PART SEVEN

Evolution

26 Legacy systems

Objectives

The objectives of this chapter are to introduce legacy systems and to describe how many of these systems have been designed. When you have read this chapter, you will:

- understand what is meant by a 'legacy system', and know why these systems are critical to the operation of many businesses;
- have been introduced to common legacy system structures;
- understand the principles of function-oriented design – the most commonly used design strategy for current legacy systems;
- understand how legacy systems can be assessed to decide if they should be scrapped, maintained, re-engineered or replaced.

Contents

Companies spend a lot of money on software systems and, to get a return on that investment, the software must be usable for a number of years. The lifetime of software systems is very variable but many large systems remain in use for more than 10 years. Some organisations still rely on software systems that are more than 20 years old. Many of these old systems are still business-critical. That is, the business relies on the services provided by the software and any failure of these services would have a serious effect on the day-to-day running of the business. These old systems have been given the name *legacy systems*.

These legacy systems are not, of course, the systems that were originally delivered. External and internal factors, such as the state of the national and international economies, changing markets, changing laws, management changes and structural reorganisation, mean that businesses undergo continual change. These changes generate new or modified software requirements, so all useful software systems inevitably change as the business changes. Therefore, legacy systems incorporate a large number of changes which have been made over many years. Many different people have been involved in making these changes and it is unusual for any one person to have a complete understanding of the system.

Businesses regularly replace their equipment and machinery with modern systems. However, scrapping legacy systems and replacing them with more modern software involves significant business risk. As I discussed in Chapter 4, most managers try to minimise risks and therefore do not want to face the uncertainties of new software systems. Replacing a legacy system is a risky business strategy for a number of reasons:

1. There is rarely a complete specification of the legacy system. The original specification may have been lost. If a specification exists, it is unlikely that it incorporates details of all of the system changes that have been made. Therefore, there is no straightforward way of specifying a new system which is functionally identical to the system that is in use.

2. Business processes and the ways in which legacy systems operate are often inextricably intertwined. These processes have been designed to take advantage of the software services and to avoid its weaknesses. If the system is replaced, these processes will also have to change, with potentially unpredictable costs and consequences.

3. Important business rules may be embedded in the software and may not be documented elsewhere. A business rule is a constraint which applies to some business function and breaking that constraint can have unpredictable consequences for the business. For example, an insurance company may have embedded its rules for assessing the risk of a policy application in its software. If these rules are not maintained, the company may accept high-risk policies which will result in expensive future claims.

4. New software development is itself risky so that there may be unexpected problems with a new system. It may not be delivered on time and for the price expected.

Keeping legacy systems in use avoids the risks of replacement but making changes to existing software usually becomes more expensive as systems get older. Legacy software systems which are more than a few years old are particularly expensive to change for several reasons:

1. Different parts of the system have been implemented by different teams. There is, therefore, no consistent programming style across the whole system.
2. Part or all of the system may be implemented using an obsolete programming language. It may be difficult to find staff who have knowledge of these languages and expensive outsourcing of system maintenance may be required.
3. System documentation is often inadequate and out of date. In some cases, the only documentation is the system source code. Sometimes the source code has been lost and only the executable version of the system is available.
4. Many years of maintenance have usually corrupted the system structure, making it increasingly difficult to understand. New programs may have been added and interfaced with other parts of the system in an *ad hoc* way.
5. The system may have been optimised for space utilisation or execution speed rather than written for understandability. This causes particular difficulties for programmers who have learned modern software engineering techniques and who have not been exposed to the programming tricks that have been used.
6. The data processed by the system may be maintained in different files which have incompatible structures. There may be data duplication and the data itself may be out of date, inaccurate and incomplete.

Businesses which have a large number of legacy systems are therefore faced with a fundamental dilemma. If they continue using the legacy systems and making changes as required, their costs will inevitably increase. If they decide to replace their legacy systems with new systems, this will be costly and the new systems may not provide as effective business support as the legacy systems. Consequently, many businesses are looking at software engineering techniques which extend the lifetime of legacy systems and which reduce the costs of keeping these systems in use. I discuss some of these techniques in Chapter 27, which covers software evolution in general and in Chapter 28, which covers software re-engineering.

26.1 Legacy system structures

Legacy systems are not simply old software systems although the software components of these systems are the main focus of this chapter. Legacy systems are socio-technical computer-based systems (discussed in Chapter 2) so they include

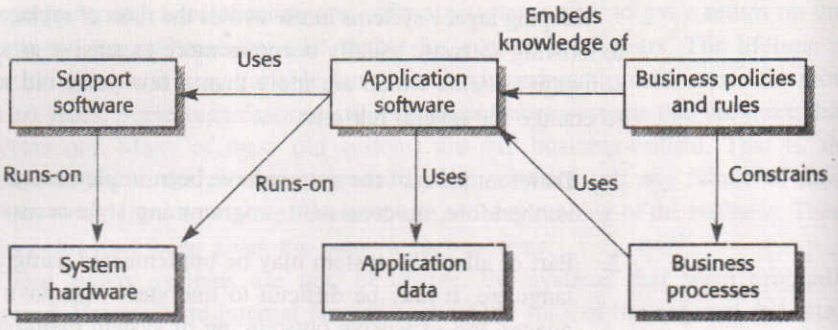

Figure 26.1 Legacy system components

software, hardware, data and business processes. Changes to one part of the system inevitably involve further changes to other components. Decisions about these systems are not always governed by objective engineering criteria but are affected by broader organisational strategies and politics.

Figure 26.1 illustrates the different logical parts of a legacy system and their relationships:

1. *System hardware* In many cases, legacy systems have been written for mainframe hardware which is no longer available, which is expensive to maintain and which may not be compatible with current organisational IT purchasing policies.
2. *Support software* The legacy system may rely on a range of different support software from the operating system and utilities provided by the hardware manufacturer through to the compilers used for system development. Again, these may be obsolete and no longer supported by their original providers.
3. *Application software* As I discuss later, the application system which provides the business services is usually composed of a number of separate programs which have been developed at different times. Sometime the term *legacy system* means this application software system rather than the entire system.
4. *Application data* These are the data which are processed by the application system. In many legacy systems, an immense volume of data has accumulated over the lifetime of the system. This data may be inconsistent and may be duplicated in different files.
5. *Business processes* These are processes which are used in the business to achieve some business objective. An example of a business process in an insurance company would be issuing an insurance policy; in a manufacturing company, a business process would be accepting an order for products and setting up the associated manufacturing process.
6. *Business policies and rules* These are definitions of how the business should be carried out and constraints on the business. Use of the legacy application system may be embedded in these policies and rules.

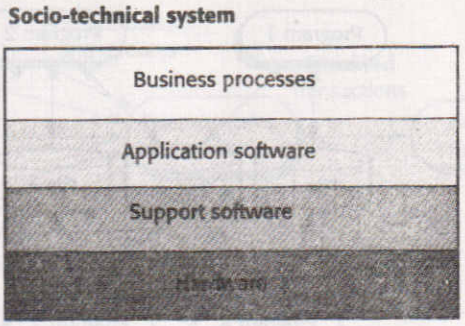

Figure 26.2
Layered model of a legacy system

An alternative way of looking at these different components of a legacy system is as a series of layers as shown in Figure 26.2. Each layer depends on the layer immediately below it and interfaces with that layer. If interfaces are maintained, then it should be possible to make changes within a layer without affecting either of the adjacent layers.

In practice, however, this simple encapsulation rarely works and changes to one layer of the system may require consequent changes to layers that are both above and below the changed level. The reasons for this are:

1. Changing one layer in the system may introduce new facilities and higher layers in the system may then be changed to take advantage of these facilities. For example, a new database introduced at the support software layer may include facilities to access the data through a web browser and business processes may be modified to take advantage of this facility.
2. Changing the software in the system may slow it down so that new hardware is needed to improve the system performance. The increase in performance from the new hardware may then mean that further software changes which were previously impractical become possible.
3. It is often impossible to maintain hardware interfaces, especially if a radical change to a new type of hardware is proposed. For example, if a company moves from mainframe hardware to client–server systems (discussed in Chapter 11) these usually have different operating systems. Major changes to the application software may therefore be required.

The application software in a legacy system is not a single application program but usually includes a number of different programs. The system may have started as a single program processing one or two data files but, over time, changes may have been implemented by adding new programs which share the data and which communicate with other programs in the system. Similarly, the initial system data files are added to as new information is required. This is illustrated in Figure 26.3. Different programs share data files so that changes to one program that affect data inevitably result in changes to other programs.

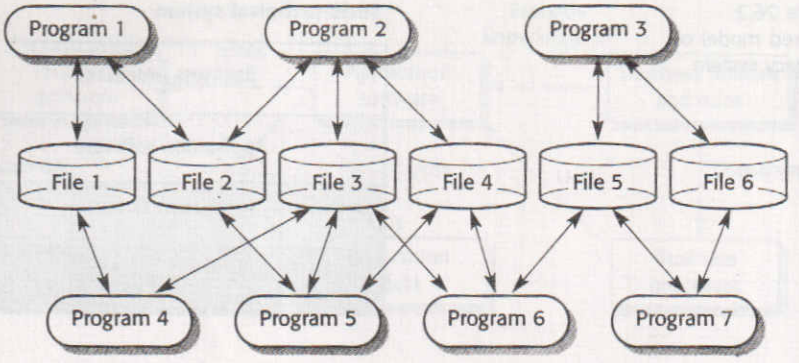

Figure 26.3 Legacy application systems

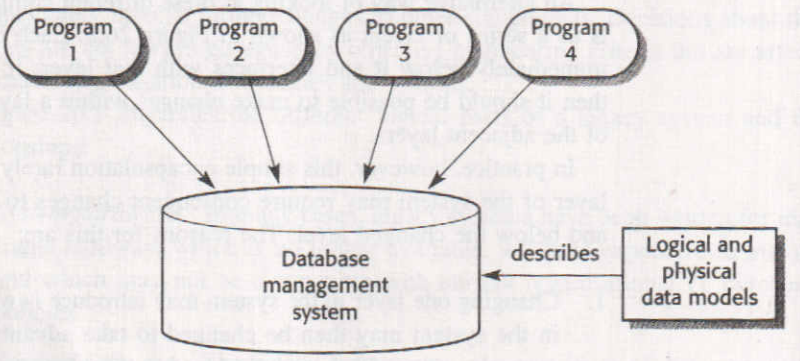

Figure 26.4 Database-centred systems

The different programs in the legacy application system have usually been written by different people and are often written in different programming languages or in different versions of a programming language. For example, the original software may have been developed in COBOL-72 but later programs implemented in a new version of the language, COBOL-80. Compilers and support software for all of these languages may have to be maintained. This all adds to the complexity of the system and increases the costs of making changes to it.

While there are still legacy systems that use separate files to maintain their data, a large number of business systems have centralised their data management around a database system (Figure 26.4). The advantage of adopting this structure is that the data in the system is described using logical and physical data models. Redundancy and data duplication are less likely and it is easier to assess the impact of system changes which affect the system data. Databases also provide transaction processing capabilities where changes to the data can be made in a recoverable way. This allows interactive updates of the data to be made.

The requests for interactive updates of the data come from different terminals and different times. For example, a banking system may have hundreds of terminals used by counter staff in branches and by the public as ATMs. These individual transactions are all made against the central accounts database and they must be collected and organised in such a way that they do not interfere with each other.

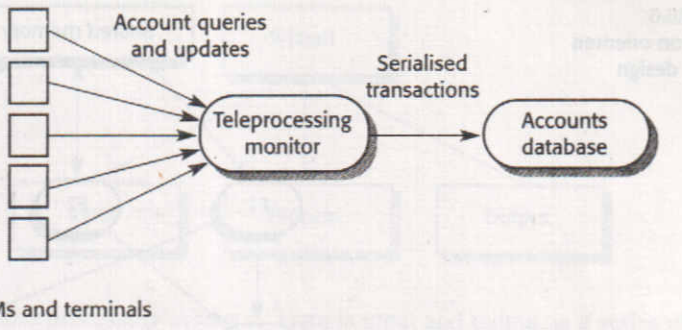

Figure 26.5 Transaction processing using a TP monitor

A teleprocessing monitor, such as IBM's CICS system, is a software system which can handle and buffer inputs from many different sources. In banking systems, a TP monitor accepts transactions from branch terminals and ATMs and may carry out some local processing. It then buffers these transactions and presents them as a serialised list to the account database which updates the customer's account and confirms that the transaction has been processed. This is illustrated in Figure 26.5.

There are two major legacy issues in database-centred systems:

1. The database management system which is used may be obsolete and incompatible with other DBMSs used by a business. Relational database management systems are now the most effective database management systems for business applications. However, many legacy systems rely on older database systems that are based on hierarchical and network models. These systems were designed to allow the performance of the system to be optimised rather than for simple data management. Modern hardware may make this performance optimisation unnecessary but the costs of changing to a relational data model are very high.
2. The teleprocessing monitor which is used may have been designed for use with a particular database system and for mainframe hardware. Therefore, it may not be possible to use the same TP monitor with a new database. This part of the system may also have to be replaced and this increases the costs and risks of system change.

26.2 Legacy system design

Virtually all of today's legacy systems were designed before object-oriented development was widely used in industry. Rather than being organised as a set of interacting objects, the programs in these systems are usually structured as a collection of subroutines or functions. Each subroutine provides part of the functionality of the system and is called, as required, by other subroutines. In some languages,

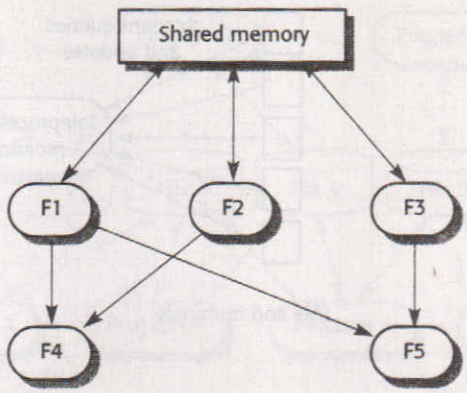

Figure 26.6
A function-oriented view of design

subroutines have their own private data but also access shared data areas. In other languages, such as older versions of COBOL, data is shared and accessible by all subroutines.

A function-oriented design strategy relies on decomposing a program into a set of interacting functions or subroutines with a centralised system state shared by these functions (Figure 26.6). The local state information in functions is only maintained while they are in execution. This design strategy is embedded as 'top-down design' or 'structured design' in a number of structured methods which were invented in the 1970s and early 1980s (Myers, 1975; Wirth, 1976; Constantine and Yourdon, 1979). Hundreds of thousands of application programs have been developed using these methods and associated CASE tools.

Function-oriented design conceals the details of an algorithm in a function but system state information is not hidden. This can cause problems because a function can change the state in a way that other functions do not expect. Changes to a function and its use of the system state may cause unanticipated changes in the behaviour of other functions. This is a particular problem with legacy systems as the programs have usually been changed by many different people. It is unusual for one person to completely understand how the different parts of a program interact.

A functional approach to design is most effective when the amount of system state information is minimised and information sharing is explicit. Systems whose responses depend on a single stimulus or input and which are not affected by input histories are naturally functionally oriented. Many business data-processing systems are concerned with processing discrete records. The processing of one record is not dependent on the results of processing the previous record. It is therefore quite natural to use a function-oriented approach when designing these systems.

Business data processing systems are the largest class of legacy systems. There are two main types of business system:

1. *Batch processing systems* Data is input and output in batches from a file rather than input and output to a user terminal. Examples of batch processing systems are payroll systems, billing systems, etc.

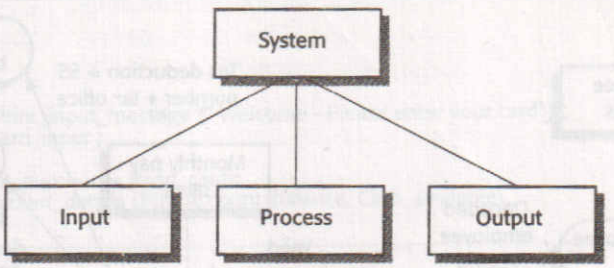

Figure 26.7
An input–process–output model

2. *Transaction processing systems* Data is input and output as a series of transactions against a database with the transaction generated from a user terminal.

Of course, these different types of system may share data. Therefore a bank uses a transaction processing system to manage account transactions but generates bank statements for customers using a batch processing system.

Both batch processing and transaction processing systems generally follow an input–process–output model as illustrated in Figure 26.7. These systems collect inputs from one or more sources, carry out some processing on these inputs and then generate outputs that are related, in some way, to the inputs. For example, a telephone billing system takes inputs which are customer records and telephone meter readings from an exchange switch, computes the costs for each customer and then generates printed bills as output.

The input, processing and output components may themselves be further decomposed into an input–process–output structure. For example:

1. An input component may read some data (input) from a user terminal, check the validity of that data and correct some errors (process), then queue the valid data for processing (output).
2. A processing component may take a transaction from a queue (input), perform some computations on the data and create a new data record recording the results of the computation (process), then queue this new record for printing (output).
3. An output component may read records from a queue (input), format these according to the output form (process) and then send them to a printer (output).

The design of function-oriented systems is often modelled using data-flow diagrams. I introduced data-flow diagrams in Chapter 7. Data-flow diagrams are a functional representation where each round-edged rectangle in the data flow represents a function which implements some data transformation and each arrow represents a data item which is processed by the function. Files or data stores are represented as rectangles in the data-flow notation which I use for data-flow diagrams. Data-flow diagrams show end-to-end processing, i.e. all of the functions which act on data as it moves through the different stages of the system.

To illustrate how the data-flow diagram can describe a function-oriented design, consider Figure 26.8 which shows the design of a salary payment system. This would

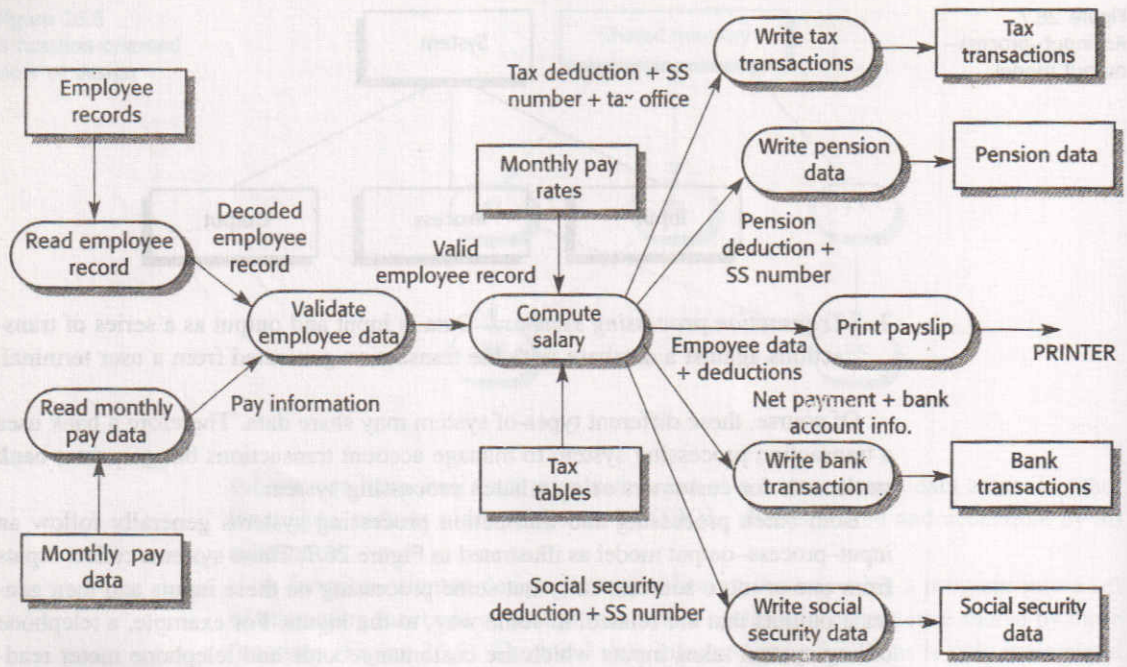

Figure 26.8
Data-flow diagram of a payroll system

normally be implemented as a batch processing system. In this system, information about employees in the organisation is read into the system, monthly salary and deductions are computed and payments are made. You can see how this system follows the basic input–process–output structure which I discussed above:

1. The functions on the left of the diagram, 'Read employee record', 'Read monthly pay data' and 'Validate employee data', input the data for each employee and check that data.
2. The 'Compute salary' function works out the total gross salary for each employee and the various deductions which are made from that salary. The net monthly salary is then computed.
3. The output functions on the right of the diagram write a series of files which hold details of the deductions made and the salary to be paid. These files are processed by other programs once details for all employees have been computed. A payslip for the employee recording the net pay and the deductions made is printed by the system.

An example of a transaction processing system is the software which controls automatic bank teller machines (ATMs). The service provided to a user is independent of previous services provided so can be thought of as a single transaction. Figure 26.9 illustrates a simplified functional design of such a system. I have used a function-oriented design description language rather than Java. Java is an object-

Figure 26.9 Design description of an ATM

```
INPUT

loop
    repeat
        Print_input_message (" Welcome - Please enter your card") ;
    until Card_input ;

    Account_number := Read_card ;
    Get_account_details (PIN, Account_balance, Cash_available) ;

PROCESS

    if Invalid_card (PIN) then
        Retain_card ;
        Print ("Card retained - please contact your bank") ;
    else
    repeat
        Print_operation_select_message ;
        Button := Get_button ;
        case Get_button is
            when Cash_only =>
                Dispense_cash (Cash_available, Amount_dispensed) ;
            when Print_balance =>
                Print_customer_balance (Account_balance) ;
            when Statement =>
                Order_statement (Account_number) ;
            when Check_book =>
                Order_checkbook (Account_number) ;
        end case ;
        Print ("Press CONTINUE for more services or STOP to finish") ;
        Button := Get_button ;
    until Button = STOP ;

OUTPUT

    Eject_card ;
    Print ("Please take your card ) ;
    Update_account_information (Account_number, Amount_dispensed) ;

end loop ;
```

oriented language and I don't think it is natural to use it to describe function-oriented designs.

In this design, the system is implemented as a continuous loop and actions are triggered when a card is input. Functions such as Dispense_cash, Get_account_number, Order_statement, Order_checkbook, etc. can be identified which implement system actions. The system state maintained by the program is minimal. The user services operate independently and do not interact with each other.

Each ATM in the bank's network is generating a transaction, so the banking system must have some way of managing potentially simultaneous transactions and updating its database in a controlled way. This is usually achieved using a tele-processing monitor as shown in Figure 26.5.

Keeping legacy systems in operation is one good reason why function-oriented design will continue to be used for many years. However, the use of this approach is not confined to legacy systems. It may be appropriate to use this technique for new systems development:

1. Where data processing systems are to be implemented which rely on processing transactions and updating a database. The program processing the transactions does not need to maintain information about previous transactions, so objects maintaining private data are unnecessary. These follow the input–process–output model that I have discussed.
2. Where a company has invested heavily in structured methods, associated CASE tools and staff training. The risks and costs of changing to an object-oriented approach to program design may be not justified.

Although many software developers consider function-oriented design to be an outdated approach, object-oriented development may not offer significant advantages in these situations. An interesting challenge that we face is to ensure that function-oriented and object-oriented systems can work together.

26.3 Legacy system assessment

Organisations which depend on many legacy systems and which have a limited budget for maintaining and upgrading these systems have to decide how to get the best return on their investment. This means that they should make a realistic assessment of their legacy systems and then decide on what is the most appropriate strategy for evolving these systems. There are four strategic options:

1. *Scrap the system completely* This option should be chosen when the system is not making an effective contribution to business processes. This occurs when business processes have changed since the system was installed and they are no longer completely dependent on the system. This situation is most common when mainframe terminals have been replaced by PCs and off-the-shelf software on these machines has been adapted to provide the computer support that the business process needs.
2. *Continue maintaining the system* This option should be chosen when the system is still required but where it is fairly stable and users do not request a large number of system changes.
3. *Transform the system in some way to improve its maintainability* This option should be chosen when the system quality has been degraded by regular change and where regular change to the system is still required. This is covered in Chapter 28 which discusses techniques of system re-engineering.
4. *Replace the system with a new system* This option should be chosen when other factors such as new hardware mean that the old system cannot continue in operation or where off-the-shelf systems are available which allow the new system to be developed at a reasonable cost.

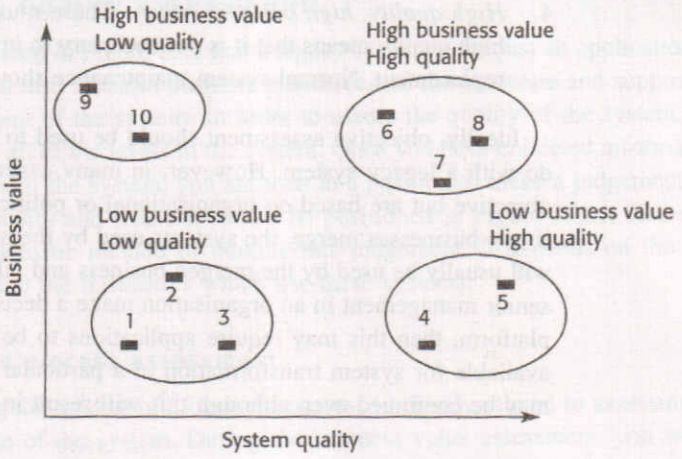

Figure 26.10
System quality and business value

Naturally, these options are not exclusive, so where a system is composed of several different programs, different options may be applied to different parts of that system.

When you are assessing a legacy system, you have to look at it from two different perspectives (Warren, 1998). From a business perspective, you have to make an assessment of the value of that system to the business. From a system perspective, you have to make an assessment of the quality of the application software and the system's support software and hardware. The combination of the business value and the system quality is then used to help inform the decision on what to do with the legacy system.

To illustrate this assessment, let's assume that an organisation has 10 legacy systems. The quality and the business value of each of these systems is assessed and compared with others by plotting it on a chart showing relative business value and system quality. This is illustrated in Figure 26.10.

From Figure 26.10, you can see that there are four clusters of systems:

1. *Low quality, low business value* Keeping these systems in operation will be expensive and the rate of the return to the business will be fairly small. These are candidates for scrapping.
2. *Low quality, high business value* These systems are making an important business contribution so cannot be scrapped. However, their low quality means that operational costs are high so these are candidates for system transformation or replacement if a suitable system is available.
3. *High quality, low business value* These are systems which don't contribute much to the business but which may not be very expensive to maintain. It is not worth the risk of replacing these systems, so normal system maintenance may be continued or they may be scrapped.

4. *High quality, high business value* These must be kept in operation but their high quality means that it is not necessary to invest in transformation or system replacement. Normal system maintenance should be continued.

Ideally, objective assessment should be used to inform decisions about what to do with a legacy system. However, in many cases, these decisions are not really objective but are based on organisational or political considerations. For example, if two businesses merge, the systems used by the most politically powerful partner will usually be used by the merged business and other systems may be scrapped. If senior management in an organisation make a decision to move to a new hardware platform, then this may require applications to be replaced. If there is no budget available for system transformation in a particular year, then system maintenance may be continued even although this will result in higher long-term costs.

26.3.1 Business value assessment

The assessment of the business value of a system is a subjective judgement and there is no reliable objective method that can be used. As with all subjective processes, if you simply rely on one opinion then you are likely to get a very skewed value. Therefore, I recommend that you adopt a viewpoint-oriented approach (see Chapter 6) where you identify a number of business viewpoints and make a value assessment from each of these viewpoints. Viewpoints which should be considered and possible questions are:

1. *End-users of the system* How effective do they find the system in supporting their business processes? How much of the system functionality is used?
2. *Customers* Is the use of the system transparent to customers or are their interactions constrained by the system? Are they kept waiting because of the system? Do system errors have a direct impact on customers?
3. *Line managers* Do managers think that the system is effective in contributing to the success of their unit? Are the costs of keeping the system in use justified? Is the data managed by the system critical for the functioning of the manager's unit?
4. *IT managers* Are there difficulties in finding people to work on the system? Does the system consume resources which could be deployed more effectively on other systems?
5. *Senior managers* Does the system and associated business process make an effective contribution to the business goals?

Once viewpoints have been identified, people from each of these viewpoints should be interviewed and their answers collated. This will give you an overall picture of the value of the system to the business and you can then make an informed assessment of its business value.

26.3.2 System quality assessment

We have seen in Figure 26.2 that a legacy system is not just an application software system but also includes business processes and the hardware and support software environment of the system. In order to assess the quality of the system, you have to look at all of the levels in the system. Once you have collected information about all aspects of the system, you are then in a position to make a judgement about the system quality and where it should be positioned in Figure 26.10. However, there is no systematic method of making this judgement. It depends on the individual systems and the businesses which use these systems.

Business process assessment

Assessing the quality of a business process is closely related to assessing the business value of the system. During the business value assessment, you will have to ask questions about the business process and this will give you some understanding of the effectiveness of that process in supporting business goals.

However, as the process is part of the legacy system, you may wish to discover more detailed information about the process to help you make a judgement whether or not the process is of high quality. This is important because poor quality processes require improvement and this leads to associated changes to the system software.

To assess the process, I recommend a comparable viewpoint-oriented approach to that used for business value assessment. Examples of questions which you might ask are:

1. Is there a defined model of the process and are there procedures in place to check that the model is followed?
2. Do different parts of the organisation have the same processes for the same functions?
3. How have the people involved in the process adapted it to make it more suited to their work?
4. Are there relationships with other business processes necessary? If so, are they clear to the people involved in the process?
5. Is the process effectively supported by the legacy application software? Does it provide the information required? Does the process require the same data to be entered several times in different places?

You should not be surprised if the answers that you get to these questions from different people are totally different. Management in an organisation may think that they have high-quality processes but the actual processes used may be totally different from that assumed by managers. When making process assessments, you should always focus on the actual process which is used and you should not rely only on process documentation to make judgements of the process quality.

Environment assessment

From Figure 26.2, we can see that the environment of an application software system includes the support software (operating systems, compilers, utilities, etc.) used by the system and the hardware platform on which the application system executes. You need to assess this environment as environmental factors are often the drivers of changes to the application software.

Assessing the system's environment is again a judgemental process that is informed by measurements of the system and its maintenance processes. Examples of data which may be useful include the costs of maintaining the system hardware and support software, the number of hardware faults which occur over some time period and the frequency of patches and fixes applied to the system support software.

Factors that you should consider during the environment assessment are shown in Figure 26.11. Notice that these are not all technical characteristics of the environment. You also have to consider the reliability of the suppliers of the hardware and support software. If these suppliers are no longer in business, this means that there may not be maintenance support for their systems.

Figure 26.11 Factors used in environment assessment

Factor	Questions
Supplier stability	Is the supplier is still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, are the systems maintained by someone else?
Failure rate	Does the hardware have a high rate of reported failures? Does the support software crash and force system restarts?
Age	How old is the hardware and software? The older the hardware and support software, the more obsolete it will be. It may still function correctly but there could be significant economic and business benefits to moving to more modern systems.
Performance	Is the performance of the system adequate? Do performance problems have a significant effect on system users?
Support requirements	What local support is required by the hardware and software? If there are high costs associated with this support, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licences? Older hardware may have higher maintenance costs than modern systems. Support software may have high annual licensing costs.
Interoperability	Are there problems interfacing the system to other systems? Can compilers etc. be used with current versions of the operating system? Is hardware emulation required?

Figure 26.12 Factors used in application assessment

Factor	Questions
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures which are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent and up to date?
Data	Is there an explicit data model for the system? To what extent is data duplicated in different files? Is the data used by the system up to date and consistent?
Performance	Is the performance of the application adequate? Do performance problems have a significant effect on system users?
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? Is there an explicit description of the versions of components that are used in the current system?
Test data	Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills	Are there people available who have the skills to maintain the application? Are there only a limited number of people who understand the system?

Application software assessment

Assessing the quality of existing application software is different from the quality assurance activities that take place during software development (see Chapter 24). The legacy system may have been developed using techniques and standards that are no longer in use, the system structure will inevitably have been corrupted through change and the documentation of the system may be out of date.

Some of the factors which you might use to make a judgement of the quality of the application software are shown in Figure 26.12. Finding the answers to the questions associated with these factors helps you assess the system quality. You may also collect quantitative system data that will give you more information on which to base your judgement.

Examples of quantitative data which might be collected are:

1. *The number of system change requests* System changes tend to corrupt the system structure and make further changes more difficult. The higher this value, the lower the quality of the system.

2. *The number of different user interfaces used by the system* This is an important factor in forms-based systems where each form can be considered as a separate user interface. The more different interfaces, the more likely that there will be inconsistencies and redundancies in these interfaces.
3. *The volume of data used by the system* The higher the volume of data (number of files, size of database, etc.), the more complex the system.

Although this data is often useful, it can be very expensive to collect. This may mean that it is impractical to collect it for the assessment of the system. Furthermore, there are no absolute values that may be used. The age and size of the system have to be taken into account when making quality judgements based on measurements.

KEY POINTS

- A legacy system is an old system that still provides essential business services.
- Legacy systems are not just application software systems. They are socio-technical, computer-based systems, so include business processes, application software, support software and system hardware.
- Most legacy systems include a number of different programs and shared data associated with these programs. This data may be held in files or in an obsolete database management system.
- Most legacy systems have been designed from a functional perspective and are composed of sets of interacting functions which communicate through parameters and global shared data areas.
- In the business systems domain, most legacy systems are either batch processing systems or transaction processing systems. In both cases, their general organisation can be represented using an input–process–output model.
- The business value of a legacy system and the quality of the application software and its environment should be assessed to help decide whether to replace, transform or maintain the system.
- The business value of a system is an assessment of the effectiveness of the system in supporting business goals.
- The quality of the system depends on the quality of the business processes, the quality of the application software itself and the quality of the hardware and software which is used to support the system.

FURTHER READING

'Legacy information systems: Issues and directions'. An overview of the problems of legacy systems with a particular focus on the problems of legacy data. (J. Bisbal, D. Lawless, B. Wu and J. Grimson, *IEEE Software*, September/October 1999.)

The Renaissance of Legacy Systems. This book is mostly concerned with a method for the evolution of legacy systems. However, it includes a good general discussion of these systems, case studies which illustrate legacy system structures and a chapter on system assessment. (I. Warren, 1998, Springer.)

'Cash cow in the tar pit: Reengineering a legacy system'. A readable description of practical experiences of a legacy system evolution project. (W. S. Adolph, *IEEE Software*, May 1996.)

EXERCISES

26.1 Explain why legacy systems may be critical to the operation of a business.

26.2 Suggest three reasons why software systems become more difficult to understand when different people are involved in changing these systems.

26.3 What difficulties are likely to arise when different components of a legacy system are implemented in different programming languages?

26.4 Why is it necessary to use a teleprocessing monitor when requests to update data in a system come from a number of different terminals? How do modern client–server systems reduce the load on the teleprocessing monitor?

26.5 Most legacy systems use a function-oriented approach to design. Explain why this approach to design may be more appropriate for these systems than an object-oriented design strategy.

26.6 Expand the 'Compute salary' function in Figure 26.8 and draw a data-flow diagram which shows the computations carried out in that function. You need the following information to do this:

- The employee record identifies the grade of an employee which can be used to look up the table of pay rates.
- Employees below a particular grade may be paid overtime pay at the same rate as their normal hourly pay rate. The extra hours for which they are to be paid are indicated in their employee record.
- The tax deducted depends on the employee's tax code (indicated in the record) and his or her annual salary. Monthly deductions for each code and a standard

salary are indicated in the tax tables. These are scaled up or down depending on the relationship between the actual salary and the standard salary used.

26.7 Under what circumstances might an organisation decide to scrap a system when the system assessment suggests that it is of high quality and high business value?

26.8 Suggest 10 questions that might be put to end-users of a system when carrying out a business process assessment.

26.9 Explain why problems with support software might mean that an organisation has to replace its legacy systems.

26.10 The management of an organisation has asked you to carry out a system assessment and has suggested to you that they would like the results of that assessment to show that the system is obsolete and that it should be replaced by a new system. This will mean that a number of system maintainers are made redundant. Your assessment actually shows that the system is well maintained and is of high quality and high business value. How would you report these results to the management of the organisation?