rebuild existing systems?

If the decision is taken to rebuild some (or all) of the existing systems, high-productivity tools should be chosen together with a systems environment that provides adequate flexibility for the future.

An element of risk is inevitable in any major systems development, and the use of new development tools in a new systems environment will certainly increase the risk. It can be reduced by choosing products that have been proven in organisations like your own but, beyond that, the risk has to be balanced against the costs and benefits.

Are the existing systems fit for their purpose?

Many organisations are still running systems that only partly meet the business needs, but which would demand a prohibitive amount of time and effort to rewrite. Of 20 senior systems executives from large organisations who replied to a Butler Cox questionnaire in the autumn of 1984, two-thirds said that they would make their systems "largely or radically different" if they were able to start again from scratch.

If the systems are inadequate it is clearly desirable to replace them with satisfactory ones. By reducing the time and effort required for systems development, system building tools, used in a new systems environment, may make this possible.

Can you afford to rebuild existing systems?

Even if existing systems are clearly inadequate, the costs of rebuilding them with conventional tools may be prohibitive. But system building tools may make it possible to rebuild systems at much lower costs. Savings of 80 per cent or more compared with conventional tools are quite possible, and can be obtained for maintenance and enhancements as well as for new developments.

No system lasts indefinitely, so the decision to rebuild using system building tools may be seen as bringing forward the rebuilding date, rather than as a completely new cost.

Even so, rebuilding substantial systems is a costly undertaking, and the most important costs may be lost opportunity costs. Separate decisions must therefore be made for each system or group of systems.

The expected rate of change to existing systems also has a material effect on the decision whether to rebuild (see below).

Overall, for most organisations, we believe that the rebuilding of applications cannot usually be justified on cost savings alone. The costs may be acceptable, however, if there are other benefits such as greater flexibility or ease of use.

Can you justify the time required to rebuild existing systems?

Converting all the existing systems to a new systems environment is a daunting prospect. Not only is it expensive, it is also likely to require a temporary ban on enhancements, which may be unacceptable to users.

Using system building tools will often allow systems of up to 1,000 function points to be rebuilt by five people in six months. In many organisations a six-month ban on amendments will be acceptable to users, especially if the resulting system will include some enhancements. The size of systems that can be rebuilt is therefore related to the longest period of time for which users can be persuaded to accept a ban on enhancements. The larger the system, the longer the ban, and the less likely users are to accept it.

RATE OF CHANGE TO EXISTING SYSTEMS

Maintaining and enhancing existing systems is a major part of the work of almost every information

systems department. The maintenance level usually depends on the size of the system being maintained, on its quality, and on the rate at which the users' requirements change. Allen Albrecht provided delegates on the 1983 Butler Cox Foundation Study Tour with data that showed the annual maintenance level in IBM in 1980 was about 30 minutes per function point.

Another measure, the one that we will use in this report, is the ratio of annual maintenance cost to development cost (ignoring inflation). The IBM figures imply a ratio of about two per cent, much lower than is usual in Europe, where five per cent and even 10 per cent are not uncommon. For the purpose of this analysis the division between systems maintenance and systems enhancement is not significant. We will distinguish, however, between changes and additions to existing systems. (Additions are discussed in the next section, beginning on page 9.)

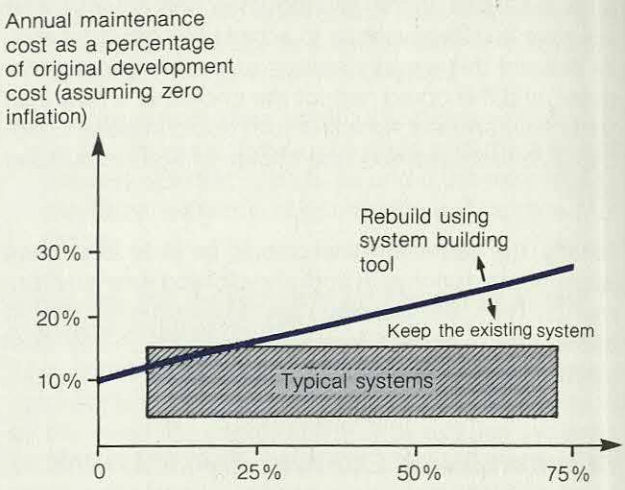
Changes may be required either to correct errors or to amend the system to meet changed user requirements. Additions are almost always enhancements, consisting of new reports and new transactions, but not additional processing for existing transactions. The reason for making this distinction is that it is usually possible to find system development tools that provide substantial benefits in adding new reports and transactions to existing systems, but it is more difficult to find tools that are effective for making changes to systems originally developed with conventional tools.

In most organisations, corrections and small improvements are subject to some kind of overall budget limit. Larger improvements may be subject to separate justification exercises. In general the rate at which changes are made can, to a considerable extent, be controlled by the information systems department. The way in which this control is exercised reflects the department's priorities. Some departments discourage changes in order to increase the resources available for new development.

If the rate of change for a system is expected to be high, it may be possible to reduce the cost of maintenance by rebuilding the system with advanced system building tools. If a system is redeveloped using advanced tools then the maintenance costs, including the costs of changes, will be reduced as well, the productivity improvement factor being similar to that obtained for original development. Against this saving must be set both the cost of redevelopment and, usually, additional operating costs.

Every organisation has its own rules for such evaluations. We have performed some calculations to investigate the trade-off between lower maintenance costs and higher operational costs for tools giving a four-fold productivity improvement.

Figure 2.1 Maintenance versus operating cost trade-off for system rebuilding with a tool giving a four-fold improvement in productivity



The results are shown in Figure 2.1. In performing the calculations, we have used discounted cashflow with a 10 per cent discount rate over five years. We have assumed that operational costs fall by 10 per cent per year, and that systems built with system building tools cost 30 per cent more to run than those built with Cobol.

Figure 2.1 shows that, even for a system with low running costs, rebuilding is only cost-justified for moderately high maintenance levels (above about 10 per cent). The implication is that, for typical levels of change and operating costs, rebuilding systems will not be cost-justified. (Rebuilding could be justified over a longer period or if the tool gave either a greater increase in productivity or a smaller increase in running costs.)

Nevertheless, there are organisations that have many old systems and a high maintenance workload. In April 1985, for example, the United States Office of Manpower and Budget specified a 25 per cent reduction in government computing through the use of system building tools. Departments were instructed to replace old systems.

Where rebuilding is not justified, the maintenance task may be eased somewhat by using a data dictionary to identify where changes need to be made and by using development workbenches. There may also be scope for systematic improvement of the existing systems by appropriate planned maintenance and techniques such as structured retrofit. Because our research has concentrated on development, rather than maintenance, we will not discuss these techniques further.

ADDITIONS TO EXISTING SYSTEMS

In most organisations it is often necessary to extend systems, in the existing systems environment, by adding new transactions and reports. Tools that may be used for such additions are available for most systems environments. Nevertheless, when deciding whether to use advanced system building tools in this way, a trade-off has to be made between the additional operational costs and the savings in development effort. High development productivity and acceptable operating costs are most usually achieved with reports and online queries.

Figure 2.2 shows the circumstances in which extra operational costs due to the use of advanced system building tools will be less than the savings in development costs. Again, we have used discounted cashflow over five years with a discount rate of 10 per cent, and we have assumed that improved hardware price/performance reduces operational costs by 10 per cent per year.

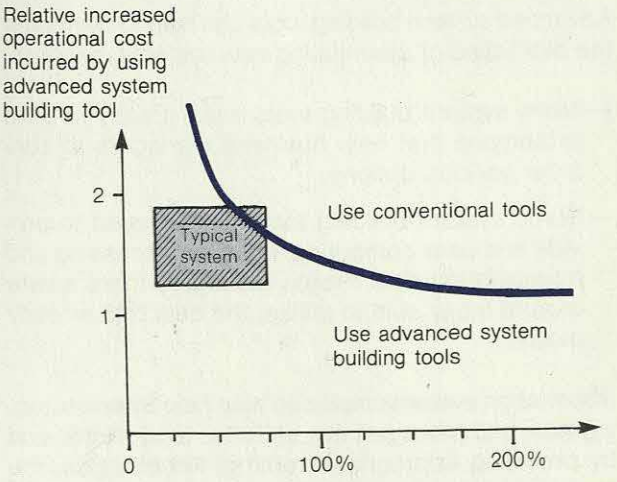
The figure clearly shows that system building tools are usually appropriate for additions. Our analysis indicates that it will almost always be less expensive to use advanced system building tools for additions to existing systems. Indeed, we regard the development of queries and reports in conventional languages as now justified in only the most exceptional circumstances, where the function is used intensively by many people, for example, or where highly complex processing is essential.

Nevertheless, it is important to systems' users that existing transactions can be mixed with transactions built with the new tools. Even if a system building tool can interface with the existing systems environment (IBM's CICS and DL/1, for example) it may not be compatible with existing applications running in that environment.

The new system building tool should also be able to produce screen formats that comply with the standards used in existing systems, and should allow function keys to be used in a uniform way across a mixed set of transactions. It is also desirable that a system built with the new tool should be able to use existing modules and transactions.

System building tools may be used to increase the number of additions to existing systems rather than reduce the cost of making additions. In this situation it will be necessary to consider whether the organisation can assimilate and make good use of the new functions at the rate at which they can be delivered. We will return to this issue in the next section, where we discuss the demand for new systems.

Figure 2.2 Trade-off curve for system enhancements



THE DEMAND FOR NEW SYSTEMS

System building tools are most effective when they are used for developing new systems. In some organisations the new systems required are not closely linked with the existing systems base. In this situation it may be best to use a new systems environment with advanced system building tools for developing the new systems. When deciding on which systems environments to use, and in determining the criteria for selecting tools for use within those environments, it is necessary to consider both the nature and likely volume of new systems. The following questions arise:

- How many new systems does the organisation need during the next few years?
- Are the new systems closely related to existing systems?
- Is the introduction of new systems likely to be seriously constrained by budget limitations?
- Can the organisation assimilate and exploit new systems as fast as system building tools enable them to be produced?

How many new systems does the organisation need?

Almost every organisation has a known backlog of applications. In addition, there is usually a hidden backlog of applications that managers would like but which they do not request because they believe that the information systems department cannot provide them sufficiently quickly (or at a price) to make them useful. If conventional development methods continue

to be used, therefore, the amount of systems development that needs to be done over the next few years will continue to exceed the resources available to do it.

Are new systems closely related to existing systems?

In some organisations all the systems use a few key databases and any new systems must also access those databases. In these cases new systems should be considered in a similar way to additions; they may be provided by using system building tools in the current systems environment.

In other organisations systems are not closely linked, and in most organisations there are some systems that are not closely linked to the mainstream systems. Examples include personnel systems in a retail organisation and fixed asset systems in most organisations. In selecting tools with which to build such systems, the existing systems environment is not a limiting factor.

Indeed, the mainstream systems environment may have been chosen for its stability, network management features and ability to handle large volumes of data. If there are new systems that do not require these features, but do need end-user computing facilities and high development productivity, then a new systems environment may be chosen. This approach has been extremely successful for some organisations:

— The Santa Fe Railway was an IBM user but used Sperry's Mapper to build a real-time control system for its operations. It still prefers IBM for some corporate systems, however.

— A large retail organisation was an ICL user, but now builds all its online systems on minicomputers running the Pick operating system.

New systems environments will also usually be preferable when developing new kinds of systems such as computer integrated manufacturing, decision-support systems and office systems.

Will the introduction of new systems be restricted by budgetary limitations?

Some information systems departments can install any systems environment and build any application that the customer will accept and pay for. Others are subject to overall budgetary constraints that are often set by an essentially political process without reference to the value of particular systems. In many large organisations, such limits are applied much more strictly to capital than to revenue budget items.

Though troublesome and frustrating to information systems managers, such limitations may actually be in the best interests of the business. Research in the PIMS MPIT programme of the Strategic Planning Institute shows that, for companies with an inferior strategic position, additional expenditure on information systems may reduce managerial productivity and, hence, profits. (These results were given by Gus van Nievelt at the Foundation conference held in The Hague in May 1984.) We do not believe that such counter-intuitive results are restricted to commercial organisations — they doubtless also apply, with appropriate modifications, in non-profit organisations.

In other organisations, however, it appears that budget limitations come about because of a failure of nerve. In essence, senior managers, and, sometimes, information systems managers, simply do not believe the things they say about the benefits of computer systems. If the information systems department is competent, senior management should be pleased to see its costs rise because the benefits to the business will also be rising.

Can the organisation assimilate many more new systems?

The ability to deliver many more new systems is only valuable if the business needs, and can assimilate, new systems. This is a question that must be answered individually by each organisation. Nevertheless, our view is that most European businesses make less use of computer systems than they should do, because:

— Implementation lead-times discourage managers from asking for systems.

— Development costs are too high.

— Business (and information systems) managers are too preoccupied with introducing basic systems to give attention to the strategic systems that could really enhance their businesses.

Advanced system building tools can help to overcome the difficulties of assimilating new systems, because:

— Many system building tools make it easy to build prototypes that help business managers to consider various options.

— Some system building tools can be used to provide end-user computing facilities. Accessing and manipulating data makes managers more aware of, and more able to utilise, the data that already exists.

Information systems staff can also help by encouraging new and more positive attitudes to systems, and by providing appropriate training. Nonetheless, the best way of developing the requisite awareness is by involving users through prototypes and end-user computing.

Another important factor in motivating the business to make more (and better) use of systems is the extent to which computer systems are used in running the information systems department. If the department is seen to be making effective and productive use of systems, then the rest of the business will be more inclined to use systems as well.

SUMMARY

In this chapter we have discussed several options for the future of information systems, some of them quite radical. In so doing, we have considered issues ranging from overall strategy to detailed cost accounting.

Although it is not usually possible to provide a direct cost-justification either for a programme of system replacement or for a move to a new systems environment, we believe that these are options that all computer users should consider very seriously. Many of the most successful computer systems, the kind that people talk about at conferences, originated from a change to a new systems environment.

Today, many installations are hemmed in by systems and software that they feel obliged to keep but would rather be without. Advanced system building tools provide a way out of this dilemma. But it requires determination to follow this route.

CHAPTER 3

CHOOSE THE APPROPRIATE SYSTEM BUILDING TOOLS

Having first determined the required system environments and the objectives for using system building tools within those environments, the next stage is to select the appropriate tools. Hundreds of system building tools are now available from scores of suppliers. Within the scope of this report it is not possible to discuss them all separately, and so we have divided them into five types:

—Incremental aids.

—Data dictionaries.

—Development workbenches.

—Discrete tools.

—Integrated toolkits.

We use the term 'advanced system building tools' (ASBTs) to indicate discrete tools and integrated toolkits. (Our rationale for this classification is given in Appendix 2.) We now describe briefly each of the five types.

Incremental aids
We describe tools that provide a single function as 'incremental aids'. Examples include flowcharters and test harnesses. Typically these tools are used in only one stage of the development process and, by themselves, would not be sufficient to develop a complete system.

Data dictionaries
Data dictionaries are usually based on proprietary software products and are used to provide a measure of consistency throughout the development process. A typical example is MSP's Datamanager.

Development workbenches
Examples of development workbenches include Philips Maestro, IBM's CMS, ICL's Program Master, the Unix Programmers' Workbench and Burroughs' PROFS.

Discrete tools
Many system building tools have been designed for constructing only one or two specific types of program, and we refer to these as discrete tools. We identify four types of discrete tools, corresponding to the four types of program that may be found in a data processing system (batch updates, batch reports, online updates and online reports). Thus discrete tools are classified as:

—Batch update tools (although there are not many of these).

—Report writers, such as Mark IV.

—Teleprocessing development systems, such as Oxford Software's UFO and Pansophic's Gener/ol.

—Query processors, such as ASI's Inquire.

Integrated toolkits
Integrated toolkits differ from discrete tools in having their own file definition mechanisms (or data dictionary) and often their own database management systems as well. Also, they are usually applicable to at least three of the four types of program mentioned above. Examples include Cincom's TIS, Microdata's All, Burroughs' Linc, Information Builders' Focus, NCSS's Nomad and Mathematica's Ramis.

The process of selecting the appropriate tools is best done by following four steps:

—Identify the tools that meet the constraints.

—Narrow the field.

—Evaluate the shortlisted tools.

—Select the most appropriate tools.

Initially, the best tools for a single systems environment are selected, but the final step also considers the tools selected in the wider context of the whole systems department.

IDENTIFY THE TOOLS THAT MEET THE CONSTRAINTS

For many systems environments, only a few system building tools are available but, in general, IBM users have a much wider choice of system building tools than users of other mainframe computers, and DEC users have a wider choice than users of other minicomputers.

Figure 3.1 System building tools with a good reputation

Application area	*Product*	*Supplier*
Operational systems	Linc All UFO Gener/ol Applications Master	Burroughs Microdata Oxford Software Pansophic ICL
Management information and decision support systems	Focus Nomad Ramis	Information Builders NCSS Mathematica
User-controlled operational systems	Mapper	Sperry

The ideal starting point for a selection exercise is for there to be no constraint, so that the system building tool that best meets the objectives can be chosen. The ideal tool necessarily depends on the applications requirements, making it impossible to generalise as to what the best products are. Nevertheless, we have several products that have recently won competitive evaluations or helped specific users to achieve impressive results. These systems are listed in Figure 3.1.

It will usually not be possible to make a completely unconstrained choice, however, because the tools selected will need to meet at least one of the following requirements:

—The tools must produce applications that can coexist with existing applications.

—The tools must run on existing computers.

—The tools must produce systems that can communicate with existing systems or terminals.

Compatibility with existing applications

The highest degree of compatibility with existing applications will be obtained by continuing to use the current programming language. This has the further advantage that operational efficiency need not be reduced. In this case, the possible tools are development workbenches, incremental aids and data dictionaries. Some, at least, may already be in use in the information systems department. Others, notably workbenches, are compatible with several different host computers.

A lesser, but often quite adequate, degree of compatibility may be obtained with discrete tools — that is, tools that can be used with standard systems environments. The range of discrete tools available depends on the systems environments currently being used. Many IBM mainframe users, for example, run most of their online systems under CICS. For CICS systems that are to access DL/1 databases there are a variety of discrete tools available including UFO, Mantis and Gener/ol.

For IBM mainframe users with non-standard teleprocessing monitors or database management systems, or for users of non-IBM compatible mainframes, the choice is much more restricted. Indeed, there may be only one or two tools that are compatible with the existing systems environment. In the case of the ICL 2900, for example, the only teleprocessing development systems appear to be ICL's Applications Master and Cincom's Mantis. For some of the more unusual systems environments there is no tool at all.

For minicomputers there is a wide range of tools for DEC machines and a much narrower range for other minicomputers. Some minicomputer operating systems have features that help with system construction, however — the best known being the IBM System 38 operating system, Unix and Pick. Development with one of these operating systems will typically be two or three times as fast as equivalent mainframe developments in a standard language.

Suitable tools for microcomputers are even scarcer, but the requirement for compatibility with existing applications is also less common.

Compatibility with existing computers

The need for system building tools to be compatible with existing computers imposes fewer constraints than those discussed above because it does not require compatibility with existing applications and software. Thus, for example, an Adabas user with this need might consider ADF and Mantis, though both are incompatible with Adabas and hence with the existing applications.

Also, once the requirement to run under a conventional teleprocessing monitor and database management system is removed, the integrated toolkits that are available for the particular hardware environment can also be considered (see Figure 3.2).

Figure 3.2 Some integrated toolkits available for particular hardware environments

Hardware environment	*Integrated toolkit*
Burroughs	Linc
DEC VAX	Focus Info Powerhouse Ultra
IBM	Focus Nomad Ramis TIS
Sperry	Mapper
Wang	Focus

These toolkits often provide higher development productivity than the discrete tools required for compatibility with existing applications. This is because they address a wider range of systems and, usually, more of the development stages. They also reduce the effort involved in passing between development stages or between batch and online operations — a process that often requires great effort.

As with the discrete tools and incremental aids, the choice of integrated toolkits is greatest for IBM and DEC users.

Ability to communicate with existing systems or terminals

Occasionally, system building tools will be used to develop systems that are completely independent of existing systems. More commonly, however, there will be a need for the new systems to communicate with existing systems or terminals, because:

— Developers may need to access both the old and new systems environments, preferably using the same terminals.

— Users may need to access both old and new systems and, again, they may need to use the same terminals.

— Systems built with the new tools may need to access existing systems, possibly to extract data.

A considerable degree of de facto terminal standardisation now exists, and many computers support ASCII, 2780 and 3270 terminal protocols. These protocols can therefore be used to provide systems running on a new computer with access to existing systems, provided that extra software is written to interface with the existing applications. If this approach is acceptable then the communications requirement need not greatly restrict the choice of tools. It will, for example, be possible to use a minicomputer with, say, Pick in what is otherwise a mainframe installation.

The need for terminals to access both old and new systems environments can often be met by adopting a standard terminal and using telecommunications switching techniques (although protocol conversion may also play a part).

In each case the constraint will not be too restrictive if the existing system supports one of the standard terminal protocols. If, however, the existing system supports only some unusual, probably proprietary, protocol then there may be few systems environments, except those provided by the original supplier, that meet the constraint, and the choice of system building tool will be restricted accordingly.

NARROW THE FIELD

The purpose of this second step is both to restrict the number of potential products to be evaluated and to provide information for the evaluation. We have divided this step into two activities:

— The one-day test.

— Detailed study.

If only a few possible system building tools have been identified in the first step then the first activity may be omitted.

The one-day test

The one-day test was devised by Scaffolding (GB) to enable a list of 20 possible system building tools to be reduced to a shortlist of two for detailed evaluation.

To carry out a one-day test, a specification for a small, but real, system is prepared. Each supplier is then challenged to implement as much of the system as possible in a day whilst you observe. If any specific facilities or kinds of complexity are of particular interest, these should be included in the specification.

By focusing on the organisation's needs, rather than the strengths of the tool, you will rapidly come to appreciate the real value and limitations of the tool.

Usually it will become apparent that some system building tools are very unreliable, whilst others are difficult to use for your applications. You should therefore be able to construct a shortlist for further study. We recommend that the shortlist should contain no more than four products.

Detailed study

Each of the shortlisted products will then be studied in detail by:

— Arranging for the suppliers to make presentations about the product.

— Allocating development staff to become thoroughly familiar with the capabilities of the tools.

— Using the tools to construct further test systems.

— Simulating system maintenance and enhancement activities.

— Considering the operational implications of adopting the tools.

For each tool you should seek to provide data for the next stage — evaluation. Special attention should be paid to the way in which the tool handles complex situations, because this is the area in which many

otherwise valuable tools prove inadequate, or even counter-productive.

EVALUATE THE SHORTLISTED TOOLS

Each of the shortlisted system building tools (or set of tools) should now be evaluated under the following four headings:

- Implications for systems development.
- Impact on systems lifecycle costs.
- Deficiencies of the tool.
- Future prospects for the tool and its supplier.

Implications for development

The implications of using a particular system building tool for systems development should be assessed in five ways:

- Applicability: Can the tool be used to build the kind of systems that are needed now and will be needed in the future?
- Productivity: Is system development really faster than with existing tools and techniques? Is maintenance faster? By how much?
- Usability: Who can use the tool? Does it provide features for users, even end users, or is it usable only by professional developers?
- Compatibility with the current development process: Will the tool fit in with existing tools and methods? Does the tool allow new methods to be used?
- Impact on existing skills: Does the tool require developers to acquire new skills? Does it make existing skills obsolete?

In each case the assessment must be made on the basis of the practical experience gained in the previous step, rather than on theories about which features are the most valuable.

Applicability

Some system building tools can be used only for simple systems or for specific types of systems. The best tools, however, can be used for almost as wide a range of data processing systems as can Cobol and PL/1. The following examples confirm that large, complex and critical systems can be built using advanced system building tools:

- The whole operation of the Santa Fe Railway is controlled by a system developed with Mapper. This system runs on four Sperry mainframes, has 67G bytes of storage and supports 2,200 terminals and printers.
- At Morgan Stanley, New York, most of the teleprocessing systems on a multi-mainframe configuration with 122G bytes of storage are developed in Natural.
- At Electricité de France no Cobol has been written for several years, and all development is now carried out using program generators developed in-house.

Nevertheless, even the best system building tools have their limitations. For example:

- ADF uses its own peculiar screen formats.
- Focus has no transaction or file recovery functions (although these features are promised for mid-1985).
- Gener/ol has no batch capability.
- Mapper is unsuitable for large and complex databases, such as bill-of-materials.
- Powerhouse systems can be very inefficient.
- UFO has poor security features.

But these limitations can be circumvented. Thus, ADF can be modified to produce different screen formats, a security system can be written in Cobol and called from UFO, and recovery features can be added to Focus systems. The costs of overcoming such limitations must be included when calculating the overall productivity improvement provided by the tool. In addition, the risks of using a non-standard version of the tool must be considered.

Productivity

The productivity that can be obtained with various system building tools varies greatly. Figure 3.3 gives some of the values for productivity that we have measured, or that have been published elsewhere, for a selection of advanced system building tools. Equivalent values are given for the more commonly

Figure 3.3 Productivity figures for some system building tools

Tool	*Productivity range found (function points per man-day)*	*Representative value (function points per man-day)*
Cobol	0.03 to 0.8	0.35
PL/1	0.06 to 1.8	0.35
APL	0.2 to 1.8	0.18
Focus	1.4 to 8.8	2.3
Linc	3.5 to 10	4.2
UFO	1.2 to 8	1.8
DMS	—	3.5
Mapper	—	2.3
Delta	—	0.8

used programming languages. The figure illustrates that the best integrated toolkits can be ten times as productive as Cobol or PL/1.

It is more difficult to give productivity figures for workbenches, partly because the relative improvements are much smaller. For example, the experience of the most advanced users of Maestro, some of whom now have several years' experience, indicates an improvement of between 10 and 20 per cent. The improvement would presumably be greater with a workbench that, unlike Maestro, could support compilation and interactive testing, in addition to providing good text handling facilities. We are not aware of any such product and, even if one existed, we would not expect the improvement to exceed 25 per cent — thus, productivity might be increased from 0.35 to 0.44 function points per man-day.

We believe that no single incremental aid is likely to provide a productivity improvement of more than a few per cent. Research by Barry Boehm at TRW suggests that a set of such aids can increase development productivity by no more than about 10 per cent.

The high productivity of integrated toolkits is due to five key features:

— The languages used to describe data or define processes are much less oriented to the computer than are conventional programming languages.

— The developer has to deal only with a limited number of languages or subsystems, and these present consistent human interfaces.

— Meta-data (descriptions of files, records and programs) can be defined once and are then automatically passed between the various tools, usually via a data dictionary.

— The tools operate interactively, both in helping the user to define his needs and at the coding and testing stages.

— Analysts can use the tools to develop programs, and therefore no longer need to waste time in writing detailed specifications for programmers.

None of the other types of system building tools provides all of these five key features, as Figure 3.4 shows. Teleprocessing development tools usually do not provide meta-data definitions, are not suitable for use by analysts and may actually increase the number of languages a programmer must learn. Thus, if a system has batch and online elements a programmer will probably be needed both to write the batch parts and to link them with the online parts.

The relatively poor productivity improvement obtained from the use of workbenches reflects the fact that workbenches generally provide only one of the key features — interactive operation. Nevertheless, they do provide support for management activities and for documentation. Although these features are useful, they are secondary as far as improving development productivity is concerned.

The limited value of workbenches was captured, inadvertently, by one workbench designer who described programmers' work as "essentially being concerned with text handling". This is rather like describing a pianist as "essentially being concerned with pushing piano keys". Whilst true, it misses the point. The real basis of programmers' work is logical structures — both for data and procedures. Substantial improvements in development productivity will be produced only by tools that assist in defining and manipulating these logical structures.

An example of a development system that produces program source code, but does not assist directly

Figure 3.4 System building tool features influencing productivity

	Type of system building tool			
Feature	*Integrated toolkit*	*Teleprocessing development system*	*Workbench*	*Line editor, Cobol compiler, JCL, etc.*
High-level language	Yes	Yes	No	No
Limited set of languages	Yes	To some extent	No	No
Meta-data definitions	Yes	Usually no	No	Yes[1]
Interactive use by developers	Yes	Yes	Yes	No
Usable by analysts	Yes	No	No	No
Supports management activities	No	No	Sometimes	No
Supports documentation	No	No	Yes	No

[1] Only if data dictionary is used.

with logical structures, is the Delta toolkit. Its productivity approaches, but does not reach, that of the discrete and integrated toolkits (see Figure 3.3).

Another reason for the relatively low productivity of workbenches is that, because they are aimed principally at programmers, they have to be accommodated within conventional development disciplines. Thus, prototyping and integrated development are not supported by workbenches, and the use of workbenches may actually preclude the use of the advanced tools needed for prototyping.

Usability

The range of people who can use a system building tool depends both on the tool and on the function for which it is to be used. Thus, managers can be taught to use the reporting functions of Mapper or Focus in a few hours, but may be unwilling to accept the longer tuition needed to use the more sophisticated functions.

In general, end users can master online query languages and report writers, whether provided as discrete tools or as parts of integrated toolkits. But allowing users access to these tools often generates substantial, and largely unpredictable, machine loads. Although technical solutions to this problem exist, most tools do not use them.

Rather more training is required before end users can define their own files and write their own applications using integrated toolkits. Probably only a minority of users will ever make the effort needed to master systems such as Mapper and Focus, but the development of intelligent pre-processors (such as Information Builders' Fidel) is enabling users to achieve these results without mastering the full toolkit.

In our view, it is unlikely that even the most advanced system building tools will allow ordinary users to develop substantial transaction processing systems. Although users can be given the required skills, as has been done extensively at Santa Fe Railway, what this does is to convert them to programmers.

Discrete tools, such as UFO and Gener/ol are usually too complex to be used effectively by anyone other than a data processing professional, preferably a programmer. Workbenches can generally be used only by data processing professionals and are really effective only when used by programmers.

Most incremental aids are aimed exclusively at data processing professionals.

Compatibility with the current development process

All types of system building tools can be used with the conventional development process. Workbenches and incremental aids, however, can only be used in this way, which presumably is the reason for the recent popularity of development workbenches. In general, workbenches emphasise the traditional division between analysts and programmers.

Discrete tools can generally fit into the traditional development process, but some of them also support prototyping. Integrated toolkits can be used with a conventional development process but they produce better results with a new development process based on prototyping, analyst-programmer teams and reduced documentation levels. (We shall discuss this further in Chapter 4.)

Impact on existing skills

The introduction of any new tool requires developers to learn a new language, and often a new set of underlying concepts. In the case of an integrated toolkit it may be necessary to learn several new languages — one for data definition, one for online transactions, a variant for batch processes, and a report writer. It is not unusual for conventional programmers already to have to master eight, or even a dozen, different computer languages. If the new tool simply adds to this number it will increase the dependence on technical skills and reduce flexibility. The skills impact of a tool depends on whether it can replace existing tools (at least for most applications) and, if so, whether it will be used in this way. The former will have been established when the selection criteria were determined; the latter depends on the way in which the introduction of the new tool is managed. We shall return to this topic in Chapter 6.

We expect that incremental aids and workbenches will increase the technical content of the developer's job. Incremental aids are almost always used in addition to existing tools and, if several are to be used, they generally do not use the same concepts or have consistent user interfaces.

Discrete tools require developers to acquire new skills, although these tools may mitigate the deficiencies of conventional interfaces to database management systems and teleprocessing monitors. If a discrete tool is well designed this may reduce the level of technical knowledge required. Certainly CICS development tools like UFO have often been used by staff without CICS and DL/1 expertise to build online database systems under CICS.

Integrated toolkits generally reduce the technical complexity of the systems environment, allowing less technically skilled people to develop applications. Thus systems can be developed by analysts and, sometimes, by users.

Impact on systems lifecycle costs

To determine the total cost of a typical system over its useful life it is necessary to consider development, maintenance and operating costs and the likely life of the system.

Development costs

The main elements of development costs are staff, computer power and the cost of acquiring and introducing the development tools.

We have already shown that system building tools generally reduce the amount of manpower required to develop systems. This element of development costs will therefore fall, but the reduction may be partly offset by a need for people with greater skills and a need to retain people to ensure continuity with previous methods.

Almost all system building tools require more computer power than conventional tools. Thus, one Focus user finds that its development staff are online for twice as long and that each online session uses 50 per cent more cpu time. This increase is offset by the productivity gains obtained from the tool, however, and the same user estimates an overall reduction of 50 per cent in development machine costs. It is worth noting that development staff (assuming their number stays constant) will need additional computer resources in order to produce more systems, once a new system building tool is introduced.

The costs of acquiring and using a system building tool include purchase, rental, service and training costs and, sometimes, the costs of reduced productivity during the introductory period. If these costs are capitalised and spread over the life of the tool they will rarely amount to more than $1,000 per development man-year (except for a development workbench, which might cost as much as $4,000 per development man-year).

Maintenance and enhancement costs

Maintenance and enhancement costs are determined by the level of maintenance, the rate of enhancements, the productivity of the relevant staff, and computer costs.

The maintenance level is the amount of work needed simply to keep the system running. Ideally this should be zero, but it rarely is. We find that systems produced quickly and with high-productivity tools do not usually contain hidden problems that will adversely affect the maintenance level (certainly no more than with conventional methods).

The rate of enhancements is determined by the volatility of the business environment. It should not depend on the development tool used but, if the tool makes it less expensive to include enhancements, users will usually ask for more to be done.

Most system building tools provide improvements in maintenance productivity equivalent to that for development productivity. The main exceptions are:

— Simple program generators. In this case maintenance productivity may be reduced.

— Libraries and data dictionaries, which provide greater benefit for maintenance and enhancement activities than for the initial development.

Operating costs

Tools that generate programs in conventional languages generally produce systems that are, at best, as efficient as programs written directly in those languages. Beyond this type of tool it is not possible to generalise about the impact of system building tools on machine efficiency.

Some development tools — All, Gener/ol, Linc and Protos, for example — produce more efficient systems than Cobol. Most tools, however, use substantially more cpu time and memory. Figure 3.5 gives comparative figures for cpu usage for several system building tools.

Experience has shown that it is much harder to predict the performance of systems developed with advanced system building tools than of those developed with Cobol. This is especially so for integrated toolkits, where optimising the procedures and data structures may make enormous improvements. One Focus user produced a 20-fold improvement in one system by careful tuning.

Advanced tools often also require more main memory, but the requirements for disc storage, peripherals, operator support and telecommunications are generally little different from those for conventionally developed systems.

Figure 3.5 CPU efficiency of systems generated with system building tools

Tool	*CPU usage relative to Cobol*
ADF	0.8 to 2.4
All	less than 1
Focus	0.8 to 3
Gener/ol	not more than 1.1
Info	3 to 5
Mantis	2 to 3
Linc	not more than 1
Protos	0.8
Powerhouse	10

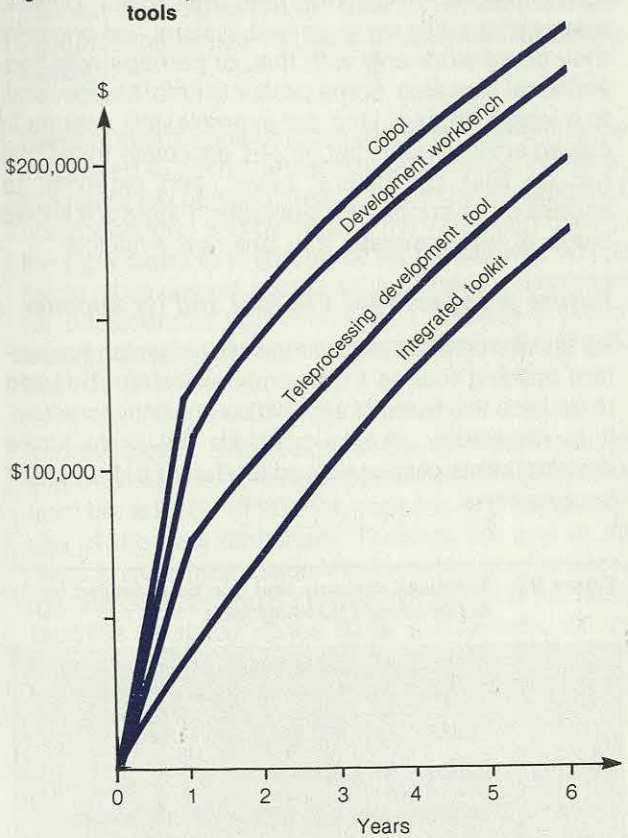
Figure 3.6 System building tools used in calculating relative life cycle costs

Tool	*Relative productivity*	*Relative operating cost*
Cobol	1	1
Development workbench	1.2	1
Teleprocessing development tool	2	1.2
Integrated toolkit	4	1.5

Life of the system

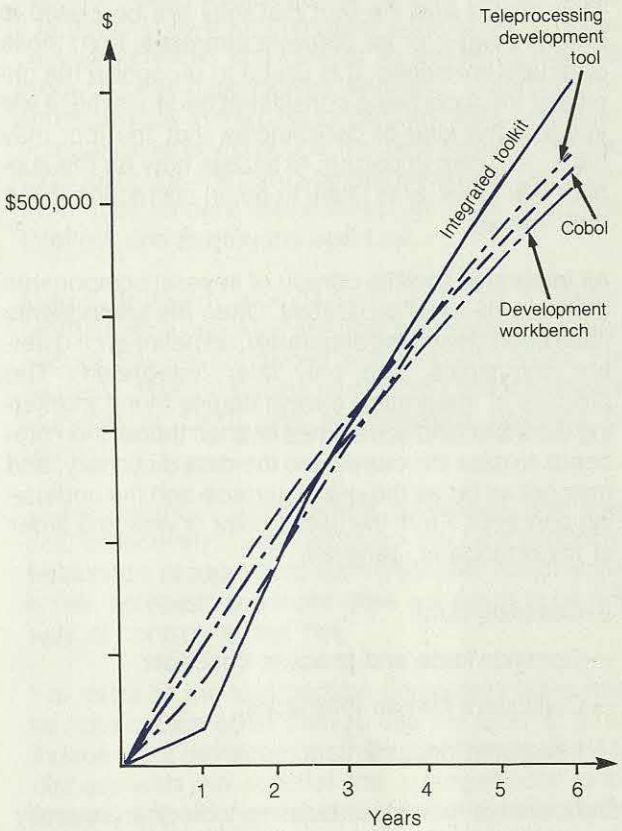
The typical lifetime of applications will have been established earlier when the objectives for using system building tools were determined. To illustrate the relative lifecycle costs of four different tools we have calculated the costs over time for two different types of system. The four tools, and the assumptions made about their productivity and operating costs (relative to Cobol) are shown in Figure 3.6. Both types of system are assumed to have maintenance costs of five per cent per year of the original development cost. One system has annual operating costs equivalent to 20 per cent of the original development cost, and the other has annual operating costs equivalent to 80 per cent of the original development cost. The lifecycle costs of developing, maintaining and operating these two types of system, using each of the four development tools, are shown respectively in Figures 3.7 and 3.8.

Figure 3.7 Life cycle costs for four different development tools



Assumptions:
Annual maintenance = 5% of development cost
Annual operating cost = 20% of Cobol development cost

Figure 3.8 Lifecycle costs for four different development tools



Assumptions:
Annual maintenance = 5% of development cost
Annual operating cost = 80% of Cobol development cost

Figure 3.7 shows that for a system where annual operating costs are relatively low (assumed to be 20 per cent of the Cobol development cost), the integrated toolkit is the least expensive option over any reasonable period of time.

On the other hand, Figure 3.8 shows that, for a system with higher annual operating costs (assumed to be 80 per cent of the Cobol development cost), using Cobol becomes less expensive than the integrated toolkit after three years and less expensive than the teleprocessing development tool after four-and-a-half years. After six years the total costs of using the integrated toolkit are about seven per cent higher than with any of the other three options. By this time there is little difference between the lifecycle costs of the other three, although assuming a different level of maintenance would change their order.

Deficiencies of the tool

The deficiencies of many system building tools reflect their origins and the fact that they are now used in different ways, or for different purposes, from those originally envisaged. It is useful to recognise the origins of the tools being considered because these will suggest the kind of deficiencies that the tool may have. It is then important to assess how far the supplier has gone, or is likely to go, in correcting these deficiencies.

All integrated toolkits consist of several components that may be used separately. Often the components have been developed separately, sometimes by different companies, and only later 'integrated'. The process of integration always begins in the marketing literature (and sometimes finishes there), then proceeds to data structures and the data dictionary, and may get as far as the user interface and the underlying concepts. From the users' point of view, the order of importance is, generally:

—Common data.

—Common data and process concepts.

—Consistent human interfaces.

—Integrated documentation.

Deficiencies in system building tools are generally caused either by an integrated toolkit having originally been developed as a set of separate tools, or by the tool having been built originally for use in a specific environment such as a batch environment, or for use by Assembler programmers, or for high-volume teleprocessing systems, or for end-user timesharing.

In a batch environment, programs are treated as files of static text and testing is based on the use of program listings. Online operation requires the programs to be treated as structures and, preferably, interpreted. It also requires testing aids that both relate directly to the program and can benefit from the interpreter's knowledge of program structure and data names. Cobol compilers usually show their batch, and card-orientated, origins very clearly. It is extremely difficult to modify a batch-based tool for interactive use and this has rarely been done successfully.

Many of the older system building tools (such as Cobol and PL/1) were conceived as a means of producing Assembler programs. Debugging has therefore required a knowledge of Assembler, and again this deficiency is difficult to remedy.

In order to enable high-volume systems to be developed with reasonable operational performance, some system building tools provide the developer with a great deal of control over the management of system resources. In many cases the use of these tools absolutely requires the programmer to exercise such control which, in turn, forces him to become an expert in the internal operation of a teleprocessing monitor or database management system.

Many integrated toolkits (Ramis, Nomad and Focus, for example) were originally developed for end-user timesharing. As a consequence, these products have deficiencies in the areas of security, integrity and efficiency, especially for multi-access systems, and in access by inexperienced users. They are also likely to be very inefficient for any residual, but nonetheless necessary, batch applications.

Nevertheless, as the use of system building tools for developing operational systems increases, the suppliers are acting to remedy these deficiencies. For example:

—Information Builders has added automatic recovery and semi-tutorial facilities to Focus.

—Sperry has supplemented the Mapper interpreter with a compiler that will generate more efficient systems.

—Mathematica has added better security features, and a compiler, to Ramis II.

Most integrated toolkits use their own, private, integral database management system, and originally they could work only with this, or perhaps with conventional files also. Some products (Info, Mapper and, to a lesser degree, Linc, for example) still operate in closed environments but others, especially those that run on IBM computers, have been extended to access other suppliers' databases. Figure 3.9 shows some of the accesses that are now available.

Future prospects for the tool and its supplier

So far we have considered the way in which the system building tool as it presently exists can be used to develop the types of application currently foreseen. It is necessary also to consider the likely future developments of the tool and the future prospects of its supplier.

Figure 3.9 Database systems that can be accessed by some system building tools

Tool	*Database systems that can be accessed*
Focus	IMS, IDMS, Adabas, System 2000, Vsam, Isam, Osam, SQL/DS, Total, Model 204
Gener/ol	Adabas, IMS, Total, Vsam, Isam
Nomad	Vsam, Qsam, IMS, IDMS, SQL, DB2, Focus
Natural	Adabas, Vsam, DL/1, IMS
Intellect	Adabas, IDMS, SQL, Focus, Vsam

The system building tool

In the longer term we expect the coverage, power and usability of advanced system building tools to be increased by their suppliers.

Most system building tools are presently aimed either at end-user computing or at the professional development of data processing systems. Suppliers will need to address both these markets and they can do this most effectively by developing a family of compatible tools. Such families are already available from suppliers such as Software AG, Burroughs, ICL, IBM and Mathematica. Several suppliers have added graphical, statistical and text facilities to their families in the last few years, although these are often rather primitive. In the future we expect to see these additional features enhanced. Where appropriate, suppliers will also add compatible products for areas such as computer-integrated manufacturing and text systems development. Communications support also will be enhanced in the areas of microcomputer-mainframe communications, videotex, electronic mail and inter-business communications.

Report writers are now very powerful and so the scope for improving them is limited to integrating the output of data, text and image, and providing assistance to the user. For online query processors the next step will be the introduction of simple learning functions similar to those already available with Intellect.

There remains considerable scope for improving the tools used to build transaction processing systems. Programs will be simplified by making data validation non-procedural and by moving the validation rules to the data dictionary. (Dictionaries themselves will be made more powerful and usable.) There is also room for considerable improvement in the way in which logic is specified, though it is not clear how this should best be done.

Improvements in usability will come in part through improvements in the languages, but more significantly from the addition of learning features and the greater use of the data dictionary. Towards the end of the decade we expect to see intelligent knowledge-based systems intervening between the user and the basic facilities. In many cases these subsystems will be located in an intelligent workstation, through which the user may, transparently, gain access to a variety of hosts and applications.

The supplier

The future survival and, indeed, prosperity of the supplier is of basic importance to a prospective user of system building tools. Possible suppliers now fall into three groups: computer companies, major software companies and independent developers.

In general, computer companies are reasonably secure (although the withdrawal of a major mainframe company would surprise no one). Nevertheless, it is not unknown for computer companies to fail to deliver promised software or to cease to develop products once described as 'strategic'. Unfortunately, and especially in the case of DEC and IBM, computer companies rarely produce the best system building tools, although they are sometimes able to buy them from independent developers, as with IBM and Intellect and Burroughs and Linc.

Major software companies such as Cullinet, Mathematica and ADR are now well-established and secure. Several are owned by large corporations. For example, Martin Marietta owns both Oxford Software (supplier of UFO) and Mathematica (supplier of Ramis).

There are many independent developers and they are, collectively, responsible for the best products. Selecting a product from such a supplier does involve a risk, however, and there does not seem to be any way of controlling this risk.

For users of the less popular computers there may be little choice other than to use the product of an independent developer. In this case it is important to discuss with the supplier the arrangements to be made if support and maintenance of the product were to be discontinued. For example, will the supplier provide the source code and all internal documentation, and under what terms? It is also important to consider whether the benefits from using the tool could continue if it were not enhanced, because it may not be.

As Figure 3.9 illustrated, the user of a popular system building tool has a further line of defence. There is an increasing tendency, especially in the IBM environment, for other suppliers to provide access to the more popular tools. Some suppliers provide conversion functions as well. If such a supplier should withdraw from the market, or cease to enhance its product, the user always has the option of moving to a competitive product.

SELECT THE MOST APPROPRIATE TOOLS

At this final step of the selection stage the results of the evaluations should be set against the objectives established in the second stage for using system building tools in each of the systems environments.

It is often quite easy to make a good choice, but hard to identify the best choice. The standard methods of weighted scores and risk analysis may be used to resolve this difficulty. However, if there are now two or three advanced system building tools that meet

the criteria, it may not much matter which is chosen. The best rule for breaking the tie is probably to choose the tool with the largest number of European users.

The selection should be carried out for each systems environment. Here there is a danger that the desire to start afresh with new tools in a new environment, whilst still needing to maintain existing systems, will lead to a proliferation of 'best' tools. There are, however, good reasons for restricting the number of system building tools in use in a particular organisation, and therefore using tools other than the best in some cases, in the interests of flexibility and standardisation. Ideally the experienced developer should be competent in using all of the selected tools and should be able to choose the best one for any particular purpose.

CHAPTER 4

DEFINE THE NECESSARY CHANGES IN THE DEVELOPMENT ENVIRONMENT

The system building tools that we have called incremental aids and workbenches fit naturally into conventional systems development processes and are compatible with traditional programmer and analyst functions. These tools can therefore be used in conventional development environments.

Advanced system building tools, which we have divided into discrete tools and integrated toolkits, generally provide greater benefits to their users. Many organisations do not achieve the benefits because the tools are used with inappropriate development approaches and methods. In most installations, the following changes in the development environment will be needed before the benefits of using advanced tools can be realised:

- —Abolish or reduce the split between programmers and analysts.
- —Use prototyping wherever possible.
- —Increase the amount of interactive computer support available to development staff.
- —Reduce the amount of documentation.
- —Accept the limitations of the system building tools.
- —Deliver databases rather than systems.

ABOLISH OR REDUCE THE PROGRAMMER-ANALYST SPLIT

The traditional division between programmers and analysts should be reduced or, preferably, eliminated in order to get the best value from advanced system building tools and to match staff skills to developing needs.

Traditional programming skills, and particularly knowledge of complex systems software, are not needed in order to use advanced system building tools. This is especially true for integrated toolkits because these products can shield the user from a great deal of tedious and complex detail. The need for specialised language-related knowledge has caused development staff to describe themselves in terms of the software, as CICS-Cobol programmers for example, rather than in relation to their organisation. The consequent high mobility of development staff has been a great problem for many organisations.

Most advanced system building tools can be used by analysts and some even by users. Some organisations have found that analysts make better use of system building tools than programmers because their thinking is not constrained by their experience with conventional languages. There are many examples of inexperienced staff achieving remarkably high productivity with advanced tools.

Where separate people perform the roles of analyst and designer, considerable effort is required to transfer knowledge between them. This process, which often takes the form of writing and maintaining copious documentation, is very time-consuming and can easily lead to errors.

Where a new development team is being established, as will usually be the case if systems are being rebuilt, we recommend that all staff should perform the full range of development functions, though they may do so with differing frequency. With system building tools, many projects can be handled by only one or two people carrying out all the development functions, but for larger projects some form of team structure may be needed.

If some systems have to meet strict performance requirements, possibly because of high transaction volumes, it may be necessary to use a less productive but operationally more efficient language. Such requirements should be rare, and should be met by concentrating expertise in a small conventional programming team.

If there is a long-term commitment to the maintenance of old systems, this should be allocated to a separate maintenance team. (In some organisations the maintenance team consists largely of freelance staff with traditional programming skills.)

USE PROTOTYPING

Prototyping is feasible only if it is very much faster to build a prototype than a full system. The use of system building tools enables prototypes to be built quickly and inexpensively.

Wherever possible, prototyping should replace formal specifications as the usual means of identifying requirements, because:

—Conventional methods are expensive and are based on the false assumption that users can relate to an abstract specification.

—Prototyping improves quality, increases user commitment and reduces risk.

Prototypes are an excellent way of helping users to clarify their requirements — and this is not confined to cases where there is only a single 'customer' for the system. Prototyping worked well at VP-Centralen, where there were over 200 'customers'. (The experience of VP-Centralen formed the case history in Foundation Report No. 45 — Building Quality Systems.) Where there are many customers, however, conflicts of requirements will arise, due both to technical disagreements and to vested interests. In this situation an effective means of resolving disputes as they arise is essential.

Conventional methods are expensive

In many organisations, producing system specifications consumes 30 per cent or more of the total development effort. Thus it may take up to eight man-hours to specify each function point for the desired system. By contrast, it may take as little as 20 minutes per function point to build a prototype. Even if several prototypes have to be built, there will still be a substantial reduction in the overall development effort.

Conventional methods are based on false assumptions

Conventional development methods assume that users can specify systems before they have any experience of those systems and can relate effectively to static, largely verbal, descriptions. Some users are able to do this in some cases — when specifying the replacement of a satisfactory system, for example. In general, however, users find it very much easier to see what is wrong with a system than to visualise a complete new system. The conventional assumption is even less valid for management information systems and systems that support the moment-to-moment operation of the business than it has been for conventional 'back-office' administrative systems.

Conventional methods also assume that existing formal methodologies are based on rigorous engineering principles, in that the correct use of the methodology will necessarily produce the required system. Anyone who has substantial experience of developing systems will know that this assumption is very doubtful. This fact was recognised as long ago as 1947 by computer pioneer Alan Turing when he told the London Mathematical Society that, based on his own work, "up to a point it is better to let the (programming) errors be there than to spend such time in design that there are none". We believe that this remains true, at least for the process of establishing user requirements.

Conventional development approaches place a heavy emphasis on pre-specifying all details because it is very expensive to develop and modify systems with conventional tools. Using advanced system building tools, prototypes may be built at a rate of up to 20 function points per man-day, which is up to 100 times as fast as conventional development. Changes to the prototypes can also be made very quickly.

Prototyping improves systems quality

Users will review the prototype system and, if it fails to meet their needs, will get it changed. Thus the users' insights are brought to the problem at an early stage, ensuring that the system provides the right functions in a convenient way.

Prototyping increases user commitment

In reviewing a prototype, users are more intimately involved in the specification process than is usual. Apart from its value to the final specification, this involvement encourages the user to feel that the system being built belongs to him. In addition, users will be able to consider, at an early stage, how the system is to be introduced. Thus prototyping not only improves the system but increases user commitment both to the design and to the implementation of the system.

Prototyping reduces risks

Prototyping reduces the risks both of total failure and of costly overruns, because prototypes are an inexpensive means of detecting serious specification errors and/or wrong business assumptions. Estimates of development and operating costs will also be more accurate if they are made at the end of the prototyping stage because, by then, more will be known about the development and operational requirements.

PROVIDE GOOD COMPUTER SUPPORT

Unless advanced system building tools are being used principally to reduce the number of development

staff, their introduction will require substantial investments in development computer power.

In our view, computer support for advanced system building tools requires personal terminals, response times below two seconds, and access to computer power at any reasonable time. Research by IBM, TRW and others has confirmed that providing these facilities significantly improves development productivity.

The use of advanced system building tools requires that development staff be provided with good computer support, because:

—A high proportion of development staff will use the tools.

—Staff consume more computer resources when using the advanced tools.

—Better machine support allows extra functions, such as documentation, to be provided.

—Inadequate access to terminals or excessive response times reduce the productivity of development staff.

A higher proportion of development staff will use the tools either because the programmer-analyst division will have been abolished, or because analysts will use the tools for prototyping.

Staff consume more computer resources with the new tools because the tools have been designed to take advantage of online operation. Machine efficiency has, in effect, been traded for human efficiency.

Many conventional information systems departments are curiously ambivalent about their own use of computers. Their staff program computers, but rarely use them to support other activities. Development computers can be used (but often are not) for preparing and storing systems documentation, for electronic mail, for budgeting and for project control purposes.

Development staff will not make full use of computer facilities unless doing so is as easy as writing. They therefore need their own terminals, just as they need their own telephones and pens. When developing teleprocessing systems, staff may need either two terminals (one to show the screen and one to access the development system) or a large-windowing system such as the IBM Display Panel.

But merely providing a personal terminal is not sufficient. Without adequate response times its use will be inconvenient and staff will be obliged to fall back on paper listings. Many useful facilities require the computer to search a file and this in turn requires a substantial amount of computer power if the search is to be completed in an acceptable time.

The use of computerised development tools is still inhibited in some organisations by restrictions on the operating hours of the development computer, by the times when compilations may be done or by the amount of storage available to development staff. Effective use of the tools certainly requires these constraints to be reduced or eliminated.

In most information systems departments, adequate computer support for development staff can be provided only through the use of a separate development machine. Some organisations compromise by locating the development machine in an operational computer centre, remote from the development staff, so that it can serve as a backup for an operational machine. This approach should be discouraged because it leads to excessive response times and periodic loss of service during which the developers are unable to work effectively.

REDUCE THE AMOUNT OF DOCUMENTATION

Most information systems departments have a great deal of documentation for their systems. In the best-run departments the documentation is comprehensive and current, but in many organisations it is neither. As a result it is rarely referred to, and the considerable effort that goes into writing and revising it is largely wasted. Advanced system building tools allow the amount of documentation, especially design documentation, to be reduced substantially, because:

—Prototyping is generally better and faster than producing a detailed specification.

—Existing documentation is often out of date.

—Design documentation often encourages maintenance staff to make changes without thinking through the implications for the application logic.

Advanced system building tools allow prototypes to be built at a rate of up to 20 function points per man-day (that is, each function point will take about 30 minutes to produce). By contrast, the effort required to document a conventionally produced system is typically in the range of two to six man-hours per function point.

It is not possible to eliminate documentation altogether. Operating instructions and user guides (where these are not provided online) are essential parts of the delivered systems. Top-level documentation, such as systems flowcharts, will be necessary to provide users and maintenance staff with an overview of the system context. In maintaining complex

systems it is often difficult to locate all the places in which a particular data item is used. The need to do this may be reduced by a database management system that provides users and programmers with a view of the data that reflects their actual interests, rather than the physical storage arrangements (a property known as data independence). Alternatively (or additionally), a data dictionary can be employed.

Reducing the amount of documentation should not be restricted to simple in-house systems. At VP-Centralen, for example, data structure diagrams and structure charts were not retained during the development phase. At the end of the two-year project the documentation consisted only of:

—The structured specification.

—Source code.

—Operating guidelines.

—The user manual.

In many cases the source code will be reasonably self-explanatory, and its readability can be improved by the use of meaningful data names and careful use of indentation. Where the purpose of the code is not immediately clear, annotation should be used.

ACCEPT THE LIMITATIONS OF THE SYSTEM BUILDING TOOLS

Most system building tools have some limitations as to the types of system that can be produced. The most common limitations are restrictions on screen and report formats and on the use of function keys. These restrictions may be inherently obstructive (for example, the inability to use the same function keys in ADF and Cobol modules of a teleprocessing system), or they may simply be incompatible with existing standards. Such restrictions may justify the rejection of the tool but, if the decision has been made to use it, then the restrictions should be accepted and compromises should be devised. Nevertheless, a great deal of effort can be wasted in circumventing irritating, but ultimately trivial, features of system building tools.

Incremental aids, and workbenches such as Maestro, allow the programmer to access all the features of the chosen programming language, so that no design compromise will be needed. However, the use of such tools may require changes in the way in which staff carry out their work. Maestro, for example, requires the results of compilations to be transferred to its own files for inspection rather than allowing the IBM timesharing features to be used for this purpose.

The implication is that the necessary procedures and support must be put in place to make the mode of operation of a tool a productive and convenient one. It may also be necessary to change the department's standards in order to obtain the greatest benefits from the tools.

DELIVER DATABASES RATHER THAN SYSTEMS

An increasing proportion of information systems provide management information rather than automate routine operations. Although these systems often depend on basic business data, they need to meet managerial requirements for information that may change very rapidly.

In these situations, placing a systems development professional between the user and the database may only delay the production of the required information. A reasonable approach, now being adopted by several Foundation members, is for the information systems department to provide suitably structured databases together with tools for data retrieval, analysis and reporting. Such systems are sometimes referred to as 'reportless systems' because no reports need be defined in their specifications.

The tools provided to users may be statistical analysis packages, such as SAS and SPSS, or integrated toolkits such as Ramis and Nomad. In the latter case, the same tools may be used both by development staff and by users, and this may allow some flexibility in the allocation of work.

A PAUSE FOR BREATH

By the end of this stage, the system building tools most appropriate for the organisation will have been evaluated and selected, and the changes required in the development environment to make the best use of the tools will have been defined. The next stage is to begin to use the tools.

CHAPTER 5

BUILD THE PILOT APPLICATION

Once the tools have been selected and the changes in the development environment have been defined, it would be ideal to introduce the new tools and practices at once into all parts of the information systems department. This is usually impractical because of:

—The number of people affected.

—The need to continue supporting existing systems, tools and methods.

—The inevitable uncertainties about some of the proposed practices.

It is therefore usually necessary to start with a pilot system or set of systems.

The objectives of a pilot application are often misunderstood. A pilot should not, in general, be another stage in the process of selecting system building tools; it should be the first real application of the tools and methods that have already been selected and defined. The pilot may reveal some critical mistake in the selection or definition of new methods, but this should be regarded as a very unlikely outcome. If the previous stages have been performed properly and management is visibly committed to the new tools and methods, there should be no last-minute surprise.

The objectives of a pilot application are to check that the selected tools and new development methods will work in the organisation and to begin the process of converting from older tools and methods. The process will normally proceed in four stages:

—Select the pilot application.

—Establish the pilot team.

—Build the application.

—Evaluate the experience.

SELECT THE PILOT APPLICATION

The pilot application selected should be reasonably representative of the systems that will be built in the future, and it should be capable of being implemented within a few months — ideally, three or four. In addition, the application should be significant to the business. The advantages of choosing a representative application are two-fold. The lessons learnt will be of general application and the success of the pilot will be seen as being relevant to other systems. On the other hand, choosing an atypical application will encourage the sceptics and pessimists (who are found in every information systems department) to resist the further use of the new tools and methods.

The pilot application should be of modest size so that the lessons can be learnt without undue delay and management can retain control of the change process. Experience has shown that it is all too easy to lose control. In one British company the first Focus systems were so successful that users were demanding that their systems should be developed in Focus even before the systems department had evaluated the pilot applications.

The application should be significant to the business in order to maximise the benefits to the business and to the information systems department, and to secure any senior management support that may then be needed — for training courses or new hardware, for example.

ESTABLISH THE PILOT TEAM

Where possible, the members of the pilot team should be chosen for their interest in business results rather than in the technology of the new methods. They must be people who can adapt the changed development environment to the requirements of the project without losing sight of the overall objectives.

Because of the high productivity of advanced system building tools the pilot team can be quite small. In two months a team of three should be able to complete a system of between 200 and 300 function points (equivalent to about 15,000 lines of PL/1). All members of the team should be properly trained in the use of the tools and should be committed to the new development procedures.

Ideally, the team should be given time to gain some experience of using the tools before embarking on

the pilot. If this is not done the pilot experience will not be fully representative of later project experience.

BUILD THE APPLICATION

When the team is ready, work can commence on the pilot application.

The pilot project should operate in as normal a way as the abnormal circumstances allow. However, proper records should be kept of the effort expended on the project and its progress even if this will not usually be done. Problems, errors and solutions to problems should also be carefully recorded.

EVALUATE THE EXPERIENCE

The main aim of the evaluation should be to determine whether the tools and new development procedures will allow the department to meet its objectives for the use of system building tools. The secondary aim should be to identify any changes needed in the tools or procedures to increase their effectiveness.

The evaluation should be conducted by someone who was not personally involved with the pilot application but who is familiar with the objectives, tools and procedures concerned. Given the importance of the consequent decisions, the systems development manager will often be the best choice.

In addition to the project team, the evaluation should involve users, technical support staff and operations staff. The evaluation should assess the development productivity actually achieved and the quality of the systems delivered. It should also look carefully for any unforeseen problems.

The results of the evaluation should be:

—A recommendation for the future use of the new tools and procedures. Should the tools and procedures be brought into general use and, if so, what systems or functions should be excluded?

—Recommendations for any necessary changes or extensions to the tools and procedures.

—Estimating guidelines for systems developed in the new way.

—Any consequent recommendations for changes in staffing, terms of employment and office organisation that will be needed to exploit the new tools and procedures.

The evaluation will show whether it is possible to meet the objectives originally set, using the tools selected, in the revised development environment. If the decision to proceed remains valid, the tools and procedures should be fully installed.

CHAPTER 6

EXTEND THE USE OF SYSTEM BUILDING TOOLS

Peter Drucker has argued that the most important management decisions are always negative ones — decisions to stop doing things. That is certainly true for the introduction of system building tools. The key decision is to abandon the old methods of systems development in favour of the new ones. Information systems management should make this completely clear and should express its commitment to the new tools and methods by making appropriate changes in staffing, facilities, standards and procedures. There will also be a need to manage the changeover period.

MAKE THE REQUIRED STAFFING CHANGES

For most of the system building tools discussed in this report, though not for incremental aids and programmers' workbenches, the new development environment will require analysts to have much closer contact with users and to perform much of the development themselves. This will usually require:

— New criteria for staff selection.

— Business and technical training for existing staff.

— Reorganisation of development teams.

— New staff policies.

New criteria for staff selection

The new development environment will require rather different people from those presently found in many information systems departments. It is rarely possible, let alone desirable, to replace all of the existing development staff. But it is desirable to rethink the criteria for recruiting new staff. Analyst-programmers should be recruited for their interest in results and business knowledge, rather than for their technical knowledge and experience. Indeed, some types of experience — Cobol programming in a large conventional development team, for example — will be a definite disadvantage. The attitudes of such people are unlikely to be compatible with the effective use of system building tools.

Analyst-programmers should be self-motivating, results-oriented and able to relate well to users. They will need to operate in small teams, or even alone, and it will not be possible to manage them en masse. They must be people who will set their own priorities and who will gain satisfaction from the results they achieve for their clients, rather than from technical virtuosity for its own sake.

Analyst-programmers need skills in business, data analysis and in the use of advanced system building tools. That is also the skills priority order because it is easier to learn to use a tool than it is to learn data analysis, and it is easier to learn data analysis than to learn business skills.

Business skills are needed to ease communication with clients, to avoid ignorant mistakes and to secure the respect of users. It is no accident that Morgan Stanley Bank in New York obtains very high productivity from its development workforce, nearly half of whom have banking qualifications.

Data analysis skills are needed to ensure that data structures are not unduly tied to the requirements of individual applications, but provide flexibility for user access and future enhancements.

Skill in using the particular tools is needed to avoid gross inefficiencies and to allow the rapid development work on which prototyping, in particular, depends.

Business and technical training for existing staff

Existing programmers will need some business and systems training if they are to function effectively as analyst-programmers. Regrettably, not all programmers will be willing, or able, to make this transition.

Existing analysts will need some technical training to convert them to analyst-programmers. There should be no serious difficulties in achieving this, but the idea, found in some installations, that personal use of a computer (or terminal) constitutes a loss of status may have to be firmly repudiated.

Reorganisation of development teams

To exploit fully the opportunities provided by system building tools it will be necessary to reorganise the development staff. The best results will usually be

achieved by small integrated teams that are allocated to particular business areas. For many Foundation members, existing systems teams may be used as the basis for the new development teams by introducing retrained programmers and dividing the teams into smaller units.

In the longer term, it is desirable that development staff should rotate between business areas, but this should be left until the staff have properly assimilated the methodological changes.

New staff policies

Existing staff policies will reflect the existing development environments. Introducing new development environments based on system building tools may require changes in remuneration and career progression.

The skills needed by the new analyst-programmers are greater than those of conventional analysts and programmers. Although job satisfaction will be greater for those using system building tools, the organisation will probably need to increase salaries and/or fringe benefits if it is to retain its increasingly skilled and effective development workforce.

The overall effect of these changes will be to distance the new analyst-programmers from existing programmers and technical support staff. It will no longer be possible to rotate staff either between applications development and technical support functions, or between maintenance of older systems and development of new ones.

It will therefore be necessary to identify distinct career paths for maintenance programmers, analyst-programmers and technical support staff. Some organisations may have to accept that they cannot provide continuing career development for some existing categories of staff, and should plan for them to leave and be replaced through further recruitment.

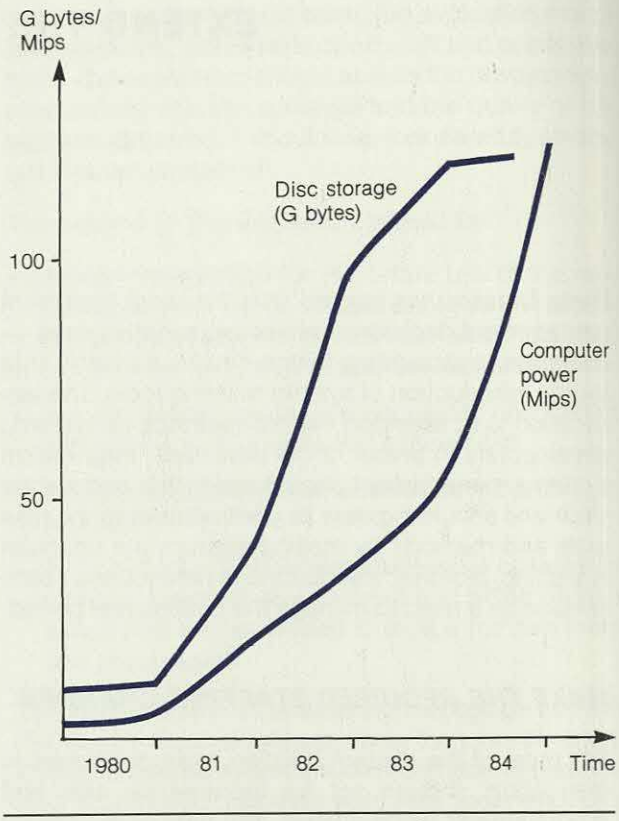
PROVIDE ADEQUATE FACILITIES

For the reasons given in Chapter 4, every developer should have his own terminal, and some may need more than one. In many organisations this will require a significant increase in the number of terminals installed in the information systems department. In turn, this will require sufficient computer power to provide the necessary support. The new way of working may also require some changes in the working environment. The necessary steps are to:

—Install extra computer capacity.

—Select software for additional support functions.

Figure 6.1 Growth in computer capacity at Morgan Stanley



—Consider a move away from open-plan offices.

—Provide adequate technical support for the tools.

Install extra computer capacity

Extra computer capacity will be needed both to support developers with their new tools and to run the new applications that will be built. As an extreme example, we show in Figure 6.1 the growth in computer capacity at Morgan Stanley over the four-year period since the decision was made to standardise on Natural for applications development. Computer power has increased at an annual compound rate of more than 130 per cent during this period. Most of the growth in capacity has been required to run new applications and is thus a measure of the success of the policy, not of its failure.

There may also be a need, as discussed in Foundation Report No. 38 — Selecting Local Network Facilities — to change the supporting network in order to provide very fast, preferably subsecond, response times.

Select software for additional support functions

Developers will benefit from computer support for activities other than system development, including document preparation and presentation, messaging,

planning and modelling. Software for word processing, graphics, electronic mail, project management, estimating and systems modelling may therefore be useful. Appropriate systems should be selected, or developed, and installed on the development computer.

Consider a move away from open-plan offices

Open-plan offices have become increasingly fashionable since the 1960s for all kinds of office work. The case for open-plan offices for systems development staff has, in our view, never been made convincingly. In general, systems development staff differ from ordinary office staff because they are motivated more by results and personal development and less by social factors.

In the last three years, experiments at IBM and TRW have shown that a move away from open-plan offices to individual, or small-team, offices can help to make substantial improvements in productivity, even when conventional programming languages are used.

Some further physical facilities may also be needed to make the best use of the prototyping opportunities provided by system building tools. The physical facilities needed for effective prototyping have been described in some detail by Bernard Boar of AT&T in "Application Prototyping". He argues that a demonstration room with a large-screen projector driven from a terminal is a valuable addition to project rooms.

Provide adequate technical support for the tools

Any significant piece of software requires proper technical support. This is especially true for advanced system building tools, which are very complex and which affect the integrity and efficiency of all the systems that are built with them. The technical support staff should provide advice about using the tools as well as maintaining them. They should also (probably) provide a specialist service for tuning applications where operational efficiency is a significant concern.

UPDATE THE STANDARDS AND PROCEDURES

The existing standards and procedures will have been developed for the existing tools and methods. In revising the standards and procedures, changes will typically be needed to define:

- —The rules for using the new tools.
- —The procedures for prototyping.
- —New technical standards.
- —The procedures for regular monitoring of productivity.

The revised standards and procedures should also reinforce the view that systems are to be regarded as corporate, rather than personal, property — an attitude sometimes termed 'egoless programming'. This view is best encouraged by design and code reviews, inspections or walkthroughs. These procedures are especially valuable as a means of sharing expertise when unfamiliar tools and methods are being introduced.

Rules for using system building tools

To meet the objectives identified in Chapter 2, system building tools must become the normal means of systems development. Unless this is achieved the systems department will not be able to make the quantum jump in productivity that can be obtained.

Once the tools have been installed, exceptions to their use should be permitted only when authorised by senior management and should be carefully tracked. In general, exceptions should be discouraged.

Procedures for prototyping

Prototyping requires its own six-stage development cycle, as shown in Figure 6.2 overleaf.

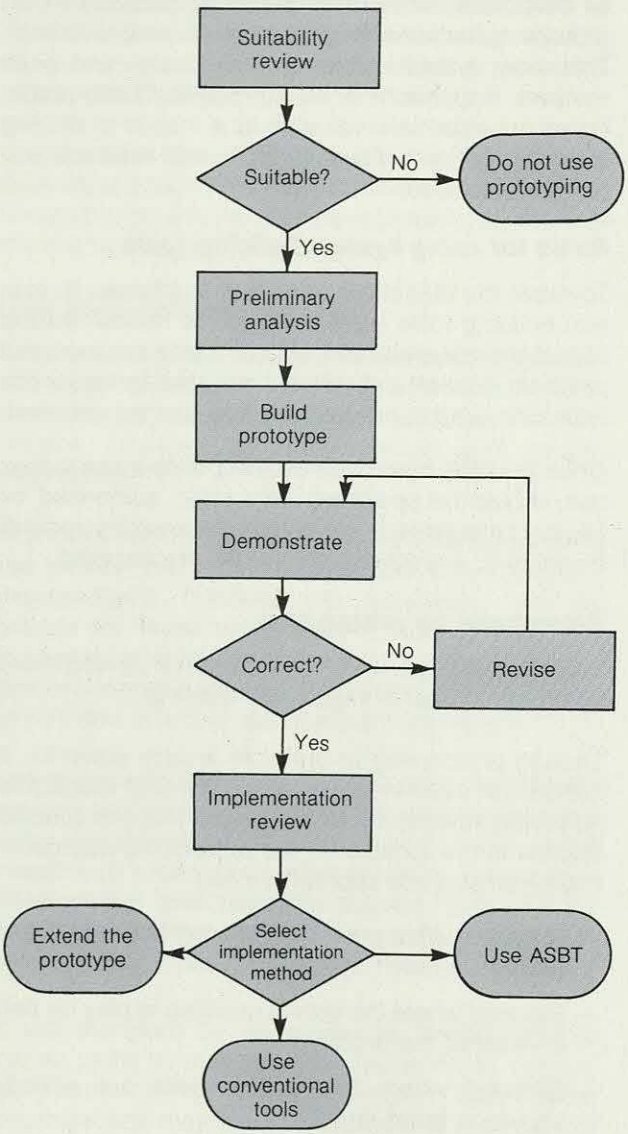
Though prototyping in principle is very powerful, it cannot be used in every case. The first stage (the suitability review) therefore checks that the specific application is suitable for the prototyping approach. Prototyping is not appropriate for:

- —Systems with a great deal of complex logic, especially in a batch system.
- —Systems where the user is unwilling to play his part in refining the prototype.
- —Systems where the requirements are already known in great detail.
- —Systems where rapid completion is more important than correct functioning.

The second stage (preliminary analysis) establishes the aims of the application, the essential functions, the main logical data structures and whether any of the data needed already exists in machine-readable form. This stage is similar to conventional systems analysis, but is much shorter because it produces a reasonable first approximation, not a perfect final answer.

The third stage is to build the first prototype, which should be sufficiently complete to allow meaningful discussion with the user. The prototype should implement the essential functions identified by the preliminary analysis, though batch updating and system security and integrity features can usually be omitted. There is no need to spend too much time on the details of screen and print layouts at this stage.

Figure 6.2 Prototype cycle



At Stage 4, the first prototype is demonstrated to the prospective users. Discussion should concentrate on the overall correctness of the model so that serious errors and omissions can be detected. Discussion of later versions should involve all those with any interest in the system, and should focus on details as well as on the overall structure.

Once a revised specification has emerged, the prototype should be revised (Stage 5), although it is best not to do this in the presence of the users. Additional functions and improvements in screen and print formats can be added, and the prototype is demonstrated again.

The number of iterations of the demonstrate-revise cycle should be controlled by keeping the user informed of the costs of continued refinement and by setting some limits to the process of refinement. The limit may be determined by a completion date or by a predetermined number of iterations (some organisations use three). Nevertheless, the option of continuing beyond the limit must remain open if the information systems department and the user judge that further refinement is necessary.

Once the user has accepted that a system based on the prototype will meet his requirements, an implementation review (Stage 6) should be carried out. This review decides whether the system should be built by extending the prototype, by starting again with advanced system building tools, or by using conventional development tools. The review will also produce accurate cost estimates for development and operation.

New technical standards

Most of the changes in the systems environment required to make the best use of system building tools will be contrary to, or at least out of sympathy with, existing technical standards. New standards and procedures will therefore be needed for estimating, documentation, program-structure and program-coding conventions. These technical standards should be reviewed periodically in the light of experience and to take account of changes in the tool.

Measures of the development productivity obtained by using system building tools should be used for estimating purposes. Feedback for the other items may be obtained from inspections.

Regular monitoring of productivity

Development productivity is the key factor that links development technology with business benefits. If the initial gains in productivity are not maintained then the benefits are lost; if the gains are exceeded, new opportunities may be revealed.

Productivity monitoring requires that the time expended on each project and the size of each system delivered be recorded. It is also useful to record the use of computer resources.

MANAGE THE CHANGEOVER PERIOD

Several transitional arrangements will be required to accomplish the changeover from the old methods. They concern users, development managers and development staff.

Users will need to be educated about prototyping. The need for them to play an active role should be stressed. It must also be made clear to them that prototyping does not produce a usable system but

only a model. On occasion it may be possible to derive a usable system directly from the prototype, but that is a matter for the information systems department to resolve.

Project development managers should not be expected to carry the costs of training and of gaining experience of the new tools, especially if they are to be judged on their ability to meet timescales and budgets. These costs should be carried as a departmental overhead. It is, after all, the information systems department as a whole that benefits from the increased skills and productivity.

Some development staff will probably be unable or unwilling to become analyst-programmers. Many programmers identify themselves with a particular range of computers and see their personal development in terms of an increasingly close involvement with those computers, often as software support specialists. Such staff will perceive the new focus on business requirements and the preference for integrated development as a distraction from their preferred career paths, and may be concerned at the prospect of learning new skills or the possibility of losing their jobs.

It may be possible to transfer some of these staff to maintenance, technical support or other specialist roles. Others may prefer to leave, and this should be accepted and positively managed.

CONCLUSION

By following the six-stage process described in this report, an organisation should find itself, having installed the most appropriate system building tools in the most appropriate systems environments, in a position to begin to realise the substantial benefits that such tools can offer.

We believe that the only way in which most systems departments will be able to meet the future demands that will be placed on them is to abandon traditional methods of systems development in favour of the use of advanced system building tools. Furthermore, we would recommend that, in the process of making this change, serious thought should be given to using the new tools to redevelop the existing systems base.

APPENDIX 1

MEASURING PRODUCTIVITY WITH FUNCTION POINTS

Information systems departments need to measure development productivity in order to manage projects, monitor progress and measure the impact of changes in the development process. It is helpful if the measures used also allow objective comparisons to be made between organisations so that they can benefit from each other's experiences.

Lines-of-code produced per man-day has been very widely used as a measure of programming productivity. This measure breaks down, however, if the language is changed, or if it is used in different ways. Moreover, it does not measure the work of requirements analysis and systems design, which most organisations now regard as more important than programming.

Dr Allen Albrecht of IBM has devised 'function points' as a measure of system size. The number of function points produced per man-day is a good measure of development productivity because:

—Function points measure functions delivered, not effort or program size.

—Function points are language and machine independent.

—Function points are intelligible to users.

—The measure is independent of changes in information systems and user organisation.

—Reference data is available from several sources.

FUNCTION POINTS MEASURE FUNCTION NOT EFFORT

The biggest defect of measuring lines-of-code produced is that some lines are much more valuable than others, either because of differences in the language used or because of programming style. Some programmers write verbose programs that, though they have more lines than terse ones, perform the same functions and may be no easier to maintain.

Function points address this problem precisely because they measure function, not programming complexity. They are thus preferable both to lines-of-code and to computer science measures such as Halstead metrics.

The function point concept has been validated by a study of 22 applications systems developed by IBM's DP Services organisation in the United States. Function points are counted by the process of function point analysis (FPA), devised by Dr Albrecht and described in a paper to a SHARE/GUIDE/IBM joint symposium in 1979.

Detailed practical advice is available from IBM and Burroughs. Briefly, the process comprises four stages:

—Count and classify the functions.

—Calculate the 'crude' function points.

—Calculate the adjustment factor.

—Multiply the crude function points by the adjustment factor.

FPA counts only functions that are available to system users. Functions used only in development, to assist in computer operations or for the location of system errors in normal operations are not counted. Functions are divided into three types:

—Input transaction types, including reference files maintained by other systems.

—Output transaction types.

—Inquiry types.

FPA counts files as they appear to the user. Thus indexes required to support convenient and rapid retrieval, and partial files required to circumvent the addressing limits of the operating system, are not counted. Files are classified either as:

—Master files, if they are maintained by this system, or

—Interface files, if they are either transaction files generated by another system but used in this system or transaction files generated by this system.

Figure A1.1 Points for functions and files

System attribute	Complexity		
	Simple	Normal	Complex
Functions			
Input	3	4	6
Output	4	5	7
Inquiry	3	4	6
Files			
Master	7	10	15
Interface	5	7	10

In analysing database systems it is currently necessary to consider the equivalent set of files. This is a drawback of FPA as it is currently defined.

Functions and files are then classified as simple, normal or complex, and each function and file is allocated a number of points according to Figure A1.1. The total of points for all the functions and files in the system is the crude function point count for the system.

Different implementations of the same function may have different function points, depending on the type of system and the way in which it will be used. (The clearest example of this is illustrated by the difference between online and batch systems.) To allow for this, the system must now be rated on each of the 14 factors shown in Figure A1.2. For each factor a rating of between 0 (no influence) and 5 (strong influence throughout) is assigned. The sum of the ratings is then used to calculate an adjustment factor, by using the following formula:

Adjustment factor = 0.65 + 0.01 × (sum of ratings)

Figure A1.2 Function point analysis adjustment factors

Data communications
Distributed processing
Performance objectives
Tight configuration
Transaction volume
Online data entry
Interactive online transaction
Online update of master files
Complex processing
Design for reusability
Ease of conversion and installation
Ease of operation
Multiple site installation
Ease of change and use

The factor may therefore vary from 0.65 to 1.35, but it is almost always in the range 0.85 to 1.15 for substantial business systems. The crude function point count is then multiplied by the adjustment factor to obtain the final function-point count.

FPA may be applied at any stage in a project and, in particular, at a very early stage. If a complete specification, or working system, is available a system of up to 1,000 function points (equivalent to between 50,000 and 100,000 lines of Cobol or PL/1) may be analysed in just a few hours.

FUNCTION POINTS ARE LANGUAGE AND MACHINE INDEPENDENT

Because FPA deals only with the system as seen by the user, and with logical functions, rather than lines of code or numbers of screen formats, it is independent of the language or computer used to implement the system. This enables function points to be used to compare systems written in different languages, to different standards and on different computers.

It also enables FPA to be used for purposes other than assessing development productivity, including:

—Providing a measure of the overall effectiveness of information systems.

—Monitoring maintenance levels.

—Planning system conversions.

—Evaluating packages.

FPA can provide a figure for the total amount of application function installed in an organisation, whether provided by users, system professionals or in the form of packages. The ratio of this figure to total information systems costs provides a crude measure of effectiveness.

The ratio of function points altered to total function points installed provides a measure of the level of maintenance. Similarly, the effort expended per installed function point may be used as a measure of maintenance efficiency, provided that clear definitions of maintenance, excluding enhancements, are used. In 1983 Dr Albrecht provided data of this type to the Butler Cox Foundation Study Tour in which he showed that maintenance efficiency at IBM had increased between 1970 and 1980.

In planning system conversions, FPA can be used to measure the size of the required conversion work. When evaluating packages, FPA can be used to count the total 'value' of the functions required, provided and likely to be used. These measures can be used

to supplement subjective judgements about how well a package 'fits' the organisation's needs.

FUNCTION POINTS ARE INTELLIGIBLE TO USERS

The concepts of files and functions are readily understood by users, who can also easily accept that some functions and files are more complex than others. Though they should not be expected to perform FPA, users should have little difficulty in understanding sizings or comparisons based on FPA.

The function point concept therefore enables users to understand the relative complexity of their systems without having to involve themselves with difficult technicalities.

THE MEASURE IS INDEPENDENT OF CHANGES IN SYSTEMS AND USER ORGANISATION

Because function points deal with system functionality, not with the way the functions are implemented, they are equally applicable to any mixture of professional and user development, and to development departments organised both with and without separate programming teams. Nevertheless, to provide a basis for fair comparisons, all development effort, whether expended by users, analysts, programmers or other staff must be counted. (Some of the function point figures quoted by computer users or in research papers do not include all of the development effort, and care should be taken if these figures are used for comparison purposes.)

A consistent basis for measurement of effort is also necessary. Man-days, man-hours, man-months and man-years are all in use but have different bases in different organisations. For example:

— One Foundation member assumes 26 hours of development work per week, but 52.14 weeks per year (i.e. 1,356 hours per year).

— Another assumes 35 hours per week but only 42 weeks per year (i.e. 1,470 hours per year).

— One major research study was based on 168 hours per man-month (presumably 22 days of seven-and-a-half hours each).

In this report we have used man-days as our unit of effort because:

— It is the shortest reasonable unit for allocating development staff.

— It is, at least with the better tools, a sufficient period for a useful result to be achieved.

— It is easier to measure than man-hours.

— It allows periods of absence and training time to be readily excluded.

Where necessary, we have assumed that a day comprises seven working hours and that there are 210 days available for project work in a year.

REFERENCE DATA IS AVAILABLE FROM SEVERAL SOURCES

Many organisations collect data for lines-of-code produced, including productivity rates. But this is unsuitable as a basis for comparison and for future planning.

After lines-of-code, function points is the measure for which the most data is available. Papers have been published in academic journals by researchers such as Allen Albrecht of IBM and Rudolph Eberhard of the University of Auckland. IBM and Burroughs now both promote FPA. We have drawn on this work, as well as on our own researches, for the productivity data quoted in Chapters 1 and 3 (pages 1 and 15 respectively) and in Appendix 3.

We do not know of any other measure of development productivity for which comparable amounts of data are available.

APPENDIX 2

A CLASSIFICATION OF SYSTEM BUILDING TOOLS

Many classifications of system building tools have been suggested by consultants and other experts. Unfortunately, these classifications are constantly undermined by the suppliers, who:

—Constantly extend their products to provide new features.

—Frequently lay claim to features that are thought to be significant, even though their products lack them.

Terms such as 'relational' and 'non-procedural' refer to important specific concepts in software science, but the terms have been applied to a wide variety of products. Thus VisiCalc, Datastar, Querymaster, All and Ramis have all been described as non-procedural. Yet, clearly, they have little else in common.

The term 'fourth-generation language' is now widely used, most notably by James Martin, to indicate advanced system building tools. This is a rather unhelpful term because:

—It implies that the language is the most important aspect of these products, whereas other aspects (such as system software; methods for writing, amending and compiling programs; and program testing methods) are equally important.

—It groups together products such as Linc, Ramis, Delta, UFO and Querymaster, which have little in common.

The term 'program generator' is also widely used, often as a synonym for 'system building tool'. In this report we use this term to describe tools that produce programs in a high-level language, such as Cobol. Some program generators (Linc, for example) are very sophisticated and it is rarely necessary for a programmer to change the generated code. Others, (Microsoft's Sourcewriter, for example) can only generate simple programs, and we call these 'simple program generators'.

Advanced system building tools have usually been aimed at either professional development of large systems or user development of small systems. Thus, some tools are too complex for anyone who is not a system professional, whilst others can tackle only simple systems or make very inefficient use of computer resources. Within these categories many tools are restricted to particular classes of problem, such as financial or statistical applications, or to particular systems environments.

From the point of view of a potential user these aspects are at least as important as the technological factors — an excellent solution to the wrong problem is of little use.

OUR CLASSIFICATION

We have divided system building into five main (and several subsidiary) categories:

—Incremental aids.

—Data dictionaries.

—Development workbenches.

—Discrete tools.

—Integrated toolkits.

In addition, we describe discrete tools and integrated toolkits collectively as advanced system building tools (ASBTs).

Incremental aids

We use the term incremental aids to describe tools that provide a single function, generally usable at only one stage of the development process. Examples include flowcharters and test harnesses, editors and program library systems.

Data dictionaries

A data dictionary holds in a single place all the data definitions used in a set of applications. Some data dictionaries provide online support to system building tools and support to analysts.

Development workbenches

The best known development workbench is Philips' Maestro which consists of Softlab software running

on a Philips minicomputer. There are several other hardware-software workbench products, including one based on an IBM-compatible computer (and thus able to compile and run programs).

The most significant competition comes from the editors, compilers, filing systems and utilities provided by computer manufacturers — for example IBM's CMS, ICL's MAC and the Unix Programmers' Workbench.

Despite the name, then, workbenches are primarily software. Recognising this, some computer suppliers have recently improved the quality of their workbenches, announcing such products as Program Master (ICL) and PROFS (Burroughs).

Discrete tools

The development of a substantial system generally involves the definition of files and the construction of four kinds of program: batch updates, batch reports, online updates and online reports (though not all systems include all of these). Many system building tools have been designed to construct only one or two kinds of program. We refer to these products as discrete tools.

Thus, there are four kinds of discrete tools, corresponding to the four kinds of program:

—Batch update tools, though there are rather few of these.

—Report writers, such as Mark IV.

—Teleprocessing development systems, such as UFO and Gener/ol.

—Query processors, such as ASI's Inquire.

Integrated toolkits

Integrated toolkits differ from discrete tools in having their own file definition mechanisms, and often their own database management system, and by addressing three or four of the four kinds of program. However, they vary considerably in the types of systems and users for whom they are mainly intended. For example: TIS, All and Linc are intended for the professional development of transaction processing systems; Focus, Nomad and Ramis are intended for the development of management information systems by users or business analysts.

APPENDIX 3

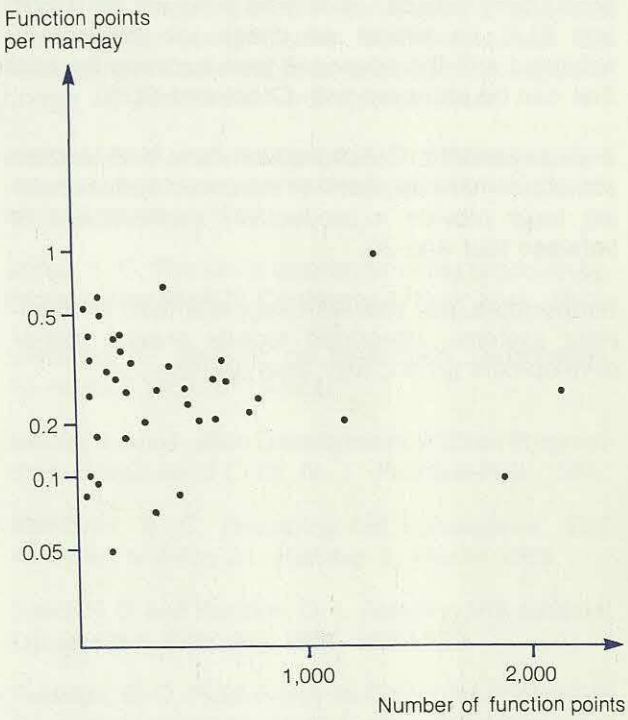
DEVELOPMENT PRODUCTIVITY IN PRACTICE

Our research has established that development and maintenance productivity are the key factors that link system building tool technology with business benefits. We have found that effective tools (and especially advanced system building tools) can make a major contribution to improving development productivity.

In this appendix we present our detailed findings of the productivity obtained with Cobol or PL/1 and with system building tools.

We have measured the size of systems in function points and have included all development activity, whoever performs it. We are aware, however, that the function point approach does have limitations. Skill and experience are needed to apply it, and different estimators may produce values that differ by 20 per cent. But function points are preferable to lines-of-code, at least for this purpose, because they are a measure (albeit imprecise) of the right thing rather than a precise measure of the wrong thing. (We gave our reasons for preferring function points in more detail in Appendix 1.)

Figure A3.1 Cobol and PL/1 development productivity



Note: Each point represents the development productivity achieved (function points per man-day) in developing a system with the specified number of function points.

COBOL AND PL/1 DEVELOPMENT PRODUCTIVITY

Figure A3.1 shows productivity data published by Dr Allen Albrecht of IBM and Dr Rudolph Eberhard of the University of Auckland, or collected during the research for this report. The data relates to system development in Cobol and PL/1 during the period 1977 to 1983. Though the precision of the data is not high, the figure highlights several features:

— Most systems have less than 800 function points (that is, about 50,000 lines of Cobol). In some cases larger systems have been divided to ease the management problems.

— Development productivity varies over a very wide range, from as much as 1.0 function points per man-day to as little as 0.05 function points per man-day. The variations are almost as large within one organisation as between organisations.

— The average productivity for systems of less than 800 function points is about 0.2 function points per man-day.

DEVELOPMENT PRODUCTIVITY WITH ADVANCED SYSTEM BUILDING TOOLS

Figure A3.2 overleaf shows development productivity data for six advanced system building tools. The choice of tools reflects the availability of data rather than the merits of the individual tools.

Each bar on the diagram represents an actual level of development productivity achieved with the particular tool.

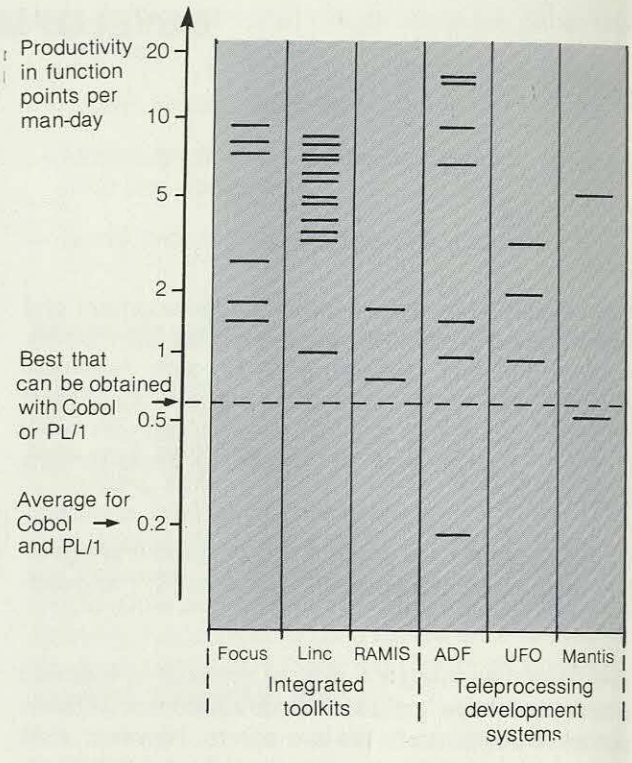
As with Cobol and PL/1 the data shows considerable variations — from 1.5 to 9 function points per man-day for Focus and from 0.2 to 15 function points per man-day for ADF, for example. (We believe that the very high productivity values for ADF may relate to applications that have been carefully selected to be suitable for ADF, whilst the lowest value is for a highly unsuitable system.)

Despite the variability, it is clear that advanced system building tools provide much higher development productivity than can usually be achieved with Cobol and PL/1. In almost all cases the productivity achieved with the advanced tools exceeds the best that can be achieved with Cobol and PL/1.

If we assume that Cobol productivity is 0.35 function points per man-day, then the advanced system building tools provide a productivity improvement of between four and 20.

Furthermore, our research suggests that, for complete systems, integrated toolkits provide higher development productivity than discrete tools.

Figure A3.2 Productivity obtained with advanced system building tools



BIBLIOGRAPHY

Albrecht, A. The role of function point analysis in application development and maintenance. Butler Cox Foundation 1983 Study Tour, IBM Presentation Summary, October 1983.

Albrecht, A. and Gaffrey, J. Software function, source lines of code and development effort prediction: a software science validation. IEEE Transactions on Software Engineering, Volume SE-9, Number 6, 1983.

Anderson, S. M. Santa Fe Railway's OX Project. Santa Fe Railway, 1984.

Baxter, M. Automating the data processing office. Butler Cox Foundation UK Management Conference, Session Summaries, Cambridge, 30 September — 2 October 1984.

Boar, B. H. Application Prototyping: A Requirements Definition Strategy for the 80s. Chichester: John Wiley, 1984.

Boehm, B. W. Software Engineering Economics. Englewood Cliffs, N. J.: Prentice-Hall, 1981.

Dearnley, P. A. and Mayhew, P. J. In favour of system prototypes and their integration into the systems development cycle. Computer Journal, Volume 26 Number 1, 1983, p36-42.

Drummond, S. Measuring applications development performance. Datamation, 15 February 1985, p102-108.

Ehrman, J. R. The new tower of Babel. Datamation, March 1980, p157-160.

Hansen, H. D. Up and Running. New York: Yourdon Press, 1984.

Hyldon, M. Adopting a fourth-generation language to support prototyping. Butler Cox Foundation UK Management Conference, Session Summaries, Cambridge, 30 September — 2 October 1984.

Jenkins, A. M. and Naumann, J. D. Prototyping: the new paradigm for systems development. MIS Quarterly, September 1982.

Jones, T. C. The limits to programming productivity. Proceedings SHARE Conference, New York, 1979.

Lansman, G. Banking on Innovation. Datamation, 15 August 1984, p114-122.

Martin, J. Application Development Without Programmers. Englewood Cliffs, N. J.: Prentice-Hall, 1984.

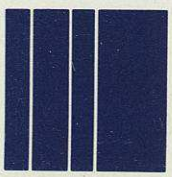
McNurlin, B. C. Replacing old applications. EDP Analyzer, Volume 21, Number 3, March 1983.

Read, N. S. and Harmon, D. L. Assuring MIS success. Datamation, February 1981, p109-120.

Rudolph, E. C. Productivity in Computer Application Development. University of Auckland Department of Management Studies Working Paper 9, March 1983.

Sharples, T. Experiences using Application Program Generators on a mainframe at Trafford MBC. NCC Conference, 25 June 1984.

Xephon Buyers Guide. On-line Application Generators Maidenhead: Xephon Technology Transfer Ltd, 1983.



Butler Cox & Partners Limited
Butler Cox House, 12 Bloomsbury Square,
London WC1A 2LL, England
☎ +44 1 831 0101, Telex 8813717 BUTCOX G

Belgium & The Netherlands
SA Butler Cox NV
Avenue Louise – 479 – Louizalaan,
Bte – 47 – Bus.
Bruxelles 1050 Brussel
☎ (02) 647 15 53, Telex 61963 BUTCOX

France
Butler Cox SARL
Tour Akzo, 164 Rue Ambroise Croizat,
93204 St Denis-Cedex 1, France
☎ (1)820.61.64, Telex 630336 AKZOPLA

United States of America
Omni Group Limited
115 East 57th Street, NY 10022, New York, USA
☎ (212) 486 1760

Switzerland and Germany
Butler Cox & Partners Limited
Butler Cox House, 12 Bloomsbury Square, London WC1A 2LL
☎ (London) 831 0101

Italy
Sisdo BDA Srl
20123 Milano – Via Caradosso 7 – Italy
☎ 498 4651, Telex 311250 PPF MI

The Nordic Region
Statskonsult AB
Stortarget 9, 5-21122 Malmo, Sweden
☎ 46-401 03 040, Telex 127 54 SINTAB