1999-4

An Investigation into the Causes and Effects of Legacy Status in a System with a View to Assessing both Systems Currently in use and Those Being Considered for Introduction

Patricia O'Byrne
Technological University Dublin, patricia.obyrne@tudublin.ie

Recommended Citation

O'Byrne, P. An investigation into the causes and effects of legacy status in a system with a view to assessing both systems currently in use and those being considered for introduction. Staffordshire University, 1999.

An investigation into the causes and effects of legacy status in a system with a view to assessing both systems currently in use and those being considered for introduction

Patricia O'Byrne

A dissertation submitted in partial fulfilment of the requirements of Staffordshire University for the degree of M.Sc.

This research was carried out in collaboration with The Dublin Institute of Technology

April 1999

Abstract

This dissertation analyses the area of legacy systems and determines the effects that are exhibited in legacy systems, presenting them in a legacy effect determination framework, so that management can ascertain whether the system they have is a legacy system. An analysis of legacy causal criteria is carried out, resulting in a table of legacy causes. A new definition of legacy systems is put forward, by defining legacy status as a status held by a legacy system. "*A system exhibits legacy status if it is deficient in terms of its suitability to the business, its platform suitability or application software quality, with the effect that its asset value diminishes, as does its ease of operation, maintenance, migration or evolution*." Legacy status is split into three dimensions, that of system suitability, platform suitability and software quality. These dimensions are analysed and practices shown that enable good quality within them.

Solution strategies for handling legacy systems are analysed and broken down into components. These components are analysed in regard to their impact on the legacy causes. A mapping takes place between each strategy component and legacy cause.

A legacy causal criteria framework enables management to assess their systems for possible legacy status. This framework can be used on current existing systems or on new proposed systems. This legacy causal criteria framework is cross-referenced to the legacy effect determination framework, allowing management to see the real or potential effects that a weakness in one of the legacy causes may have. These frameworks can be applied both to existing systems to evaluate their legacy status or to potential new systems to evaluate how they will behave in the future.

Acknowledgements

I would like to thank my supervisors Dr. Bing Wu, for his advice, guidance and encouragement along the way and Mr. Denis Kelly for providing me with information and contacts relating to the ESB and for reviewing early drafts of my dissertation. I am also grateful to staff in the D.I.T. and Staffordshire University who listened, read and advised me.

My thanks also goes to Vivian Young in the ESB and all of the IT managers and staff in CIE who discussed decisions made about systems and their outcome and issues about which decisions need to be made.

My deepest thanks go to my husband Paul who acted as a sounding board and kept the household running and my sons Daragh, Stephen and Brian who tolerated my absences and tiredness due to working on this dissertation, without complaint.

I am eternally grateful to my parents, who fostered an interest in learning and a tenacity to complete a task undertaken in all of their children.

Table of contents

Table of tables

Table of figures

Chapter 1 Introduction

1.1 Background

1.1.1 Legacy systems

As the computer industry ages, more and more organizations are relying on systems that are so established within the organization that they are taken for granted. As such, these systems can be treated almost like a family member, whose idiosyncrasies are indulged and whose failings are forgiven. It is often only when a system's failure to keep up with the changing environment becomes critical, that a decision is made that something needs to be done about it. Such a system is classified as a legacy system. Definitions of legacy systems vary greatly, with many definitions focussing on one aspect of what it means to be a legacy system. Ning *et al.* (1994) describe legacy systems as systems that inhibit an organization's business growth and capacity to change. Arnold (1989), Sneed (1995), Adolph (1996) and Gibson (1998) are among those who recognise the difficulty of maintaining many of these systems, because of frequent modification over their life span and diminishing resources of staff with the necessary skills to maintain them. Gold (1998), Fitzgerald (1998) and Alderson & Shah (1998) state that the occurrence of an event or series of events trigger legacy status, as a system that was previously compatible with business requirements can become incompatible due to changes in its external environment. A system's legacy status is highly dependent on how it matches business requirements. Another aspect of an information system that can cause it to become legacy is the platform on which it exists. Sneed (1995), Ning *et al.* (1994), Ward (1995), Bennett (1995) and Bancroft *et al.* (1997) discuss the limitations that can be imposed on a system due to the hardware or system software on which they are based. The suite of programs within and the design of an application can cause problems if they are not properly developed in the initial stages, or if change is not managed in such a way as to preserve the integrity of the design or implementation of the application. Arnold (1989), Sneed (1995), Adolph (1996) and Bancroft (1997) address different aspects of this problem.

Despite the fact that much research has been done in the area of legacy systems, it is only recently that analysis has been focused on a broad description of what constitutes a legacy system (Gold 1998, Alderson & Shah 1998, Ransom *et al.* 1998). There is, however, in the opinion of this author, a need to be able to evaluate a system that is suspected to be a legacy system, so that a more suitable solution can be chosen to solve the inherent problems. To do this, there is a need to identify the characteristic effects of legacy systems and the related causal criteria, in such a way that a problem or set of problems can be identified in a given system.

These same characteristics can be used to assess a solution, so that a more suitable solution can be chosen and the risks associated with this solution weighed.

1.2 Aim and objectives

The aim of this dissertation is to research into the concept of legacy status and related issues regarding transition from that status and develop a set of frameworks that can be used by management to identify legacy status in a current or planned business information system. The results can be used to provide guidelines to management to enable them to choose a suitable solution to any legacy aspects that are present and avoid immediate potential legacy status in the new system.

In order to achieve this aim, the following objectives need to be met:

1. To identify the characteristic effects that are evident in legacy systems so that they can be related to a legacy problem.

2. To develop a Legacy Effect Determination Framework so that a system's legacy effects can be documented.

3. To identify the characteristic causes of legacy systems and define legacy status.

4. To develop a thorough definition of causal criteria, to enable assessment to take place.

5. To develop a legacy Causal Criteria Framework, so that the causal criteria of legacy status can be identified within a system.

6. To develop a Legacy Status Cause / Effect Framework, so that if a weakness exists in one of the causal criteria, the possible effects of this can be seen. Alternatively, if the system is exhibiting legacy effects, this framework identifies what the possible underlying causes are.

7. To analyse components of existing strategies for dealing with legacy systems and the effects of these components on legacy status, in order to guide strategic managers in the task of choosing an approach towards transition from a current legacy system.

1.3 Research method

The methods employed in the course of this research include literature reviews, interviews with management and staff in the co-operating organizations, critical analysis and design.

1. To fulfil the first and second objectives, identification of the characteristic effects that are evident in legacy systems and determination of the characteristic causes of legacy systems, a literature review was carried out. Interviews also took place with personnel from IS management in both of the co-operating companies. The results of these reviews and interviews were critically analysed, by comparing previous definitions of legacy systems with each other and to practical problems that are arising.

2. To fulfil the third and fifth objectives – developing legacy effect determination and causal criteria frameworks - design work, based on research findings and original thought processes was carried out.

3. The fourth objective – a thorough definition of legacy causal criteria - required that the author undertake further literature reviews and recognise from interviews and practical experience that other aspects that have been separately described but not related to legacy status before could come into play.

4. The sixth objective required the author to design a procedure through which managers can assess their existing and replacement / new systems, comparing and

contrasting their legacy status and enabling them to make decisions on a sound basis.

5. The seventh objective is analysing components of existing strategies for dealing with legacy systems. This involved further literature reviews and analysis of the findings, comparing and contrasting experiences in different case studies with each other, to specify how a wide variety of strategies can be covered by a smaller number of strategy components. The components were analysed with reference to their positive or negative effects on legacy causal criteria and mapped against those criteria.

Representatives from two organizations co-operated in this research by providing information regarding legacy system problems that have been solved, are in the process of being handled or remain as unresolved problems within their organization. These organizations are C.I.E., the Irish bus and rail transport company and the E.S.B., the Irish Electricity Supply Board. This author worked as a programmer and as an analyst and system designer in C.I.E. for several years.

1.4 Expected Deliverables and Potential Benefits

The expected results of this dissertation are outlined below.

1. A new definition of legacy status. This offers a solution to the dilemma of defining legacy systems, by giving a range of causal criteria and effects that can be identified in a system.

2. A new Legacy Effect Determination Framework. This enables management to determine whether or not a current system is a legacy system and in what areas legacy status exists.

3. A discussion of causal criteria relating them to legacy effects. This explains the reasons why the exhibited effects may occur or what may happen if a problem with a causal criterion is not addressed.

4. A new legacy Causal Criteria Framework. This allows the user to assess an existing or proposed system, finding which of the causal criteria are enabled or inhibited by the system.

5. A new Legacy Status Cause / Effect Framework. This allows the user to note possible effects that may result in systems where one or more causal criteria are inhibited or, conversely, to discover possible causal criteria for effects that are being exhibited in an existing system.

6. A new mapping of strategy components, describing them as enablers or inhibitors for relevant causal criteria.

The overall benefit of this dissertation is to allow the user to evaluation or asses the legacy status of a current system, noting its strengths and weaknesses. Depending on the areas of weakness, strategic decisions can be made relating to the type of components required in a solution strategy. Many of the offered solutions can undergo a preliminary assessment by combining the mapping of strategy components with the Causal Criteria Framework. If a solution nis seriously being considered, it can be assessed more thoroughly, by applying the Causal Criteria Framework to it. Potential effects of using an inhibiting strategy component are illustrated in the Legacy Status Cause / Effect Framework.

1.5 Organization of This Dissertation

This dissertation has nine chapters.

Chapter 1, the current chapter, introduces the subject area, gives research methods, the organization of the dissertation and the results and expected benefits.

Chapter 2 identifies and lists legacy effects, grouped into four effect groups. A new Legacy Effect Determination Framework is developed. It then analyses existing research and determines legacy causes, grouping them into three legacy causal dimensions. A new table of legacy causal criteria is presented.

Chapter 3 considers the first causal criteria group, System Suitability. It defines System Suitability in the context of this dissertation. Modern practices in the area are discussed and analysed giving reasons why these practices are not always used and the legacy effects that can result due to lack of System Suitability are determined.

Chapter 4 considers the second causal criteria group, Underlying Platform Suitability. It defines what a platform is in the context of this dissertation. Modern practices in the area are discussed and analysed giving reasons why these practices are not always used. The legacy effects that can result due to lack of suitability of the underlying platform are determined.

Chapter 5 considers the third causal criteria group, Software Quality. It defines software, software engineering and Software Quality in the context of this dissertation. Modern practices in the area are discussed and analysed giving reasons why these practices are not always used and the legacy effects that can result due to lack of Software Quality are determined.

Chapter 6 brings together the frameworks that have been developed so far. It reiterates the Legacy Effect Determination Framework and places it in context. A legacy Causal Criteria Framework is developed and presented. Both of these frameworks are combined to give an overall framework, the Legacy Status Cause / Effect Framework, working from the results obtained in Chapters 3, 4 and 5. Procedures for assessing current systems and new systems are designed and presented using these frameworks. The usefulness of these frameworks is argued.

Chapter 7 addresses the objectives involved in correcting a legacy problem. Some of the legacy handling strategies that are being put forward at present are listed. The components of these strategies that are under review in this dissertation are described. The chapter identifies which of the legacy causal criteria they enable, if any and which they inhibit, if any. A mapping of strategy components against causal criteria is presented.

Chapter 8 is a case study in applying the frameworks to a solution strategy. It describes a widely used legacy replacement strategy – an enterprise resource planner called SAP R/3 – in terms that are used throughout Chapters 3, 4 and 5. A preliminary assessment of SAP R/3 based on its strategy components is presented. A more thorough assessment, using the Causal Criteria Framework is then presented. The results of these assessments are discussed and compared with experiences of other authors regarding this solution strategy.

Chapter 9 summarises the dissertation and offers conclusions and suggestions for further research in this area.

Chapter 2 Effects and causes of legacy status

2.1 Introduction

The aim of this chapter is to identify the effects that can occur in systems that are classified as legacy systems and what the causal criteria are. The effects are used to develop a Legacy Effect Determination Framework, while the causes are used to define the causes of legacy status table. These effects and causes are then used to contribute to the definition of legacy status that is central to this dissertation.

Managers often ignore aspects of systems that are part of an organizational infrastructure until they start going wrong. These systems may be cosseted and cajoled into performing their intended function for quite a while before action is taken to remedy the situation. It is therefore of strategic interest to be able to recognise symptoms that indicate that a system may be a legacy system. These symptoms show themselves as legacy effects.

The underlying causes of these effects need to be tackled in such a way as to ensure that those causes are addressed, without detrimentally affecting other criteria relating to the success of the system. It is therefore also of great strategic importance to identify the criteria that have caused legacy status in the past and are likely to contribute to it in the future.

There are certain characteristics that can cause a system to be classified as legacy or old or outdated. Although it is not necessary for a system to be old to be legacy (Brodie & Stonebraker 1995, Young-Gul 1997, Slee & Slovin 1997), many legacy systems have been developed five or more years ago (Levey 1995). In five years technology, techniques, business environments and requirements are liable to change dramatically. Any aspect of a legacy system that depends on circumstances that were relevant at development time but are no longer relevant may be discussed as being old or outdated. A system that is to be replaced or upgraded may also be called the old system to differentiate it from the replacement, or new system.

A legacy system is one that has one or more elements of legacy status.

Section 2.2 identifies the effects that legacy systems show and categorizes them in a legacy effect determination framework. Section 2.3 analyses the underlying causes of legacy systems and develops a framework of legacy status causal criteria. A new definition of legacy status is proposed in section 2.4 and the chapter finishes with a summary and conclusion (section 2.5).

2.2 Effects of legacy status

While it is normal for users and management to notice the effect of legacy status on a system, the underlying cause may not be quite so evident. Within this section, effects will be addressed from three different angles; 1) Section 2.2.1 analyses effects that have been attributed to legacy systems. 2) Section 2.2.1 derives and restructures the list of legacy effects into groups, tabulating them in Table 1. 3) Sections 2.2.3 to 2.2.6 discusses exactly what these effects are. Section 2.2.7 incorporates these effects into a Legacy Effect Determination Framework (

Legacy effect determination framework		
	Effect	**P**resent **A**bsent or **U**ndetermined
Asset value	Mission criticality	
	Reliability	
Ease of operation	User satisfaction	
	Ease of testing and auditing	
Ease of maintenance	Cost of maintenance and resistance to it	
	Availability of maintenance resources	
	Program size and complexity	
	Dependence on individuals	
Ease of migration / evolution	Ease of use of new technology	
	Scalability	

Table 2), to allow management to make a preliminary assessment of the extent of the problem in the system under consideration.

2.2.1 Characteristic effects exhibited by legacy systems

The research method used to determine the effects exhibited by legacy systems is to review literature that has been published over the years. Much of this literature is devoted to introducing a new way to handle legacy systems and as such, does not analyse the effects in detail. As the purpose of this chapter is to enable management to identify one or more characteristics of their system as being a legacy effect, the author's approach is to classify these effects.

Publications on legacy systems over the last ten years show a progression from those which provide a one-sided definition (Arnold 1989), with the purpose of proposing a solution, through to current research which examines how legacy status can be viewed and how it comes about (Gold 1998, Gibson *et al.* 1998). Alderson & Shah (1998) discuss how these definitions can be attributed to the differing viewpoints of their definers.

The definitions discussed here (further definitions can be found in section 2.4) are chosen because each one of them either covers the concept of a legacy system in a very broad sense, or it states a specific attribute that can be an effect of legacy status. Some of them are specific to one particular legacy system on which the researcher was working at the time of definition. However, one or some of the effects of legacy status on that system can be generalised. None of these definitions cover the full range of effects exhibited by legacy systems in a way that can be used to identify legacy status in a system. It the author's intention to derive a fuller, more specific definition of legacy status, in the understanding that a legacy system is one that suffers from legacy status. With this intent, the oldest of the chosen papers is analysed first, with later papers adding to or reinforcing the definition.

Arnold (1989) suggests software restructuring as a possible cure for legacy systems. The effects that he is trying to conquer are a degradation of the systems asset value to the organization, shortened system lifetime, difficulty in auditing and testing, low job satisfaction among programmers, high software complexity, poor understanding of the code, reduced programmer productivity, dependence on individuals for maintenance or enhancement. Also outdated software structures and software engineering practices, high maintenance costs, low standards, difficulty in maintaining systems, inability to use tools to analyse software or to convert software and inability to add new features to the software.

Ning *et al.* (1994) try to understand legacy code. They recognise the effects as being that many legacy systems inhibit their business growth and capacity to change and cannot take full advantage of the new computing environments. Business rules may be embedded in the software, but cannot be reused without reusing the whole system. Also, the systems may run slowly, on outdated platforms, making them more tedious to use.

Sneed (1995) looks at the possibility of reengineering legacy systems, restating that existing legacy systems are difficult to migrate or maintain**.** He introduces the idea that the reliability of the system may be suspect.

Bennett (1995) gives a very general effect of legacy systems as "large software systems that we don't know how to cope with but that are vital to our organization". While this does not add to any of the individual groups, it is probably an overall statement with which many managers can identify.

Brodie *&* Stonebraker (1995) suggest migration of the legacy system. They see the effects as a resistance to modification and evolution, but they also note that the system may "lack the power or the agility to meet current organizational requirements".

Adolph (1996) suggests reengineering and recognises that although a legacy system may be operating competently at present, this does not imply that the software represents a set of stable requirements. Legacy effects include inability to put in new features, constant patching making the system unreliable.

Bancroft *et al.* (1997) promotes the idea of replacing existing systems with an enterprise resource planner, SAP R/3. She notes that legacy systems use old technology, are lacking in flexibility, are highly complex and possibly diverge with corporate strategy.

Liu *et al.* (1998) state that "IT systems become inadequate in reflecting business needs, either operationally or economically, and so become legacy systems …Many legacy systems remain supportive to core business functions and are ‘indispensable’ to the business”.

Gibson *et al.* (1998) concentrate on the business and technology dimensions of legacy systems. They recognise that part of the legacy aspect of the system is the structure, culture, job designs, workflow and managerial approaches that affect how an organization operates and that existing business processes which are facilitated by the system may not be beneficial to the organization at all.

Ransom *et al.* (1998) see a legacy system as a system which was developed sometime in the past and which is critical to the business in which the system operates, but maintaining it incurs unjustifiable expense.

2.2.2 Derived effects of legacy status

Other authors, alongside those listed above have mentioned the effects exhibited by legacy systems or those with legacy status (Markosian *et al.* 1994, Wu *et al.* 1997, Waters & Chikowsky 1994, Levey 1995). Although these effects have been mentioned, due to different objectives, the discussions provided by these authors lack the depth or breadth of scope required in this dissertation.

This author now analyses the effects that have been listed, organizing them into four groups, based on interviews and observations at the co-operating organizations. The defined four groups are *asset value, ease of operation*, *ease of maintenance* and *ease of migration / evolution.*

Asset value

Effects relating to the *asset value* of the system affect the organization’s ability to function in the area serviced by the system. If this diminishes, the organization is no longer competent in this area. The listed effects that are in this group are:

- Degradation of the systems’ asset value to the organization (Arnold 1989).
- Lacking power or agility to meet current organizational requirements (Brodie & Stonebraker 1995).

- Possibly divergence with corporate strategy (Bancroft *et al.* 1997).

- Inadequacy in reflecting business needs (Liu *et al.* 1998).

- Existing business processes may not be beneficial at all (Gibson *et al.* 1998).

These effects all relate to the system becoming less of an asset to the organization in that they no longer serve the *mission critical* need they may have served in the past, due to lack of power or focus. The other listed effect in this group is:

- Suffering from suspect reliability (Sneed 1995, Adolph 1996)

This is also an effect that diminishes the asset value of the system.

Ease of operation

Effects relating to the ease of operation of a system affect the users, auditors and support staff on a day-to-day basis. Even if the system provides all of the functionality that is required by the business process, the effects in this group may still be present. There are two sub-groups, the first affects core users and is classified as *user satisfaction* and incorporates the following effects:

- Systems run slowly, on outdated platforms (Ning *et al.* 1994)
- May not suit organizational structure, culture, job designs (Gibson *et al.* 1998)

The second sub-group relates to non-core users:

- Difficulty in auditing and testing (Arnold 1989)

Ease of maintenance

This group comprises the effects that relate to the constant changes that need to be made to most systems, to keep them in line with current business practice. This group is sub-divided into four. The first sub-group is *cost of maintenance and resistance to it.* This sub-group relates to the everyday cost of maintenance and unwillingness to undertake it. It contains the listed effects:

- Maintaining it incurs unjustifiable expense (Ransom et al. 1998).
- Resistance to modification (Brodie & Stonebraker 1995).
- Difficulty in maintaining systems (Arnold 1989).
- Outdates software structures and software engineering practices (Arnold 1989).
- High maintenance costs (Arnold 1989).

The second sub-group relates to the availability of staff to work with technology inherent in the system and the style of systems development. This sub-group *availability of maintenance resources* comprises the following effects:

- Low job satisfaction among programmers (Arnold 1989).
- Low standards (Arnold 1989).
- Poor understanding of the code (Arnold 1989).

- Reduced programmer productivity (Arnold 1989).

The third sub-group arises when the business process grows or changes and corresponding code is added to the software, without removing redundant code. It is the *program size and complexity* sub-group and contains the following listed effects:

- High software complexity (Arnold 1989).
- Highly complex (Bancroft *et al.* 1997).

The fourth sub-group relates to the effects that mean an individual or group becomes indispensable to the organization, possibly due to the fact that they have personalised the system to an extent where others may not understand it.

- Dependence on individuals for maintenance or enhancement (Arnold 1989).

Ease of migration / evolution.

The fourth group of effects relates to major or strategic changes that are required to enhance the system to meet new business needs or to move it onto new platforms or to scale it up. The effects of this group are sub-divided into two sub-groups. The first is the *ease of use of new technology* sub-group and contains the following listed effects:

- Difficult to migrate (Sneed 1995).
- Resistance to evolution (Brodie & Stonebraker 1995).
- Business rules cannot be reused without reusing the whole system (Ning *et al.* 1994).
- Inability to put in new features (Adolph 1996).
- Inability to use tools to analyse software or to convert software (Arnold 1989).
- Inability to add new features to the software (Arnold 1989).

The second sub-group relates to the *scalability* of the system and contains the following effects:

- Inhibt their business growth and capacity to change (Ning *et al.* 1994)

- Cannot take full advantage of new computing environments (Ning *et al.* 1994).

The groups and sub-groups are tabulated for the convenience of the reader in Table 1 Effects of Legacy Status.

Inability to cope	
Asset value	Mission criticality
	Reliability
Ease of operation	User satisfaction
	Ease of testing and auditing
Ease of maintenance	Cost of maintenance and resistance to it
	Availability of maintenance resources
	Program size and complexity
	Dependence on individuals
Ease of migration / evolution	Ease of use of new technology
	Scalability

Table 1 Effects of Legacy Status

Having analysed the effects and tabulated them, part of the definition of legacy status can be stated:

The legacy status of a system affects the system's asset value, the ease with which it can be operated and maintained and the ease with which it can be migrated or evolved.

The following sections describe each of the four derived effects in detail.

2.2.3 Asset value

If a legacy system is causing concern, then it is considered by those concerned to be of some value to the organization. In some cases, the value would be classed as critical, whereas in others, the asset value has diminished over the years, or the reliability of the system may be suspect.

Mission criticality

A system is critical to the mission of the organization if the organization cannot function to optimal effect without it. Most legacy systems have a high asset value to their organization (Arnold 1989, Bennett 1995, Gibson *et al.* 1998, Ramage 1998(1)). They contain critical rules of how the business operates, which are valuable assets to the company. As these system requirements may be uniquely defined within this business process, the process cannot be discarded. The algorithm operated in the process may contain rules, embedded in the code, that are not documented anywhere else (Ning *et al.* 1994). Another factor which could make the system critical to the organization's mission is that it contains and provides the only access to valuable data but lacks the power and agility to meet current organizational needs (Brodie & Stonebraker 1995).

The asset value that a system has may diminish. A system that used to be critical to the business can remain constant while the business process changes. In this case, users may find themselves feeding the system to get results. These results, while still useful, may become disproportionate in their usefulness to the effort that is required to gain them. It is possible that a large percentage of what the system offers has become redundant (Fitzpatrick 1997). I.T. systems can become inadequate in reflecting business needs, either operationally or economically and so become legacy systems (Liu *et al.* 1998).

Reliability

The reliability of these systems may be suspect (Sneed 1995, Adolph 1996). When discrepancies start to creep into a system, it may be very difficult to isolate exactly what the problem is or how widespread it is (Arnold 1989, Markosian *et al.* 1994).

2.2.4 Ease of operation

Often the first point of discontent with the system is its ease of use. This may be a minor irritation that the screen formatting is not as up-to-date as that of another system, or it may be more serious, in that a new user would have extreme difficulty in getting the system to produce the required results correctly.

User satisfaction

This situation can occur where the only concrete copy of the rules under which the organization operates exist in the software (Ning *et al.* 1994). In this case, the user may not be sure of why certain data is entered or what it means. If data entry or result distribution is done without the user’s comprehension, it is likely that user dissatisfaction will result. User satisfaction represents the current level of satisfaction of a user who is using a current model of an existing system.

Ease of Testing and auditing

A system that is difficult to operate or maintain will also be difficult to test and audit (Arnold 1989, Markosian *et al.* 1994). A system that cannot be tested or audited fully will gradually lose reliability. Confidence in the system will fall.

2.2.5 Ease of maintenance

Cost of maintenance and resistance to modification

Legacy systems can become very difficult to maintain. (Arnold 1989, Sneed 1995, Bennett 1995, Brodie & Stonebraker 1995, Adolph 1996, Gibson *et al.* 1998, Gold 1998, Ransom *et al.* 1998). They are often monolithic systems. Maintenance programmers generally require a long time to become familiar and confident with the code and what it does, partly because the systems are heavily modified over their life spans. This results in the maintenance of these systems taking up larger and larger portions of legacy funds. (Arnold 1989, Brodie & Stonebraker 1995, Bennett 1995, Adolph 1996, Wu *et al.* 1997, Gold 1998, Gibson *et al.* 1998, Ransom *et al.* 1998).

Availability of resources to maintain them

There is a smaller pool of experts familiar with older technology than would be available for more modern systems. Coupled with this, programmer productivity is low due to the complex nature of the task and the inadequacy of the tools. Programmer job satisfaction is similarly suffering due to frustration in working with poorly structured software (Arnold 1989, Waters & Chikofsky 1994, Bancroft *et al.* 1997). Resources to maintain legacy systems are becoming scarce, while the need for development programmers is artificially high.

Program size and complexity

The programs in many legacy systems are very large and complex. Structural enhancement is a very difficult task and maintenance within a possibly poorly structured program can involve some guesswork (Arnold 1989, Sneed 1995, Levey 1995, Bancroft 1997).

Dependence on individuals

There is a heavy dependence on individuals who alone understand poorly structured software (Arnold 1989, Adolph 1996), making it difficult to interchange people who are maintaining the software.

2.2.6 Ease of migration / evolution

Ease of use of new technology

Many systems need to adapt to the use of new technology. This can vary from using an upgrade of the same platform, to the use of a completely new piece of hardware, a new database engine or operating system. The cost of new technology is dropping so fast that each year opens up new horizons for what a system should be able to do. A legacy system may be running on a slow, outdated platform and cannot take full advantage of the new computing environments (Ning *et al.* 1994, Waters & Chikofsky 1994, Bancroft *et al.* 1997).

Scalability

Legacy systems can be monolithic in nature and confined to a single machine or to a limited network. To extend their capacity to allow for an increase in demand may not be possible. They are costly and difficult to scale adequately for growing business demands. There is a lack of flexibility among these systems (Bancroft 1997).

2.2.7 Legacy effect determination framework

The identified effects shown in Table 1 can be used by management to document the effects that are occurring in the system under consideration. This is done by an assessment of the system. Some or all of the effects will be obviously present or absent. Others may need further investigation. The observation of effects is done by staff who are experienced in the use of the system and in the needs the system is trying to serve. To this extent, it may be relatively subjective. However, it is only within the context of the system, that its asset value, ease of operation, maintenance and migration / evolution can be assessed. The Legacy Effect Determination

Framework can be filled out partly or in full. The purpose of it is to start an investigation, rather than to provide definitive answers. To fill out the framework, the manager, in conjunction with experienced user staff and I.T. staff places a “P” in the “Present, Absent or Undetermined” column if the effect is present, an “A” if it is absent and a “U” if it is undetermined.

Legacy effect determination framework		
	Effect	**P**resent **A**bsent or **U**ndetermined
Asset value	Mission criticality	
	Reliability	
Ease of operation	User satisfaction	
	Ease of testing and auditing	
Ease of maintenance	Cost of maintenance and resistance to it	
	Availability of maintenance resources	
	Program size and complexity	
	Dependence on individuals	
Ease of migration / evolution	Ease of use of new technology	
	Scalability	

Table 2 Legacy Effect Determination Framework

2.3 Causes of legacy status

The effects of legacy status have been summarised in Table 1 and a framework for documenting these effects has been designed and presented in

Legacy effect determination framework		
	Effect	**P**resent **A**bsent or **U**ndetermined
Asset value	Mission criticality	
	Reliability	
Ease of operation	User satisfaction	
	Ease of testing and auditing	
Ease of maintenance	Cost of maintenance and resistance to it	
	Availability of maintenance resources	
	Program size and complexity	
	Dependence on individuals	

Ease of migration / evolution	Ease of use of new technology	
	Scalability	

Table 2. To users, customers and managers, effects that are marked as "P" are the signs that the system is suffering from legacy status. Having identified a set of effects that legacy status can cause, attention now turns to the criteria that have caused these effects. It is only by understanding the underlying cause that a suitable solution can be found. The causes are wide-ranging in nature, but in the author's opinion, they can be categorised in such a way as to enable an organization to check their systems for these causal factors. This section approaches legacy causes from two angles: 2.3.1 quotes causes of legacy systems from publications and section 2.3.2 looks at methods of assessment that have been put forward, to glean the criteria for which a system is being checked. Section 2.3.3 distils the resultant causal factors into a table (**Error! Reference source not found.**). Sections 2.3.4 to 2.3.6 introduce these causal factors. Section 2.3.7 summarizes the discussion on deriving legacy causes.

2.3.1 An analysis of causes that have been related to legacy status

Various definitions of legacy systems indicate a certain number of causes. These range from very specific causes (Arnold 1989, Bennett 1995), which emanate from what Alderson & Shah (1998) have termed the developmental viewpoint, to one general cause, the occurrence or anticipated occurrence of an event (Gold 1998). Gold's (1998) analysis is important in a few ways: 1) it gives dignity to legacy systems that has often been denied, by showing that before the occurrence of the event, the system was not legacy. 2) It highlights the fact that legacy status is inevitable in any system and that the fight against legacy status is a series of battles rather than one major war. 3) It encourages readers to examine their systems in a strategic way, by anticipating events and incorporating them into their strategic plan. However, it is the author's opinion that further classification is needed of areas in which events can occur that may cause legacy status. While very specific events that have caused legacy status in the past may not be relevant to a particular system that is being checked; an analysis of recorded events shows areas of concern.

In accordance with the approach to legacy effects, definitions discussed here are chosen because each one of them states a specific cause or set of causes for legacy status. To add to the author's derived definition of legacy status, the chosen papers are analysed chronologically.

Arnold (1989) concentrated on quality of software as being a major problem. However, his definition of software also included the software development environment and those who operated within the environment.

Sneed (1995) judged systems according to their technical quality and their business value. He states that existing legacy systems are difficult to migrate or maintain and that, due to the low level of understanding of the code, the reliability of the system may be suspect.

Bennett (1995) states that a system may be written in assembly or an early version of a third generation language; was probably developed using state-of-the-art software engineering techniques; many perform crucial work for the organization and is generally large, hard to understand and hard to maintain.

Adolph (1996) recognises that although a legacy system may be operating competently at present, this does not imply that the software represents a set of stable requirements or that it is still one of the organization's core competencies. In other words, the system has become unsuitable.

Bancroft et al. (1997) notes that legacy systems use old technology, are lacking in flexibility, are highly complex and possibly diverge with corporate strategy.

Slee & Slovin (1997) see the legacy problem as destiny – "the ongoing challenge of managing evolving I/S assets in the era of hybrid computing".

Gold (1998) considers that legacy systems have a system component and a software component, where legacy software is "critical software that cannot be modified efficiently" and a legacy system is "a socio-technical system containing legacy software". Other system components are people, hardware, data, and business processes, the maintenance process, development cultures and the system's relationship to the environment.

Gibson et al. (1998) recognise that part of the legacy aspect of the system is the structure, culture, job designs, workflow and managerial approaches that affect how an organization operates and that existing business processes which are facilitated by the system may not be beneficial to the organization at all.

These causes can be grouped and restated as follows:

System suitability

- System has become unsuitable (Adolph 1996)
- System diverges with corporate strategy (Bancroft *et al.* 1997).
- Relevance of business processes to the system (Gold 1998).
- The people who use and maintain the system and the job designs within the organization (Gibson *et al.* 1998)
- The system's relationship to the environment (Gold 1998)
- Organizational structure or culture (Gold 1998, Gibson *et al.* 1998)

Platform suitability

- Technical quality (Sneed 1995)
- Old technology (Bancroft *et al.* 1997)
- Development environment (Bennett 1995)

3. Software quality

- Low level of understanding of the code (Arnold 1989)
- Quality of software (Arnold 1989)
- Software engineering techniques (Bennett 1995)
- Ongoing challenge of managing the maintenance process (Slee & Slovin 1997)

This list covers three groups and mentions very diverse areas. If a comprehensive view of legacy causes is expected, then the groups need further specification. Several authors have suggested methods of assessment for legacy systems. To assist in the specification of causal criteria, a review of methods of assessment follows.

2.3.2 Methods of assessment

In previous decades, when legacy systems were exhibiting problems, these problems were addressed in a piecemeal fashion. However, as the problem of legacy system grew, it was generally recognised that a more thorough and holistic approach was needed to get a satisfactory outcome.

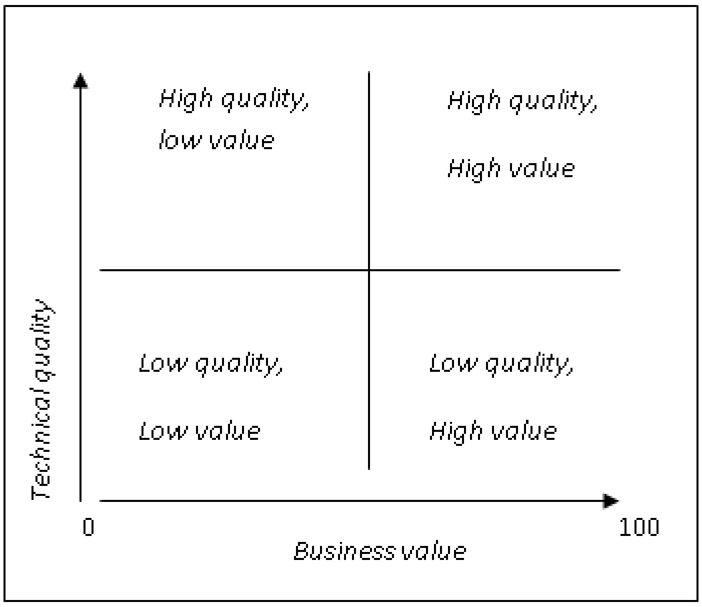


Figure 1 Sneed (1995) evaluates legacy systems

Sneed (1995) suggests that as the size of the system increased, the need to plan its evolution became greater. This planning has to take into account the *business value* of the system as well as its *technical quality*. This grid is useful when assessing an application for immediate use, but it ignores management of change, which is needed to ensure that the current legacy system is not replaced by a system that has legacy status at the time of introduction.

Neumann (1996) puts forward a Legacy System Transformation process. While this assumes that the system's legacy status has already been confirmed, it can add to a definition of legacy causes, in that it addresses the impact of various aspects of the system on its legacy status. The first three steps in this process are:

- Understand the business strategy
- Gather system information
- Conduct an impact analysis of the system on the business strategy, where areas of impact include database, system interfaces, user interface and functionality.

Slee & Slovin (1997) recognise the pace of change as being very rapid, both in terms of business strategy and technological advances. They also discuss the impact that these changes have on the people and infrastructures in the organization. The technical issues that are cited are the hardware, maintainability and size, integration between old and new hybrid systems.

Ransom *et al.* (1998) put forward the Renaissance method for legacy assessment:

- Establish an assessment technique
- Assess business value
- Assess external environment
- Assess application
- Interpret results

This assessment considers the business value, the external environment and the application as three distinct areas for assessment. This is done by breaking down the technical quality aspect of the system into two separate areas: external environment and application. The external environment consists of the hardware, the software and the organizational infrastructure. The application consists of the quality of the application software, both at system and component level.

2.3.3 Derived causal factors

The issues that have been put forward in the last section show a growing number of causal criteria for legacy status. Sneed's (1995) two-dimensional grid has been heavily used in the literature. The two dimensions of business value and technical quality do indeed address the quality of the system and its suitability to the business strategy as defined, but the grid makes the assumption that the business strategy is currently correct and makes no account for future change.

Ransom *et al.* (1998) state that there are in fact three issues that need to be assessed, *Business Value, External Environment* and *Application* and they give details as to what areas are covered in each assessment.

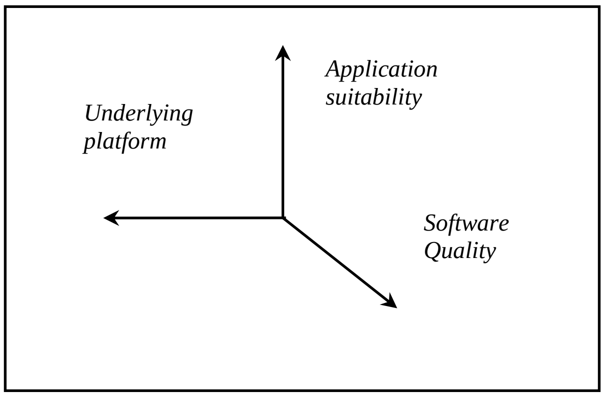
According to Ransom *et al.* (1998) business value is the value of the system to the business process and its benefit to the organization. However, in the opinion of this author, this issues lacks emphasis on the sociotechnical aspect of the system and also fails to emphasise the fact that the business as is may change, thereby requiring the processes to suit the organization's mission rather than its current business practices.

Ransom *et al.*'s (1998) assessment of external environment includes an assessment of organizational infrastructure, hardware and software. In the author's opinion this encompasses both technical and human resources aspects whereas the human resources aspects could be more suitably assessed along with the suitability of the system to the business process and organizational mission. The technical assessment should perhaps be dedicated to the assessment of technology, both hardware and non-application software.

Finally, Ransom *et al.* (1998) propose that the assessment of the application consists of the quality of the application software, both at system and component level. This author contends that the quality of management of process change within the system is also of paramount importance in this area.

The author therefore proposes a new model of causal criteria for legacy status. This is a three-dimensional model (see Figure 2), where each dimension is represented by a vector. As the legacy status of the system increases along that vector, the co-ordinate for that vector moves away from the centre point. As the system improves, its co-ordinate will move back towards the centre. Ideally, this dimensional chart should be accompanied by metrics, which would allow the centre point to indicate a co-ordinate of (0,0,0) showing zero legacy status. However, the investigation required to metricate this would be too detailed in the context of this dissertation and may be the subject of further work. The dimensions themselves, however, will be used throughout the dissertation.

Figure 2 Dimensions of legacy status



The first dimension is called *System Suitability*. The *System* Suitability of the system is a dimension that is addressed by both the IT and business management and is under constant review. This suitability includes the alignment between business and IT strategy and the internal domain, which includes staff, culture and workflow. In this respect, it incorporates Ransom *et al*'s. (1998) 'business value', but goes further, in that it includes the suitability of the technology used by the system to the organizational environment. As this suitability improves, the system can be assessed as travelling along a vector towards the origin, i.e. towards a point where it has no legacy status relating to System Suitability. A perfectly suitable system will have zero legacy status in this dimension.

Causes of legacy status

The second dimension is called *Underlying Platform Suitability*. This is a purely technical dimension. It involves the technical hardware and non-application software that is used in the system. It excludes the suitability of this technology to the organizational environment – that factor is part of the first dimension., *System Suitability*. The *Underlying Platform Suitability* includes those technical aspects of the system, which are managed by the IT department but are not developed by them. Although the platform consists of several components that may or may not be legacy, the vector can be justified by assessing the impact of legacy status of each of these components on the entire platform.

The software quality dimension includes the quality of the current software, both at component and design level and the quality of the software change management process. This dimension comprises the creative aspect of Information systems within an organization. As an organization's software quality mechanisms improve, the system can be assessed as travelling along a vector towards the origin. Alternately, as they degenerate, the legacy vector in this direction grows, indicating a larger legacy status in this dirction.

The author's conclusion is that the three dimensions causing legacy status in a system are:

- The suitability of the system to the business area which it needs to service the suitability and viability of the underlying platform.
- The suitability and viability of the underlying platform
- The quality of the software code, the software design and the software change management in the system.

For the convenience of the reader, the dimensions of causal criteria and their sub-groups are tabulated and presented in Table 3.

System suitability	System suitability to business process
	Business process to organizational mission
	System technology to organizational environment
Underlying platform suitability	Hardware suitability
	Operating System Suitability
	Network suitability
	Development environment suitability
	Data management suitability
Software quality	Quality of change management
	Design quality
	Component quality

Table 3 Causes of legacy status

2.3.4 System suitability

While users are quite happy to complain about a system that is currently in use in their organization on a daily basis, wishing that it was gone and forgotten, the reality is that this system may be a major asset to the organization. The likely asset value of legacy systems is well recognised (Arnold 1989; Ning *et al.* 1994; Brodie & Stonebraker 1995; Bennett 1995; Gibson *et al.* 1998; Ramage 1998(1)). However, the reverse may also be true. The suitability of a system to an organization can change over time. The system that at one stage addressed the organization's needs may have failed to adapt to change in the outside world and may have become less suitable and perhaps even unnecessary.

This suitability has three aspects:

- suitability of the system to the business process (Neumann 1996),

- suitability of the business process to the organization's mission (Henderson & Venkatraman 1993, Sneed 1995),
- suitability of the technology used by the system to the organizational environment (Slee & Slovin 1997) .

A more detailed description of application suitability is addressed in Chapter 3.

2.3.5 Underlying platform suitability

The platform components on which systems run include:

- Hardware
- Operating System
- Networking
- System development environment
- Data management suitability.

Advances in microprocessor technology mean that processor speeds have increased by ten orders of magnitude since the 1950s (Parkinson 1991). In the 1950s, the programmer needed to thoroughly understand the way in which the machine carried out the instructions it was given in order to make it perform a task. As the years went by, various efforts were made to improve programmer productivity, by adding layers of software that shielded the developer from the intricacies of the machine. Programming languages went from machine language which the machine could understand, to assembly language, which needed an assembler, to early third generation languages such as Cobol (Hooper 1959) and Fortran. These languages needed either an interpreter or a compiler and linker. The emergence of Fourth Generation languages added further layers of software and this trend continues today, where systems can be designed using a CASE tool, generated to work on a target platform and reverse engineered into the CASE tool format when changes are required. All of this has led to computer solutions moving from a machine-oriented point of view to a problem-oriented point of view – i.e. one in which new business requirements can be satisfied in a shorter length of time. These developments have diverged widely over the years. The variety and quality of platforms now available for developing and operating application software is vast.

Ning *et al.* (1994) point out that legacy systems that are based on outdated platforms are inhibiting their business' growth and capacity to change.

In the opinion of the author, these advances have advantages and disadvantages. It is obviously advantageous when a programmer can adapt a system to a new business requirement in a short time, especially in a business environment where change is rapid and unrelenting. However, because of the wide variety and complexity of means of developing and operating applications now available, more expertise is required to choose the correct combination of platform components, to install that combination and to support users and developers in their operation of these components. Furthermore, because of the series of complex translations that is now required between developer or user and machine, there are many more potential sources of error, making the tracing of errors much more difficult. In the opinion of the author, this complexity promotes a less scientific and more cavalier approach towards the solving of problems whereby the search for the cause of the problem may be abandoned once the problem can be avoided. In many cases, it is not possible to trace causes, as full information is not available to the maintainer. This approach to problem solving leads to a throwaway mentality, where every system is seen as being good enough until the next version comes along.

The suitability of the underlying platform depends on the careful choice and expert installation, operation and maintenance of suitable platform components. A more detailed description of underlying platform suitability is addressed in Chapter 4.

2.3.6 Software quality

Even systems that are based on legacy platforms may be viable, depending on the need for change of that system area, the need for integration of this system with others and the state of the software in the system. In some cases, despite the age of the software and the number of changes that has been made, conscientious adherence to standards of system development and maintenance mean that the system is quite workable. However, in many cases, constant changes cause the code to become convoluted, without organization or structure, adhering to no standards (Arnold 1989, Sneed 1995, Bancroft *et al.* 1997). The resulting code is often called spaghetti code and is virtually impossible to unravel.

Apart from the quality of code in the modules making up the system, understanding and maintaining the system as a whole can only be done efficiently when interactions between these modules are understood. This requires that rigorous design techniques be used, understood and maintained (Parkinson 1991).

As technology advances, techniques for programming and paradigms for designing applications go in and out of fashion (Royce 1970, Jackson 1975, Myers 1978, Nierstrasz 1992, Downs *et al.* 1993, Fowler & Scott 1997). A change in technique or paradigm that does not carry forward existing software can render existing software legacy with immediate effect. To avoid this situation, process change requires careful implementation.

Software quality can be measured at different levels:

- Quality of implementation of process change (Quality of change management),
- quality of inter-component design as built and maintained (Design quality),
- Quality of the component as built and maintained (Component quality).

Taking each of these levels individually, software quality can be assessed. A more detailed description of software quality is addressed in Chapter 5.

2.3.7 Summary of legacy causes

Legacy systems can be categorised in terms of effects and causes. The effects, while they are likely to be visible to users and management, do not always indicate a solution. The causes are multifarious and need to be associated with criteria that can be assessed both in the system that is causing concern and also in any proposed solution. This dissertation proposes three dimensions of criteria as a) system suitability, b) software quality and c) platform suitability (see Figure 1). These dimensions are further broken down into criteria (see **Error! Reference source not found.**).

2.4 New Definition of Legacy Status

As seen in Sections 2.2 and 2.3, definitions of the meaning of legacy in the context of legacy computer applications vary widely and are usually based on the aspect of legacy status in which the author has an interest. It is the intention of this dissertation to examine criteria that can either cause legacy status in a system or make it more susceptible to events that can cause legacy status. For this reason, the definition must be specific in terms of causal criteria and effects. It is not intended to promote any particular solution to a legacy problem, but to allow the reader to use these criteria in assessing the problem that may be present and the solution that may be being considered. Therefore, the definition must show a variety of *possible* causal criteria, without excluding systems that do not suffer a deficiency in all of the criteria.

The definition of legacy status, which is proposed by this dissertation and used in the remaining chapters, is as follows:

"*Legacy status is a deficiency in a system in terms of its suitability to the business, its platform suitability or application software quality, with the effect that the system's asset value diminishes, as does its ease of operation, maintenance, migration or evolution*."

The critical level of legacy status that is acceptable in any of the three dimensions is dependent on the owner organization. It is incumbent upon management to determine the level of legacy status and its acceptability to the organization. At a strategic level, management determine their approach to the system. This may be an existing system that is intrinsic in the organization or a new system, for which a new strategic approach is required. The strategic approach adopted for a system can enable or inhibit the causal criteria. If there is a weakness in one or more of the causal criteria (see Table 3), this may cause one or more of the legacy effects that have been tabulated (see Table 1). This relationship is shown in Figure 3.

Conversely, if a legacy effect is present in the system, the cause of this can be one of several causal criteria, which may be inhibited by the strategy followed by them.

The following chapters explain each of the causal factors in more detail, outlining the effect poor practice in any dimension can cause.

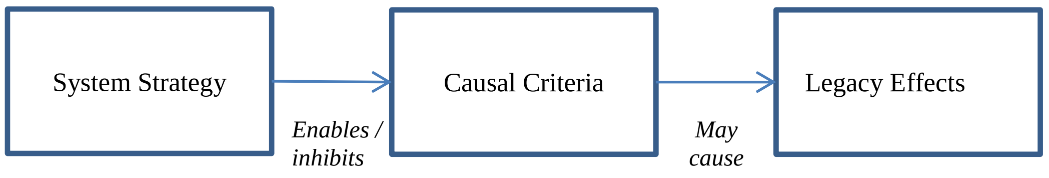


Figure 3 Relationship between system strategy and legacy effects

2.5 Summary

The term "legacy system" covers a wide variety of problems, often with wide-ranging and deeply rooted causes. While none of the definitions are wrong, few address the full scale or range of causes and potential or real effects that are addressed here. This chapter has analysed the effects exhibited in legacy systems and produced A Legacy Effect Determination Framework. Following on from that, further analysis and specification have resulted in a table of the causes of legacy status.

A new definition has been put forward. It speaks in terms of legacy status rather than of a legacy system, introducing the concept of a system suffering from a degree of legacy status. Legacy status has been shown to have three dimensions, with a legacy system having legacy status in one or more of these dimensions. A perfectly suitable system has zero legacy status. These dimensions will be further explored in the next three chapters.

Chapter 3 System Suitability

3.1 Introduction

As stated in the chapter 2, legacy status is defined relative to has three dimensions, the first of which is System Suitability. In order to be able to assess or ensure System Suitability, it is necessary to understand exactly what is meant by it. The purpose of this chapter is to examine current practices in ensuring that the system is suited to the task and organization for which it is intended. It starts with a definition of System Suitability. Section 3.2 outlines ways in which suitability can be assured. Section 3.3 shows the problems that can arise to prevent suitability from being assured and section 3.4 shows the effects that are caused by failing to ensure system suitability at development time and throughout the lifetime of the system. These causes are cross-referenced to the Legacy Effect Determination Framework (

Legacy effect determination framework		
	Effect	**P**resent **A**bsent or **U**ndetermined
Asset value	Mission criticality	
	Reliability	
Ease of operation	User satisfaction	
	Ease of testing and auditing	
Ease of maintenance	Cost of maintenance and resistance to it	
	Availability of maintenance resources	
	Program size and complexity	
	Dependence on individuals	
Ease of migration / evolution	Ease of use of new technology	
	Scalability	

Table 2) in Section 2.2.7.

3.1.1 Definition

System suitability is influenced by many factors, but starts with the alignment of business and information technology strategy. Slee & Slovin (1994) recognise that legacy issues must be understood in a wider business context. Despite the new technologies that are available, legacy systems play a major part in future success. In many cases, the IS organization itself is a legacy issue, tied to old technology. IS is a microcosm of larger economic forces. Most prized attributes today are speed, precision, nimbleness and effectiveness. Faster business change and faster technology change challenge the IS organization to upgrade their user and support staff skills, processes, applications and infrastructures to integrate with business-driven development.

The suitability of the system shows how it fits into the organization. This system can suit the organization by:

- doing what the organization wants it to do (suitability of system to business process),
- doing what the organization needs it to do (suitability of business process to organizational mission)
- being usable by the organization (suitability of the system to the organizational environment).

3.2 Current Practice in the Area

Over the years, different approaches and enhancements have been made to the area of achieving and maintaining system suitability. The approaches addressed here are Information Engineering, Soft Systems Methodologies, Strategic Alignment Model, portfolio assessment and various Human Computer Interaction (HCI) models which take into account socio-technical considerations.

3.2.1 Information Engineering

Information engineering is an approach that tries to ensure that IT strategy is aligned with business strategy before the process of system development begins.

Davids (1992) lists the steps involved in the Information Engineering approach:

First, a business plan is formulated by the strategic managers.

Following that, an Information Strategy plan is formulated with the strategic managers present. This plan starts by:

- Identifying the business issues, which are analysed and formally recorded in an unambiguous manner. A high level understanding of the business can be recorded in a Mission statement. From this, the company's strategy is expressed as aims and objectives. These are the medium to long term results that the organization wishes to achieve and its strategies for achieving them. Objectives lead to critical success factors which describe the essential conditions the business requires to achieve its objectives. The critical success factors should include infrastructure requirements. Goals are the translation of aims into measurable targets that can be achieved by a certain time. Goals are accompanied by their performance metrics.
- Determining the information areas and main activities of the organization.
- Grouping the main functions into business areas based on their usage of the information areas.
- Envisaging a set of conceptual systems within each business area, so that business area may operate effectively.
- Assessing the current systems that are in operation for their applicability to the business and their effectiveness.
- Determining the technology strategy that will support the business.
- Evaluating potential systems that could be used to provide a competitive advantage.

When the Information strategy plan is complete, business area analysis can begin in each of the business areas.

The Information Engineering approach ensures that:

- The systems that are installed suit the business process
- The business process suits the organization's mission
- Critical success factors could include organizational environment factors.

The IT strategy is formulated reasonably late in the proceedings, so there is not a huge emphasis on organizational environment factors.

3.2.2 Soft Systems Methodologies

The effect of the environment on the suitability of a system is ignored in the hard methodologies such as SSADM (Downs *et al.* 1992) or Select Perspective (Frost 1995) that have been derived for developing systems. Checkland (1981) devised a more suitable way of solving the problems that systems have within their organizational environments. Soft systems methodologies approach the appraisal of a problem with a system in a holistic fashion. The system is not isolated from its environment and can therefore only operate in it if the environment is compatible with it. The emphasis here is not on finding a solution to a specified problem, but on understanding the situation in which a perceived problem is thought to lie.

Stages in SSM

Avison & Fitzgerald (1995) document the stages in SSM, one of the Soft Systems Methodologies:

- Stage 1 - situation exists, which is a problem for someone.
- Stage 2 - Situation is expressed in an understandable way.
- Stage 3 - A root definition of the problem is derived.
- Stage 4 - A conceptual model of the problem is built.
- Stage 5 - The real problem is compared with the concept. (Stages 3 to 5 can be iterative).
- Stage 6 - Changes are made to the conceptual model to solve the problem.
- Stage 7 - Analogous changes are made to the real situation.

Use of SSM is ideal where a system is of high technical quality and seems to suit the business process perfectly, but is causing a problem in that it cannot or is not being operated to maximum effect. The problem here may be due to a lack of suitability of the system technology to its environment.

3.2.3 Strategic Alignment Model

System Suitability, in all its aspects, is addressed by the Strategic Alignment Model (Henderson & Venkatraman 1993) in Figure 4. It is done at a very high level, so other approaches will be required at a more detailed level. Henderson & Venkatraman

(1993) state that the inability to realise value from IT investments is, in part, due to the lack of alignment between business and IT strategies of organizations. Economic performance is directly related to the ability of management to create a strategic fit between the position of an organization in the marketplace (the external domain) and the design of an appropriate infrastructure to support its execution (the internal domain). Ideally, companies should undertake constant alignment between their business and IT strategy. Also, both business and IT strategy should be well supported by an organizational and IS infrastructure.

Henderson and Venkatraman (1993) developed a Strategic Alignment Model (see Figure 4). This model shows four domains – two business domains (external and internal) and two IT domains (external and internal). Two alignments must take place continuously in order to maintain the strategic fit between the internal and external domains and to maintain functional integration between IS and business. This strategic alignment can be operated from four different perspectives. The approach used in Information engineering is closest to the second perspective, that of technology transformation.

The four alignment perspectives are strategy execution, technology transformation, competitive potential and service level.

Strategy execution

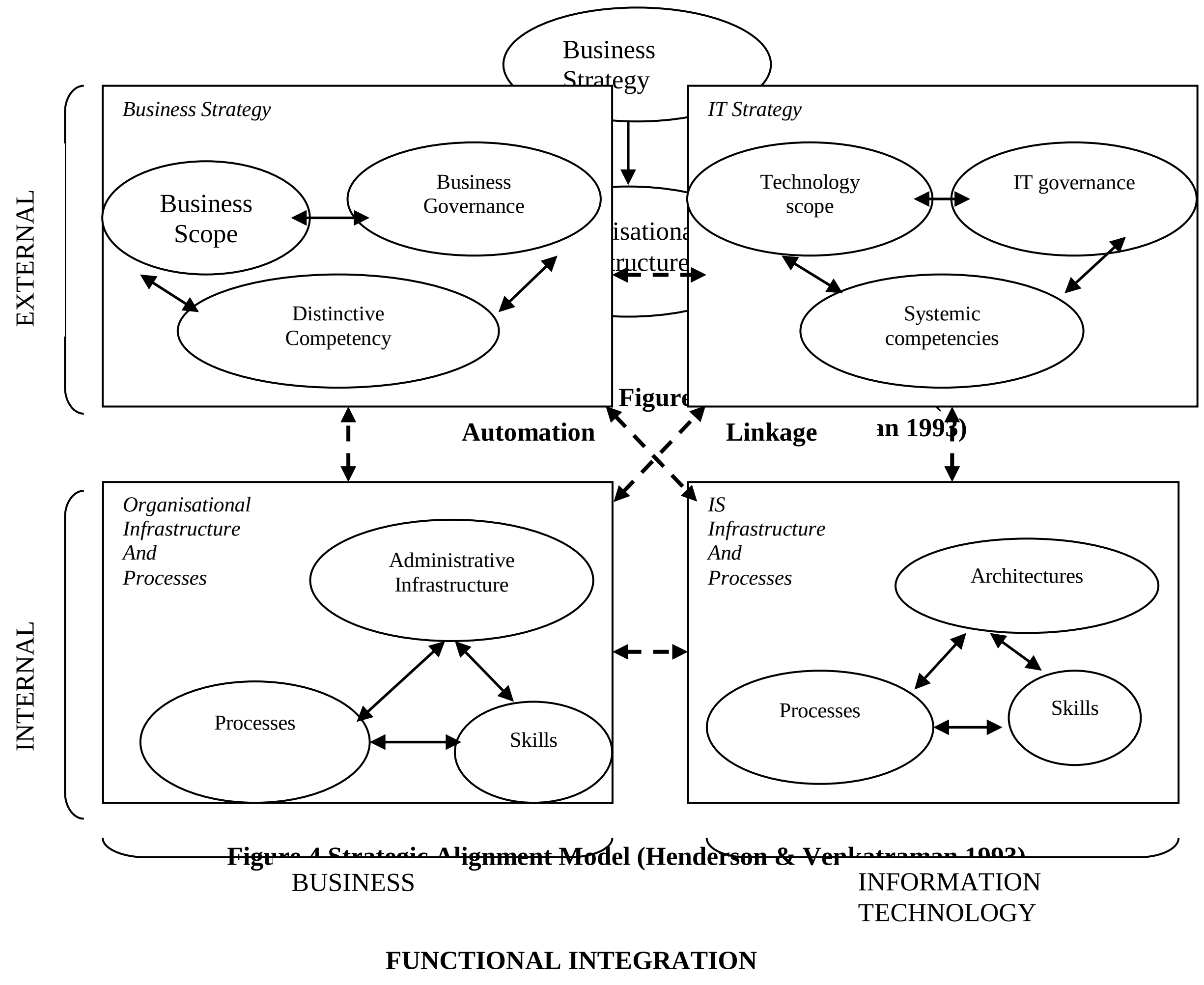


Figure 4 Strategic Alignment Model (Henderson & Venkatraman 1993)

The strategy execution alignment perspective can be seen in Figure 5. The business strategy is formulated and drives both organizational design choices and the design of the IS infrastructure. This is the most common and traditional view of the role of strategic management.

Technology transformation

Technology transformation

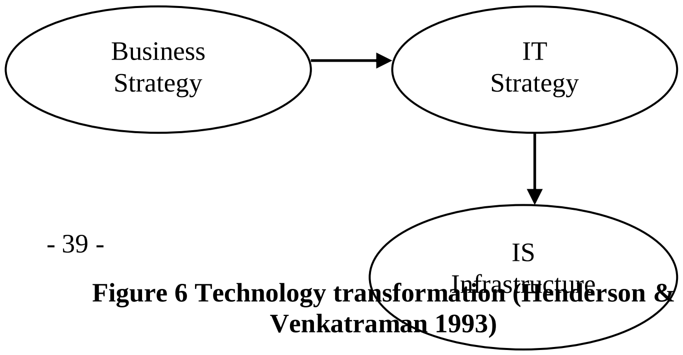


Figure 6 Technology transformation (Henderson & Venkatraman 1993)

alignment perspective can be seen in Figure 6. It involves implementing the chosen business strategy through appropriate IT strategy and basing the organizational infrastructure on this.

Competitive potential

The competitive potential alignment perspective can be seen in Figure 7. This is where IT strategy is formulated on the availability of new IT capabilities that could be exploited to expand or improve business scope. The business strategy can be changed by the IT strategy.

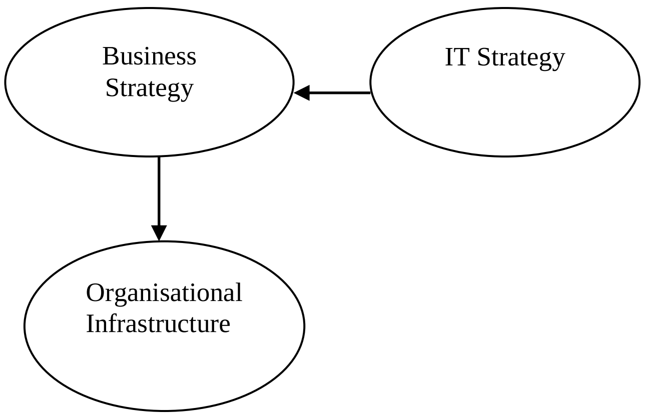


Figure 7 Competitive potential alignment perspective (Henderson & Venkatraman 1993)

Service level

The service level alignment perspective can be seen in Figure 8. This perspective suits an IS service organization. In this perspective, the role of business strategy is indirect and is viewed as providing the direction to stimulate customer demand. This perspective also links the robustness and reliability of the external IT environment with the system. For example, if an external support provider or upgrader is unreliable, this effects the suitability of the organizational environment to the system.

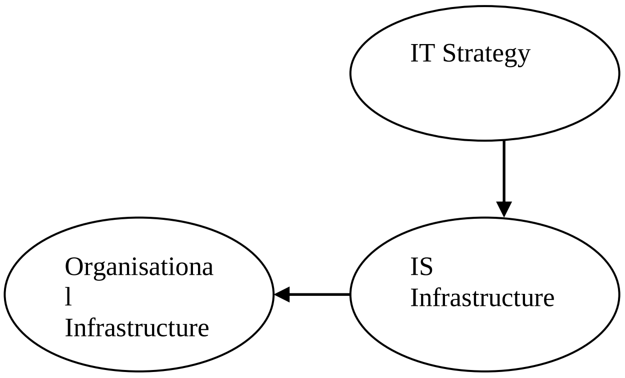


Figure 8 Service level alignment perspective. (Henderson & Venkatraman 1993)

The ideal alignment should ensure that:

- The business process suits the external domain.
- The system suits the business process.
- The internal business domain and IS infrastructure suit the system.

3.2.4 Portfolio Assessment

Slee & Slovin (1994) build on the strategic alignment model. They discuss the questions that must be answered to ensure strategic alignment of IT and business goals. They consider the Strategic execution perspective put forward by Henderson & Venkatraman (1993) (see previous section) as the old ideal. This ideal was that a corporation would express its strategic direction in clear terms, with sufficiently long lead times for IS to prepare. IS could assess its systems and infrastructure to better support business goals (Slee & Slovin 1994). They consider the competitive realignment perspective to be the new ideal , where IT becomes an engine of change, redefining what is strategically possible, so the future of IT requires holistic understanding.

In order to transform the IS organization, Slee & Slovin (1994) suggest six sources of change:

- Better alignment of business and IS goals and strategies
- Better partnering between the business and IS communities
- Better management and technical processes within IS
- Better skills, practices, tools and techniques at developer level
- Continuous enhancement of IS capabilities
- Metrics for organizational performance

Overall, the organization should be assessed as follows:

- How well does the IS organization compare to industry standards such as the Software Engineering Institute maturity model?
- How does the IS organization satisfy its customers?
- How well is IS delivering value and responding to needs?

In order to assess the portfolio of applications, Slee and Slovin have two main evaluation criteria:

- How effectively does a system support business objectives (suitability of business process to organizational mission)?

- How efficiently does it perform those support tasks (suitability of system to business process and to organizational environment)?

Those criteria are assessed using business and technical perspectives. Business objectives are derived from evolving business processes and strategy. IT assets are mapped against these objectives and evaluated using a comprehensive set of metrics. The portfolio is examined in chunks, rather than individual systems, thereby ensuring that the system integration level is also assessed.

To address the third category of system suitability - suitability of the system to the organizational environment – Slee & Slovin (1994) suggest additional questions:

- How do end-user and formal IS supported solutions interact?

- How do current IT assets support fundamental business goals and processes?

- How does the applications strategy fit with the technical infrastructure?

3.2.5 Sociotechnical considerations

The suitability of the system to the organizational environment has a large impact on the success or failure of the system. Traditional systems methodologies do not address these issues, so these interpretive methods can be utilised at a detailed level.

A system may be unsuitable to the organizational environment within which it operates. Assessment of these aspects of a system involves interpretive methods (Preece *et al.* 1994, Walsham 1993). Walsham (1993) states that interpretive methods of research start from the position that our knowledge of reality, including the domain of human action, is a social construction by human actors and that this applies equally to researchers. Therefore, there is no objective reality – only different realities depending on the way the perceiver interprets it. These evaluation methods emphasise the usefulness of findings to the people concerned (Walsham 1993).

Contextual inquiry

Contextual inquiry is a form of elicitation that can be used to assess usability. The contexts under examination are defined by Whiteside *et al.* (1988) as work, time, motivational and social. Users and evaluators identify usability issues of concern collaboratively, while users are working in their natural environments. Holtzblatt and Jones (1993) describe a contextual interview that will show up a) structure and language used in the work, b) individual and group actions and intentions, c) the culture affecting the work and d) explicit and implicit aspects of the work. This assessment can be done by a) getting as close to the work as possible, b) uncovering work practice hidden in words, c) creating interpretations with customers and d) letting customers expand the scope of the discussion. There are no metrics for contextual inquiry.

Co-operative evaluation

Co-operative evaluation allows the evaluator to work with the user who will use the software. More than one user is sampled, and they do the tasks that they normally undertake. The user explains problems as they are encountered and the evaluator takes notes. At the end of the session, the two review the notes that have been taken.

RAMESES project

The RAMESES project aims to derive a strategic model for risk assessment of business process changes in small to medium enterprises with legacy systems, with an emphasis on sociotechnical systems (Edwards *et al.* 1998).

3.2.6 Summary of current practice

The three categories of system suitability that are under review are:

- Suitability of system to business process. This can be ensured by continuously aligning IS and business both in terms of operational and strategic changes. The IS organization keeps track of new technology changes in the external IT domain and the business managers keep track of, and anticipating business changes within the organization. Both IS and business managers should work together to maintain a good strategic fit. Approaches that will assist in this objective are Information engineering, strategic alignment and portfolio assessment.

- Suitability of business process to organizational mission. The business strategy must be continuously aligned with customer needs and competitor advances. The external IT domain will show any new technologies that can be used to drive business process change. Strategic alignment and portfolio assessment ensure that this will work.

- Suitability of the system to the organizational environment. This is partly covered during the aspect of strategic alignment that aligns the IS organization with the IS and business infrastructure. The link between external IT strategy and internal IS Infrastructure relates the external provider environment to the system. However, it is likely that problems that arise in this area may need to be re-examined using a soft systems methodology or by using one of the interpretive methods.

3.3 Common Problems in the Area

There are many different political, financial and cultural reasons why these techniques are not always carried out. Slee & Slovin (1994) list some of them:

Political:

- Organizations have invested in IT for many years and are demanding accountability in regard to the investments already made.

Financial:

- Budget increases have been limited to the rate of inflation for most of the recent past.
- Companies continue to demand cost reductions and better returns on investment from every segment of the business.
- A large proportion of most IS budgets is spent on corrections and minor enhancements to legacy systems.
- The accelerated pace of change – both business and technological - means that systems are superseded more rapidly than before.

Cultural:

- The same procurement and maintenance processes are being used as were used in the past.
- The same change control and architectural approaches are being used, on a micro rather than a macro level.

3.4 Effects of Problems in the Area

The three categories of system suitability that are under review are the suitability of the system to the business process, to the organizational mission and to the organizational environment.

3.4.1 Suitability of system to business process

When system suitability is not constantly matched to business process needs, the system gradually falls out of alignment with the business process (Henderson & Venkatraman 1993, Bancroft *et al.* 1997, Edwards *et al.* 1998, Gibson *et al.* 1998, Liu *et al.* 1998, Ransom *et al.* 1998). These may be a series of minor events or a major event such as the millennium (Ramage 1998(1)). The effects of this are:

- Diminishing asset value.

- Diminishing user satisfaction.
- Increasing cost of maintenance.

3.4.2 Suitability of business process to organizational mission

Before the IT systems are examined, a preliminary analysis of the company's goals is required (Edwards *et al.* 1998). The SEBPC (SEBPC 1998) (Systems Engineering for Business Process Change) programme was established in 1996 to "release the full potential of IT as an enabler of business process change and to overcome the disabling effects that the build-up of legacy systems has on change". Therefore, a system that cannot change has the effects of:

- Diminishing asset value.
- Diminishing user satisfaction.
- Diminishing the ease of use of new technology.
- Increasing the size and complexity of programs, in an effort to get around the inability to use new technology.

3.4.3 Suitability of the system to the organizational environment

A legacy application can become socially or culturally unacceptable to its users because of demands it makes of them. The HCI is a prime example of this, where users are now slower to tolerate an interface that requires a high learning curve. Other factors that cause user intolerance involve putting the user to unnecessary or inconvenient work – an example could be the positioning of the computer interface or the fact that different applications require different sign-ons.

This can be measured using standard HCI evaluation methods (Preece *et al.* 1994), such as benchmarking, observation and monitoring and interpretive studies.

Similarly, a new application can be socially and culturally unacceptable to a user because it demands changes in working practices or operating procedure. In a service environment, the user is not always an employee – it may be a member of the public. A member of the public should not be expected to operate a system that requires more than a superficial effort. The operation of this system is the organization's interface to the public – the response of the machine is the response of the organization. If it is easy to use, does what they want and is helpful and friendly, the user leaves feeling that he / she has been well treated. If, on the other hand, the user is unsure of what is happening and how to conduct a transaction, they leave feeling badly served and do not wish to repeat the experience.

Although users may be competent employees, they may not be suitable for the task of using the application, even if the application is a suitable aid for the task that the users are doing. The environment in which the task is being carried out could be unsuitable for use with the application. In many cases, applications look perfect in head office, but are not practical in hostile environments where the tasks are being done.

Another area of suitability of technology to organizational environment involves the external technology supplier. Their attitude, robustness, competence, availability, reliability and endurance should be assessed as part of the alignment between the external and internal organizational domains.

The effects of this are:

- Diminishing mission criticality in that consistent use of the system diminishes.
- Diminishing reliability, in that the system is being misused or incorrect data is being entered by users who misunderstand the data being entered.
- Diminishing user satisfaction.

3.5 Effects of lack of System Suitability

The System Suitability causal criteria have now been defined. This section of the dissertation analyses these causal criteria to determine the effects that they may cause and produces a table. Table 4 cross-references The Legacy Effect Determination Framework (Table 2 in Section 2.2.7) with possible underlying causes in the System Suitability dimension, those causal criteria being suitability of the system to a) the business process, b) the organizational mission and c) the organizational environment. Each of the causal criteria has a row in the table. Any effects that may be caused by that causal criterion is marked with an “X”.

The *criticality of the system* will diminish if it remains stationary while the process requirements change. Although the system may still be critical to the business, it is not as focused on the needs of the process as it should be. This also applies if the organizational strategy veers away from the area to which the system is critical, as the organization's mission is no longer being served to such a high extent by this system. If the organizational environment changes in such a way that the system becomes difficult to use or support, then the criticality of that system will be compromised, because it cannot be fully utilised. In this context, change of environment generally signifies that the organization has moved on in terms of technology or that technology maintenance is no longer possible due to internal or external environment considerations.

System reliability is a measure of how consistently the system performs in terms of producing the results that it should produce according to its functional and technical specifications, particularly in relation to specified availability as agreed in the Service Level Agreement (SLA) (Fitzpatrick 1997). This should not be affected by the suitability of the system to its process, organizational mission or environment.

User satisfaction is based on the users feeling that the task that they have undertaken is effective towards fulfilling their goal. If the system is not properly focussed towards the business process it is attempting to enable, users will find themselves feeding a system to get results, only some of which are useful. Likewise, if the system does not match the organizational environment, the users may find the need to go to inordinate lengths to keep the system running properly. Once again, the users may find the amount of time and effort going in to the system is disproportionate to the results being produced. The link between user satisfaction and suitability of system to organizational mission is not quite as strong, but is definitely a potential problem. The problem here is that users may feel that they are being by-passed or made redundant, because they are not working on systems that are geared towards the future.

Ease of testing and auditing is affected by System Suitability in much the same way that user satisfaction is. If the system is not properly geared towards a business process, then the specification of the system does not match the business process. Therefore, in testing and auditing the system, it is necessary to know how the system is supposed to work, and also in what way that functionality is pertinent to the business process in hand. If the system is not geared towards the organizational environment, support staff may not have the diagnostic or auditing tools that are necessary to test the system.

Cost of maintenance and resistance to it and *availability of maintenance resources* are effected by the suitability of the system to the organizational environment. As technology has developed over the years, the skills required to develop and maintain applications has varied widely from system to system. If the IS department depends on the same individuals to develop and maintain systems using differing technology, then there may be a shortfall of some of the skills required. Another source of resistance to maintenance may be that staff are reluctant to work on technology that has become outdated, as they are not enhancing their marketability. These problems can be avoided by hiring staff that are appropriate to the technology in the system.

Program size and complexity is a repercussion of either allowing inappropriately skilled staff to maintain a system or failing to implement Software Quality criteria (see Chapter 5).

Dependence on individuals is an indication that the system has legacy status. Although the match of the system to its process and mission may be causes for leaving it to one or two individuals to maintain, they are not effects of this. However, if the system is not suited to the organizational environment, it is likely that there will be some staff who remain from when the organizational environment did suit the system. These staff members will know the system and have the skills necessary to keep it working and as such, will become a life-line for those systems.

Legacy effects / **System suitability**	Asset value		Ease of operation		Ease of maintenance				Ease of migration / evolution	
	Mission Criticality	*Reliability*	*User satisfaction*	*Ease of testing and auditing*	*Cost of maintenance and resistance to it*	*Availability of maintenance resources*	*Program size and complexity*	*Dependence on individuals*	*Ease of use of new technology*	*Scalability*
System suitability to business process	X		X	X						
Business process to organisational mission	X		X							
System technology to organisational environment	X		X	X	X	X	X	X		

Table 4 Effects of failure of system suitability

Ease of use of new technology and scalability may *cause* a system to become unsuitable to its process, organizational mission or environment, but they are not an effect of this occurring.

Scalability may have repercussions for system suitability, but is not caused by it. There may be a tenuous link between scalability and organizational environment, but problems of scalability is much more likely to be related to platform criteria.

The area of System Suitability overlaps somewhat with that of Software Quality, in that the management of process change is closely related to strategic alignment of business and IT strategy. However, the management of process change is much more detailed and requires rigour at every level.

3.6 Conclusion

This chapter has examined definitions of System Suitability and revisited current practices in the area.

Each of the approaches cited has areas that it covers well, but none of them are all-encompassing:

- The Information Engineering approach ensures that installed systems are compatible with business processes currently operating and planned for future operation within the organization. However, it is not strong on the issue of suitability of the system to its environment.
- Soft Systems Methodologies do address the environment of the system, but need to be used in conjunction with any of the development methodologies.
- The strategic alignment model is theoretically very useful, and if competently addressed, from all aspects, will ensure that systems stay focused on the processes which they must carry out in the present and the future and also blend in well with the environment in which they operate.
- Portfolio Assessment offers a broad-ranging view of the issues that can cause a system to become unsuitable, without supplying a consistently reliable methodology for changing this.
- Interpretive methods rely largely on the environment in which the system operates.

There are enough approaches available to enable suitable systems to be developed and maintained. However, at present, a combination of approaches is required to attain a suitable system. There are enough real and potential problems in existence to discourage the maintenance of the necessary alignments and thereby cause a legacy situation to arise. The effects of a diminishment of System Suitability are presented as Effect columns marked “X” in Table 4.

It is not easy to achieve and maintain System Suitability, but it is incumbent on management to strive towards it.

Chapter 4 Underlying Platform Suitability

4.1 Introduction

Platform Suitability is the second dimension used to define legacy status in Chapter 2. The purpose of this chapter is to examine the aspects of Platform Suitability that cause legacy effects. In order to do this, a definition of platform follows in this section. As Platform Suitability is a diverse area, the philosophy behind developing an open system is also discussed in this section. In Section 4.2 current practice in choosing platforms is discussed. Section 4.3 shows the problems that can prevent the use of these practices and section 4.4 shows the effects that divergence from these practices can cause. The chapter gives references for further reading and a conclusion based on the findings within the chapter.

4.1.1 Definition of Underlying Platform

On breaking down the causal dimensions of legacy status (see Chapter 2), the underlying platform suitability is one dimension. This dimension differs from software quality and system suitability in that it is an area that is relatively independent of the logic of the system. The platform enables the system to be developed, used and expanded if necessary. While it can enable or inhibit the scope of change, it is a separate issue from that of system suitability or Software Quality. For the purposes of this dissertation, the platform on which a system runs includes:

- Hardware
- Networking
- Operating system
- System development environment
- Data storage mechanisms

4.1.2 Open systems

A system is understood to have a boundary, chosen according to the human observer's particular purpose and priorities, which separates it from its environment. Given this understanding of boundary and environment, systems may be classified as closed where nothing from outside, except pre-chosen parameters, can cross the boundary, or as open where possibly unknown elements from the environment can have an effect (von Bertalanffy 1968). In simpler terms, a truly open system is one that can be built on one manufacturer's machine and moved to, or communicate with another with little or no change. If there are several manufacturers' machines in use and networked throughout the organization, then systems or components of systems, which are resident on one machine, should be able to inter-work or communicate with those on another. This allows them to transfer information and carry on working when new manufacturers machines and environments are added to the network and new systems are written. Graham (1995) sees open systems as having this double aspect of future-proofing systems against the exigencies of commercial hardware manufacturing and also enabling systems to work together and co-operate in a way that can be understood and utilised by the business.

The users of today expect to be able to use the latest communications devices to get the service they require. Organizations find that their staff mobility is increasing and users now expect to be able to access systems from different sites, using Internet web servers and browsers, by Electronic Data Interchange (EDI), or by use of a third-party agent. These facilities must be integrated in to a situation where the processing that is currently on-going in an organization - on-line and batch systems - are also accommodated. It is therefore a necessity to be able to support systems that are split over different platforms.

In order to accommodate requirements in the most flexible way, an I/S organization needs to take advantage of the latest in information and communication technology.

4.1.3 The ideal

Organizations vary in their use of mainframe machines, private networks, intranets and the Internet. Intranets enable organizations to link their home-based users to sensitive corporate data, allowing them to share information with distributors and partners. These intranets offer a high level of openness, good performance and good interaction. The similarities between LANs, WANs and intranets are being exploited. “Several companies are going from the situation where they had in-house applications on a private network, where for example, resellers of their goods could order products on-line, to a situation where they are using a Web-server to host their applications off an intranet. This allows greater traffic and gives access to a wider number of users.” (McCarthy, 1997).

The opportunities offered by this level of information sharing are the subject of James Martin’s (1996) deliberations. He describes the possibility of vast international corporations with a flat structure based on virtual operations on a web of electronic links both internally and to other corporations. The Internet is almost world-wide at present, giving users access to information from around the globe. The fact that millions of users are accessing the same network gives rise to a global collective consciousness, where chains of technology feed more technology. Aside from the Internet, companies can use their own set of links to make knowledge available, in a controlled fashion, world-wide. This in turn means that if a service is required, the best, not the nearest can be used.

Bernstein (1996) explains the ideal vision: “Each knowledge worker has a desktop appliance that connects to an information utility. The utility is an enterprise-wide network of information services, including applications and databases, on the LANs and WANs. Servers on the LAN typically support files and file-based applications such as E-mail, bulletin boards, document preparation and printing. Local area servers also support a directory service, to help a desktop user find other users and find and connect to services of interest. WAN servers support access to databases and electronic libraries, or transaction processing applications, such as purchasing, billing and inventory control. Some servers are gateways to services offered outside the enterprise, such as travel or information retrieval services, news feeds and electronic document interchange with business partners. In response to such connectivity, some businesses are redefining their business processes to use the utility to bridge formerly isolated component activities. In the long term, the utility should provide the information that people need when, where and how they need it.”

Not all organizations need or want world-wide access. However, all organizations should be aware of the possibilities that are offered by distributed systems, so that they can make an informed choice. The intention in this chapter is to examine the criteria that make a platform component suitable or unsuitable for the system under consideration.

4.2 Current Practice in the Area

Choosing and managing a platform involves a combination of different choices. In order to manage this combination, the ideal, as described in the previous section, must be kept in mind. The requirements of the application or suite of applications that need to be accommodated can be used to shape this ideal. Once this is done, aspects of the platform can be chosen for their technical qualities within the parameters of the ideal described.

4.2.1 Achieving the Ideal

To achieve this ideal, systems should be built in as flexible and open a manner as possible. This can be done when designing a system by:

- Splitting the problem a system addresses into component parts, each component containing data and logic.
- Layering the architecture on which the application sits, so that higher layers are platform independent.
- Controlling the message traffic between components by using a message broker.
- Providing an appropriate development environment for the application.

Splitting the problem

There have been, broadly speaking, three information technology eras. In the first - 1950-1970, the emphasis was on the process. Data was input to the process, the process executed and results were output. In the second era – 1970-1990, thought was given to the fact that the data for an organization could be reused. The data was organised into a structure – hierarchical, network or relational – and stored in this order. Applications were tied into the database where necessary and were granted the access they required. In the latest era, the logic and the data are tied together and are treated as one component or object. The data is treated as attributes of the component. Rather than requesting the data, the component is asked to provide a service, which will give a result. Neither data nor logic is disposable. This is the basis of object technology and of component-based development (Schulte 1996).

The object-oriented paradigm is based on the idea that data and its functionality are highly interdependent. Functions operate on data, to give or store information. The data on its own is useless - functions are needed to interpret the data to give meaningful information. The definition of an object includes the object's data and its behaviour. In object-oriented systems, data and the operations on that data are stored together. An object is accessed through the operations that act on it - its data is hidden or encapsulated inside the workings of the object. Objects communicate by sending messages to each other. Objects, like any other application data and software, can be distributed over different platforms and networks. Distributed objects can be used by clients on remote platforms, with the client needing only to know how to invoke an operation on the server object, without needing to know on which server the object resides (Orfali *et al.* 1996).

A component is an object that is language and platform independent. Components can be grouped together to form applications, or bigger components. A component offers a service to the client. Orfali *et al.* (1996) define a component as having the following properties:

- It is a marketable product.
- It is not a complete application.
- It can be used in unpredictable combinations.
- It has a well-specified interface.
- It can be invoked across address spaces, networks, languages, operating systems and tools.

Service-oriented architecture

"This is a particular style of multi-tier computing that helps enterprises share logic and data among multiple applications. It assumes multiple software layers and usually has thin clients and fat servers. It works on the principle that many aspects of processing logic are inherently tied to the data, rather than being associated with a particular application” (Schulte 1996).

As technology changes from year to year, there is no one ideal platform solution. In order to provide a truly open system, it is necessary to separate the business problem from the platform. This requires a change in architecture of the platform or platforms on which the business system sits. System architecture has changed over the years, from a single tier to three or more tiers (Frost & Allen 1997) (see Figure 9).

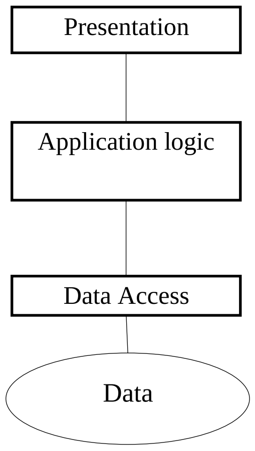


Figure 9 Three-tier architecture

Centralised applications are based on a single tier - all the application logic is together - data access logic, business rules (including communication with other processes) and presentation logic. A huge number of legacy applications are based on this type of architecture.

The next generation consists of systems that are split over two layers or tiers - one on the client machine and one on the server machine. The split differs from application to application, generally with data on one side and presentation logic on the other.

A new three- or multi-tier architecture allows logic to be split into layers. The most distinct layers would be the data access layer and the presentation layer. The data access layer communicates with stored data. This layer contains all functionality and business rules that are specific to the stored data object. At the other end, the presentation layer contains all functionality relating to application interfaces, including some business rules that are specific to a particular interface. In the middle, there are layers or components, which look after any inter-process, inter-application or cross-platform communication. The content of these layers will vary, with some being standard across applications and others being highly application specific.

In order for integrated systems to benefit from component-based software, they require three- or multi-tier architecture. A range of services that provide that extra layer of separation is known as middleware services. These are distributed system services that have standard programming interfaces and protocols. These services reside in a layer above the Operating System and networking software and below industry-specific applications. Middleware services are distributed and include a client part, which supports the service's application programming interface (API) running in the application's address space and a server part, which supports the service's main functions and may run in a different address space.

Middleware services allow an application on one machine, with one type of network software to talk freely to another application on another machine with a different type of network software.

One part of the software in the service needs to be specific to the machine - this part needs to be tailored to the operating system and networking software, but independent of the application - i.e. a system protocol. Support of standard protocols enables programs to interoperate – i.e. one system can access programs and data on another system. This is only possible if the two systems use the same protocol (i.e. the same message formats and sequences) and the applications that are running on those systems have similar semantics, so the messages map to operations that the applications understand (Bernstein 1996).

The other part of the software needs to communicate with a standard application, without needing to know the operating system or networking software on which it resides - i.e. an Application Program Interface. Common APIs or Application Program Interfaces can solve user interface problems, and make it easier to port applications to a variety of server types, giving the customer some independence from the vendors. The interface requirements include database, communication, presentation and other services.

At its most basic, middleware can be defined as the methods which hide the send/receive semantics that are handled by the application software in two-tier systems. Types of middleware are :

- Remote procedure calls (RPCs). An RPC sends a call from one machine / process to another for some service. (Tucker 1997) When the developer is writing the server object, it is first defined using an interface definition language (IDL). The IDL file goes through a pre-compiler, which produces the skeleton server class. The developer then fills out the skeleton with the functionality for the method and compiles it. This compilation is multi-purpose; it describes the object to an interface repository, it produces client and server stubs for the method and produces code to implement the code on the server. A utility is provided to compile the IDL information in a persistent data store that can be accessed by programs at run-time. The run-time objects are instantiated on one or more servers. At instantiation, the run-time objects are registered with the implementation repository. The developer need not get involved in developing communications code, or tracking the exact location of run-time object instances (Orfali *et al.* 1994). RPCs are synchronous.

- Message-oriented middleware (MOM). Messages are sent by the application to the MOM. The MOM saves the messages in a queue for receipt on another machine. Messages are asynchronous, which means that the client does not wait for a server. The MOM makes sure that the message reaches its target at some stage, but the receiving program can control the timing of the reading of the message. One of the big advantages to this is that clients and servers can communicate across a network without being linked by a private, dedicated, logical connection and can run at different times. (Orfali 1996). MOMs also provide the ability to hide the communications protocol from the application.

Middleware methods free applications from send/receive semantics, which bind them to platforms.

Traffic control

One of the big problems in scaling up system size is that of managing message traffic. Message brokers provide a solution. The use of object technology can add a new element to the solution, by packaging application services into components and brokering those components.

Message brokers

Inter-application communication can either be managed directly by the application, or through a message broker. A message broker is “an intelligent third party (hence “broker”) working between information sources and information consumers. It makes communication an independent, shareable function rather than something that is limited to two parties” (Schulte 1996). Traditionally, if an event occurred at one task and this task needed to inform three other tasks, this task would send out three messages and receive back three messages, as shown in . Likewise, any of the other tasks that needed to communicate a message to the group would need to communicate to three tasks and receive three replies.

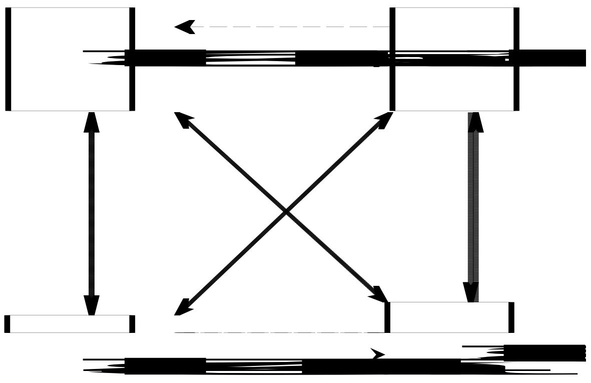


Figure 10 Inter-application communication without a message broker

However, with the use of a message broker, the task needs only to send out one message and receive back one reply. The message broker handles communications with the other three tasks (Figure 11).

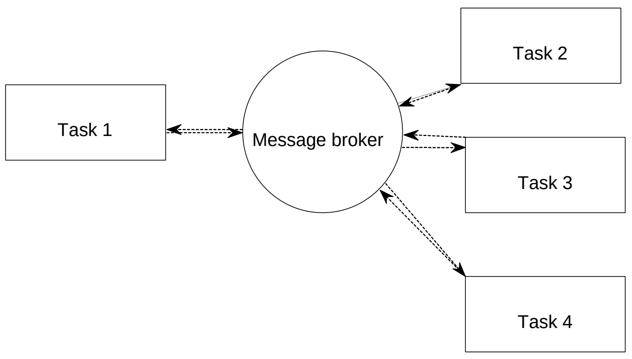


Figure 11 Inter-application communication with a message broker

Message brokers allow communication between compatible tasks. The tasks still need to be connected to the message broker, which is just as difficult as making traditional connections. However, once connected, the task can reuse this connection to communicate with new tasks.

The broker itself has an API through which applications can communicate with it. A message broker can be based on messaging, message-queuing or RPCs (Schulte 1996).

Object request brokers

An object request broker (ORB) is a message broker that works on objects or components. Objects are registered with an ORB and the ORB manages all requests for that object (Tucker 1997). The ORB is the middleware that establishes the client-server relationships between objects. A client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, passing it the parameters, invoking its method and returning the results (OMG 1997). Most ORBs follow either a standard called CORBA - common object request broker architecture, or DCOM - Distributed Component Object Model produced by Microsoft. Internet Inter-ORB Protocol (IIOP) is a CORBA based protocol, promoted by the Object Management Group (OMG), for communications between objects and applications, particularly on the Internet and within intranets (Tucker 1997).

ORBs are set up to deal with objects only. This restricts their use with legacy systems, making them suitable only for object-oriented legacy systems or those that have object wrappers.

The concepts of brokers handling messages and application services being released as components gives the system designer flexibility at a higher level of granularity.

Message brokers allow functional integration. This is where programs communicate with each other by sending messages and waiting for replies. This communication may be synchronous or asynchronous. In general, functions are shared within one domain and functions are transferred between domains. Within an application domain, data is processed by a shareable "service" (a set of business rules and data access logic). Between application domains, production databases are encapsulated on a coarser level. Only their respective, native application programs access them, so edits, integrity checks and business rules can be reused. Both within and between domains, only one development group need know the data models and semantics of the data (Schulte 1996).

4.2.2 Development environments

Multi-tiered systems require a new type of development environment, both to allow development of new components and to allow reuse of components that are already available. Traditional systems development environments are unsuitable, from the point of view of business logic, database logic, programming interface logic or human computer interface logic. The type of development environment usually put forward is a framework.

Frameworks

A framework is an environment that is defined by an API, a user interface and a set of tools. It may also have framework-private middleware services in addition to ones that it imports. It generally maintains context across services and specialises the user interface. It can also offer a simplified sub-set of a service, which is all that is necessary in the context of the framework. The framework includes tools, which are generic applications that make the framework easier to use. Frameworks have a platform isolation layer, which is a set of services for memory, file, process and environmental management. To enable portability of higher layers, these services have the same semantics across various versions of many platforms, such as Microsoft Windows, Apple Macintosh, IBM OS/2, Novell NetWare and several variants of UNIX (Bernstein 1996).

The business application is developed using components from the framework and with some additional process-specific components.

4.2.3 Attributes of hardware

Although integrated systems are the ideal, a system should be measured, not just on its level or capacity for integration, but also on its need for integration. There is a wide range of hardware available, and processors with differing capacities, advantages and disadvantages can be chosen.

According to Laudon & Laudon (1998) management should be involved in the procurement of hardware. They should understand the capabilities of various computer processing, input, output and storage options as well as price-performance relationships. They should be involved in hardware capacity planning and decisions to distribute computing, downsize or use network computers. Attributes that need to be considered are:

- Hardware type: mainframe / minicomputer/PC(networked/standalone)

- Processor size compatibility
- Upgradeability
- Reliability of vendor
- Standardisation of ports and peripherals
- Scalability
- VLSI/RISC/CISC etc.
- Robustness
- Cost

Rightsizing is the process of choosing between platforms, that is the process of selecting the correct or most appropriate hardware platform for the given business application (Robson 1997). Management should weigh up the considerations involved in downsizing as shown in Table 5.

Downsizing advantages	Downsizing disadvantages
Greater user control and power	Weakened central control
Increased flexibility	Technical complexity
Decentralised costs	Hidden costs
Lower costs	Large initial capital outlay
Improved responsiveness	Disruptive to business
Encourages purchased systems	Discourages common systems
Reduces IS workload	Increased user workload and skills demand
Encourages innovation	Staff resistance/ Skill shortfall
Eases and speeds integration	Database disintegration
Business responsiveness	User management distraction
Moves to open systems	Encourages parochialism
Faster system development	Fragments strategic direction

Table 5 Downsizing considerations

4.2.4 Attributes of operating systems

If an organization wants an operating system for its mainstream business applications, it needs an operating system that is compatible with the software required by these applications. It should be easy to use and install. Its user interface features should be easy to learn.

Mission critical applications need an operating system that will provide reliable support for multitasking and memory management. These systems typically have large volumes of transactions to process and require operating systems that can handle large complex software programs and massive amounts of data (Laudon & Laudon 1998). Attributes that need to be considered are:

- Vendor reliability
- Standardisation of hardware requirements
- Obsolescence
- Dependence on specific hardware
- Application availability
- Middleware / ODBC availability
- Scalability
- Security
- Performance
- Robustness
- Support for graphics, calculation power
- Cost

4.2.5 Attributes of networks

Fitzgerald & Dennis (1996) list the network evaluation criteria, from the management focus as:

- Time - Are elapsed time, transaction time, overall processing time, response time or other operational times quick enough?
- Cost - Are annual network cost, per unit cost, maintenance cost, or others, such as operational, investment and implementation costs, in line with expectations?
- Quality - Is a good product or service being produced? Is there more rework because of the network? Has the quality of data/information diminished?
- Capacity - Does the network have the capacity to handle workloads, peak loads and average loads, as well as the long-term future capacity?
- Scope - Is the network's scope properly defined? Does the network interconnect all the necessary business functions?
- Efficiency - Is the network efficient?
- Productivity - Is productivity of the user (information provider) and management (information user) as expected? Is decision making fast and accurate?

- Accuracy - Are there few errors? Can management rely on this network?
- Flexibility - Can the network perform diverse operations that may be required?
- Reliability - Are there fewer breakdowns of this network compared with network goals?
- Acceptance - Have the information providers, the information users and the management accepted the network?
- Controls - Are there adequate security and control mechanisms in place to prevent threats to the network, such as errors or omissions, fraud and defalcation, lost data, breaches of privacy, disastrous events, and the like?
- Documentation Does the network have adequate written / pictorial descriptions documenting all its hardware, protocols, software, circuits and user manuals?
- Training Are training courses adequate and are they offered continually, especially for users? Are training manuals adequate and updated regularly?
- Network life Is the future life of the network adequate? Does it have sufficient capacity for long-term growth?

Goals of network design are :

- Minimum circuit distance between the various computers.
- Adequate circuit capacity to need today's data transfer needs, as well as those required three to five years in the future.
- Efficient software / protocols that can be used on a variety of circuit configurations including satellite circuits that permit the network to interconnect with national or international networks as well as with e-mail systems, use multi-vendor hardware, and connect to public packet switched networks.
- A very high level of reliability (network uptime) must be met. This may be the most important factor. The network designer always should remember that when business operations move into an online, real-time data communication network, it is as if the company had closed its doors to business when the network is down.
- Reliable hardware that offers minimum cost, adequate speed and control features, a high meant time between failures (MTBF), and good diagnostic / serviceability features.
- Reasonable costs.

- Acceptance of the network by both day-to-day users and managers who must use its data or information.
- Sufficient security and control for the highest risk application using the network.

4.2.6 Attributes of development environment

The development environment consists of the tools available to develop the application software for the business system. It consists of a set of translators that translate code into a format that can be read by the system. The development environment, depending on its sophistication, may allow inclusion of components that are already available as part of the development framework or from a component repository.

Management should be aware of the strengths and weaknesses of various software tools, the tasks for which they are best suited, and whether these tools fit into the firm’s long-term strategy and information architecture. Tradeoffs between efficiency, ease of use and flexibility should be carefully analysed. (Laudon & Laudon 1998). Attributes that need to be considered are:

- Openness.
- Standardisation.
- Portability.
- Adaptability.
- Suitability to platform.
- Suitability to application.
- Suitability to database.
- Suitability to human computer interface.
- Adaptability to programming interfaces.
- Robustness.
- Efficiency.
- Adaptability to batch processing.
- Object or component-based or not.
- Documentation.

4.2.7 Attributes of data management

Data storage mechanisms have changed over the years from flat files, through hierarchical, network and relational data models and on to object-oriented, multi-media and hybrid databases. Databases have been distributed over networks. Additional integrated informational needs have been supplied by operational data stores and data warehouses.

Conventional database management systems were designed for homogeneous data that can be easily structured into predefined data fields and records. But many applications today and in the future will require databases that can store and retrieve drawings, images, photographs, voice and full-motion video.

Attributes that management needs to consider when assessing a database system are:

- Data administration with regard to
  - Sharing.
  - Disseminating.
  - Acquiring.
  - Standardising.
  - Classifying.
  - Inventorying.
  - Data planning and modelling (see Software Quality).
  - Database technology and management.
  - Organise structure and content.
  - Develop security procedures.
  - Develop documentation.
  - Maintain database management software.
  - Users.
  - Training.
  - Privacy.

Managers need to evaluate the costs and benefits of implementing a database environment and the capabilities of various DBMS or file management technologies. Key technology decisions should consider the efficiency of accessing information, flexibility in organising information, the type of information to be stored and arranged, compatibility with the IS infrastructure and data or object model (Laudon & Laudon 1998).

4.2.8 Platform Configuration

The choice of an appropriate platform is not based solely on the quality of an individual piece of hardware or support software. It is based on defining the needs that the organization will have with regards to network scalability and versatility of hardware, operating system, development and operating environment and data management systems. Once the platform characteristics that are ideal for the organization have been identified they can be matched as closely as possible with configured components.

4.3 Common problems

Within many organizations, different groups, using a variety of tools, databases and languages, on a variety of platforms and operating systems have developed systems over the years. Packages were bought in to service a particular need, which did not communicate with any other systems. This diversity has occurred even within a business unit, resulting in several stand-alone systems being used by the same operating personnel. Users may use different logons, user interfaces and different physical peripheral devices to access different systems in the course of a day's work. Quite often, the user enters the same piece of data into two or more systems independently. Systems grew around the area where a problem needed to be solved and were written to serve only those customers who were customers at the time of writing and only for the specific service requested. Any data that is transferred between systems is sent by means of extract and update programs or database gateways, using two-tier client/server systems, where communication logic is written in to the system software at client and server ends. There is no sharing of data processing (Schulte 1996).

4.3.1 Hardware

Adolph (1996) listed some of the problems that can be encountered when a system ages. The hardware may suffer from "bizarre restrictions such as 64Kbyte segments or 80-column record restrictions". He also mentioned a) performance problems and memory limitations, b) host computer becoming difficult to obtain and service, and c) scalability being limited by the host computer's processing capacity. He also showed how the size and complexity of the program lead to constant patching, which made the reliability questionable.

Adolph's (1996) criticisms of older systems are based in a modern context. Although these restrictions were necessary and valid for the hardware that was used at the time, hardware has now moved on, with the consequence that many of these restrictions are no longer meaningful, especially where software has been moved off the restrictive hardware. Likewise, although modern programs are also large and complex, there is not the same dependence on hardware restrictions as there was then.

Although there are cases where the hardware on which a system runs has been tailor-made for the task, this is not generally the case in the area of legacy business systems. That scenario is more likely to present in process control systems. Most legacy business systems are running on a large mainframe or mini-computer or cluster of mini-computers that are provided by a single manufacturer with a proprietary operating system. In general, these older machines are quite conservative and restrictive in the use they can make of new technology. While this was not a problem in the hey-day of these systems, it has become a problem, because customer expectations are raised to new heights. This is not just a problem with the machine, but with the operating system and software development environments offered by the older platforms. These systems are therefore rarely capable of incorporating new business processes using the latest technological innovations quickly or easily (Gibson *et al.* 1998). This is a modern problem, which has arisen due to raised expectations regarding what services a system should provide and how quickly that service can be incorporated or changed. Modern mainframes and minicomputers have overcome these problems and have the added advantage over personal computers of a much better security system.

There are two areas where custom-built hardware can cause problems in business systems.

a) Custom-built peripherals are used for input or output that has been done over the years in a specific way. These peripherals offer an interface to the customer that has been established as being reliable and trustworthy. However, these high-cost devices are unable to adapt to changing technology such as electronic transfer or change in format of the data being offered for input / output.

b) Incompatibility between process control system (e.g. power monitoring or automated vehicle control) hardware and financial system hardware. The approach taken towards developing process control systems has differed radically to that of developing business systems. In most cases, within the same organization, different teams will build these two types of systems on different types of hardware, using different operating systems, completely different development approaches, different programming languages and different data storage mechanisms. Semi-state organizations are providers of infra-structural services to the community that are likely to involve process control systems. This leads to problems when trying to share data or processing among the systems in the organization. The status of process control systems is outside the scope of this dissertation, but information systems that need to interface to process control systems must have the flexibility to do so.

Moving from legacy hardware can cause problems, especially in the areas of accountability for cost (Slee & Slovin 1996), worries about reliability, security and ability to maintain (Slee & Slovin 1996) and need for new training and change of attitudes (Adolph 1996). Staying with old hardware can incur risks of obsolescence, dependence on a single supplier and restrictions on business change and growth. It is necessary, therefore, not just to decide to leave legacy hardware behind, but also to evaluate future options very carefully, so that the advantages offered by older hardware are not lost.

4.3.2 Operating system

Operating systems can be tied to the hardware that they service. Unless an operating system is in reasonably widespread use, it is unlikely that it will support the required range of middleware or supplied software. Operating systems can limit use of new technology.

Problems related to changing from one operating system to another include lack of openness of applications already running on the operating system, worries about robustness, security, life-span, compliance to standards, performance and availability of components to run on it. In regard to middleware products, it is difficult to choose which protocols and APIs to use, as there is a risk that the chosen one will become obsolete due to uncertain market forces (Bernstein 1996). Therefore, in choosing new technology, it is necessary to thoroughly check out the reliability and stability of the technology, so that it will endure for the lifetime of the system that is based on it. This requires a balancing act – certain technology leaders will risk unproven technologies to gain competitive advantage. While this is a valid option, there is no doubt that it is a risk. Organizations who are not at the leading edge of technology should be slow to opt for a technology that has not been proven. This does not exclude these organizations from upgrading their systems. Platform technology does not need to be the newest on the market to provide those attributes that are cited above.

4.3.3 Networking

Along with problems of incompatible hardware, there can also be communications problems. Communications may have inadequate bandwidth to suit expected needs projected over the next five years, or be using non-standard communication protocols. Network gateways may be bottlenecks, which will stifle the growth of the network. The cost of maintaining the current network infrastructure may be disproportionate to its value to the business or even to networks providing similar value elsewhere. It is difficult to move these systems from one platform to another. It is costly and difficult to scale adequately for growing business demands. As semi-state organizations often own or have control of physical networks such as railways, power lines or telecommunications lines, there is a possibility for them to own their own communications network. While this gives them more control over its use, it also lessens the incentive to upgrade bandwidth or communications technology, thereby hampering their ability to take advantage of advances in this technology. It also gives rise to the possibility of hybrid networking techniques, which cause problems unique to the site.

"Likewise, distributed databases have not replaced centralised databases and end-users have not taken over. C/S has become dominant, but is changing. New delivery channels; mobile, Internet and inter-enterprise messaging are supplementing C/S." (Schulte 1996).

Once again, it is necessary to achieve a balance when acquiring new technology. There are definite restrictions in staying with outdated networks, but there is also a risk in going to new technology. The stability and security of older systems should not be thrown away lightly, but often a combination between old and new technology can offer the optimal solution.

For networking, speed of access, security and privacy require relatively new technology. This should not be allowed to exclude the use of other aspects of the platform that have aged gracefully.

4.3.4 Development environment

"Early tools and design practices are giving way to more sophisticated technology and management approaches." (Schulte 1996)

The development environment consists of:

- the development mechanism for business rules software.
- the development mechanism for interfacing software, to data, human / computer and other applications.

Arnold (1989) Levey (1995) and Bancroft (1997) are among those who recognise the difficulties that arise from trying to maintain software on older environments. The language and operating system used to develop a system are vitally important to its ability to age gracefully. If the language is relatively standard and is upgraded frequently, then this gives the system a longer lifetime. Adolph (1996) cited the obscurity of the development environment (programming language, line editor, assembler and linker and cryptic command language) as difficulties in maintaining code.

While systems that were written in Cobol in the 1970s are still in use today and can be modified to take advantage of modern development practices, amendments are still slower than they would be if the language were more flexible. Most business systems are written in languages that, although they are more tedious than modern languages, are still maintained by their suppliers. However, the lack of flexibility in these development environments causes problems. The business software may not be independent of the human-computer interface (HCI) or the data management. This problem is even worse when a development environment that is no longer maintained by the supplier has been used. The impact of having to learn an outdated and obscure environment is very damaging to programmer morale, in that the experience gained is unlikely to be useful in the future. In many cases, some code generation is done, meaning that the executable code could reside in obscure libraries throughout the system. This cuts down the traceability of the system considerably. It is more difficult and therefore more expensive to employ personnel to maintain or enhance the systems based on outdated platforms. Programmer productivity is down due to the complex nature of the task and the inadequacy of the tools, causing the need for development programmers to remain artificially high. In the opinion of this author, this is a problem that is in its infancy. As platforms become more and more diverse, they also become obsolete more quickly. This problem is particularly evident in the PC end of the market, towards which a lot of organizations are heading.

This problem can be overcome by choosing the development environment with care. No technology is proven when it is brand new. Languages such as Cobol will take a very long time to disappear and may never completely disappear. However, approaches at upgrading these languages to a more modern environment can often lead to an environment that is clumsy and not as reliable as its predecessors. Choice of development environment should be based on the suitability of the environment for the job in hand and also its prevalence in the marketplace. An environment with a higher profile, although it will eventually become obsolete, will have a bigger following and will therefore be upgraded for longer and will offer a wider range of experts.

It is often the case that more than one development environment is in use in an organization, and these environments are not compatible. Different groups, tools, databases, platforms, operating systems, languages are in use and are not integrated, even within a business unit.

This problem can occur where process control systems are in use in the organization, which can effect interfaces to business systems or non-standard devices, is that the programming of these is non-standard and may be device-specific. Some process control systems are written in an assembly language that is specific to the hardware on which they run. When evaluating an environment for business systems, its openness to these process control environments is a factor.

Risks that are incurred in updating to new development environments are:

- development environment complexity.
- development environment standardisation.
- development environment life-span.

Despite the emergence of new tools to help in the construction of open and shareable systems, the process of splitting the problem over layers - particularly the client/server split - is still left to the service developer and is not always straightforward. (Bernstein 1996).

4.3.5 Data management

"Relational database management systems (RDBMS) are now ubiquitous. Distributed databases have not replaced centralised databases and end-users have not taken over. Most enterprise IT portfolios consist of many application domains that are joined, where necessary, in a clumsy and non-integrated way. In some cases, systems within a domain will have consistent technology, data models and semantics, but not always. Between heterogeneous domains, this consistency tends to disappear…. Operational data stores are generally read-only databases, operating something like a data warehouse, more for predictable transactional look-ups than ad-hoc decision support queries. These are forever redundant with legacy and purchased application data models, which sharply define their limits". (Schulte 1996)

Data management is an area that has changed a lot since the 1970s. It also differs greatly between process control and business systems. Process control systems tend naturally towards an object-based design, which may use flat files or be supported by an object-oriented database. Business systems tend to be based on either a hierarchical or relational database management system, or on a file management system

A legacy business system may contain valuable data but lack the power and agility to meet current organizational needs (Brodie & Stonebraker 1995). This data may be in the wrong format or out-of-date and difficult to integrate with other data. If the data management system being used is particularly old or non-standard, it is likely that any attempt at reading or updating it requires programming using a third generation language. Older systems generally had a lower user involvement in their development and ownership of the data or system is often not established (Bancroft *et al.* 1997). As responsibility goes with ownership, there may be no notion of who is responsible for the accuracy or security of the data. As users may be entering the same data into two or more systems, there may be a breach of data integrity. The definition of the same piece of data may differ between systems, or be too rigidly defined within a system, leaving little room for change, as is happening with the Year 2000 crisis. Data management systems vary from file management, which involves coding the data structures into the system programs, through hierarchical, network and relational database systems and on to object-oriented or hybrid database systems. During the last few years, advancements in the area of sharing data from different data management system providers have begun to emerge.

Problems that exist in the area of data management are:

- Diversity of data management systems in use.
- Incompatibility of data management systems in use.
- Inflexibility (in terms of business change and growth) of current data management systems.
- Current systems have an infrastructure that is data-based, so it is difficult to veer from this.
- Despite the centralisation of data, this data can be updated from different processes, giving rise to possible integrity losses between processes.

Risks incurred in adapting new data management methods:

- Chosen proprietorial database may become obsolete.
- May not be compatible with existing systems.
- Large investment in changing I/S infrastructure and support processes.

Object-oriented databases such as POET are in existence today, but have not yet gained a wide market-base in the area of business applications. Object-based systems are now more likely to use a relational or hybrid database to store the data part of the object. Many database management systems, particularly relational databases, are used as part of a fourth generation environment, which also provides a development and operating framework. These environments allow rapid application development and many of them have adapted over the years to become hybrids of their former classification (Oracle (www.oracle.com)). Those with a wide market-base are also likely to endure throughout changes in the world-wide environment. They have also established themselves to such a degree that other software developers have used them as a basis for their products. However, it would be foolish to underestimate the disruption to the organizational environment that is incurred by introducing a new database management system throughout the organization. It is likely that some incompatibilities with existing systems will occur and also highly likely that existing staff skills will not meet the challenges of the new technology without substantial retraining and raising of staff numbers.

4.3.6 Platform configuration

Each aspect of a platform can cause problems, of and within itself. Technology changes so rapidly that a platform can become outdated in a short space of time. It is not possible to update platforms every time new technology emerges. However, by careful planning and evaluation of future platform needs, the lifetime of a system that resides on it can be greatly elongated.

4.4 Effects of problems

Lack of openness can lead to a downgrading of service, in that communications with other systems or platforms cannot be direct. These communications may take the form of semi-automatic (i.e. a tape or disk may be used to transfer data from one system to another. Examples of this are certain instances of Electronic Funds

Transfer at Point of Sale EFTPOS and Electronic Data Interchange EDI) or manual (the data is printed out from one machine and typed in to another). The former method suffers from the fact that there cannot be two-way communication. This stifles the system's ability to adapt to change in the environment by integrating with other systems and platforms. Table 6 cross-references the Legacy Effect Determination Framework (

Legacy effect determination framework		
	Effect	**P**resent **A**bsent or **U**ndetermined
Asset value	Mission criticality	
	Reliability	
Ease of operation	User satisfaction	
	Ease of testing and auditing	
Ease of maintenance	Cost of maintenance and resistance to it	
	Availability of maintenance resources	
	Program size and complexity	
	Dependence on individuals	
Ease of migration / evolution	Ease of use of new technology	
	Scalability	

Table 2) with possible causes relating to the underlying platform suitability dimension. Each of the causal criteria has a separate row in the table. In the row, a column marked with "X" indicates an effect that may result because of a weakness in that causal criterion.

Legacy Effects / **Underlying platform suitability**	Asset value		Ease of operation		Ease of maintenance				Ease of migration / evolution	
	Mission criticality	*Reliability*	*User satisfaction*	*Ease of testing and auditing*	*Cost of maintenance and resistance to it*	*Availability of maintenance resources*	*Program size and complexity*	*Dependence on individuals*	*Ease of use of new technology*	*Scalability*
Hardware		X			X	X	X		X	X
Operating system				X	X	X	X	X	X	X
Network	X	X	X	X	X	X	X	X	X	X
Development environment		X	X	X	X	X	X	X	X	X
Data management		X		X	X	X			X	X

Table 6 Effects of platform unsuitability on legacy status

4.4.1 Hardware

In conclusion, hardware problems can cause the effects of:

- Reliability may diminish, because the hardware may not have the capacity to enable complex journalling or increased numbers of conflicting users.
- Cost of maintenance and resistance to may rise if the hardware is limiting the capacity of change to the system.
- Availability of maintenance resources will suffer if the manufacturer no longer supports the hardware.
- Program size and complexity can be affected by certain hardware limitations.
- Use of new technology is more difficult if the hardware is not compatible with current standards.
- Scalability depends on the hardware being amenable to networking and to extension.

Although other effects may be noted from legacy hardware, these effects tend not to be the primary problems that present themselves:

- Mission criticality may diminish as a result of other legacy effects relating to hardware being ignored.
- User satisfaction is more likely to be based on the service provided – it may suffer through slowness and lack of reliability.
- Lack of capacity or complex or large programs may affect ease of testing and auditing.
- Dependence on individuals is less likely to be a problem with legacy hardware in the business applications area as most business applications use relatively standard hardware.

4.4.2 Operating System

Note: many older operating systems, especially those on larger machines have stood the test of time and are not obsolete, in that the applications they support are also supported by their successors. However, some relatively new operating systems, specifically those on smaller machines, have a shorter life span and their applications may not be supported by operating systems that supersede them.

Legacy operating systems affect:

- Ease of testing and auditing, if development staff need to learn an obsolete or obscure operating system that imposes limitations on the development environment
- Cost of maintenance and resistance to it will be similarly affected.
- Availability of resources to maintain systems will diminish when the operating system becomes obsolete, especially if it had a short life span.
- Program size and complexity will depend on how much functionality needs to be incorporated into the programs to work the operating system adequately, and also on operating system size limitations. These size limitations may depend directly on the hardware, or on a limitation in the operating system.
- Ease of use of new technology depends heavily on the adaptability of the operating system to it.

- Scalability requires an open operating system that supports the required networking and can adapt to use with other machines, possibly using different operating systems.

Legacy effects that may not be caused by the operating system:

- Diminished mission criticality should not be one of the early symptoms caused by operating system problems. It may eventually become a problem, but other symptoms will be present first.
- Reliability should not be changed by the operating system alone.
- User satisfaction should not depend heavily on the operating system. Older operating systems user interfaces may cause irritation in some users but most operating systems that are now in use allow for the implementation of a good user interface design and adequate system functionality.

4.4.3 Network

Legacy networking problems can affect:

- Mission criticality, in that if the system is not universally available, its use will be inconsistent and users will find ways around depending on it where possible.
- Reliability and speed are highly dependent on the networking quality.
- User satisfaction depends on reliability, consistency and availability, which are heavily affected by networking.
- Ease of testing and auditing can be greatly hampered by lack of adequate tools that can be configured over an inappropriate network.
- Cost of maintenance and resistance to it will rise where the network either causes regular problems or causes problems that are difficult to trace.
- Availability of maintenance resources depend on the adherence of the network to standards that are established either by widespread practice or are well documented.
- Dependence on individuals can be prevalent where a hybrid network is in place.
- Program size and complexity can rise in situations where inter-application communication over a network needs to be handled by the application programs.
- Ease of use of new technology is hampered by non-standard network operation.

- Scalability in a distributed system is dependent on the network.

4.4.4 Development Environment

Legacy development environments cause the effects:

- Diminished reliability, where environment lacks the security provisions necessary to ensure a robust system.
- Diminished user satisfaction where the environment does not enhance usability or functionality.
- Difficulty in testing and auditing can be a problem where the environment suffers from lack of traceability.
- High cost of maintenance and resistance to it results where the environment is obscure or obsolete or does not provide a reliable mechanism for tracing and eradicating errors or implementing standard testing procedures.
- Availability of maintenance resources will be low where the environment is obscure or no longer supported by the supplier, as is often the case where maintenance contracts have lapsed.
- Program size and complexity may grow, especially where developers do not fully understand how the current system works and do not have the facilities to find out.
- Difficult to adapt to new technology – many of the more complex development environments are quite rigid in their interfacing abilities. Often the use of new technology requires that the developer work around the development environment rather than work through it.
- Scalability can be a problem due to lack of openness.

Lesser effects of the development environment:

- Mission criticality may diminish as a knock-on effect of those mentioned above, but should not be one of the primary problems.

4.4.5 Data Management System

Problems with data management systems will cause the effects:

- Reliability may be suspect due to redundant information or lack of security.

- Difficulty and high overheads in testing and auditing if there is a lack of rigorous data management or the system does not provide adequate tools.
- Cost of maintenance and resistance to it will rise if the data organization becomes corrupt or redundancies creep in.
- Availability of maintenance resources will depend on the system being relatively standard or well documented.
- Availability of resources to maintain it will depend on the supplier maintaining the system and maintenance contracts being upheld.
- Difficult to use new technology if the data management system is not amenable.
- Scalability depends on ease of replication / partition and openness.

Lesser effects:

- Mission criticality will be a knock-on effect of diminished reliability, but should not be the first symptom.
- User satisfaction may diminish as a secondary effect.
- Program size and complexity may be related to the data management system where the development environment is part of the data management system or data requests require complex programming.
- Dependence on individuals may be caused by badly documented designs or obsolete or obscure data management systems being in place.

4.4.6 Platform configuration

Each platform component has a variety of problems that are inherent to it. However, the combination of platform elements is also a likely cause of legacy status. Platform elements can be combined in such a way that they enable or inhibit key areas of growth in a system. The management of platform development and integration is of paramount importance and failure to plan and implement this can cause legacy problems in a relatively new system.

4.5 Conclusion

A platform consists of the technical infrastructure that supports a system or systems throughout their lifetime. If this platform is well planned and maintained, and is suited to the system and organizational needs, then choices regarding platform components are made easier. Platform components can be upgraded or added in a compatible manner, which will not cause difficulties. The philosophy of integrating systems from different platforms is relatively new in legacy terms and many of the problems result from a failure to evolve compatible platform components.

Chapter 5 Software Quality

5.1 Introduction

The aim of this chapter is to establish the meaning of software and the term “Software Quality” and how the software engineering process affects the quality of software. The rest of this section consists of definitions of software, Software Quality criteria, the software engineering process and how it is broken down for evaluation in this dissertation. Software Quality is broken into software code or component quality, software design quality and finally the quality of Change Management. The mechanisms for implementing quality software are described in Section 5.2. Section 5.3 shows problems that cause these mechanisms to be ignored or to fail. Section 5.4 lists the effects that are caused when these mechanisms fail. These effects are illustrated in Table 7. Section 5.5 concludes the chapter.

5.1.1 Software definition

There are many definitions of the term “software”. For The purposes of this dissertation, the definition given by Pressman (1997) is adopted. He suggests the following formal definition of software: “Software is (1) instructions (computer programs) that when executed, provide desired function and performance, (2) data structures that enable the programs to adequately manipulate information, and (3) documents that describe the operation and use of the programs.” He states the characteristics that are exhibited by software:

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software does not wear out.
- Most software is custom-built, rather than being assembled from existing components. Software components are built using a programming language that has a limited vocabulary, an explicitly defined grammar and well-formed rules of syntax and semantics.

5.1.2 Software quality aspects

The Institute of Electrical and Electronic Engineers (IEEE 1983) define software quality as “the degree to which software possesses a desired combination of attributes”. These attributes are typically referred to as quality factors. McCall *et al.* (1977) and Boehm (1978) suggested quality characteristics and models for these, which include software correctness, reliability, efficiency, integrity, usability, maintainability, testability, flexibility, portability, reusability and interoperability. Later, one of the international standards for software quality that was developed was ISO 9000-3 (1991). This included a part on “Guidelines for the application of ISO 9001 to the development, supply and maintenance of software”. Factors mentioned here are testability, maintainability, reliability and interoperability. A new draft international standard (ISO/DIS 9000-3, 1996) lists functionality, reliability, usability, efficiency, maintainability and portability.

The factors or aspects of quality mentioned correspond to a large degree to the effects that are listed in Chapter 2 regarding legacy status. Many of these software quality aspects relate to the suitability of the system or the platform. However, the process of engineering the software plays a large role.

Kitchenham & Pfleeger (1996) discuss five views of software quality. The first is the transcendental view, where perfect quality is an ideal towards which we strive. The second is the user view, which sees quality as fitness for purpose. The third is the manufacturing view, which sees quality as conformance to specification. The fourth is the product view, which sees quality as tied to inherent characteristics of the product and finally the value-based view sees quality as dependent on the amount a customer is willing to pay for it.

5.1.3 Software Engineering

Software engineering is a discipline that has been in existence since the 1960s. The original definition of software engineering was given at a NATO conference (Naur & Randall 1969) by Fritz Bauer and is as follows:

"Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines". The IEEE (1993) expand this definition: "Software engineering is (1) the application of a systematic, disciplined, quantifiable approach to the development,

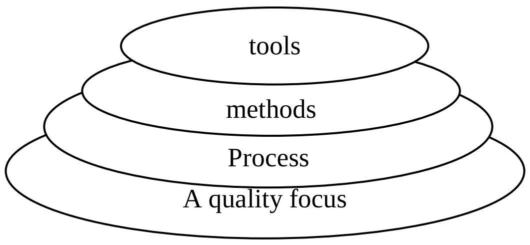


Figure 12 Layered Technology of Software Engineering (Pressman 1997)

operation and maintenance of software; that is, the application of engineering to software, or (2) the study of approaches in (1)".

Software Engineering is a layered methodology (Figure 12): Any engineering approach must rest on an organizational commitment to quality. The bottom layer is a quality focus. The next layer is the process layer. Process defines a framework for a set of key process areas that must be established for effective delivery of software engineering technology. Methods provide the technical 'how to's' for building software (e.g. requirements analysis, design, program construction, testing and maintenance. Tools provide the automated or semi-automated support for the process and the methods. There are also umbrella activities, which must be carried out, to maintain a quality system. These include:

- Software project tracking and control
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Document preparation and production
- Reusability management
- Measurement
- Risk management

Parkinson (1991) offers a slightly different and more detailed view of the components of a development approach (Figure 12). From these diagrams it can be seen that the process model, methods and tools chosen have a high impact on the quality of the final system. Parkinson (1991) gives a methodology as having a set of five steps, but does not put a sequence on them. Process models and methodologies are explained in section 5.2.2.

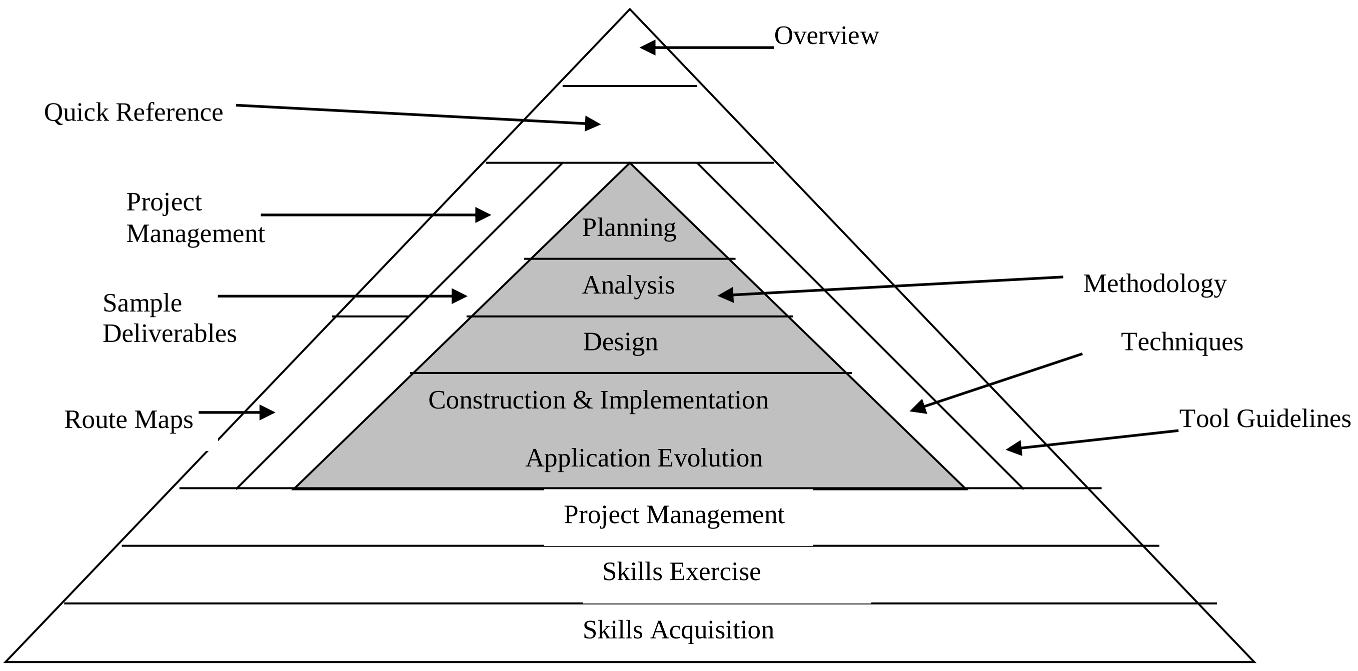


Figure 13 Components of a development approach (Parkinson 1991)

Although the construction of software components is treated no differently here to any other step in the process lifecycle, when a system is being examined from a reengineering point of view, the constructed component may be the only remnant of the original development. For this reason, component quality is treated separately in this dissertation (see Section 5.2.1).

Pressman's (1997) 'quality focus' and Parkinson's (1991) 'Application Evolution' show that the quality of the software depends not only on the process model, methods and tools that have been chosen, but also on the implementation, maintenance and upgradeability of the system. The management of process change is therefore vital to the continuing assurance of quality (see Section 5.2.3).

The management of process implementation and change is unique to each organization and in some cases, to each system. Although there is no specific set of rules which governs exactly how an organization should manage process change, there are standards against which an organization can check to see if they are succeeding.

5.1.4 Software Quality in context

Although the term "Software Quality" can cover all quality aspects of a system, within the context of this dissertation, "Software Quality" is confined to the quality of the application software as written, designed and evolved.

5.2 Mechanisms for Improving Software Quality

Attempts at improving software quality have abounded since the first computer program was written. Early attempts focused on the program and how it could be simplified and structured. These were partly related to the development environment and platform on which the programmer worked, but later extended into a set of standards that can be adapted from organization to organization. Early design work focused on criteria for developing structured programming (Dahl *et al.* 1971) using a top-down structure (Wirth 1971) and modular programs (Dennis 1973). However, it became apparent that the quality of code within a program was not the only problem. As applications became larger and more modular, software programs stopped working in isolation. An application could be made of hundreds or thousands of components, each containing at least one program and probably more. The quality issue, therefore, does not stop at code quality.

Parkinson (1991) charts the result of several studies, carried out in a variety of environments in the USA during the 1970s and 1980s (Figure 14). This shows that within system development, more than 60% of the defects and errors reported in information systems

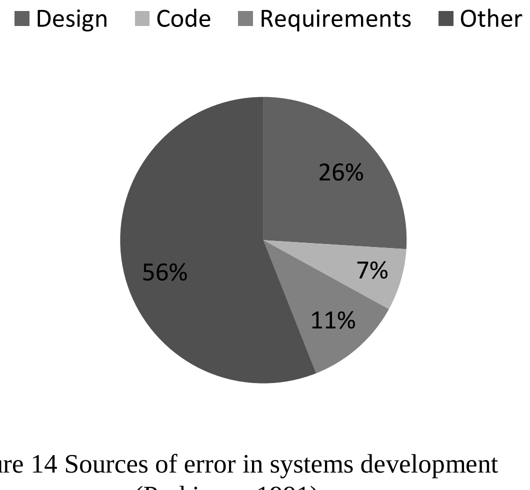


Figure 14 Sources of error in systems development (Parkinson 1991)

after they had been installed can be traced back to incorrect requirements analysis and specification (Parkinson 1991).

A further chart (Figure 15) shows that the resources required to fix errors in software were much higher for requirement errors (82%) than for any other errors.

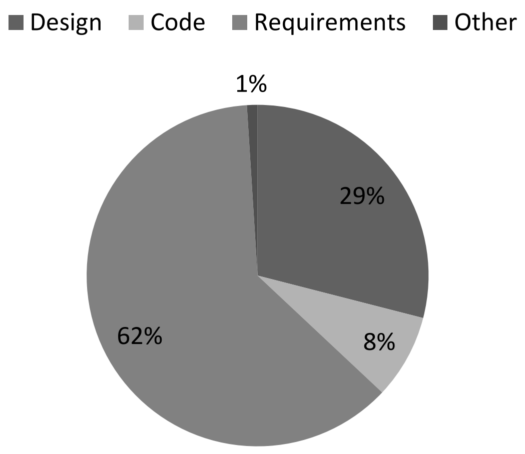


Figure 15 Resources required to correct errors in software (Parkinson 1991)

While still trying to improve the design of the system, Stephens *et al.* (1974) and Yourdon & Constantine (1978) developed graphical notations for representing data and the processes that transformed it. DeMarco (1979) introduced and named the key graphical symbols that enabled an analyst to create information flow models, suggested heuristic for the use of these symbols and the use of data dictionary and processing narrative. Gane & Sarson (1982) came up with one of the many variations and Ward & Mellor (1987) and Hatley & Pirbhai (1987) extended the ideas for real-time systems.

The earliest process model was the Waterfall Model (Royce 1970). However as applications grew and became more diverse and less stable in their requirements, this model was seen to have its drawbacks. Later models offer variations that may be better suited to some applications or environments.

Therefore, although software quality at component level and at design level has been considered important for the last thirty years, it is obvious that over that time, the particular practices that were carried out varied greatly. As technology changes and business requirements change more and more rapidly, the focus now moves on to include the problem of transition from one software engineering paradigm to another.

In conclusion, the issues to be addressed when assessing Software Quality in the context of this dissertation are:-

- Quality of the code within a module or component henceforth known as component quality.

- Quality of the current system process documentation that is in use for an existing system, henceforth known as design quality.
- Quality of implementation of process change, with particular regard to quality assurance processes.

5.2.1 Component quality

In order to be able to ensure software quality, the quality of the code within each software module or program needs to be structured. The program structure should follow a standard procedural design technique.

Dijkstra (1965) and Bohm & Jacopini (1966) worked on principles of procedural design. The flowchart was once the most widely used graphical representation for procedural design, but it allows violation of the principles of structured programming, including sequence, selection and iteration as the only logical constructs that should be used (Dijkstra 1976). Nassi-Shneiderman diagrams (Nassi & Shneiderman 1973) offer a more structured diagrammatic technique. Stepwise refinement (Wirth 1971) takes the design to a higher level, where the problem solution is listed as a set of instructions and complex instructions can be broken down further at a later level, thereby providing a top-down design. This leads to an increase in the opportunities for modularity. A well-designed module (Myers 1974) should show a low level of coupling and a high level of cohesion. These are the principles of procedural design of the program. Along with the procedural design of the program, the design of its interfaces must be formalised. The data structures used within programs vary in complexity, with some of the more complex structures having algorithms that are designed and published (Aho *et al.* 1983). Wasserman (1980) proposed principles of data design such as data abstraction and information hiding. Increasingly, the way in which programs and persistent data interact is changing, with the change in data management systems. The structure of data that is used by a module but is defined outside the model can relate back to Chen's (1977) Entity Relationship Diagrams, where entities can be converted to files or database tables. Another area of design concerns the calling structures used on and by a module. As platforms become more distributed, these structures can be very complex and need to be very well specified. Interface design addresses the design of three types of interfaces. The first is between software modules and can be addressed by Data Flow Diagrams (DeMarco 1979) or object sequence or interaction diagrams (Rumbaugh *et al.* 1991). The second is the design of interfaces between the software and the human user (HCI) (Shneiderman 1987, Preece *et al.* 1994, Dix *et al.* 1993) or a peripheral device and the third is the design of interfaces between the software and external data.

Although these practices attempt to provide a mechanism for high quality design, quality cannot be assured by their presence. This is the task of the quality assurance group. To assure quality in design and implementation, as well as having high design quality criteria, quality of conformance must also be high. "Quality of design refers to requirements, specifications and the design of the system. Quality of conformance is an implementation issue. If the implementation follows the design and the resulting system meets its requirements and performance goals, conformance quality is high." (Pressman 1997).

Paulk (1993) defines software quality assurance activities for design and implementation that can be addressed by two groups, 1) the Software Quality Assurance group and 2) the Software Engineering group.

The Software Quality Assurance group is responsible for:

- Preparing a Software Quality Assurance plan for a project.
- Participating in the development of the project's software process description.
- Reviewing software engineering activities to verify compliance with the defined software process.
- Auditing designated software work products to verify compliance with those defined as part of the software process.
- Ensuring that deviations in software work and work products are documented and handled according to a documented procedure.
- Recording any non-compliance and reporting to senior management.

The Software Engineering group is responsible for:

- Applying solid technical methods and measures
- Conducting formal technical reviews
- Performing well-planned software testing

A full description of the review and testing process and other topics relating to software quality assurance is available in Pressman (1997) Chapter 8. Testing techniques are covered in Pressman (1997) Chapters 17 and 22 (object-oriented testing).

Conclusion

The quality of a module of code can endure throughout multiple changes, provided the style, organization and structure of that module follows the prescribed standards for the system. If the standards change, then all of the code should either remain in the old standard or be converted to the new standard. This is a policy decision to be taken by the quality assurance group. High quality components are one of the criteria necessary to fulfil Kitchenham and Pfleeger's (1996) manufacturing view of software quality.

5.2.2 Design quality

Along with the modular design that is needed for each software module, an overall architectural design is needed for the system. There are several process models that have been proposed over the years and most well designed systems will follow one of them. A process model is also known as a software-engineering paradigm. It is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required. Each model should address the generic phases that are required: the definition phase, the development phase and the maintenance phase. Commonly used models are the Linear Sequential model, prototyping, the incremental model, the spiral model and the component assembly model.

Linear Sequential model

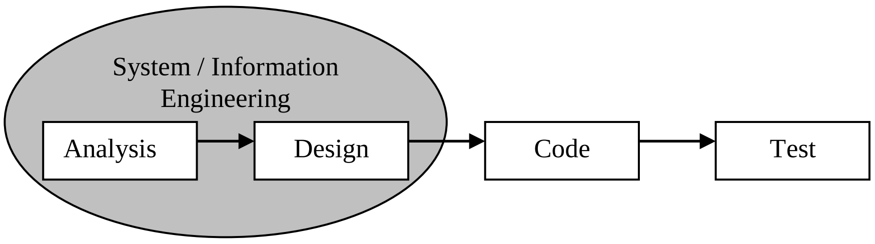


Figure 16 Linear Sequential model (Royce 1970). Diagram from Pressman (1997)

Also known as the classic life cycle or waterfall model (Royce 1970), it suggests a systematic, sequential approach to software development (Figure 16).

Problems with this approach are:

- Real projects rarely follow the sequential flow and changes can cause confusion.
- This model has difficulty accommodating requirements change.
- The customer will not see a working version until the project is nearly complete.
- Developers are often blocked unnecessarily, due to previous tasks not being done.

Prototyping

Brooks (1975) describes the prototyping paradigm. The developer and customer define the overall objectives for the software. A quick design focuses on what the customer will see. From this, a prototype is constructed. The user evaluates it and improvements are made. This continues in an iterative fashion until a satisfactory product is achieved (Figure 17)

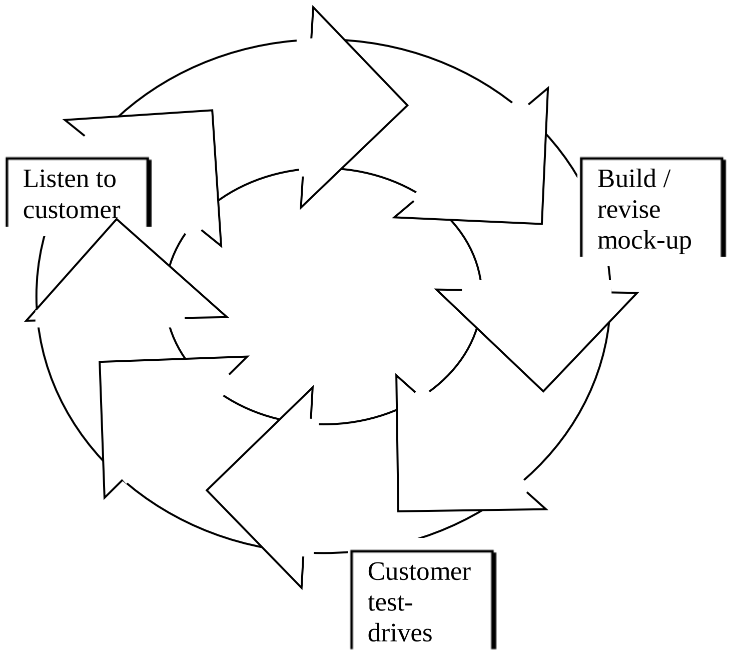


Figure 17 Prototyping paradigm (Brooks 1975)

Problems with this approach: -

- The customer sees a working version and expects the finished product to be available in a short time. This puts pressure on the developer to take short cuts, at the expense of quality and maintainability.
- The developer may make compromises for speed. Inappropriate tools may be used or inefficient algorithms may be used, which then become integral parts of the system.

The RAD model

Rapid Application Development (Martin 1991) is a linear sequential software development process model that emphasises an extremely short development cycle.

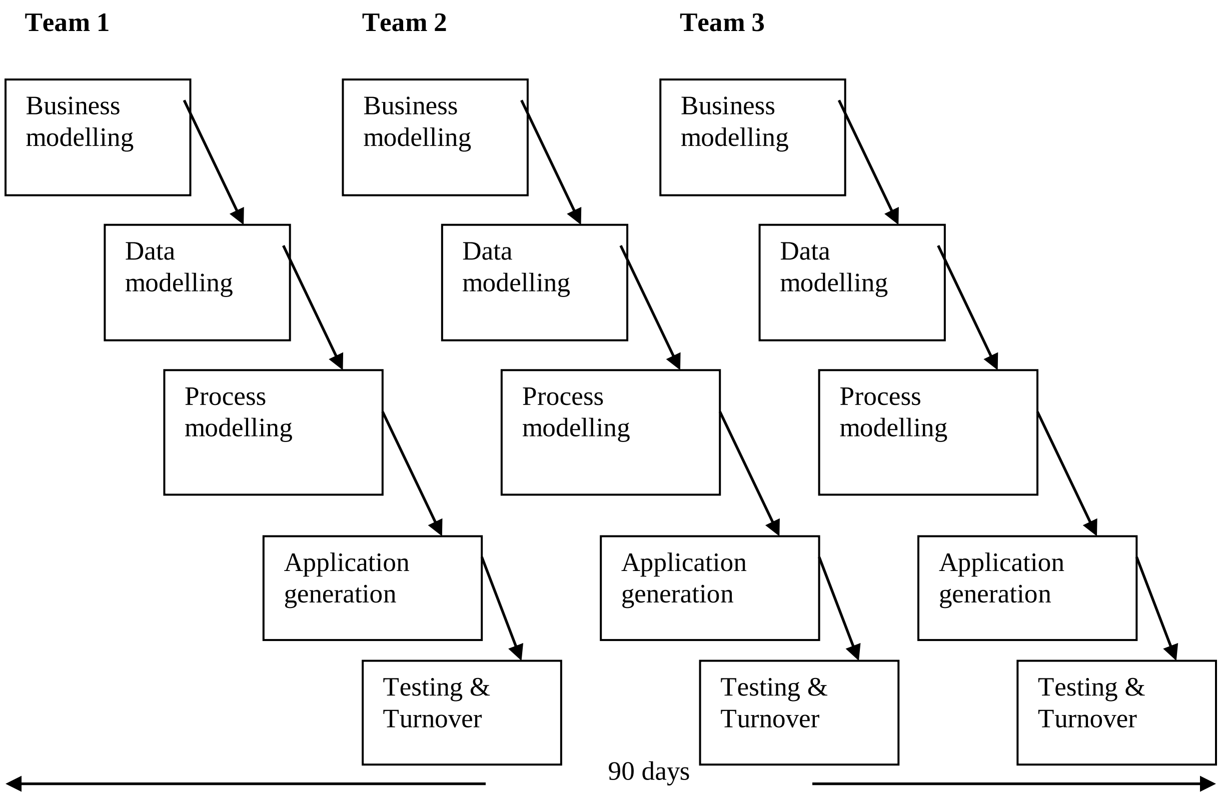


Figure 18 The RAD model (Martin 1991)

A component-based construction approach is used. To use this approach, the project scope must be constrained and the requirements should be well understood. A task that should take no more than ninety days to complete is modelled, generated and implemented. There can be several teams working on different components during this ninety day time-box (Figure 18).

Problems with RAD: -

- For large, scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
- RAD requires developers and customers who are committed to the rapid-fire activities necessary to complete a system in this time frame, or failure will result.
- RAD is not suitable for many project types.

The incremental model

This model is described by McDermid & Rook (1993) as a combination of the linear sequential model and the iterative model. The problem is broken into increments, and each increment is tackled as a linear sequence. Further increments can either be done after the previous ones, or can overlap with the previous ones. Incremental delivery focuses on the delivery of an operational product with each increment. Early increments are stripped-down versions of the final product (Figure 18). This approach

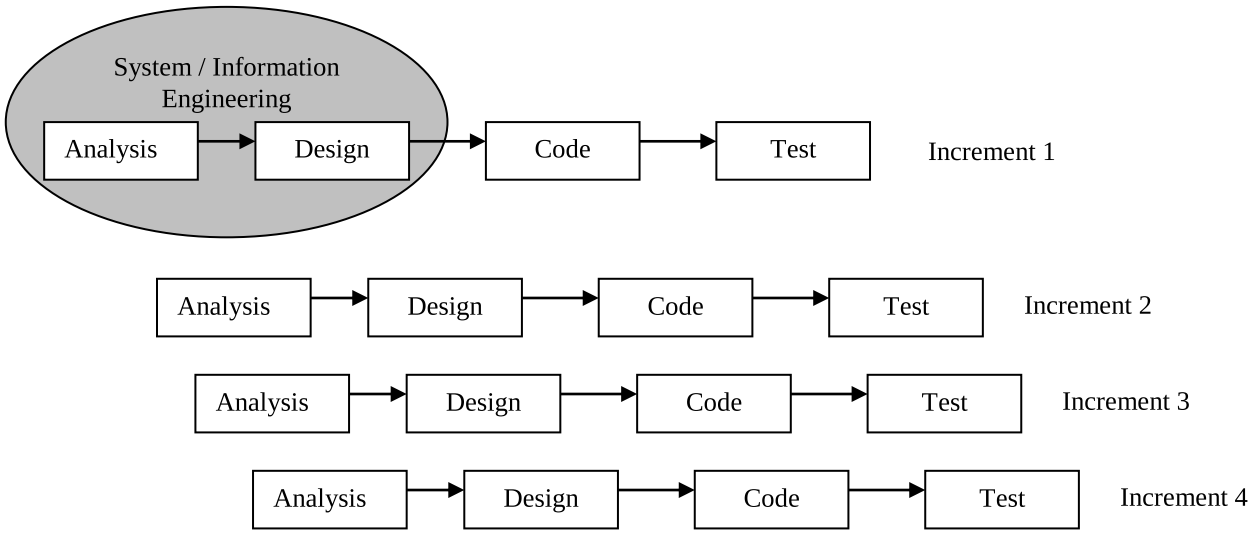


Figure 19 Incremental Model (McDermid & Rook 1993)

has the advantages that : -

- Less staffing is required than in a RAD project.
- Early delivery is guaranteed.
- Progress of the whole project is not delayed if one of the resources is not available for part of it.

The Spiral model

Boehm's (1988) spiral model couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. Software is developed in a series of incremental releases. During the early releases, there may be just a paper model, but the system becomes increasingly more complete. There are a number of framework activities, as shown in Figure 20(Customer communication,

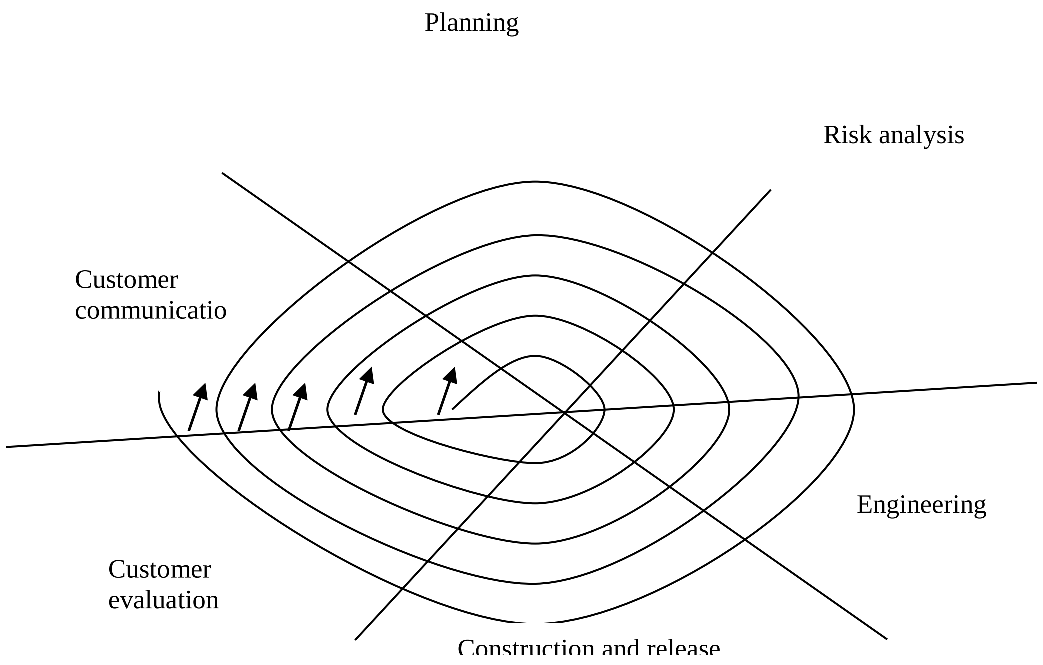


Figure 20 Boehm's (1988) Spiral model

Planning, Risk analysis, Engineering, Construction and release, Customer evaluation). Unlike any of the other models, this model keeps revisiting the system throughout its lifetime.

The Component assembly model

This incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software (Nierstrasz 1992). However, it composes applications from pre-packaged software components. The "construction & release" activity in Boehm's model is replaced by an "Engineering construction and release" activity (Figure 21).

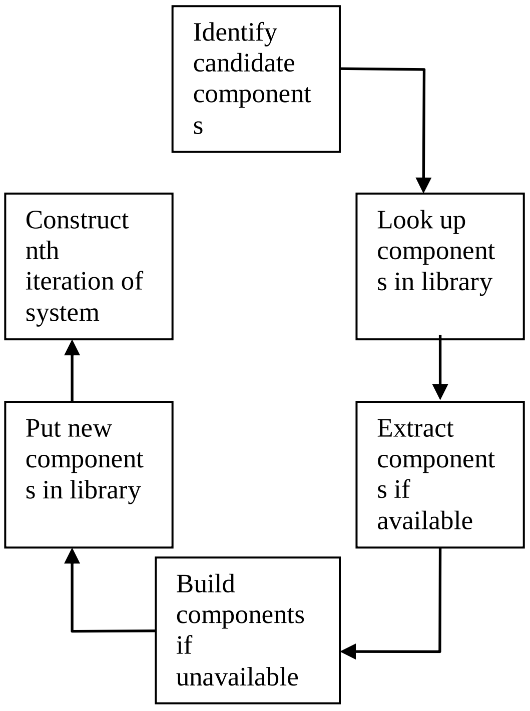

Identify candidate component s
Look up component s in library
Extract component s if available
Build components if unavailable
Put new component s in library
Construct nth iteration of system

Figure 21 Component Assembly model – engineering construction and release activity (Nierstratz 1992)

Methodologies

The process model dictates which methodologies are suitable. Methodologies are based on a process model and contain a set of techniques and a framework that will force the developer to follow the methodology. Parkinson (1991) categorises methodologies into three. The first category includes those that are visionary, providing new ideas of ways to tackle the development, but lacking practical guidance on how to do it - e.g. Information Engineering (Davids 1992). The second category consists of those that provide a host of techniques that can be used, but are weak on providing the framework for using them in the correct order and tracing requirements from one stage to the next - e.g. UML (Fowler & Scott 1997). The third category consists of those methodologies that provide a rigorous and rigid framework, which guides the developer through the stages and techniques, ensuring that all necessary tasks are carried out before advancing to the next stage – e.g. SSADM (Downs *et al.* 1992).

In the opinion of the author, each of these categories has its advantages and disadvantages. The first provides great ideas, but requires the developer to develop or adapt techniques to suit these ideas. Few commercial application developers are in a position to do this. The second allows for slippage of quality assurance mechanisms. Those mechanisms are not provided, so in order for an organization to use this methodology and maintain a high level of quality, the framework needs to be developed in-house. This option also requires great dedication. The final option looks perfect – the techniques are provided, as is a framework for carrying them out. While this is good, the methodology must be adaptable, so that the organization can ensure that all steps and stages add value to each and every development process.

To assess a methodology for a system, Parkinson (1991) suggests that the assessor needs to know:

1. How IS is currently organised and the strengths and weaknesses of the organization.

2. The skills of developers, project managers and departmental managers.

3. The opportunities and threats that face the IS function.

4. The portfolio of current development and maintenance activity.

5. The problems and issues that are likely to impede a successful migration to an automated development environment.

CASE tools

Computer Aided Software Engineering tools are available throughout the spectrum of software requirements. Within the scope of this dissertation, only tools that implement a methodology are addressed. Many of these tools are sold along with a commercial methodology, such as Arthur Andersen's Foundation/ (Flaatten 1989), or IEF (Gane 1990). These tools are generally large pieces of software that rigorously follow the methodology for which they are bought. Many of them will see a project from the planning stage right through to implementation.

The second category of methodologies is served by many CASE tools, mostly workbench in nature. These include Select Enterprise, which implements a commercial methodology called the Select Perspective (Frost 1995), which is close to the more widely known object-oriented methodology OMT (Rumbaugh *et al* 1991) or UML (Fowler & Scott 1997). Another toolset in this range is System Architect (Popkin Software 1996) which provides a set of techniques and claims to be compatible with several methodologies. While these tools do some checking, they do not provide full quality assurance automatically.

A thorough review of CASE tools is undertaken by Ovum (1998).

In the opinion of the author, CASE tools can greatly enhance and speed up the development process, but only where the tools used are adding value to the process. For a CASE tool to add value to the process it must automate some aspect of it that needs to be done to enhance the quality of the end product. The CASE tool, as a system in its own right, should follow system suitability criteria as laid out in Chapter 3.

Conclusion

There are a variety of process models, each of which can be used successfully. Once a process model has been used to develop a system, documentation style, organization and structure should either remain in the format of that process model, or all be converted to a different process model. This is particularly important where automated tools are used. High design quality is necessary to fulfil Kitchenham and Pfleeger’s (1996) manufacturing view and user view of software quality.

5.2.3 Quality of Change Management

In order to achieve a quality product, an organization must be dedicated to the production and maintenance of quality products. The software process involves more than following coding standards or system development methodologies. In order to maintain software quality, there needs to be continuity, not just through the development lifecycle, but also throughout the lifetime of the system and between systems. Organizations approach the process in different ways, but their capability at producing and maintaining quality software can be optimised. There are models for software process assessment (SPA) and software process improvement (SPI). These models are designed to allow an organization to assess its maturity, and give guidelines for improvement. Among them are the Software Engineering Institute's Capability Maturity Model (CMM) (Humphrey 1989), SPICE (SPICE consortium 1994), BOOTSTRAP (Haasse *et al.* 1994) and AMI (Applications of Metrics in Industry) as described by (Debou *et al.* 1995). Although these models have their critics, they do offer a framework for process improvement, enabling the organization to improve quality of software and to incorporate change without compromising previously high quality systems.

Capability Maturity Model (CMM)

This is a framework of software development and managerial processes originally put forward by Humphrey (1989), with a revised version being published by Paulk (1993) for the Software Engineering Institute (SEI). Companies who implement the CMM want to improve their ability to meet cost, schedule and product functionality goals. There are five levels of maturity, the fifth being the best. Key process areas (KPA) are associated with each of the maturity levels. Each key process area has the following characteristics:

- Goals
- Commitments
- Abilities
- Activities
- Methods for monitoring implementation
- Methods for verifying implementation

Level 1 – The initial level

Few, if any, organised processes exist. Each developer utilises his / her own chosen methods or techniques. Software quality depends on the capabilities of specific individuals in an organization.

Level 2 – The repeatable level

A software development organization must implement basic project management practices, including metrics for estimating size of software to be produced (e.g. Function Point Analysis) and the resources required to execute the project, and tracking procedures against the metrics. Also, software configuration management and quality assurance practices should be in place, the capability to effectively manage the requirements definition process and the capability to manage subcontractors (if applicable). There is still a heavy dependence on individuals. During times of stress, this level reverts to level 1. For an organization to have reached level 2, the key process areas that should be in place are: -

- Software configuration management
- Software quality assurance
- Software subcontract management
- Software project tracking and oversight
- Software project planning
- Requirements management

Level 3 – The defined level

The organization has defined and established the software development and maintenance practices specific to the types of applications they produce. Standards and procedures to codify these practices are implemented consistently. Training is provided and peer reviews are carried out to evaluate product quality. Integrated project management exists. The emphasis is on product quality. For an organization to have reached level 3, the required key process areas are: -

- Peer reviews
- Inter-group co-ordination
- Software product engineering
- Integrated software management
- Training program
- Organization process definition
- Organization process focus

Level 4 – The Managed Level

The emphasis is on process quality. The organization focuses on establishing a set of process measures and uses them to initiate corrective actions. Once these have been established, the organization is ready to use them to implement continuous process improvement. The key process areas for level 4 are: -

- Software quality management
- Quantitative process management

Level 5 – The Optimized level

Process quality measures are used to improve existing processes and to evaluate candidate new processes. They are also used as the basis of efficacy of introducing new technologies into the organization. The key process areas for level 5:

- Process change management
- Technology change management
- Defect prevention

An organization can see what areas need to be improved by checking their procedures and procedure development techniques against this standard. The CMM is extensively used in industry for:

- Identifying strengths and weaknesses of a software development organization, i.e. benchmarking to industry's current practices.
- Supporting process improvement initiatives.
- Helping software procurers to evaluate the capability of contractors (Debou *et al.* 1995).

SPICE

This is a set of standards that has been put forward by the International Standards Office. This set of ISO standards have been derived by the SPICE project (SPICE consortium 1994), based on the SEI process maturity framework or CMM. The objective of SPICE (Software Process Improvement and Capability determination) is to provide a common approach and framework for assessment and improvement.

The SPICE components:

- Introduction and concepts within SPICE
- Activities such as system design and requirements specification which are core to software engineering

- Overall framework used when conducting a SPICE assessment. It sets out the processes whereby a company is assessed and a rating developed which reflects the capability of the company
- How to conduct process assessments in any organization.
- Elements required to develop an assessment instrument to assist the assessor
- Qualifications, backgrounds and training required for assessors
- Format of output of assessment so that it can be used for a process improvement program.
- How the output can be used to determine the organizations process capability
- Vocabulary of terms.

SPICE provides a continuous scale of capability measurement:

Initial or not performed – 0. Software anarchy. No standards or procedures for activities such as system design or requirements specification are in place.

Performed or performed informally – 1. System design, integration and testing are performed, but not planned and monitored. Dependent on individuals.

Managed or Planned-and-tracked – 2. Projects are monitored and verified against expected delivery time and correctness. Reviews are scheduled and corrective action taken when problems arise. Configuration control is in use.

Defined or Well-defined – 3. Well-defined processes are in place and a series of process templates are developed, defining how each process is carried out in terms of staff qualifications, actions to be taken, documentation to be processed, documentation to be produced and verification products to be generated. At this level a company is capable of monitoring data which can be used for process improvement, e.g. data on the number of defects which slip through the system and acceptance testing.

Measured or Quantitatively Controlled – 4. Developer can specify measurable goals such as ‘only 5% of reworking during system testing will be due to errors which should have been trapped by previous activities’. Developer can predict process performance. He/she should be able to predict the level of errors generated when a particular design notation is used or a particular tool is not used.

Optimising or Continuously Improving - 5. Developer can quantify the effectiveness of processes and can carry out process improvement programmes based on the modification of existing processes.

The SPICE organization (http://www-sqi.gu.edu.au/spice/suite_intro.html) claim that their framework for process assessment:

a) Facilitates self-assessment.

b) Takes account of the context of the process being assessed.

c) Produces a process rating profile rather than a pass/fail result.

d) Addresses the adequacy of practices relative to the process purpose and

e) is appropriate across all application domains.

BOOTSTRAP

The BOOTSTRAP project was performed within the frame of the ESPRIT programme (Haasse *et al.* 1994) to develop a method for software process assessment, quantitative measurement and improvement. It uses the SEI's process assessment method as a basis and adapts it to the needs of the European software industry to include ISO 9000-3 attributes and the European Space Agency's (ESA) PSS-05 software engineering standards.

The BOOTSTRAP methodology includes:

1. A diagnosis of the software development environment – organization, methodology and support tools.

2. The development of an action plan, which lays out the steps necessary for improving quality, productivity and timeliness through an evolution to a higher maturity level.

The diagnosis is done through interviews of senior management, quality assurance personnel and software developers and maintainers. The investigation covers three main areas:

1. Organization internal structure and application domains, corporate quality assurance structure and resource management.
2. Methodology and engineering know-how.
3. Technology and technology transfer.

The AMI approach

AMI (Application of Metrics in Industry) is described (Debou *et al.* 1995) as having twelve steps, grouped by three in activities:

Assess

1. Assess weaknesses and critical parts of the software development process.
2. Define primary goals for metrication.
3. Validate goals against the assessment conclusions.

Analyze

4. Build a goal-tree (translation of primary goals into sub-goals and metrics).
5. Verify the consistency of the tree.
6. Derive metrics for leaf goals.

Metricate

7. Write the measurement plan, which is the reference document for collection and analysis of data and for ease of tracing of these tasks.
8. Collect the data.
9. Verify the data.

Improve

10. Present the measurement data using graphics
11. Relate data to goals
12. Determine whether goals are fulfilled.

Criticisms of Software Process Assessment and Improvement models.

Pfleeger *et al.* (1997) discuss 'highly theoretical results are never tested empirically, new metrics are defined but never used and new theories are promulgated but never exercised and modified to fit reality". Their arguments against measurement efforts are that they are not useful to practitioners, who want short-term, useful results, Customers who are forced to specify their needs in an unfamiliar fashion and are unsure what the results will be. They suggest that there is a place for measurement, but only where that measurement is shown to have value. Metrics must be used while keeping the development goals in mind. In the CMM case, the goal is to improve productivity by introducing reuse. Rather than prevent movement, the model should suggest which steps to take first. Any application of software measurement should be an integral part of a general assessment or improvement program, where the measures support the goals and help to evaluate the results of the actions. To use measurement properly, we must understand the nature and goals of measurement itself.

More specifically, CMM has many critics. Bach (1994) criticises the CMM very forcefully, saying that it is applicable only to those organizations who are either very poor at managing software or very good at it. Some of the problems he sees are:-

- Lack of formal theoretical basis.
- Only vague empirical support, which is not specific to CMM.
- Reverence of process over people.
- Reverence of institutionalising processes.
- Displacement of goals from improving process to achieving a higher maturity level.

Bach (1994) maintains that personal mastery is at the centre of heroism, but that it has no place in the CMM.

Hartley (1996) points out some of the difficulties faced by organizations who are trying to improve their maturity levels:-

- The move to the next level of maturity (CMM) is very complex – the areas of quality, productivity, technical capability and maturity for the future are interrelated. No one area can be omitted when trying to understand the underlying root issues within the organization.

- Metrics programs must be used to measure where an organization is, where it is going and how to get there.
- Comparative analysis requires a common unit of measurement to be established
- Continuous process improvement is required on the development processes themselves.

5.2.4 Discussion on software quality

Despite criticisms of the controls placed on individuals through the use of process assessment and improvement models, the author believes that this is a conflict between software development as an art and software development as an engineering discipline. As the major effort required in developing a system moves from being machine-oriented to being problem-oriented, the solutions become less granular in nature, but also become more difficult to handle. Traceability is now seen as an extremely important factor (Graham 1995) in managing a system throughout its lifecycle and as tools become more sophisticated and technology life-cycles shrink, this traceability could enable easier migration or evolution of systems. However, traceability depends on consistent formatting of documentation, which is closer industrial engineering practice than art or craft. Kitchenham and Pfleeger's (1996) transcendental view of software quality reflects the same ideals as those towards which process assessment and improvement models strive.

When developing components it is important to make up a set of rules and assure that those rules are followed. When designing a system, a choice of process model should be made, and the rules and quality assurance protocols that are required for this model should be followed. The area of process change management is much more difficult, in that this is where the rules are devised and aligned to the organizational business and I.T. needs. Software Process assessment and improvement is an area that is still in development. Although there are many very good ideas and philosophies around, very few organizations are following these models successfully in practice. However, in the author's opinion, a rigorous, yet flexible model will be needed to guide organizations through the complex minefields of integrating more and more complex systems with increasingly diverse components.

5.3 Common Problems in the Area

Because of the high rate of change in requirements, code and environment problems are cited over and over again as being at the root of many legacy systems. The problems that can arise and the effects they have are outlined here.

5.3.1 Component quality problems

Software modification often leaves behind software that is difficult to understand for those other than its author. The result is software that is harder to change, less reliable when it is changed and progressively less likely to be changed (Arnold 1996).

The effects of incoherent coding on a system are outlined by Levey (1995):

- It takes too long to make superficial changes.
- It takes too long to implement programs that have been changed
- There is a need for constant changes.
- It is necessary to edit the data that is input to a program.
- The program that controls or drives the process is too large
- The program is producing a lot of temporary files.
- Detour systems are in place which pre- and post-process the data, to add functionality that is not available in the core.
- Limited understanding of the entire system.

Slee & Slovin (1997) add that the conceptual integrity of the design, documentation and implemented system rapidly degrades.

Software quality standards have changed considerably since the 1970s. Evolution of programming standards meant that each generation of programmers had a different style of programming and way of organising a program. Unless new standards were retrospectively enforced, a program could end up with different standards being used in different parts of the program. The advent of object-oriented programming reinforced the ideas of giving modules a well-defined structure and interface, by making them operations that could be done by an object. However, Casais (1998) addresses the fact that the number of object-oriented systems that are becoming legacy is growing. This is partly because of the weaknesses in analysis and design methods that were available until recently. These methods were geared towards single applications instead of families. The situation has been further complicated by the use of automatic code generators, which often produce very poor quality code, which is difficult to find, let alone understand.

Sneed (1995) points out that in a lot of cases, the programs in legacy systems are too large and complex for structural enhancement. This can result from code alterations being made to allow for a single exceptional occurrence, which is obsolete, but has never been removed. Many of the conditions that are tested in the code never apply and some conditions that are imposed during an amendment can prevent code that was there previously from being accessible.

Slee & Slovin (1997) mention some of the problems with current coding practices:

- Methods are personal and not standardised
- Code is written in idiosyncratic ways by employees who must be retained to update their personal handiwork
- A closed-shop approach makes formal peer review ineffective.
- Fix-on-fix errors occur, where an amended piece of code is again amended.
- No record is kept of changes that are made.

Levey (1992) describes how a legacy system becomes inflexible. Most systems are originally written in a clear, legible and flexible way. However, even using the most up-to-date and open systems that are available, a system can become unwieldy and incomprehensible in a reasonably short period of time. Such problems occur when systems are changed without taking into account the initial style, organization and functionality of the system. For example, if a system is written using unconditional branch instructions such as GO TO’s and a later maintainer decides to use controlled looping structures, without eliminating all GO TO’s from the code, this will lead to inconsistencies in style and will probably cause structural problems. If a program’s organization contains input, processing and output sections and a maintainer, when making a quick change, puts processing code into the output section, the organizational structure of the code is lost. If a maintainer wishes to change the functionality of a system and adds code which makes current functionality obsolete, without removing the code for the current functionality, then the system becomes less structured and more incomprehensible (Levey 1995).

While many of these difficulties could have been avoided by rigid adherence to quality assurance methods, these methods require a rigour that is often absent. While, in some cases, mechanisms are present to ensure quality of design, quality of conformance may be low.

5.3.2 Design quality problems

Systems developed in the past that now communicate with several other tasks suffer from a problem similar to spaghetti code - by spaghetti integration (Slee & Slovin 1997). These problems prevent reuse and decrease asset value, scalability and maintainability. The system becomes resistant to change. A system that is resistant to change cannot adapt to new requirements. If a change is made, it cannot be easily tested. This brings the reliability of the system into disrepute.

As systems became more complex, the need for a higher level of organization of code became apparent. Instead of producing programs of tens of thousands of lines of code, programs were broken down into modules. Systems were designed from the top down, using a variety of systems development methodologies. Design problems came about in much the same way as code problems. Most of these systems were originally carefully designed according to the process model of the day. A significant problem is the extent of the paradigm shift that has taken place over the last ten or fifteen years. A lot of the older methodologies were either entirely manual, or were implemented using tedious and low-value tools, which exacted too high a price in terms of the time spent keeping documentation up-to-date and consistent. Because of this, a shift of paradigm meant either a loss of, or a discontinuity in the maintenance of documentation relating to design.

5.3.3 Change Management problems

As shown in the Capability Maturity model, if robust procedures are not in place and followed, change can cause chaos. There is no consistency between previous procedures and current procedures. There may not even be written or taught procedures, or any communication that procedures have changed. In this case, regardless of the quality of previous design or code quality, a system can degrade very rapidly. This situation could arise if a system was developed by one organization or team and maintained or migrated by another.

It is not easy to attain high levels of quality assurance. It requires attention to detail and careful planning. In order to implement a good quality assurance program at all levels, a high amount of training and reviewing is required. At present, the practice of formalised software process assessment and improvement is done on a relatively small scale.

5.4 Effects of Problems in the Area

Software lies at the heart of every system. The quality of the software, regardless of its age, can make the task of moving from one platform to another much easier. Even if the system is to be abandoned, the exact task done by the software is easy to recover where the software quality software is good.

Table 7 cross-references the Legacy Effect Determination Framework with possible causes relating to the software quality dimension. Each causal criterion has a row in the table. Within each row, any column representing an effect of a lack of quality in this causal criterion is marked with an “X”.

The asset value of the system is broken down into *mission criticality* and *reliability*. The criticality of a system to the organization’s mission is dependent on its current functionality. As such, it may be indirectly affected by, for example, process change management, but this is not inevitable. The reliability of the system, however, is highly susceptible to improper coding or maintenance practices and is affected by software quality at every level.

Legacy Effects / Software Quality	**Asset value**		**Ease of operation**		**Ease of maintenance**				**Ease of migration / evolution**	
	Mission criticality	*Reliability*	*User satisfaction*	*Ease of testing and auditing*	*Cost of maintenance and resistance to it*	*Availability of maintenance resources*	*Program size and complexity*	*Dependence on individuals*	*Ease of use of new technology*	*Scalability*
Quality of Change Management		X		X	X	X	X	X	X	
Design quality		X		X	X		X	X		X
Component quality		X		X	X			X	X	X

Table 7 Effects of poor quality software

User satisfaction, like mission criticality, can change over time if process change is not properly managed, but is not directly linked to the quality of change management, or to the quality of the code or design. *Ease of testing and auditing* is directly related to the quality, logic and clarity of the code and to the ease of use of the design. Process change also effects both auditing and testing, in that most system upgrades require a comparison of before and after results. This is only possible when the origin of those results can be traced from the old system to the upgraded system.

Maintenance costs are very closely related to the quality of the software. Software maintenance involves monitoring and altering the code in components, altering the design to allow components interact in a different way, or upgrading part or all of a component or set of components while maintaining system integrity and traceability. This is only possible where code, design and process can be clearly understood and competently changed. If this is not the case, maintenance will take longer and be a less enjoyable task, thereby increasing the *cost of maintenance and resistance to it*.

Availability of maintenance resources can be affected by the quality of change management. If process change is mismanaged, leaving systems between paradigms and without clear and consistent documentation, then the system will require a hybrid of skills and tools to maintain or fix. While this could also apply to the design quality and the component quality, these factors are more likely to effect maintenance costs. *The size and complexity of individual programs* will be affected by the quality of the existing code. As code becomes less understandable, more changes are forced into the program, with rigorous selection criteria being imposed on statements, to ensure that they are carried out only when appropriate. In a very large and unstructured program, it is not always possible to see what conditions have already been eliminated when processing reaches a certain point. The software design can also cause problems in this area, where a component has unclear functionality or is too large to be contained within a single program.

If the quality of software is poor or done in a hybrid fashion, then it is likely that there are one or two individuals who understand what is going on. These individuals become critical to the well-being of the system and it is very difficult for anyone else to take over from them. The *dependence on individuals* is directly related to software quality.

The adaptability of a system to new technology depends on how ingrained the current technology is in the software. This is usually at code level. However, the change management process will also affect the ease of adaptation to new technology, in that a good software process management system will anticipate and embrace new technology. Although the design of the system could *affect the use of new technology,* it is not likely to be as critical.

System scalability is more dependent on the platform and organizational environment than on the software, but it could be affected by the ease of use of new technology.

5.5 Conclusion

The Software Quality dimension of legacy status is an important one. The quality of software may not initially impact on the use of a system. There are examples of systems containing very poor quality software that have endured and are being used to great benefit for many years. However, once change is required, the quality of software is of paramount importance. When legacy effects that can be attributed to poor quality software begin to appear, this dimension needs to be addressed. However, the heart of the application lies in the application software. Poor quality software is a major limiting factor when migrating legacy systems. This poor quality can result from changes in coding standards, design models or process management.

Chapter 6 Frameworks used in assessing systems

The purpose of this dissertation is to allow management to assess their current or proposed systems for existing or potential legacy status. With a view to providing such a facility, three frameworks are proposed.

The aim of this chapter is to bring together the results of analysis in previous chapters to define two new frameworks (the Causal Criteria Framework and the Legacy Status Cause / Effect Framework) and to propose a set of assessment techniques based on these and the Legacy Effect Determination Framework already defined in Section 2.2.7.

In Section 6.1 the Legacy Effect Determination Framework is repeated, for the convenience of the reader. In Section 6.2 the newly formulated Causal Criteria Framework is presented. In Section 6.3 the Legacy Status Cause / Effect Framework is presented as a combination of the previous two. Section 6.4 describes how the frameworks can be applied in different scenarios using the LACE (Legacy Assessment through Cause and Effect) techniques and Section 6.5 discusses the usefulness of the frameworks.

6.1 Legacy Effect Determination Framework

The first of these has already been presented in Chapter 2. It is the Legacy Effect Determination Framework.

Management can use table 18 to document the effects that are occurring in an existing system that is causing concern. With the help of staff who use or maintain this system, the effect can be assessed as being present “P”, absent “A” or undetermined “U” and marked in to the table. The observation of effects may be relatively subjective, but it is done by staff who are experienced in the use of the system and what needs the system is trying to serve. These are the people who are in the best position to assess its asset value, ease of operation, maintenance and migration /

evolution. The legacy effect determination framework can be filled out partly or in full. The purpose of it is to start an investigation, rather than to provide definitive answers. To fill out the framework, the manager, in conjunction with experienced user staff and I.T. staff places a "P" in the "Present, Absent or Undetermined" column if the effect is present, an "A" if it is absent and a "U" if it is undetermined.

Legacy Effect Determination Framework		
	Effect	**P**resent **A**bsent or **U**ndetermined
Asset value	Mission criticality	
	Reliability	
Ease of operation	User satisfaction	
	Ease of testing and auditing	
Ease of maintenance	Cost of maintenance and resistance to it	
	Availability of maintenance resources	
	Program size and complexity	
	Dependence on individuals	
Ease of migration / evolution	Ease of use of new technology	
	Scalability	

Table 8 Legacy effect determination framework

6.2 Causal Criteria Framework

The second framework being proposed is the legacy Causal Criteria Framework. This framework takes the legacy causal criteria identified in section 2.3 and provides a column where management can mark in whether their system is enabled "E" or inhibited "I" in the causal area. In order to do this, a thorough investigation of the system must be undertaken. To investigate a system for system suitability, a combination of the practices discussed in chapter 3 can be used. Chapter 4 advises on the assessment of suitability of the underlying platform and chapter 5 advises on the assessment of software quality criteria. While a criterion is under assessment, "C" can be marked into the column, to show that it is under consideration. The purpose of this framework is to show where weaknesses lie in the system.

Legacy Causal Criteria Framework		
Evaluation criteria		**E**nabler, **I**nhibitor, **C**onsideration
System Suitability	System suitability to business process	
	Business process to organizational mission	
	System technology to organizational environment	
Underlying Platform suitability	Hardware suitability	
	Operating System suitability	
	Network suitability	
	Development environment suitability	
	Data management suitability	
Software Quality	Quality of change management	
	Quality of static design of current system	
	Quality of software written into components	

Table 9 Legacy Causal Criteria Framework

6.3 The Legacy Status Cause / Effect Framework

The third framework is the Legacy Status Cause / Effect Framework. It cross-references causal criteria with legacy effects exhibited as a result of them being inhibited. This is a combination of the tables produced at the ends of chapters 3, 4 and 5. There is a row for each of the causal criteria and a column for each of the

Causal Criteria	**Legacy Effects**	**Asset value**		**Ease of operation**		**Ease of maintenance**				**Ease of migration / evolution**	
		Mission criticality	*Reliability*	*User satisfaction*	*Ease of testing and auditing*	*Cost of maintenance and resistance to it*	*Availability of maintenance resources*	*Program size and complexity*	*Dependence on individuals*	*Ease of use of new technology*	*Scalability*
System suitability	*System suitability to business process*	X		X	X						
	Business process to organizational mission	X		X							
	System technology to organizational environment	X		X	X	X	X	X	X		
Underlying Platform Suitability	*Hardware*		X			X	X	X		X	X
	Operating System				X	X	X	X	X	X	X
	Network	X	X	X	X	X	X	X	X	X	X
	Development environment		X	X	X	X	X		X	X	X
	Data management		X		X	X	X			X	X
Software Quality	*Quality of Change Management*		X		X	X	X	X	X	X	
	Design Quality		X		X	X			X		X
	Component Quality		X		X	X		X	X	X	X

Table 10 Legacy status cause / effect framework

legacy effects. In the row for a causal criterion, every effect that it causes has its column marked with an “X”. This allows management to see what effects could be caused by a weakness in a causal criterion. Therefore, if there is an “I” in against this causal criterion in the *Causal Criteria Framework*, then the corresponding row in the *Legacy Status Cause / Effect Framework* will show the effects that may result.

Similarly, if the *Legacy Effect Determination Framework* has been filled out for an existing system and some of the effects have been marked as "P" for present, then the corresponding columns in the *Legacy Status Cause /Effect Framework* will indicate possible root causes that are marked "X".

6.4 How to Use the Frameworks – the LACE Techniques

Figure 3 (Section 2.4) shows how the strategy that is being implemented in a current system or may be adopted in a new system can result in legacy effects. Figure 22 is an adaptation, which shows how effects can be used to determine causal criteria and indicate which part of a system strategy may need addressing.

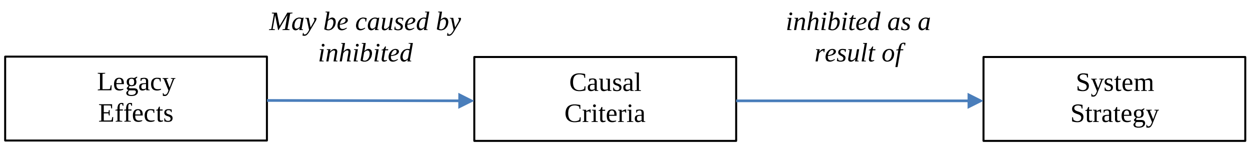


Figure 22 Relationship between Legacy Effects and System Strategy

The combined frameworks can be used for Legacy Assessment through Cause and Effect (LACE):

a) Assess an existing system through the effects it exhibits.

b) Assess an existing system for legacy causal criteria.

c) Do a preliminary assessment of a solution strategy in a preliminary fashion, based on its components.

d) Assess a solution system for potential legacy status.

6.4.1 Assessing an existing system through effects

If the framework is being used to check the legacy status of an existing system, then the suspicious effects must be mapped onto the legacy effect determination framework (

Table 8). A combined IT and business user team who are expert in the use and

Legacy Effect Determination Framework		
	Effect	**P**resent **A**bsent or **U**ndetermined
Asset value	Mission criticality	
	Reliability	
Ease of operation	User satisfaction	
	Ease of testing and auditing	
Ease of maintenance	Cost of maintenance and resistance to it	
	Availability of maintenance resources	
	Program size and complexity	
	Dependence on individuals	
Ease of migration / evolution	Ease of use of new technology	
	Scalability	

maintenance of the system assess the system for the listed effects. If the effect is present, it can be marked with a "P". If it is absent, it is marked with an "A". If the team do not know whether or not the effect is exhibiting, it is marked with a "U" for undetermined. The "U" indicates that further investigation must be undertaken, to attain a result of "A" (Absent) or "P" (Present). This can be done, but it is not always necessary, as this assessment is preliminary and merely indicates a possible need for further investigation.

When all of the effects been assessed, then those that are present "P" can be cross-referenced to the overall *Legacy Status Cause / Effect Framework*. The column for the effect that is present will have an "X" in any row that corresponds to possible underlying causes of the effect.

For example, if a system has an effect that there is poor availability of maintenance resources, then it the *Legacy Effect Determination Framework* will have a "P" against this effect. The corresponding column in the *Legacy Status Cause / Effect Framework* has "X" in rows corresponding to *suitability of the system technology to the organizational environment, suitability of the underlying platform (hardware, operating system, network, development environment* and *data management system).* This indicates that the system is weak in one or all of those areas.

The purpose of this exercise is to enable management to assess legacy status in systems, thereby providing the information needed to address that legacy status. In order to move on from here, it is necessary to do an assessment of the system for legacy causal criteria.

6.4.2 Assessing an existing system for legacy causes

This is the most definitive assessment proposed for the system. It involves the use of the *Legacy Causal Criteria Framework*, by using the definitions of those causal criteria and the practices available to ensure high quality in chapters 3 through 5. The criterion for the system is assessed in line with the advice given and then the corresponding row in the framework is marked with an "E" if the criterion is enabled and an "I" if it is inhibited. The values relating to some of the criteria will be known earlier than others. While a criterion has not had its value decided, the framework row is marked with a "C" for under consideration. Investigation should continue until the evaluation column contains only "I" or "E".

When the framework has been fully filled out, any criterion that is marked with an "I" should be looked up in the *legacy status cause / effect framework*. The row corresponding to the causal criterion will have an "X" in any column where it can be a cause for this effect. This indicates to management what the risks are if the causal criterion's inhibiting state is left unattended.

6.4.3 Assessing a prospective system in a preliminary fashion

When management are looking at the vast array of solutions that are proposed for legacy systems, they need a method for cutting back the options. Chapter 7 below shows the components that may be present in a solution strategy. Table 20 maps strategy components against causal criteria. If a solution is in the preliminary stages of evaluation, then its strategic components can be identified by consultation with the provider. If a strategic component is present in the solution, then the corresponding values in this column in table 17 can be copied into the column in the *Legacy Causal Criteria Framework*. This is done for each of the component present in the solution strategy, giving a partially filled *Causal Criteria Framework*. Any criterion that is marked with an "I" should be looked up in the *Legacy Status Cause / Effect Framework*. The row corresponding to the causal criterion will have an "X" in any column where it can be a cause for this effect. This indicates to management what effects are risked if this solution's strategy is a genuine inhibitor for that causal criterion. It must be stressed that this is a very general assessment and therefore lacks accuracy.

6.4.4 Assessing a solution system for potential legacy status

Prospective systems should be planned based on the underlying legacy status in the current system. In some cases, the changes undertaken may mean that criteria that show an "E" in the *Causal Criteria Framework* for the existing system may be adversely affected in the proposed solution. If a system is new, not a replacement system, then this assessment is also necessary, so that possible future problems can be anticipated and their risk weighed. When a system is proposed, management should fill out the legacy *Causal Criteria Framework* as shown in section 6.4.1. If a criterion has been evaluated as an inhibitor, then the potential effects of this, when the system is installed, can be seen from the *Legacy Status Cause / Effect Framework*. If the criterion is an enabler, this means that it does not contribute to a legacy status in the system, however the effects associated with it may be incurred by another cause.

6.5 Usefulness of the frameworks

These frameworks allows management to compare and contrast systems that are considered legacy with their proposed replacements, thereby giving them a more robust foundation for their decision on which solutions to take. Positive results in an existing system should not be abandoned. If the current system shows high quality in a causal criterion, then it may be possible to retain this component. When a new system is being evaluated, it is necessary to ensure that it does not introduce an unacceptable number of inhibiting criteria.

Chapter 7 Components of migration strategies

7.1 Introduction

The aim of this chapter is to examine components of strategies that can be used to resolve legacy problems and evaluate their impact on legacy causal criteria.

Chapters 3, 4 and 5 have thoroughly described the criteria that cause legacy status, cross-referencing each causal factor with its effects in the Legacy Effect Determination Framework. Chapter 6 has described how all three frameworks can be applied. In this chapter, the strategies that can be used to resolve legacy problems are examined. Their use is assessed in terms of their solution to each of the causal criteria. In order to do this causal criteria are listed as evaluation criteria.

The number of approaches and products that promise a solution to legacy problems is vast. Because of this, the solutions are categorised using the 4R portfolio matrix devised by Slee and Slovin (1997), based on Sneed's (1995) evaluation of legacy systems (see Figure 1). Solutions may come exclusively from one of these quadrants, or may be a combination of one or more of them.

Section 7.1 describes the objectives towards which organizations strive when they approach migration of or from legacy systems. It then describes the quadrants of Slee and Slovin's (1997) 4R portfolio matrix in detail and breaks them down further. Section 7.2 outlines the components that make up a strategy for legacy migration. Section 7.3 to 7.10 describes each component and the options that can be chosen for them, mapping each strategy component against the causal criteria (Table 12 to Table 19). Section 7.11 shows the overall mapping of component strategies as enablers or inhibitors of the causal criteria (Table 20)

7.1.1 Objectives of transition

Mack (1997) gives a comprehensive overview of the issues of transition, and concludes that transitions is both a political and a technical process, the aim of it being to "balance the ability of distributed computing to improve speed and service with the stability and durability of known processes and technologies". He also points out that "the single most important factor contributing to success will be the ability to plan and conduct an IT transition that is technically sound and politically astute". The factors affecting choice will be new technology, distribution of budget, skills and IT decision-making to business units and users. The risks will be technological, organizational and political indecisiveness, costing time and resources. Mack's (1997) objectives are to choose a strategy that will: -

- Achieve the aim.
- Maximise the use potential of the factors.
- Minimise the risks.

7.1.2 Available transition options

New strategies, tools and approaches are emerging to help I.S. reengineer, recondition, coexist with or extract value from existing applications. Slee & Slovin (1997)'s 4R Portfolio Assessment Matrix (Table 14) has four quadrants. The four R's are Retire, Reassess, Redevelop and Renew. These are described in the following sections. -

Low business value Low technology condition **RETIRE**	Low business value High technology condition **REASSESS**
High business value Low technology condition **REDEVELOP**	High business value High technology condition **RENEW**

Table 11 The 4R Portfolio Assessment Matrix(Slee & Slovin 1997)

7.1.3 Retire

Slee & Slovin (1997) suggest that retirement may be gradual, and that a wrapper may be used during the phasing out of this system. However, if its business value is very low, in the author's opinion it seems hardly worth the effort. Reassessment of business needs in relation to organizational and I.T. strategy is a more worthwhile course of action.

7.1.4 Reassess

Business value is low, even though technological condition is high. This system is no longer required for the business. Slee & Slovin (1997) suggest that the technology could be moved to more critical applications, but the system itself should be retired. In this case, once again, the author believes that the business needs of the organization need to be examined to develop a strategy in relation to IT, rather than blindly moving technology about to give the impression of saving costs.

7.1.5 Redevelop

This is for applications that are still mission critical, but technological advances have outstripped them. Slee & Slovin (1997) suggest either extracting business rules (reengineering), replacing them subject to a suitable replacement being found (replace) or developing a gradual transition strategy (transition).

7.1.6 Renew

These systems are in high technological condition and have high business value to the organization. However, it has been judged that the system is a legacy system. Slee & Slovin (1997) suggest that moves should be made to address the cause of these problems. This author presents a framework that allows management to accurately assess the areas in which the system is gaining legacy status (see Chapter 6), so that appropriate adjustments can be made.

7.2 Component strategies and their effects on legacy criteria

Slee and Slovin (1997) look at the directions in which legacy systems can be taken. However, there is such a myriad of options available, it is difficult to know exactly where one ends and another begins. In order to encompass the essential components of any strategy, a summary of components that should be considered as part of a strategy follows:

1. Time-base - if change is being undertaken, it can be done in one go or it can be done iteratively or gradually (iterative approach).

2. In-house or outsource.

3. Assessment - the assessment of a legacy system depends on the size of the system, the level of understanding of the problem, the number of possible options for a solution and the extent to which the assessor is involved in offering the solution.

4. Architecture - within each application, there are possible components. These components may or may not be object-oriented, may or may not have a layered architecture and may be an assortment of packages or may be bespoke components.

5. Data Reuse - data can be reused in its native state, by wrapping it and accessing it through ODBC, it can be stored in a data warehouse or it can be migrated to a new system.

6. Code reuse - code can be wrapped by surrounding it in its native state, by vertically partitioning it and reusing some of its services or by horizontally partitioning it and replacing one or more of the layers.

7. Redeveloping the system requires recovering the requirements that the system fulfils, adapting them to suit current needs and regenerating, replacing or renewing the system.

8. Renewing the system can be done by iterative enhancement, by software restructuring and by re-hosting.

Each of the above is described below and discussed as to how it enables or inhibits the evaluation criteria for a new system. Note that as these are merely components of a solution, many of the criteria cannot be judged against the components. For this reason, where a component does not enable or inhibit the criterion, this grid-square is left blank.

7.3 Time-based Strategy

Many authors advocate an incremental approach to legacy transition (Slee & Slovin 1997, Brodie & Stonebraker 1995, Mack 1997, Beyond Software 1997, System Techniques Inc. 1995 (2), Simpson 1995) whereas others advocate a complete changeover (Wu *et al.* 1997, Bancroft *et al.* 1997). Either option will work when the entire solution is properly planned. However, there is the danger when using an iterative approach that Slee and Slovin's (1997) fear of 'spaghetti integration' may

Mapping of time-based component against legacy causal criteria			
Evaluation criteria		Iterative approach **E**nabler, **I**nhibitor	Direct implementation **E**nabler, **I**nhibitor
System suitability	System suitability to business process	E	I
	Business process to organizational mission	E	I
	System technology to organizational environment		
Underlying platform suitability	Hardware suitability		
	Network suitability	I	E
	Development environment suitability(including OS)	I	E
	Data management suitability		
Software quality	Quality of change management	I	E
	Quality of static design of current system	I	E
	Quality of software written into components		

Table 12 Causal criteria enabled and inhibited by time-based component

result, where each increment of the solution is tied on to the previously installed base, without any overall plan for the architectural result. This can cause problems regarding the *network*, *operating system* and *development environment suitability*. While the component parts of the system may work well, incompatibilities between components may enforce tight coupling, thereby inhibiting the *quality of static design of the system*. If the system is suffering in this way, then problems are inevitable when change is required. Therefore, the *quality of change management* is also inhibited. The option of an incremental approach is therefore only valid, in the opinion of this author, if it is used as part of a planned incremental approach. A non-iterative approach is more likely to favourably affect the *operating system, network* and *development environment* suitability. As the system is being designed at one time, then the *quality of static design* of the current system is enabled. These enablers and inhibitors are mapped against the causal criteria in Table 15. In turn, it will be easier to manage change in a system where the static design is good, so this enables *quality of change management.* However, it also has a downside, in that if the number of applications to be implemented is very large, the project may suffer from the same problems that can occur in the linear sequential process model (Royce 1970). These problems arise from the fact that change may have occurred during the planning phase, thereby giving the system a *System Suitability* legacy status in terms of *suitability to business process* and *suitability of business to organizational mission* before it has been used.

7.4 In-House or Outsource

The task of transition may be undertaken completely by the organization itself or part or all of the work may be outsourced. If outsourcing takes place, it is important to ensure that *System Suitability* is properly addressed. This can especially be a problem where the implementation is outsourced, but operation is in-house. Outsourcing in this instance can be an inhibitor to *System Suitability to business process* and *suitability of system technology to organizational mission*. Partial outsourcing can also cause problems with the quality of *Change Management*, where third-party consultants to introduce a system and transition to later upgrades are undertaken in-house. The use of experts in a particular implementation can mean that *the quality of the design* implemented can be initially very high. If a third party is involved in the implementation, it is also likely that configuration management will be addressed afresh, rather than being left to what was there before. This has a favourable effect on all of the *Underlying Platform Suitability* criteria, but may conflict with the *System Suitability to organizational environment* criterion, especially where the solution is a variation of a packaged solution, as in this case, the solution is technically rather than socially based. If the system is developed in-house, then it is likely that it will suit *the organizational*

Mapping of in-house / outsourcing component against legacy causal criteria

Evaluation criteria		Insourcing **E**nabler, **I**nhibitor	Outsourcing **E**nabler, **I**nhibitor
System suitability	System suitability to business process	E	I
	Business process to organizational mission		
	System technology to organizational environment	E	I
Underlying platform suitability	Hardware suitability	I	E
	Network suitability	I	E
	Development environment suitability(including OS)	I	E
	Data management suitability	I	E
Software quality	Quality of change management		I
	Quality of static design of current system		
	Quality of software written into components		

Table 13 Causal criteria enabled / inhibited by choice of in/outsourcing

environment and the *business process*. The *suitability of the development environment* is also likely to be enabled, because of the fact that the developers who will be using it, or their managers, will be involved in choosing it.

However, the knowledge of available technology may not be as deep, thereby inhibiting the *Underlying Platform Suitability* criteria.

7.5 Assessment

The assessment of a system involves checking its assets and liabilities. Several authors have put forward assessment techniques, which include the following steps:

Assess business value (Sneed 1995, Ransom *et al.* 1998, Brodie & Stonebraker 1995, Wu *et al.* 1997). Neumann (1996) and Slee & Slovin (1997) take business strategy into account. Slee and Slovin (1997) address the ability of the organization to adjust to hybrid technologies, while Neumann (1996) looks at the impact of business change and how a new system can leverage power for the organization. Ransom *et al.* (1998) also assess the external environment. Mentzas (1997) advises modelling current process threads.

Assess technological condition (Ransom *et al.* 1998). Neumann (1996) gathers legacy information. Brodie & Stonebraker (1995) and Wu *et al.* (1997) gain an understanding of the legacy system. Mentzas (1997) conceptually models the current processes and benchmarks them. Sneed (1995) prioritises systems based on their technical quality and software value.

From here, authors tend to diverge, with some deciding upon a transition strategy (Neumann 1996, Slee & Slovin 1997, SEBPC 1998), (known in Table 14 as open assessment). Others advocate a particular solution (Mentzas 1997, Mack 1997), known in Table 14 as directed assessment. Table 14 maps the assessment components against the causal criteria they enable or inhibit.

The SABA project (Ramage 1998b, 1999) proposes a general approach: to understand legacy systems and develop approaches which help companies to make decisions about such systems. Alderson & Shah (1998) develop a method of understanding legacy systems through viewpoints and events, while Liu *et al.* (1998) discusses a model of retrieving requirements from legacy system behaviour, where no source code or documentation is available. Ramage (1999) discusses a model that contains two tools, one to model an organizational scenario and the other to model a technology scenario. Ganti and Brayman (1995) propose guidelines for examining the business and the business processes, reengineering the business process and linking legacy information systems with these processes to determine which systems have data and business logic of value in the new target environment. Any differing processes are developed separately.

The strength of the assessment component in a migration strategy is directly proportional to the *suitability* of the solution *system to the organizational mission* and *business process*. The approach to assessment will affect the *suitability of the system technology to the organizational environment.* If assessment is undertaken with the intention of moulding the problem towards a particular solution, then this assessment is not comprehensive and may inhibit *suitability of system technology to organizational mission.* If assessment is done openly, without a particular solution in mind, this will enable the suitability of *system technology to organizational mission*. If a solution is in any way forced, this will inhibit the

Mapping of assessment component against legacy causal criteria			
Evaluation criteria		Open assessment Enabler, Inhibitor	Directed assessment **E**nabler, **I**nhibitor
System suitability	System suitability to business process	E	E
	Business process to organizational mission	E	E
	System technology to organizational environment	E	I
Underlying platform suitability	Hardware suitability		
	Operating System suitability		
	Network suitability		
	Development environment suitability		
	Data management suitability		
Softwar e quality	Quality of change management		I
	Quality of static design of current system		
	Quality of software written into components		

Table 14 Causal criteria enabled / inhibited by assessment component

management of change. The effect on other criteria depends on the particular implementation of the assessment component.

7.6 Architecture

7.6.1 Components

Within each application, there are possible components. Although it is desirable to integrate systems as much as possible, Slee and Slovin (1997) recognise that attempts at integrating applications can quite often lead to what they term as "spaghetti integration", where applications suffer from tight coupling and loose cohesion. Tight coupling results from the use of shared databases, redundant databases and interface files. Loose cohesion results from spreading functional logic throughout numerous programs and applications. Component architectures promote tight cohesion and controlled coupling, by using items such as desktop integration, software message bus and remote data access and data warehouses as building blocks to help applications to co-operate through standardised interfaces. Because of the quality of coupling and cohesion within these integrated applications, reuse and maintainability are supported, thereby increasing the value of I.T. assets, which Slee & Slovin see as being data, processing logic and business rules. However, componentisation does not automatically imply that a given component will be suitable to the business process or the organizational mission.

This author contends, therefore, that because of the potential for quality of coupling and cohesion within components, componentisation promotes *Software Component Quality*, thereby enabling high *Design Quality* (see Table 15)

7.6.2 Object orientation

These components may or may not be object-orientated. Some authors advocate an object-orientated approach (Mentzas 1997, McGibbon 1996, Casals 1998), while others do not consider it. Pancake (1995) commends orientation for its responsiveness, flexibility, agility and ability to reflect real-world structures in a model, resulting in a self-consistent, understandable universe that matches natural thought processes. The author contends that these factors should enable *System*

Suitability to both *business process* and *organizational mission*. However, Pancake (1995) also points out the flaws in the practicality of the object-orientated approach:

i) The tools and languages are not simple and effective enough yet.

ii) Although reuse is supposedly one of the selling points of object-orientation, current object orientated systems do not encapsulate any information on object reliability, performance or resource utilisation.

iii) The paradigm is so different that substantial retraining is necessary to introduce it.

iv) Models do not cater for subtle problems of inappropriate or incompletely defined interactions. If the model is flawed, the flaw may only be exposed at a late stage in testing (Pancake 1995).

This author contends that these flaws are due to the maturity of the object-orientated approach. As it becomes more popular and therefore profitable, further developments will enable these flaws to be overcome. Already, the development of modelling languages such as UML (Fowler & Scott 1997) and Open-1 (Henderson-Sellers 1996) are set to improve the quality and rigour of software design using an object orientated approach. Similarly, new software development languages are beginning to close the gap between visual languages that do not fully implement the object-orientated paradigm such as Microsoft's Visual Basic and those that are more faithful to the paradigm but have implementation drawbacks, such as Smalltalk. The subject of data storage is also currently a topic of much debate, with many high profile database producers such as Oracle, turning their attention towards object orientation. However, as some these new technologies are not yet proven, this author contends that their large-scale use to replace systems that are currently in existence, needs careful investigation. While object-orientation is now accepted as a paradigm and most software developers do have a knowledge of it, once again, the practical implementation of the techniques involved are dependent on the technology available.

For these reasons, the author accepts that these flaws may still represent an obstacle to many large organizations. Object orientation may therefore inhibit *system suitability* to *organizational environment*, *development environment suitability* and *data* *management suitability*. Also, it may inhibit *quality of static design of current system* and *of change management*.

7.6.3 Layering

The components may or may not have a layered architecture. If an application is wrapped for reuse, its intrinsic architecture may not change. If it is not already component based, it may be wrapped and treated as a component from outside the wrapper, but it will not act as a component internally (Makowski 1995, Beyond Software 1997, System Techniques Inc. 1995(1)). However, if redevelopment or replacement is taking place, the issue of layering needs to be considered.

If this layered architecture is poorly designed developed and implemented (*i*-Cube 1998), it can cause slow and costly change, with the following results:

i) There may be gaps in an application's ability to meet new organizational, geographic, or marketing directions and in its ability to exploit emerging technologies.

ii) Development may lack discipline, leading to weak, unstable and difficult to maintain applications.

iii) Connectivity tools are not equipped to meet the demands of a high-volume production environment.

iv) Applications developed for one solution are not feasible for others, causing a systems-management dilemma with multiple tools in the enterprise technology infrastructure.

These problems result from poor configuration management and change management. As such, the author contends that they are not intrinsic to a layered architecture. However, due to the complexity of configuration management in a layered architecture, the author contends that the organizational environment needs to be very focussed. Therefore, layered architecture is an inhibitor for the criteria *suitability of the system technology to organizational environment* and *quality of change management*.

Advantages of a layered architecture:

Correct implementation of open systems, however, according to I-Cube (1998) will have the following benefits:

i) Flexibility to adapt quickly to ever-changing business and technology conditions, enabling *system suitability to business process* and *organizational strategy*.

ii) Cost savings in hardware and software acquisition, maintenance and operating costs, integration of systems on disparate platforms, providing the means to transform mountains of data into useful, timely and manageable information. Significant improvement in systems scalability, management and administration by being able to introduce new technologies and tools, enabling the *Platform Suitability* criteria of *hardware* and *network suitability*.

iii) Enhanced ease of use with features such as GUIs and online seamless integration with other applications and data sources enabling *software design quality*.

7.6.4 Bespoke components

The components that are used may be an assortment of packages from a variety of software producers, a single component-based package provided by a third party (Bancroft *et al.* 1997) or may be bespoke components. As the arguments for and against bespoke components are similar to those of in / outsourcing relating to System Suitability, this argument will not be repeated here (see section 7.4), but the effects on these criteria are the same. Bespoke components are written for a specific system and so do not necessarily cause difficulties relating to *Platform Suitability* criteria (see Table 18).

The four options in architecture have been discussed and their enabling / inhibiting effect on the causal criteria is mapped in Table 18.

Mapping of architecture component against legacy causal criteria					
Evaluation criteria		Component Enabler, Inhibitor	Object orientated Enabler, Inhibitor	Layered Enabler, Inhibitor	Bespoke Enabler, Inhibitor
System suitability	System suitability to business process		E	E	E
	Business process to organizational mission		E	E	
	System technology to organizational environment		I	I	E
Underlying platform suitability	Hardware suitability			E	I
	Network suitability			E	I
	Development environment suitability(including OS)		I	E	I
	Data management suitability		I	E	I
Software quality	Quality of change management		I	I	
	Quality of static design of current system	E	I	E	
	Quality of software written into components	E			

Table 15 Causal criteria enabled / inhibited by component choices

7.7 Data Reuse

7.7.1 Data wrapping using ODBC

Data can be reused in its native state, by accessing it through ODBC (Makowski 1995, Beyond Software 1997). This middleware provides open database connectivity - there is a generally accepted ODBC (open database connectivity) standard. Most major RDBMS vendors offer software to link their databases to the Web. Primary examples are Oracle's Network Computing Architecture and Informix's Universal Web Architecture (Tucker 1997).

ODBC allows data in a current relational database to be used from elsewhere. While this allows for more users to use the data, it is more of an integration strategy than a strategy for solving legacy status. However, the data retains any redundancies and inconsistencies that may have been in it and the security of the current database may be compromised unless safeguards that are not inherent to the database are implemented in additional systems that access it. Another difficulty in extending the use of an existing database is that the demand for the data may rise, thereby compromising the efficiency of the database management system. Data that is being used in an environment for which it was not designed may need restructuring. ODBC therefore, as a solution strategy for legacy systems inhibits *the suitability of the system to the business process,* by making it slower and possibly less reliable. It may affect the *quality of the static design of the current system,* if the current system is designed for another purpose, and the data is being reused out of context. ODBC can be used to good effect in a restructured system.

7.7.2 Data warehousing

Data can be replicated in a data warehouse so that transaction-based systems can be transformed into knowledge-based systems. Data mining provides end-users with widespread access to information that is locked in the core business applications. Desktop tools currently in use, such as spreadsheets, databases and personal information managers can do this. A data warehouse contains data from several source applications, which is copied and undergoes a transition before it is put into the data warehouse. This is done by data transformation tools (Sachdeva 1995, Hill 1997)

The rules of data entry to the warehouse are held in the metamodel. The data infrastructure in the warehouse is described by the metadata (Sachdeva 1995, Breitendeder *et al.* 1996). From the data warehouse, the business user should be able to access a table of contents of the warehouse. This should have details of the original system from which the data came, the transformation sequence that have been applied to it, the access permissions available to different categories of user, the age of the data and a rough estimate of how long it would take to access certain data. The data warehouse administrator ensures that the data is up-to-date and accurate, using a version control facility, an estimate of the time taken to load the data into the warehouse and of the growth of the data. Decision Support Systems (DSS) analysts can also access the table of contents, the transformation and business rules, the data models and what operational data is available. The warehouse infrastructure should be made up from legacy systems metadata, operational client/server systems, enterprise data architecture, and metadata from legacy data mining.

The author sees the advantages and disadvantages of data warehouses as follows. They can be used as a repository of information and an indirect information conduit between old and new applications. Data from several systems can be merged to provide answers to queries that may not be satisfied from a single system. Data warehousing has enormous potential advantages in any organization, for use in executive support and decision support systems, giving rapid, easy access to operational data and diverse historical data from differing locations for planning and decision making. As a strategy for migration it has the advantage that it can be used to preserve and share information. It can help in scaling information that was hitherto available only in a confined environment. However, data warehouses are most useful for online analytical processing, using data mining techniques, not for online transaction processing. For this reason, their application in a solution is limited. Due to the large amount of data and the expected user base, online transaction processing would be prohibitively slow. As such, data warehousing allows data from a defunct system to be retained in the warehouse, but it does not provide a solution for new processing. Data warehousing can be used as part of a solution, in that it removes some of the need for integration across platforms that may not necessarily be compatible. In particular, if an organization has vastly different applications, such as process control or monitoring systems on one platform and information systems on another, a data warehouse can enable information to be merged from both on an independent platform, without compromising the suitability of platform for either set of applications. This enables the *underlying platform suitability* criteria.

However, as data warehousing does not contribute to the functional core of the new system, it does not enable or inhibit any of the other causal criteria. However it is important that managers should consider including a data warehouse for analytical processing.

Data warehousing is mapped against the causal criteria in Table 16.

7.7.3 Data migration

Data migration is where the data is taken from its current environment and transferred to a new system. Brodie & Stonebraker (1995) propose the Chicken Little methodology, where the current system is incrementally analysed and decomposed. The new target interfaces, applications and database are incrementally designed and installed and a gateway system is used between current and new systems until all increments are complete.

Wu *et al.* (1997) propose the Butterfly methodology whereby, after a new target system has been prepared, the data and all manipulations of the data from the current system are redirected to the target data schema, thereby bypassing the need for gateways.

The migration of existing data greatly enables the *system suitability to business process* and *system suitability of business process to organizational mission*, in that it maintains information from the previous system. Both of the methodologies mentioned above retain not just the information, but its behaviour. This is a partial strategy and is only relevant where the system is being migrated.

Data migration is mapped against the causal criteria in Table 16.

Mapping of choice of Data reuse against legacy causal criteria				
Evaluation criteria		ODBC **E**nabler, **I**nhibitor	Data migration **E**nabler, **I**nhibitor	Data Warehousing **E**nabler, **I**nhibitor
System suitability	System suitability to business process	I	E	
	Business process to organizational mission		E	
	System technology to organizational environment			
Underlying platform suitability	Hardware suitability			E
	Network suitability	I		E
	Development environment suitability(including OS)			E
	Data management suitability			E
Software quality	Quality of change management			
	Quality of static design of current system			
	Quality of software written into components			

Table 16 Causal criteria enabled / inhibited by data reuse

7.8 Code reuse

Code can be written and maintained in such a state that it has a value throughout time. If the problem being addressed by the code does not change and the code has been well written, in a standard and current language, these are good reasons for reusing the code.

There is more than one way to reuse code. The author categorises the various methods of reuse as follows:

6. It can be fully reused – i.e. wrapped in its native state.

7. It can be partially reused, by splitting the layers and replacing one layer of it – usually the front-end.

8. It can be partially reused, by splitting the application vertically, by functionality. This allows the existing code to offer services to other applications.

Code reuse covers the entire area of wrapping. This can vary from wrapping a piece of functionality from a process within an application, to wrapping a full process or a full application. In some cases, functionality from more than one application is wrapped together and used by another application in a single service request.

The more general term software reuse encompasses the idea of reusing designs or parts of models.

7.8.1 Application wrapping

The application can be left as it is and accessed by another path. This is similar to the use of subroutines in a third-generation language. The subroutine exists autonomously and is available to be used by any process that can call it, provide it with the correct parameters and understand the results it gives back. In the same way, a program or application can be reused. If code is being reused in its entirety, then the interfaces must either be presented directly to the other application or masked out, by a wrapper which converts input and output between calling and called applications. The calling application may be requesting a service, with a set of parameters, but the called application may expect, for example, a lengthy login, menu-picking, form-filling procedure to be completed before it invokes the service and returns the result. The result may be intended as a display on a screen or an update to stored data. The wrapper intervenes and supplies the missing information to be input to the called application and intercepts the result and redirects it to the calling application, in the required format.

The use of applications in their native state allows for the applications to remain available to existing users in an existing environment and also to new users in a different environment, through the wrapper. However, the wrapper does represent an overhead in terms of processing, with much of the I/O being duplicated and masked out. The calling procedure may involve quite complex emulation to gain access to the required functionality and is really only justifiable in the case where the functionality is highly complex and not easy to replace. The means of doing this is described by Rossak (1991) and has been implemented by Van Mulligan *et al.* (1995) in the HERMES project. Aronica and Rimmel (1996) used emulation as part of their strategy in encapsulating a legacy system.

Sometimes, the original code needs to be changed. Code can be integrated with new systems in its native state or with minimal change or automated change. K. Martin (1996) used automation to reuse a library of C++ routines in General Electric.

Application wrapping enables *suitability of system technology to organizational environment*, by allowing access to systems that are not entirely compatible with the calling environment, but inhibits all of the *software quality* criteria. Any software problems that were there before the system was wrapped remain and are propagated to new users, inhibiting the *quality of software written into components*. The system design effectively contains a black box in terms of the wrapped application, thereby inhibiting the *quality of static design* of the system. Also, as the system is a black box, *quality of change management* is inhibited.

Application wrapping is mapped against the causal criteria in Table 17.

7.8.2 Horizontal wrapping

System Techniques (1995a) describe wrapping as developing an overlay to transform character-based screens into GUI, client-based screens or to tying together multiple systems that do not normally interface to one another.

A form of horizontal reuse is where the user interface layer is replaced by a new front end, while leaving the lower layers intact. The success of this depends on the looseness of the coupling between the layers and the quality of the underlying layers. This can be done for more than one reason. It can be done to improve the look and feel of the system, making the system more acceptable to and therefore more useable for a new breed of user. Conversely, it can be done to upgrade existing users to integrate them with other applications that may be useful to them, while still retaining the functionality they need for their core skills (System Techniques 1995 (1)). It can also be done to make the system accessible to a wider audience, by supporting client/server access and by web-enabling the wrapped system.

Horizontal wrapping enables *the System Suitability to business process* in that it can present the user with a modern interface and more importantly, only request that information from the user that the user can knowledgeably provide. As the divergence between system needs and application provision often causes fields on forms to become defunct or to change in nature, a new interface can accommodate this. A more modern interface also enables *suitability of system technology to organizational environment*. As with application wrapping, horizontal wrapping does not substantially affect the underlying software and so inhibits the *Software Quality* criteria.

Horizontal wrapping is mapped against the causal criteria in Table 17.

7.8.3 Vertical wrapping

This is where certain functionality is taken from the application in isolation. The code is split into components and wrapped. The resultant wrapper may be object-oriented. However, the code in the background, while logically split, may still suffer from tight cohesion with non-related functions and loose coupling. If this is the case, the wrapper may need to filter out all of the redundant code. This can either be done by re-engineering the underlying code (De Lucia *et al.* 1997, Allen & Frost 1997) or by making the wrapper mask the processing that is not needed from the calling application. This can slow down the system considerably. In the HERMES project, Van Mulligan *et al.* (1995) vertically wrapped functions from several legacy systems and combined the result to offer it as a single service component to the calling program. As many instances of this type of wrapper are component-based and object-orientated, they are easily web-enabled.

Vertical wrapping enables the *suitability of the system to the business process* and the *suitability of the system technology to the organizational environment* as with horizontal wrapping. Likewise, it inhibits all of the *software quality* criteria.

Vertical wrapping is mapped against the causal criteria in Table 17.

Mapping of choice of code reuse against legacy causal criteria				
Evaluation criteria		Vertical Wrapping **E**nabler, **I**nhibitor	Horizontal wrapping **E**nabler, **I**nhibitor	Application wrapping **E**nabler, **I**nhibitor
System suitability	System suitability to business process	E	E	
	Business process to organizational mission			
	System technology to organizational environment	E	E	E
Underlying platform suitability	Hardware suitability			
	Operating System			
	Network suitability			
	Development environment suitability(including OS)			
	Data management suitability			
Software quality	Quality of change management	I	I	I
	Quality of static design of current system	I	I	I
	Quality of software written into components	I	I	I

Table 17 Causal criteria enabled / inhibited by code reuse

7.9 Redevelop

This is for applications that are still mission critical, but technological advances have outstripped them. Slee & Slovin (1997) suggest either extracting business rules (reengineering), replacing them subject to a suitable replacement being found (replace) or developing a gradual transition strategy (transition).

Brodie and Stonebraker (1995) propose the “Chicken Little” migration approach, which is incremental in nature. It analyses the legacy system and decomposes it. It then designs new target interfaces, applications, database and environment. Once the new environment has been installed and gateways created, incremental migration of the database, applications and interfaces can take place.

Slee and Slovin’s (1997) recommendations are to reverse engineer the current system to a design document, change the design document to meet business requirements and use forward regeneration of the code.

In the Broom method, Mentzas (1997) also models the ‘as-is’ system and then defines metrics and process measurement criteria to benchmark solutions. A variety of ‘to-be’ processes are designed, simulated and evaluated, resulting in one being chosen. The object-oriented models are then developed for the chosen solution.

Casals (1998) proposes an object-oriented approach, whereby the current system is analysed and represented by a model. Problems with this model are identified and decisions made about the software structures that could be used to address the problems to solve this design defect. The optimal transformation strategy is then selected and the transition is carried out.

There is agreement among many authors about the general structure of a redevelopment project, with this structure having the elements of understanding the current system, noting shortfalls and new requirements and finding a new solution. These steps are common to any of the software engineering process models seen earlier (Chapter 5). However, understanding the current system can prove difficult, especially in a system that has been deemed mission critical, but with low technological quality.

Recovering value from legacy assets

The assets that Slee and Slovin (1997) list as being important to recover are:-

Valuable data

Almost every new application involves a major database conversion. Most require reconditioning of the data for removal of redundancies and inconsistencies and to add new information.

Valuable processing logic

Processing logic that works can be an enormous asset. However, when this logic is difficult to adapt or understand, its usefulness is limited. This logic can either be discarded or reengineered.

Valuable business rules

The value of processing logic is all the more important when it is the only repository of business rules. When a system is manual, there are always people who know the rules of how it should work, but when it becomes automated, these rules are encoded in the processing logic. As time passes, staff who know the business rules either leave or forget how exactly the system is supposed to work. If the current business rules need to be retained, then reengineering of the process logic is essential. It may be the case that the business rules are standard, or could be standardised without any detrimental effect.

The AMBOLS project (Liu *et al.* 1998) proposes recovering requirements by analysing the behaviour of the system, thereby by-passing the need to examine code and documentation that may be absent or difficult to understand. Arnold (1989) proposes code restructuring as a means of understanding the current system. Newcomb (1995) proposes that a legacy system cataloguing facility be set up to help cope with the volume of legacy programs and provide a knowledge base for re-engineering. This facility can be used in conjunction with automated reverse engineering tools such as Refine (Markosian *et al.* 1994). Ning *et al.* (1994) propose another toolset for reverse engineering Cobol programs, called Cobol/SRE.

Mapping of redevelopment strategy against legacy causal criteria

Evaluation criteria		Enabler, Inhibitor, Consideration
System suitability	System suitability to business process	
	Business process to organizational mission	
	System technology to organizational environment	E
Underlying platform suitability	Hardware suitability	E
	Operating System suitability	E
	Network suitability	E
	Development environment suitability	E
	Data management suitability	E
Software quality	Quality of change management	E
	Quality of static design of current system	E
	Quality of software written into components	E

Table 18 Causal criteria enabled / inhibited by redevelopment

Redevelopment, when done correctly, can have a positive effect on all of the criteria. As redevelopment focuses on the current application, however, it does not necessarily enable *the suitability of the business process to the organizational mission* or *the suitability of the system to the business process*. Nor does it inhibit it. The effect on these criteria depends on correct assessment and direction before a decision is taken to redevelop. Software restructuring, if it takes place, will enable the *Software Quality* criteria. If Arnold's (1989) definition of software restructuring is taken into account, then this will enable the *underlying platform suitability* criteria also.

Redevelopment is mapped against the causal criteria in Table 19.

7.10 Renew

This option can be chosen when the system is in good technological condition and is valuable to the business. The system may be ageing slightly in terms of the platform on which it sits or its exact alignment with business suitability. The way in which the system is renewed will affect the software quality of the system in terms of change management. It may also have an effect on the software in other ways. In many cases, system renewal causes legacy problems.

7.10.1 Iterative enhancement

Slee and Slovin (1997) advocate iterative enhancement, whereby any change is examined in terms of:

- Its compatibility with the original design and with the strategic plan.
- Its degree of necessity and capacity to enhance the system.
- Cost / benefit ratio and cost / budget ratio.
- Criticality of the required change compared to others in the backlog.
- Lead time before the business will benefit.

Iterative enhancement enables all of the *system suitability* criteria, because it works from a basis of a system that is of high business and technical quality already and iterative enhancement, done correctly, will maintain that quality. It also enables the maintenance of all of the *software quality* criteria.

7.10.2 Code restructuring

Arnold (1989) describes software restructuring as the restructuring of code, documentation, programming environment, software engineers, management policies and external environment. He gives the advantages of restructuring as:

a) Regaining understanding of the software, by making it traceable.

b) Making the software more maintainable by putting it into a context familiar to the current generation of programmers.

c) Preserving the software's asset value to the organization.

Arnold (1989) also realises that code restructuring cannot be feasible as an end in itself. It should be related to locally defined goals and those goals should be related to perceived software value.

Software restructuring, according to Arnold (1989) restructures the software and the environment. This has an enhancing effect on the *software quality* criteria and the *underlying platform suitability* criteria.

7.10.3Re-hosting

This involves moving the system onto a new platform. This may be a physical move, by moving the system onto a new machine or network, or a partly logical move, by splitting the system into tiers and moving one or more tier to another platform. If the system is currently based on a large centralised machine, then it may be a mainframe. Several authors discuss the advantages and disadvantages of keeping the mainframe. Simpson (1995) suggests keeping the mainframe only for older applications, which are less easy to migrate and then replacing them bit by bit, in the belief that client /server is more flexible in its ability to rapidly improve code, making the enterprise more responsive to changing end-user and market requirements. Transaction processing systems in particular are based on proprietary environments and can use very large databases. Beyond Software (1997) discuss the use of Mainframe Web servers, giving advantages and disadvantages. The advantages lie in the strengths of the mainframe. These include existing access control system, security, performance on database queries and transactions, bandwidth and I/O capabilities, elimination of login overhead, reliability, scalability and availability and native data store interfaces for most large data management systems. The disadvantages are that coding is required on the mainframe to re-route I/O from applications and that mainframe programmers need training in Web principles. I-Cube (1998) gives the advantages of the mainframe as reducing cost risk and implementation time, leveraging existing investment and IS functional knowledge and minimising retraining and disruptions to operations. It gives the advantages of re-hosting onto client/server open systems as positioning applications on a platform where they can adapt to emerging technologies such as the web. Therefore, re-hosting has an enabling effect on the *underlying platform suitability* criteria.

Re-hosting is mapped against the causal criteria in Table 19.

Mapping of choice of system renewal against legacy causal criteria				
Evaluation criteria		Iterative Enhancement Enabler, Inhibitor	Software restructuring Enabler, Inhibitor	Re-hosting Enabler, Inhibitor
System suitability	System suitability to business process	E		
	Business process to organizational mission	E		
	System technology to organizational environment	E		I
Underlying platform suitability	Hardware suitability		E	E
	Network suitability		E	E
	Development environment suitability(including OS)		E	E
	Data management suitability		E	E
Software quality	Quality of change management	E	E	
	Quality of static design of current system	E	E	
	Quality of software written into components	E	E	

Table 19 Causal criteria enabled / inhibited by renewal

McGibbon (1996) and Hartley (1996) recognise that in re-engineering to a distributed object computing architecture, the development team needs new architectures, new frameworks, new patterns, new tools and new skills. The technology used will change how the user works, with whom the user works, what the business is and how it survives. Re-hosting can therefore inhibit the suitability of the system technology to the organizational environment.

7.11 Mapping of Strategy Components against Causal Criteria.

The strategy components that have been covered here cover the approach to handling a system, to outsource the problem or handle it in-house; assessing the system, whether it is done with an open mind or towards a certain solution; implementation of a solution – iterative or in one go. Other strategy components involve what goes into the system – architectural decisions, reused data or code, system redevelopment or renewal. All of these are options that can be included in a solution. The decisions made will impact on the possible success or failure of the solution strategy. Table 20 shows how these decisions can have a negative (inhibiting) or positive (enabling) effect on the solution.

Mapping of strategy components against causal criteria																					
Causal criteria		Strategy components																			
		Time base		In / out source		Assess ment		Architecture component				Data reuse			Code reuse				Renewal		
		Iterative	Direct	In-house	Out-source	Open assessment	Direct assessment	Component	Object-orient.	Layered	Bespoke	ODBC	Data migration	Data warehousing	Vertical wrapping	Horizontal wrapping	Application wrapping	**Redevelop**	Iterative enhancement	Software restructure	Re-host
System suitability	**System suitability to business process**	E	I	E	I	E	E		E	E	E	I	E		E	E			E		
	Business process to organizational mission	E	I			E	E		E	E			E						E		
	System technology to organizational env.			E	I	E	I		I	I	E				E	E	E	E	E		I
Underlying platform suitability	**Hardware suitability**			I	E					E	I			E				E		E	E
	Network suitability	I	E	I	E					E	I	I		E				E		E	E
	Development environment suitability (incl. O.S.)	I	E	I	E				I	E	I			E				E		E	E
	Data management suitability			I	E				I	E	I			E				E		E	E
Software quality	**Quality of change management**	I	E		I		I		I	I					I	I	I	E	E	E	
	Quality of static design of current system	I	E					E	I	E					I	I	I	E	E	E	
	Quality of software written into components							E							I	I	I	E	E	E	

Table 20 Mapping of strategy components against causal criteria

7.12 Summary

This chapter has described the components that make up a transition strategy for resolving legacy status in a system and mapped them as enablers or inhibitors against the legacy causal criteria. This mapping facilitates management in their quest to evaluate alternate strategies that are offered to them, as described in Section 6.4.3 and as illustrated in Section 8.3 below.

There are two important things to note here.

The first is that not all decisions affect the causal criteria. If this is the case, the cell in the table is left blank. Other factors will affect this criterion in the solution.

The second is that, because a strategy component may inhibit one of the causal criteria, it does not mean that this strategy component should not be used. It merely means that careful attention should be paid to avoiding the pitfalls of this criterion. These can be avoided by looking at the definition of that criterion in chapters 3 to 5 and taking preventative measures. Likewise, if a strategic component is an enabling factor, it merely means that it will be easier to avoid problems in this causal criterion than it could have been.

Chapter 8 SAP – evaluating a solution strategy against the causal criteria framework

8.1 Introduction

This chapter illustrates the use of the Causal Criteria Framework in a case study by using it to evaluate SAP R/3 against a hypothetical public sector organization whose business needs, when broadly defined, seem to be serviced by SAP R/3.

Section 8.2 describes Enterprise Resource Planners in general, and describes SAP R/3 in terms of functionality, architecture, process model and activity / task framework. This description is structured to enable the assessment of SAP R/3 using the Causal Criteria Framework. Section 8.3 does a preliminary assessment on SAP R/3 by looking at its strategy components and how they enable or inhibit the causal criteria (see Table 20 Mapping of strategy components against causal criteria).

An example of a migration strategy that is being adopted by many companies is the replacement of some or all of their core applications with an enterprise resource planner, SAP R/3. As there are no details of the company, only some of the criteria can be evaluated. This is because many of the criteria depend on the environment in which the system is installed.

8.2 Enterprise Resource Planners and SAP

Software can be replaced on a small scale by replacing or rewriting a module of a single program, to a large scale by replacing application suites at an organizational level. The replacement software being considered here is on an organizational level. Across the world, organizations are opting to replace software on a massive scale. For this exercise to be useful, the most prolific and large-scale replacement software options are considered. Several software suppliers are now providing enterprise solutions, which purport to provide all software needs for an enterprise. Although several companies offer these solutions – notably Baan (see website http://www2.baan.com) , Oracle Corporation's Oracle applications (see website http://www.oracle.com/applications), SSA's BCPS (http://www.ssax.com/BCPS/), PeopleSoft's (http://www.peoplesoft.com/en/products_solutions) Product and Industry solutions and JDE (see http://www.jde.com). A full evaluation of the different Enterprise Resource Planners or ERPs is available in OVUM (1999). The most commonly used of these is SAP R/3 (Dailey 1996, Dailey 1997). This product has achieved world-wide success and as such, is a relevant migration strategy.

The functionality of an ERP is that it provides a set of integrated applications that can be used to fulfil the entire software needs for an organization. It provides a wide range of applications with a good depth of standard requirements coverage in several industries.

Ovum (1999) suggest that the evaluation criteria for ERPs should include both the vendor and the product. The vendor needs to be assessed for financial performance, company character and direction and operation coverage. The product needs to be assessed according to its range of business applications, its functionality, its architecture, its information retrieval mechanism, its flexibility and its ease of implementation. Dailey (1997) concludes that SAP has a lot to offer, but that when prospective customers are evaluating SAP, they should understand their industry, the pace of functional enhancements that are possible within the ERP, the time frame for technology change and the organization's internal needs. To evaluate SAP R/3 in isolation, the framework that is useful from this dissertation is the Causal Criteria Framework.

The ERP under review is SAP R/3. Its functionality, architecture and process model is described in Section 8.2 in a format that is compatible with the assessment method proposed. Section 8.3 does a preliminary assessment on SAP R/3 based on the strategy components that are present in the solution. Section 8.4 does a thorough assessment based on the causal criteria enabled or inhibited by SAP R/3. Section 8.5 discusses the results of the assessment, comparing them to the experiences of other authors.

8.2.1 Description of SAP R/3

SAP R/3 is an enterprise resource planner, which can be used to provide all of the software for an entire organization, across physical frontiers and logically different business entities. The functionality and architecture of SAP R/3 are described here. The process model used to manage the ERP is then discussed.

This description is structured in a way that eases use of the Causal Criteria Framework.

Functionality

SAP consists of 70 complex software modules for business applications, each containing a set of sub-applications. Within the suite, there are over 1,000 business processes, using over 8,000 tables of data and business rules. The modules available are: -

- Financial accounting. The major sub-applications here are financial (FI), controlling (CO) and asset management (AM). FI includes accounts payable, accounts receivable, capital investment and general ledger. This module can also generate reports for the end user, document processes and archive data.

- Human resources. This module includes functionality to pay, schedule, hire and terminate employees. The sub-applications include payroll, benefits administration, applicant data administration, work-force planning, schedule and shift planning, travel expense reporting and personnel development planning.

- Manufacturing and logistics. The sub-applications are materials management, quality management, plant maintenance, production planning and control and project management.

- Sales and Distribution. The sub-applications here allow finding and managing customers, processing sales orders, product distribution, export controls, shipping and transportation management, billing, invoicing and rebate processing (Linthicum 1996).

SAP purports to be Year 2000 and Euro compliant (O'Reilly 1998). Dailey (1996), (Dailey 1997) considers that SAP has many advantages. In certain industries, the functionality available is very good, with enough complexity to adapt to most of the idiosyncrasies an organization can have. However, as SAP tries to expand its horizontal focus – i.e. the number of industries that it services – the vertical focus, or specialisation required, eludes it (Dailey 1997). Because of this, SAP may not be cost effective for all organizations.

Although not directed at SAP in particular, but at the idea of replacement (packaged) software in general, Mack (1997) states that this type of application is not suited to enterprises who wish to use that application as a strategic weapon to become a market differentiator, but more for providing basic applications necessary to run the business, or to enhance their current functions to improve productivity.

Architecture

SAP has a three-tier, thin-client architecture that uses proprietary components. The three tiers or layers are a user interface layer, an application server and a database. The client provides the user interface to the next layer, which is the application server.

User interface layer

The client or user interface layer of SAP can run on several operating system platforms including Unix, Microsoft Windows, OS/2 and Apple Macintosh. Transactions are initiated from here. Although all of the configuration can be done through SAP, it is possible to configure the front-end separately, using third-party tools.

Application server

The central layer is an application server. A transaction initiated by the client sends data to the application server, which invokes the correct application service, to apply the business functions to the transaction. The application server is proprietary to SAP R/3. It runs on various operating system platforms, including Unix, Microsoft Windows, AS/400, MVS and the PowerPC. Application servers can integrate with each other across networks and can be distributed world-wide. Data and functional integration is enabled by the SAP middleware product ALE (Applications Link Enabling. Web enabling technology may also be used (Dailey 1997).

Data layer

The database is the third tier. The application server interfaces to the data layer. The database server is closely coupled to the application server, with one data server for every application server. Both may reside on the same machine or on independent processors, but a one-to-one coupling is always necessary. The database may be a proprietary SAP database or third party database server software such as Microsoft's SQL server or Oracle (Linthicum 1996). If a third-party database is used it may be interrogated in a non-intrusive way by third party applications, but updating the database through means other than the application server is discouraged. Updating of SAP manipulated data by a non-SAP application is not recommended and represents a security risk (Van Haelst & Jansen 1997), by offering the system administrator more than one logical access path to the system.

Inter-layer communication

Communication between the layers is carried out by remote function calls (RFCs), which are equivalent to Remote Procedure Calls (RPCs). The RFCs are platform independent and use popular transmission protocols such as TCP/IP, SNA and IPX/SPX (Linthicum 1996). However, the application programming interface (API) is specific to SAP (BAPI). Web servers can extend the range of SAP, but are available on Microsoft Windows NT machines only (Dailey 1997). Although SAP is expected to become increasingly componentised and message-based, the focus is on improving the componentisation of SAP components, rather than allowing integration to non-SAP components (Dailey 1997). The three-tier architecture can be used to provide fail-over capability – i.e. several application servers can be used, and if one fails, its processing load can be redirected to another. This configuration also allows load balancing, where the workload is divided equally between servers, or System managers can directly connect clients to under-utilised application servers. Data caching allows repeated requests to be served without having to repeat access to the database (Linthicum 1996).

System hardware configuration

As discussed, the platforms used by SAP can vary. Dailey (1997) raises concerns about the complexity of the infrastructure required for SAP. As a client/server distributed system, which is intended to handle much of an organization's financial business, the ability to install and support the infrastructure is paramount. While SAP have kept reasonably up-to-date with the latest technology, they are becoming heavily involved with Microsoft's Windows NT. This requires customer organizations to do the same, if they wish to avail of the full functionality of the system. This will have an impact on the customer organization's support infrastructure and budget (Dailey 1997).

The scalability of a single application server is limited by its coupling to a single database server. While the management of data is less complex this way, it limits SAP R/3 to scale beyond databases that are 500Gbytes and have 2,000 users (as of July 1997). Scalability is also limited by the requirement for high-speed networking connections to handle transaction traffic. The Gartner Group expect this position to improve over the next few years (Dailey 1997).

Process model

Project methodology

The methodology consists of an activity/task framework, which uses a reference model to map the organization's process needs against the modules provided as standard.

The repository structure

SAP is presented in the form of an active repository that contains configurable modules. A pre-configured system is offered, but as this is standard, it is not suitable for most customers (Bancroft *et al.* 1997). The repository is stored as a relational database, with tables for system configuration, control, master data and

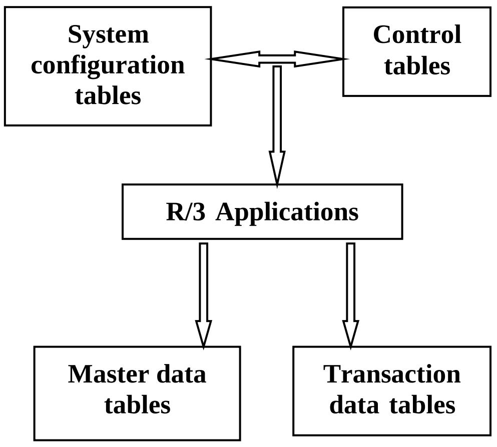


Figure 23 Table structure overview in SAP R/3 (Hinquat & Kelly 1998)

transactions (Figure 23). The system configuration tables are used to configure the applications to the organization and platform, while the control tables are used to guide user activities. The company hierarchy is mapped here. Application data tables are split into transaction tables, which govern operations, and master data tables, which contain stored business data (Hinquat & Kelly 1998). The modules reside in a repository.

Tools

SAP R/3 uses several tools to help in configuring and managing the application suite. Some of them are:

- The development workbench ABAP/4. The repository is accessed by a development workbench, which allows these modules to be configured to increase their compatibility with the organization's processes. SAP offers a workbench for manipulating the system configuration and control tables, called ABAP/4. ABAP/4 allows modification of the business logic, screen layouts, reports and fields (Linthicum 1996). It also contains tools for testing, tuning, debugging and optimising performance. End users can use an ad-hoc query feature without having to learn ABAP/4. Application modules can be customised through the workbench and functionality can be added (Hinquat & Kelly 1998).
- The Correction and Transport System (CTS). To install the designed system, configured and customised applications are moved to the Correction and Transport System (CTS), which quality checks the application and moves it into a live environment (Linthicum 1996). It also tracks changes, giving each change a correction number. CTS propagates changes through different instances of the application server in the live environment (Hinquat & Kelly 1998).
- The Business Navigator. This manipulates the reference model. Through it, the reference model can be viewed either as a set of business applications or from a process flow viewpoint. The reference model contains maps of all processes contained within the R/3 system. It is the document the project team uses to identify exactly what R/3 will do in any particular module. It can be used to understand the differences between how the company works or will work, and how R/3 operates; i.e. it provides a gap analysis. It provides graphical descriptions of the business processes threads.

- The Implementation Management Guide. This is an online documentation tool that steps the project team through the implementation process. It is created in hypertext, with links to the individual functions. It guides the project team in determining the business requirements, the documents and individual fields (Bancroft *et al.* 1997).

Activity / task framework – The Procedure Model

The process of configuring, customising and implementing SAP is guided by a project methodology called the Procedure model. It contains four steps: 1) Organization and conceptual design, 2) Detailed design and system set-up, 3) Preparations for going live and 4) Implementation Management. This model imposes an activity / task framework on the developer.

Organization and conceptual modelling

The user analyses requirements, organises the project, sets up the test environment and trains the project team. The reference model, which is the atlas to R/3, enables this process. A transition between the first and the second of the major steps is a quality check of the target concept (Bancroft *et al.* 1997).

Detailed Design and System Set-up

The team establishes global settings, the company structure and the master data. It also establishes and finalises the functions and processes. The interfaces and enhancements that were previously defined are implemented in this step. Reporting, archive management and authorisation management are established and the final test of the system is performed. This step concludes with a quality check of the application system (Bancroft *et al.* 1997).

Preparations for going live

The team prepares the production start, creates the user documentation and sets up the productive environment. It trains the users, establishes system administration and transfers data into the production system. This third step ends with a quality check (Bancroft *et al.* 1997).

Productive operation

This final step is generally done by the IS department that will support the production operation and optimise the system’s use. This involves that the technical configuration is adequate to the load created on the system by the users. The implementation framework gives the team information on whether or not they have performed each step adequately. The configuration of the system requires that both the platform configuration and the module configuration are complete. Training is stressed in the methodology, as is adequate infrastructural support. Productive operation requires a high level of commitment from the IS staff, to ensure that security and load balancing are properly administered. Dailey (1997) points out that SAP is difficult to configure and install, especially if switching from legacy mainframe systems. The configuration tools offered by SAP are slow and rigid. Third parties are beginning to produce alternatives.

Change management

Bancroft *et al.* (1997) consider that the procedural model is heavy on technical issues and light on the change management issues. Dailey (1997) thinks that one of the biggest difficulties in using R/3 is change management. When different products come out, or requirements change, it is very difficult to reconfigure the system to the new release and can incur development costs as high as those for the installation of a new system.

Operation

The role of the IS department in the operation of SAP turns to that of watchdog. Security is handled by the systems administrator, as is load balancing. The system administrator maintains records of approved users with different security access levels (Hinquat & Kelly 1998). As stated previously, if a third party database is used, reporting facilities that are specific to that database can be used, but circumventing the R/3 services could lead to data integrity problems. System security administration must ensure that independent, unauthorised access to the database layer is not possible (Van Haelst&Jansen 1997). Because of the complexity of the platform on which SAP can reside, the management of the system can be extremely difficult. Many of the management tools offered by SAP only work well when SAP is the only application on the server (Dailey 1997).

8.3 Assessment using the Strategy Component / Causal Criteria Table.

When management are looking around for a solution to their legacy system, they are often faced with a bewildering array of possibilities. Time and cost constraints will prevent a thorough assessment of all these possibilities. To do a preliminary assessment of a solution, management can split the proposed solution strategy into strategy components and look at how these components are likely to enable or inhibit causea criteria, using the mapping of strategy components against causal criteria table (Table 20).

The components in the strategy adopted for SAP R/3 are as follows:

1. The time-base in SAP may be iterative or direct.

2. The project is an adaptation of third party software, so some and usually most of it is outsourced. Table 20 suggests that this component inhibits System Suitability to business process, system technology to organizational environment and quality of change management and enables all of the underlying Platform Suitability criteria.

3. Assessment has been done when this strategy is taken, so this is an unknown component.

4. Architecture - SAP is component-based and layered, not object-oriented, or bespoke. Table 20 suggests that the component-based aspect enables quality of static design of current system and quality of software written into components.

5. Data Reuse - ODBC is not for data reuse. Data migration may or may not take place. Data warehousing is an available option and may or may not be used.

6. Code reuse – is not an option for the core product.

7. Redevelopment is not an option.

8. Renewal is not an option.

Assessment of SAP R/3 components against the causal criteria from Table 20																					
Causal Criteria	Strategy Components	Time base		In / out source		Assess-ment		Architecture Component				Data Reuse			Code Reuse			Rede-velop	Renewal		
		Iterative	Direct	In-house	**Out-source**	Open assessment	Direct assess	**Component**	Object-oriented	**Layered**	Bespoke	ODBC	Data Warehouse	Data migration	Vertical wrap	Horizontal wrap	Application wrap		Iterative enhancement	Software Restructure	Re-host
System Suitability	System Suitability to business process	E	I	E	I	E	E		E	E	E	I		E	E	E			E		
	Business process to organizational mission	E	I			E	E		E	E				E					E		
	System Technology to Organizational environment			E	I	E	I		I	I	E				E	E	E	E	E		I
nderlying Platform Suitability	Hardware Suitability			I	E					E			E					E			E
	Operating System Suitability	I	E	I	E								E					E		E	E
	Network Suitability	I	E	I	E					E			E					E		E	E
	Development Environment Suitability	I	E	E	I								E					E		E	E
	Data Management Suitability			I	E					E			E					E		E	E
ware Quality	Quality of Change Management	I	E		I					I					I	I	I	E	E	E	
	Design Quality	I	E					E	I	E		I			I	I	I	E	E	E	
	Component Quality							E							I	I	I	E	E	E	

Table 21 Preliminary Assessment of SAP R/3 using strategy components

The strategy components that have been identified as part of the SAP R/3 solution are outsourcing, component-based and layered architecture. The corresponding columns from table 20 are highlighted in Table 21. Other columns from Table 20 are shown on a grey background, so that the reader can see the context more clearly. There are three causal criteria which show up as inhibited by the components in SAP R/3. They are *suitability of system technology to organizational environment, suitability of the development environment* and *quality of change management.* The *System Suitability* *to business process* is inhibited by the outsourcing component and enabled by the layering component. Two of the components of the SAP R/3 solution strategy enable *hardware suitability, network suitability, data management suitability* and *Design quality*. The remaining three criteria are enabled by one of the strategic components of SAP R/3.

8.4 Assessment using the Causal Criteria Framework

The Causal Criteria Framework is used here to evaluate a solution strategy in isolation. It cannot be fully assessed, as the environment into which the strategy is fitting is not available for the assessment. However, the questions that may be asked pertinent to the environment can be taken from Chapter 3 System Suitability, Chapter 4 Underlying Platform Suitability and Chapter 5 Software Quality, that describe that causal criterion. The framework is shown with "I" where the causal criterion is inhibited, "E" where the causal criterion is enabled and "C" where there are further considerations. These considerations are listed, with reference to the section in the dissertation from which they are derived.

By analysing the strategic components in SAP R/3, there are indications that some of our criteria are enabled or inhibited. However, as stated in section 7.11, all that this shows is that a system which contains these strategic components has the capacity to behave as shown in Table 20. In order to assess this specific solution, it is necessary to examine the solution against the causal criteria.

8.4.1 System Suitability

System suitability to business process

As seen in section 3.1.1, the system suits the organization by doing what the organization wants it to do. Information Engineering (Davids 1992) formulates a strategic plan, building the business process needs and assessing current or potential systems for their applicability. Likewise, the Strategic Alignment Model's (Henderson & Venkatraman 1993) strategy execution alignment perspective (see Figure 5) formulates business strategy and designs the IS infrastructure around it.

If the business application is standard, then the use of standard software is a logical and cost-saving step, especially when the software comes from a reliable source and has been proven in the field. However, if the business process is non-standard there may be more than one reason for this. Either the business process has veered from standard business practice throughout the industry, for no valid reason and can easily be replaced with a standard implementation, or there are very valid business reasons for the application being non-standard. If this is the case, then standard software will not provide the solution, especially if customisation is unsupported or wide-ranging.

Considerations:

- *Can the business process be standardised?* Management should consider this very carefully – often, despite their best wishes, conditions outside their control cannot be changed; e.g. union agreements, national budget agreements, customer end-user's ability or enthusiasm to adapt to new practices, change in service level to the organization's customer due to staff having to operate the system.

- *What is the scale of diversion of the organization's practice from standard practice?*

- *Does the offered solution integrate with bespoke software that may make up the difference?*

- *Is the customised software component coupled loosely enough with the ERP to allow for independent upgrades of the ERP?*

These questions are summarised in Slee and Slovin's (1994) portfolio assessment when the ask "How does the IS organization satisfy its customers?" and "How well is IS delivering value and responding to needs?"

Note that the previous definitive marking of SAP as an inhibitor and an enabler to system suitability to business process has now been amended, to 'requires further consideration' – i.e. "C".

Business process to organizational mission

This is defined in section 3.1.1 as the system doing what the organization needs it to do. Davids (1992) formulates a strategy before developing a set of business processes, thereby avoiding this problem. Henderson and Venkatraman (1993) offer four perspectives (see Figure 4) that combine to ensure that business strategy is in line with the best available IT strategy that can work in the organization's internal environment. Slee and Slovin (1994) endorse the idea of alignment.

The software offered in SAP is standard across installations, with most of the customisation being in the configuration of the installation rather than in the ability to change modules. This can be either an advantage or a disadvantage. If the organization's requirement is to have a set of applications that are standard and reliable, then this is an advantage. If the requirement is to use software as a competitive advantage, then the standardness is a disadvantage. In evaluating this, the need for and scale of diversion from standard practice must be considered. The flexibility of the ERP in integrating with bespoke applications is also a factor.

Considerations:

- *What is the organization's position in the marketplace?* This question addresses Slee and Slovin's (1994) questions "How well does the IS organization compare to industry standards such as the SEI CMM?" and "How does the IS organization satisfy its customers?" and also Henderson and Venkatraman's (1993) competitive potential alignment perspective (Figure 6).

- *How does SAP R/3 fit in with the business strategy?*(Henderson & Venkatraman 1993)
- *How does the external I.T strategy (e.g. providers) fit in with the business strategy and internal IS infrastructure?*(Henderson & Venkatraman 1993)

The indication is that SAP 'requires further consideration' – i.e. "C".

System technology to organizational environment

This is defined in section 3.1.1 as the system being usable by the organization. ERPs work off a variety of platforms, but in general, new technology will need to be installed in order to maximise the usefulness of the application. This may involve a change in technological and business process direction. These factors must be taken into account when an application is being chosen. If the organizational support infrastructure is unable to cope, the installation will be a failure. Likewise, if the user base is unfamiliar or unable to cope with the technological and procedural changes, the installation cannot succeed. The use of third-party supplies software requires a change in staff attitude within an organization. This change in attitude needs to encompass not only the top management level, but also the organizational and IS support infrastructure as a whole, including support staff and users. If the change in working practices is too great, then users and support staff may not make that change. This is particularly the case where physical or logical user interface difficulties make it unacceptable to use. Application software must be matched not only to the business process and strategy, but to the end user for whom it is intended. This is especially the case where the organization is not in a position to train the end user – for example, when the end user is a member of the general public. The application must also be prevented from making such a change in working practices of employees as to make them unavailable to provide the service to the public that they are expected to make. Security may also be an issue here. If the correct administration is not undertaken, the system may be open to breaches.

Checkland’s (1981) Soft Systems Methodology looks at systems that are in situ but are not working as they should, possibly due to misalignment with the internal IS infrastructure. Holzblatt and Jones’ (1993) contextual interview and co-operative evaluation can be used to assess the working environment before a system is introduced.

Considerations:

- Is the organizational infrastructure adaptable to the change?
- Is the IS expertise available reliably and cost-effectively,
    - To install and support the operation of the new system?
    - To incorporate requirements change?
    - To upgrade to the next version of the ERP?
- Is the end-user base:-
    - Amenable to the new system?

- Physically, intellectually and psychologically able to operate the new system?
- Unhindered in their other tasks by the introduction of the new system?

The previous definitive marking of SAP as inhibiting suitability of system technology to organizational environment stands. This does not mean that SAP R/3 cannot be used, but that it is difficult for an organization to introduce into an environment and that the organization will need careful planning and assessment before choosing it. If it is chosen, then it is likely that retraining or hiring of new staff will need to be undertake on a substantial scale.

8.4.2 Underlying Platform Suitability

As stated, the ideal platform infrastructure is an open infrastructure, with service-oriented architecture, adequate traffic control and suitable and robust hardware and networking, operating system, database and development environment software. **Error! Reference source not found.** indicates that SAP R/3 enables all of the underlying Platform Suitability criteria due to its component-based nature. The extent of modularity and componentisation can vary from one ERP to another. As part of a strategic IS plan, the choice of platform will impact on the IS support requirements and the budget of the department. Each organization must outline its priorities in regard to scale of systems and levels of integration required. It is only against these criteria that a platform can be evaluated. The ideal outlined in 4.1.3 shows what can be achieved. However, it may not be necessary or desirable for a company to strive towards these goals.

ERP software is split into modules and the modules can be run independently or integrated. Within SAP R/3 there is a tight coupling between the application server and the data server, in that there must be a single data server for each application server.

Hardware suitability

According to Laudon & Laudon (1998), hardware choice involves understanding the capabilities of various computer processing, input, output and storage options as well as price-performance relationships. Hardware should be evaluated by type, processor size compatibility, upgradeability, reliability of vendor, standardisation of ports and peripherals, scalability, robustness and cost.

SAP R/3 has a wide variety of hardware platforms on which it can run, so hardware suitability should not be a problem. In this author's opinion, the "E" for enabling is justified in this case.

Operating System Suitability

As with hardware, SAP R/3 has a wide variety of operating systems on which it can run, thereby enabling this criterion. However, there is a policy being adopted by SAP to provide more functionality on Microsoft's Windows NT ™ operating system platforms. This policy needs to be considered in light of the need of the organization to use the extra functionality and if so, the operating system itself needs to be evaluated as laid out in Section 4.2.4. This criterion therefore requires further consideration and is assigned a "C".

Network suitability

Fitzgerald and Dennis (1996) list the network evaluation criteria, from the management focus as time, cost, quality, capacity, scope, efficiency, productivity and flexibility (see section 4.2.5).

Regardless of organizational aspirations, three-tier client/server architecture is desirable. In theory, this enables upgrading of one layer independently of the others. However, each layer should be robust, adhere to industry standards and have good vendor reliability. Security between layers and performance must be considered. If a large-scale system is being considered, then message traffic control is a consideration. However, SAP R/3 does place some limitations on communications between layers, in that Microsoft platforms provide better network capabilities in the form of web servers. There is also a tight one-to-one coupling between the application server and the data layer.

The issue of configuration management becomes more complex as networks grow. As this is as important as any of the individual pieces of hardware that are procured, this must also be considered (see section 4.2.8).

Many of the difficulties that may be caused by networks may be due to their unsuitability within the organizational environment and may therefore be covered by this criterion. However, as the network and its operability is crucial to the success of the solution, this author would advise further consideration regarding networks. This cell changes its value from “E” to “C”.

Development environment suitability

Laudon and Laudon (1998) give the attributes that need to be considered in a development environment as openness, standardisation, portability, adaptability, suitability to platform, application, database and HCI, adaptability to programming interfaces, robustness, efficiency, adaptability to batch processing, object or component-based or not and finally documentation.

In the case of SAP R/3, the development environment is a configuration rather than a development environment. This means that it has limited adaptability and suitability to application and it cannot easily be programmed. The quality and flexibility of the tools available here will have a big impact on the quality of the end system. Ideally, there should be an option for adding in bespoke or third-party modules at this level. For this to work, there must be a development environment as well as a configuration environment.

Because of the poor programmability in SAP R/3, the development environment retains its “I” as an inhibitor.

Data management suitability

Laudon and Laudon (1998) give attributes of data administration as the abilities to share, disseminate, acquire, standardise, classify, inventory, plan and model, manage, organise, secure, maintain, provide usability and privacy.

SAP R/3 allows use of a proprietary SAP database or third party database server software (Linthicum 1996). This allows for a robust database to be chosen. There are a couple of problems with this however. The first is the security issue, where third-party applications can manipulated SAP data (Van Haelst & Jansen 1997). The second is the one-to-one coupling between the application server and data server. This may cause data sharing or dissemination problems. SAP therefore ceases to be an outright enabler ("E") and requires further consideration ("C"). This consideration once again centres on the ability of the organization to operate the database efficiently and securely.

8.4.3 Software Quality

Software quality is not always easily to assess in an ERP. While the functionality and performance of the software will become evident after it is installed, the system software quality factors are not. Nevertheless, there are certain questions that need to be asked about software when choosing the ERP.

Thorough assessment of SAP R/3 against the causal criteria framework		
System suitability	System suitability to business process	C
	Business process to organizational mission	C
	System technology to organizational environment	I
Underlying platform suitability	Hardware suitability	E
	Operating System Suitability	C
	Network suitability	C
	Development environment suitability	I
	Data management suitability	C
Software quality	Component quality	E
	Design Quality	E
	Quality of change management	I

Table 22 Thorough assessment of SAP R/3 against causal criteria framework

Component quality

Component quality (see section 5.2.1) can be judged by its cohesiveness and coupling required. There are also indicators of component quality that can be derived from adherence to obvious standards, such as Year 2000 and Euro compliance. The quality of individual modules can be found out by benchmarking the module's performance in similar installations. That applies only where the module is in widespread use. Component quality should be assured by the vendors and written contractual agreements made by purchasers. Vendor reputation and reliability are factors in assessing this.

Considerations:

- Is the component performing to recognised quality standards in other installations where the requirement is similar to this organization?

- Are there written contractual agreements available from the suppliers?
- Does the vendor have a reputation for high quality?

In the case of SAP R/3, these considerations are favourable (Dailey 1997). SAP R/3 retains its “E” as an enabler for Component Quality.

Design quality

Good quality design needs a robust process model that is supported by a relevant methodology and tool set (see section 5.2.2). Along with good quality components, the framework for customising and mapping them must be adequate. This requires a robust activity / task framework and a set of visual tools, which enable the designer to make any necessary changes and see how those changes will affect the system being generated. Integrity checks and change tracking should be implemented, as should a secure design environment. Documentation production should add value to the process, rather than volume.

As seen in SAP, the workbench provides a reference model and procedure model which enables high quality configuration. However, the compliance of developers and administrators with this is voluntary. The quality of *static* design within SAP R/3 is an enabling “E” factor.

Quality of change management

For a full discussion on how change management can be carried out, see Section **Error! Reference source not found.**. **Error! Reference source not found.** has given SAP R/3 an "I" for inhibitor, due to the fact that much of it is outsourced. The use of third party software ties an organization into whatever change management is provided by that third party. A complete change of paradigm would probably mean moving away from this third party, but even a seemingly less drastic event, such as an upgrade release, may incur huge cost. Upward compatibility is only one of the factors that can cause problems. Another major difficulty is that customisation done in a previous version may not be carried through to the next version. Maintenance contracts should address these issues and the purchaser should be clear on the situation before purchase. SAP R/3 retains an "I" as an inhibitor to change management.

8.5 Conclusion of the assessment of SAP R/3

Although SAP R/3 has been evaluated against the causal criteria framework, no definite decision can be made as to whether or not to choose it as a solution strategy. There are two reasons for this.

1. The environment into which the system is to be introduced is unknown.
2. Even the definitive indicators given by the framework are only enablers and inhibitors – i.e. an indication of possible problems.

On comparing the results given in **Error! Reference source not found.** and Table 22 it can be seen that some of the criteria changed from being assumed an enabler or an inhibitor to requiring further consideration. This is because a) the component strategy in the solution has been identified as corresponding to a component strategy as described in Chapter7 and b) a status of enabler or inhibitor is only an indication of possible behaviour, not an assurance.

Effects that are indicated by this assessment

The causal criteria that have been defined as inhibited by the use of SAP R/3 are the *suitability of the system technology to the organizational environment, the suitability of the development environment* and the *quality of change management*.

By looking up the corresponding column in the Legacy Status Cause / Effect Framework (Table 10) the possible effects that may result from choosing SAP R/3 can be seen.

Legacy Effects / **Causal Criteria**	**Asset value**		**Ease of operation**		**Ease of maintenance**				**Ease of migration / evolution**	
	Mission criticality	*Reliability*	*User satisfaction*	*Ease of testing and auditing*	*Cost of maintenance and resistance to it*	*Availability of maintenance resources*	*Program size and complexity*	*Dependence on individuals*	*Ease of use of new technology*	*Scalability*
System technology to organizational environment	X		X	X	X	X	X	X		
Development environment		X	X	X	X	X		X	X	X
Quality of Change Management		X		X	X	X	X	X	X	

Table 23 Effects indicated by SAP R/3 assessment

Poor quality of change management can cause a diminishment in reliability, lack of ease of testing and auditing, availability of maintenance resources, program size and complexity, ease of use of new technology.

Third party criticisms of SAP R/3

Other researchers, as shown below, have expressed reservations about the suitability of use of SAP R/3 to many organizations.

Linthicum (1996) considers that larger organizations are more comfortable with packaged client/server applications for enterprise-level, business-critical requirements than with building applications from the ground up. However, R/3 will not fit the needs of most businesses without customisation. Dailey (1997) states that the discipline of using SAP templates may not be suitable for all organizations. These comments reinforce the assessment here that SAP R/3 may inhibit the development environment suitability. Linthicum (1996) acknowledges that there is a lot of work for client/server developers customising R/3. R/3 is becoming another platform on which to build client/server applications. The industry should respond with new tools and technologies to make this development easier. These comments reinforce the lack of suitability of the system to the organizational environment uncovered by the assessment.

Bancroft *et al.* (1997) consider that the procedural model does not address change management issues. Daily (1997) thinks that one of the biggest difficulties in using R/3 is change management. These criticisms fit in with the evaluation of SAP R/3 that has been produced by the Causal Criteria Framework.

Chapter 9 .Summary and conclusion

9.1 Introduction

This chapter gives a summary of the dissertation, followed by the conclusions in the dissertation. Other related research is described and further work is suggested.

9.2 Summary of the Dissertation

This dissertation consists of nine chapters.

Chapter 1 introduces the topic area and background, and gives the aim and objectives of the research. It describes the research method used.

Chapter 2 identifies and lists legacy effects, grouped into four effect groups. The Legacy Effect Determination Framework is designed through these groups and presented in Table 2 in Section 2.2.7. It then analyses existing research and determines legacy causes, grouping them into three legacy causal dimensions. The table of legacy causes is presented in Table 3, Section 2.3.7. A definition of legacy status is presented in Section 2.4. This chapter presents deliverables on the first three objectives listed in Section 1.2.

Chapters 3, 4 and 5 consider the three causal criteria groups of legacy status, with Chapter 3 addressing System Suitability, Chapter 4 addressing Underlying Platform Suitability and Chapter 5 addressing Software Quality. Each of the three chapters has a similar structure. The area being examined is defined in the context of the dissertation, in the first Section (3.1, 4.1 and 5.1 respectively). Modern practices in the area are discussed in the second Section (3.2, 4.2 and 5.2 respectively). Reasons why these practices are not always in use are addressed in the third Section (3.3, 4.3 and 5.3 respectively) and the legacy effects that poor practice in these groups can cause are determined in the fourth Section (3.4, 4.4 and 5.4 respectively). The fourth section also cross-references each of the causal criteria within the relevant group with legacy effects in a table Table 4, Table 6 and Table 7 respectively)

Chapter 6 presents three frameworks. It reiterates the Legacy Effect Determination Framework (Table 8) and places it in context (Section 6.1). It introduces the legacy Causal Criteria Framework (Table 9) in Section 6.2 and it cross-references the legacy Causal Criteria Framework and the Legacy Effect Determination Framework, giving an overall framework, the Legacy Status Cause / Effect Framework (Table 10) in Section 6.3, working from the results obtained in Chapters 2 through 5. Procedures for assessing current systems and new systems are designed and presented using the Legacy Effect Determination Framework and the Legacy Status Cause / Effect Framework (Section 6.4). The usefulness of these frameworks is argued in Section 6.5.

Chapter 7 addresses the objectives involved in correcting a legacy problem and some of the legacy handling strategies that are being put forward at present are listed in Section 7.1. Section 7.2 lists the components of these strategies that are under review in this dissertation. Section 7.3 to 7.10 describe these components, and identify which of the legacy causal criteria they enable, if any and which they inhibit, if any (Table 12 to Table 19). Section 7.11 presents a mapping of strategy components against Causal Criteria Framework (Table 20), showing enablers “E” and inhibitors “I”.

Chapter 8 presents a case study in which the frameworks are applied to a solution strategy. Section 8.2 describes the strategy in a structure that enables assessment. Section 8.3 identifies the inherent strategy components and uses the mapping of strategic components against causal criteria to identify the causal criteria that are likely to be enabled and inhibited (Table 21). Section 8.4 assesses the entire solution against the Causal Criteria Framework (Table 22). Section 8.5 concludes the assessment and evaluates the potential effects (Table 23) of choosing this strategy.

Chapter 9, this chapter, summarises the dissertation and offers conclusions and suggestions for further research in this area.

9.3 Conclusions

The aim and objectives stated in Section 1.2 have been met. For the convenience of the reader, the aim and objectives are repeated here.

9.3.1 Restatement of aim and objectives

The aim of this dissertation is to research into the concept of legacy status and related issues regarding transition from that status and develop a set of frameworks that can be used by management to identify legacy status in a current or planned business information system. The results can be used to provide guidelines to management to enable them to choose a suitable solution to any legacy aspects that are present and avoid immediate potential legacy status in the new system.

In order to achieve this aim, the following objectives need to be met:

1. To identify the characteristic effects that are evident in legacy systems so that they can be related to a legacy problem.

2. To develop a Legacy Effect Determination Framework so that a system's legacy effects can be documented.

3. To identify the characteristic causes of legacy systems and define legacy status.

4. To develop a thorough definition of causal criteria, to enable assessment to take place.

5. To develop a legacy Causal Criteria Framework, so that the causal criteria of legacy status can be identified within a system.

6. To develop a Legacy Status Cause / Effect Framework, so that if a weakness exists in one of the causal criteria, the possible effects of this can be seen. Alternatively, if the system is exhibiting legacy effects, this framework identifies what the possible underlying causes are.

7. To analyse components of existing strategies for dealing with legacy systems and the effects of these components on legacy status, in order to guide strategic managers in the task of choosing an approach towards transition from a current legacy system.

9.3.2 Delivered Results

This dissertation has introduced a new way of thinking about legacy systems. Rather than classifying a system as either legacy or non-legacy, it presents and defines the concept of legacy status. It then develops and presents three frameworks and asset of assessment techniques that can be used to assess current and prospective systems.

Each of the objectives is delivered as outlined below.

Objectives 1 to 3

Chapter 2 above presents deliverables on the first three objectives. It identifies legacy effects and tabulates them (Table 1). It develops a Legacy Effect Determination Framework (Table 2 - repeated in Table 8) and identifies and tabulates legacy causal criteria (Table 3) and defines legacy status.

The benefit of these deliverables is that they enable management to determine whether or not a current system is a legacy system and in what areas legacy status exists.

Objective 4

Chapters 3, 4 and 5 are devoted to the fourth deliverable. They define the three dimensions of legacy causal criteria, giving current practice in the area and problems that can arise due to a weakness in one of the causal criteria. These are cross-referenced to the legacy Effect Determination Framework in Table 4, Table 6 and

Legacy Effects / Software Quality	Asset value		Ease of operation		Ease of maintenance				Ease of migration / evolution	
	Mission criticality	*Reliability*	*User satisfaction*	*Ease of testing and auditing*	*Cost of maintenance and resistance to it*	*Availability of maintenance resources*	*Program size and complexity*	*Dependence on individuals*	*Ease of use of new technology*	*Scalability*
Quality of Change Management		X		X	X	X	X	X	X	
Design quality		X		X	X		X	X		X
Component quality		X		X	X			X	X	X

Table 7.

The benefit of this is that it enables management to accurately assess current and future systems regarding causal criteria along any of the three causal dimensions, within the context of their organization.

Objective 5

The Causal Criteria Framework is presented in Chapter 6 (Table 9).

The benefit of this is that it enables management to document causal criteria that are enabled or inhibited either by a current system or a proposed system.

Objective 6

Chapter 6 cross-references the Legacy Effect Determination Framework with the legacy Causal Criteria Framework, giving the Legacy Status Cause / Effect Framework (Table 10).

The benefits of this framework are two-fold. 1) Once the legacy effects exhibited by a system have been identified, this framework will indicate possible causal criteria and 2) if a system is an inhibitor to one of the causal criteria, the possible legacy effects that may result can be identified.

Objective 7

Chapter 7 above is devoted to the seventh deliverable. Legacy system handling strategies are broken down into components and the components mapped against the causal criteria to show which they enable and which they inhibit (Table 20).

The benefit of this is that it enables management to perform a preliminary assessment on proposed solutions to existing or planned systems, with a view to minimizing the risk of legacy problems being introduced with a new solution.

Addressing the aim

Having addressed each of the objectives, the fulfillment of the aim is now discussed.

"*The aim of this dissertation is to research into the concept of legacy status and related issues regarding transition from that status and develop a set of frameworks that can be used by management to identify legacy status in a current or planned business information system."*

This part of the aim is fulfilled as outlined in the objectives above.

"*The results can be used to provide guidelines to management to enable them to choose a suitable solution to any legacy aspects that are present and avoid immediate potential legacy status in the new system*."

Chapter 6 outlines a method of assessment of current and proposed systems using the LACE techniques on the three frameworks provided, by answering the following list of questions that could be put by management.

i) *Does the system suffer from legacy status and if so, what could be causing it?* This question can be addressed by applying the assessment technique described in Section 6.4.1.

ii) *Does our current system inhibit any of the legacy causal criteria?* This question can be addressed by applying the assessment technique described in Section 6.4.2.

iii) *When solutions are proposed, how can we do a quick assessment of them to see if they contain innate criteria that will cause likely legacy status?* This question can be addressed by applying the assessment technique described in Section 6.4.3.

iv) *When we are seriously considering a solution, how do we assess it?* This question can be addressed by applying the assessment technique described in Section 6.4.4.

The aim of the project has therefore been achieved.

9.3.3 What are the possible effects that could result from these innate criteria?

Possible effects can be indicated as soon as the *legacy causal criteria framework* is filled in. This can be done blindly by using the *"mapping of strategy components to legacy causal criteria"* table to fill in the *legacy causal criteria framework* and cross-referencing the results to the *legacy status cause / effect framework.* However, it is important to note that the actual solution needs to be evaluated against the *causal criteria framework* and then cross-referenced to the *legacy status cause / effect framework* to achieve any accuracy in this quest. To cross-reference the *legacy causal criteria framework* with the *legacy status cause / effect framework*, the row for any causal criterion that shows up as "I" in the *causal criteria framework* can be looked up in the *legacy status cause / effect framework*. Any column with an "X" indicates that the related effect may result if the solution is used.

9.4 Comparisons with other research

Investigation into legacy systems is a vast research area. Many research projects have been and are being undertaken around the world. Some of these are funded by reseach councils in Great Britain (SEBC 1998) and Europe, while others ,such as Slee and Slovin's are done as part of the industry's drive to conquer the problems faced in dealing with legacy systems.

While many of these research projects touch on the subject areas of this dissertation and have been referenced where appropriate, few have similar objectives. Those chosen for discussion here have some similar objectives.

Slee and Slovin's (1997) paper presents the findings of an inquiry into legacy systems. They describe how legacy systems come about and why they are classified as legacy. They advocate portfolio management, from both business and technical perspectives, with the portfolio being examined in "meaningful chunks", where each chunk supports a business area. They develop a 4R portfolio assessment matrix, where one of four strategies is adopted depending on the results of the portfolio evaluation. This research is very useful. This dissertation uses Slee and Slovin's (1997) research as a basis on which to build, for example, components of a solution strategy. As such, the findings in this dissertation go further than those of Slee and Slovin.

SEBPC workshops on legacy systems started with the first workshop (Ramage 1998(1)). Three other workshops followed (see http://www.dur.ac.uk/CSM/SABA/). The first workshop culminated in a definition of legacy systems (Gold 1998, Ramage 1998(1)). The second workshop offered a set of viewpoints on solutions, which split them into business versus technical, future-proofing versus coping, users versus developers and evolution versus revolution (Ramage 1998 (3)). The third addressed the conflict between social and technical dynamics in an organization. While these workshops raised many of the problems that are related to legacy systems, no definitive resolution to finding a solution has yet been put forward.

Edwards *et al.*'s (1998) research is into legacy systems in small manufacturing enterprises. The group has proposed the risk assessment model: Evaluation strategy for existing systems (RAMESES). This project "aims to identify the factors that affect the fit between business processes and IT systems and subsequently to address the issues of risk assessment for small organizations desiring change". Research undertaken by this group confirms the findings that many managers believe that large-scale, integrated IT systems are the answer to their problems, whereas in reality, the difficulties that are faced by these organizations could be overcome more efficiently and effectively by other means. The group has discovered problems in these systems that relate to software quality and system suitability. Their aim is to derive risk characteristics and quantify them, in a bid to evaluate proposed solution strategies. This research differs from the aim here, in that it tackles risk, rather than cause and effect. This research is ongoing.

The SABA (software as a Business Asset (K. Bennett, C. Brooke – University of Durham)) project has produced a model that takes in the status quo of the organization into an organizational scenario Tool (OST), prioritises possible solutions and feeds the results into the Technology Scenario tool (TST). The asset base is also fed into the two tools (Ramage 1998. This iterates around until a single solution remains, which becomes the final scenario. The Technical scenarios tool has four stages: solution routes, information capture, analysis and details of solutions. Typical solution routes are leave, discard, replace, rebuild, re-require, recreate, redesign, re-engineer, wrap and outsource. The details of solutions add issues of tools / techniques, standards / quality, time / cost, benefit, risk and transition routes to the best fitting solutions from the analysis stage (Ramage 1998b, c).similar in aim to the research presented in this dissertation. However, the approach differs in that a toolset is being developed. This research is ongoing.

9.5 Further work

This dissertation provides guidelines to management who wish to assess the legacy status in their systems and choose a solution that enhances their business system. It would be a great advantage to be able to quantify the level of legacy status associated with these criteria. Possible further research following this direction is:

1. To establish a measurement technique whereby some or all of the causal criteria could be given a quality grade.

2. To combine the resultant quality grades into a formula which would indicate a legacy status co-ordinate on the *dimensions of legacy status* graph (Figure 2).

3. To do a more rigorous definition of strategic components, so that a manager could ascertain exactly whether the solution being offered contained this component or not. This may also make the *mapping of strategy components against Causal Criteria Framework* table more meaningful in the indications it gives.

4. To take case studies from the semi-state organizations that have given advice and assess systems that have undergone, are undergoing and are under consideration for undergoing change.

5. To put the frameworks as assessment techniques into practice in an organization that is undergoing system transition.

9.6 Concluding remarks

The area of legacy system assessment and evolution / migration is very broad and detailed. As such, no simple solution can be offered towards it. The author believes that the knowledge in this area will grow and be consolidated with each new research project that is undertaken. As with the evolution of software quality and complexity, the ability to manage change will become an engineering discipline, thereby turning the current art of handling legacy systems into a science.

No solution strategy is perfect. When evaluating a solution strategy, the best that management can do is try to ensure that the solution has less legacy causal criteria in it then the existing system has. It is important to remember Slee and Slovin's (1997) remark that *"legacy is destiny – the ongoing challenge of managing evolving IS assets in the era of hybrid computing"*.

References

ADOLPH, S. 1996 “Cash Cow in the Tar Pit” IEEE Software, May Vol. 13, No. 3, pp. 41-47, CA., USA

AHO, A.V., J. HOPCROFT, J. ULLMANN, 1983 Data Structures and Algorithms, Addison-Wesley.

ALDERSON, A., H. SHAH, 1998 “Understanding Legacy Systems Through Viewpoints and Events”, Proceedings of BIT ’98 (accepted for publication).

ALLEN P, S. FROST, 1997 “Fitting Legacy Assets into the World of Components”, Object Magazine, USA.

ARONICA, R., D. RIMMEL, 1996 “Wrapper your legacy system” Datamation 15th June, Newton, MA, USA.

ARNOLD, R. 1989 “Software Restructuring” Proceedings of the IEEE, April Vol. 77, No. 4, pp 607-617

AVISON, D.E., G. FITZGERALD, 1995 Information Systems Development: Methodologies, Techniques and Tools McGraw-Hill London.

BACH, J. 1994, “The Immaturity of the CMM”, September American Programmer.Also available at: http://www.stlabs.com/testnet/docs/CMM_AP1.htm (January 1999)

BANCROFT, N., H. SEIP, A. SPRENGEL, 1997 Implementing SAP/R3. Manning publications. Also available: http://www.browsebooks.com/Bancroft/Contents.html

BENNETT, K. 1994 “Legacy Systems: Coping with Success”, IEEE Software, January Vol 12, No.1, pp 19-23.

BERNSTEIN, P. A. 1996 “Middleware - A Model for Distributed System Services” February Communications of the ACM, Vol. 39 No. 2 pp 86 - 98

BEYOND SOFTWARE 1997 “Cost/Benefits of Transforming Legacy Applications”, White Paper, Beyond Software Incorporated, available at: http://www.beyond-software.com/Customers/CostBenefits.html.

BIEMAN, J.M., L. M. OTT, 1994, “Measuring Functional Cohesion” IEEE Transactions on Software Engineering, August vol. 20, no. 8, pp 308-320

BOEHM, B. 1978 Characteristics of software quality, Vol. 1 of TRW series on software technology, North-Holland, Amsterdam, Netherlands.

BOEHM, B., 1988 “A Spiral model for Software Development and enhancement”, Computer, vol. 21, no. 5, pp61-72.

BOHM, C., G. JACOPINI, 1966 “Flow diagrams, Turing machines and Languages with only two formation rules” CACM, May vol. 9, no. 5 pp 366-371.

BREITENEDER, C.J., M. HITZ, T.A. MUECK 1996 “Metadata mining in legacy data sets”, IEEE. Available at http://computer.org/conferen/meta96/breitender/Meta.html

BRODIE, M., M. STONEBRAKER 1995 Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach. Morgan Kaufmann Series in Data Management USA

BROOKS, F. 1975 The Mythical Man-Month. Addison-Wesley.

CASALS, E., 1998, “Re-Engineering Object-Oriented Legacy Systems” JOOP January Vol 10, no. 8, pp 47-52.

CHECKLAND, P. 1981 Systems Thinking, Systems Practice. Wiley, Chichester.

CHEN, P., 1977 The Entity-Relationship Approach to Logical Database Design, QED Information Systems.

DAHL, O., E. DIJKSTRA, C. HOARE, 1972 Structured Programming, Academic Press.

DAILEY, A. 1996 “SAP: Future or Past?” GartnerGroup November Conference Presentation, Cannes.

DAILEY, A. 1997 “SAP: Getting ready for 4.0 and beyond” GartnerGroup November Conference Presentation, Cannes.

DAVIDS, A., 1992 Practical Information Engineering, the management approach. Pitman London.

DEBOU, C., N. FUCHS, M. HAUX 1995 “AMI: A Tailorable Framework for Software Process Improvement”, ISCN Newsletter, International Software Consulting Network.

DELUCIA, A., G. DI LUCCA, A. FASOLINO, P. GUERRA, S. PETRUZZELLI 1997 “Migrating Legacy Systems towards Object Orientated Platforms”, IEEE Computer Society, USA.

DENNIS, J., 1973, “Modularity” in Advanced Course on Software Engineering.(ed. F.L.Bauer) pp 128-182, Springer-Verlag, NY.

DEMARCO, T. 1979, Structured Analysis and System Specification, Prentice Hall.

DIJKSTRA, E., 1965 “Programming considered as a Human Activity”, Proceedings IFIP Congress, North Holland Publishing Co.

DIJKSTRA, E., 1976 A Discipline of Programming, Prentice Hall Englewood Cliffs, NJ.

DIX, A., J. FINLAY, G.ABOWD, R. BEALE 1993 Human Computer Interaction Prentice Hall, Englewood Cliffs, NJ.

DOWNS, E., P. CLARE, I. COE, 1992 Structured Systems Analysis and Design Method Application and Context, Prentice Hall Hertfordshire.

EDWARDS, H.M., S.C. WILLOUGHBY, G.M. MALLIEU 1998 “Legacy Systems Interaction with Business processes in SMEs” First SEPBC workshop on legacy systems, Durham University.

FITZGERALD, J., A. DENNIS, 1996. Business Data Communications and Networking. John Wiley & Sons, Inc., New York.

FITZPATRICK, R., 1997 An investigation and analysis of current methods for measuring software usability. MSc submission, Stafforshire University.

FLAATTEN, P.O., 1989 Instructor’s Manual, Foundations of Business Systems, Dryden Press, Florida.

FOWLER, M., K. SCOTT, 1997 UML Distilled – Applying the standard Object Modeling Language. Addison-Wesley Reading, Massachusetts.

FROST, S. 1995 The SELECT Perspective. SELECT Software Tools, Inc. Santa Ana, CA.

GANE, C. 1990 Computer-aided Software Engineering, the methodologies, the products and the future. Prentice-Hall International Editions, New Jersey.

GANE, T., C.SARSON 1982 Structured Systems Analysis, McDonnell Douglas.

GANTI, N, W. BRAYMAN, 1995 Transition of legacy systems to a distributed architecture, John Wiley.

GIBSON, N., C. HOLLAND, B. LIGHT, 1998 “The business and Technology dimensions of legacy systems and outline of possible solutions” February SEPBC workshop, Durham University.

GOLD, N., 1998 The Meaning of Legacy Systems, SABA Project Report: PR-SABA-01 Version 1.1. The Centre for Software Maintenance, Lincoln School of Management, Lincolnshire.

GRAHAM, I.1995 Migrating to Object Technology, Addison-Wesley, Berkshire, UK.

HAASSE, V., R. MESSNARZ, G. KOCH, H. KUGLER, P. DECRINIS, 1994, "BOOTSTRAP: Fine-Tuning Process Assessment", July IEEE Software, pp 25-35.

HARTLEY, M. 1996 Application Series Scenario, Gartner Group November Conference Presentation, Cannes.

HATLEY, D.J, I.A. PIRBHAI, 1987 Strategies for Real-Time System Specification, Dorset House.

HENDERSON, C., N. VENKATRAMAN, 1993 "Strategic Alignment: Leveraging information technology for transforming organizations" IBM Systems Journal Vol. 32, No. 1, pp4 – 15.

HENDERSON-SELLERS, B., D. FIRESMITH, I.GRAHAM, I. OPEN metamodel Ver 0.4.

HILL, J. 1997 "From Data Access to Information Sharing", Gartner Group Symposium, November, Cannes, France.

HINQUAT, B., A.F. KELLEY, 1998 SAP R/3 Implementation Guide: A Manager's Guide to Understanding SAP. Macmillan Technical Publishing, Macmillan Publishing Company.

HOLZBLATT, A., S. JONES, 1993 "Contextual Inquiry: A Participatory technique for System Design" Participatory Design; Principles and Practice. Aki Namioka, Doug Schuler (Eds.), Hillsdale, N.J.

HUMPHREY, W. 1989 Managing the Software Process. Addison-Wesley, Reading, Massachusetts.

I-CUBE INTERNATIONAL 1998 "Migrating Legacy Applications to Open Systems: An Overview of *i*-Cube's Migration Services", IT Consulting White Paper, available at: http://www.i-cube.com/inews/migrate.htm

IEEE 1983, IEEE Standard glossary of software engineering terminology, IEEE Standard 729-1983, Institute of Electrical and Electronics Engineers.

IEEE 1993 IEEE Standards Collection: Software Engineering, IEEE Standard 610.12-1990, Institute of Electrical and Electronic Engineers.

ISO 9000-3 1991 International Standard. Quality management and quality assurance standards – Part3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software. International Organization for Standardisation, Geneva, Switzerland.

ISO/DIS 9000-3 1996 Committee Draft International Standard. Quality management and quality assurance standards – part 3: Guidelines for the application of ISO 9001 to the design, development, supply, installation and maintenance of computer software, International Organization for Standardisation, Geneva, Switzerland.

JACKSON, M.A., 1975 Principles of Program Design. Academic Press. USA.

KITCHENHAM, B., S. L. PFLEEGER 1996 "Software Quality – the Elusive Target", IEEE Software, January Vol 13, No.1.

LAUDON, K., LAUDON, J. 1998 Management Information Systems – New approaches to organization and technology. Prentice Hall, USA.

LEVEY 1995, Reengineering COBOL with Objects - Step by Step to sustainable legacy systems. McGraw Hill, USA

LINTHICUM, D. October 1996 "The ABCs of SAP R/3" DBMS, pp28,30,76, USA

LIU, K., A. ALDERSON, H. SHAH, A. DIX, 1998. "Using Semiotic techniques to derive requirements from Legacy systems" February SEBPC workshop, Durham University,

MACK, B., 1997, "An approach to IT transition planning", November Gartner Group Symposium, Cannes, France.

MAKOWSKI, D., 1995 “Enhancing Legacy Systems using Client/Server techniques”, University Management System, CU.

MARKOSIAN, L., P. NEWCOMB, R. BRAND, S. BURSON, T. KITZMILLER 1994 “Using an Enabling Technology to Reengineer Legacy Systems”. Communications of the ACM, Vol. 37, No. 5. pp 58-70. USA

MARTIN, J. 1991 Rapid Application Development, Prentice-Hall.

MARTIN, J. 1996 “Cybercorp - the new business revolution”, AMACOM Books, NY as summarised by him “Only the Cyber-Fit will survive” Datamation, November Newton MA, pp 60-66.

MARTIN K., 1996 “Automated wrapping of a C++ Library into Tcl”, Fourth Annual Tcl/Tk Workshop, University of California at Berkeley, USA.

MCCALL, J. A., P. K. RICHARDS, G.F. WALTERS 1977 Factors in software quality, vol. I-III, Rome Aid Defence Centre, Italy.

MCCARTHY, V 1997 “How to switch on your Intranet” Datamation, . February pp86-90.

MCDERMID, J., P. ROOK, 1993 “Software Development Process Models” in Software Engineer’s Reference Book, CRC Press, pp15/26-15/28.

MCGIBBON, B., 1996 “Embracing the tiger: Re-engineering Legacy Systems”, Object Expo Europe, Java expo Conference proceedings p153-9, Langley, Berkshire.

MENTZAS, G. N. 1997 “Re-engineering Banking with Object-Oriented Models: Towards Customer Information Systems”, International Journal of Information Management, Vol. 17, No. 3, pp. 179-197, Great Britain.

MYERS, G. 1978 Composite Structure Design Van Nostrand Reinhold.

NASSI, I., B. SHNEIDERMAN, 1973 “Flowchart techniques for structured programming” ACM SIGPLAN notices, 8 No.6.

NAUR, P., B. RANDALL, 1969 “Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee”, NATO.

NEUMANN, D., 1996 “Evolution process for legacy system transformation” IEEE Technical Applications conference Northcon pp57-62.

NEWCOMB, P. 1995 “Legacy system cataloging facility”. Proceedings of the Fourth Reengineering Forum, Victoria, BC.

NIERSTRASZ, 1992 “Component-oriented Software Development” Communication of the ACM, September Vol. 35, No. 9, pp160-165.

NING, J., A. EGBERTS, W. KOZACZYNSKI, 1994 “Automated support for Legacy Code Understanding” Communications of the ACM, May Vol. 37, No. 5, pp 50 – 57, Chicago

OMG (1997, March 21) “What is Corba” Available at: http://www.omg.org/omg00/wicorba.htm.

O’REILLY, A.,”What is Euro Compliant?” Sunday Business Post 20/9/1998, p31, Dublin.

ORFALI R., D.HARKEY, J.EDWARDS 1996 The essential client/server survival guide, Wiley & Sons, Inc., NY, pp389-340.

OVUM 1998 Ovum evaluates: CASE Products, Ovum Ltd. Summary available at: http://www.dpu.se/ovucasee.html

OVUM 1999 Ovum evaluates: ERP, Ovum Ltd. Summary available at: http://www.ovum.com/reports/ERP_for_manufacturers.htm

PANCAKE, C. 1995 "The projise and the cost of Object Technology: A five-year forecast" Communications of the ACM, Vol. 38, No. 10, pp.33-49.

PARKINSON, J., 1991 "Making CASE work", NCC Blackwell, England

PAULK, M. 1993, Capability Maturity Model for Software, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

PFLEEGER, S.L, R. JEFFERY, B. CURTIS, B. KITCHENHAM, 1997, "Status Report on Software Measurement", March / April IEEE Software.

POPKIN SOFTWARE 1996 System Architect User Guide, Popkin Software & Systems, inc., U.S.A.

PREECE, J., Y.ROGERS, H. SHARP, D. BENYON, S. HOLLAND, T. CAREY 1994 Human Computer Interaction Addison wesley, Wokingham, England.

PRESSMAN, R., 1997 Software Engineering A practitioner's approach. McGraw Hill, USA.

RAMAGE, M., 1998a "Report on the first SEBPC workshop on legacy systems" SABA project, Durham University.

RAMAGE M., 1998B "SABA Quarterly report 3: April – June 1998" SABA project, Durham University.

RAMAGE, M., 1998b "SABA Quarterly Report 5: October – December 1998" SABA project, Durham University.

RAMAGE, M. 1998d "Building a solution space for legacy systems – Report on the second SEBPC workshop on legacy systems", SABA project, Durham University. Also at http://www.dur.ac.uk/CSM/SABA/legacy-sig/report2.html.

RAMAGE, M., 1999 "Finding solutions to legacy problems – Report on the third SEBPC workshop on legacy systems" SABA project, Durham University.

RANSOM, J., I. SOMMERVILLE, I. WARREN, 1998 "A method for assessing Legacy Systems for Evolution" February SEBPC workshop, Durham University

ROBSON, W. 1997 Strategic Management and Information Systems. Pitman, Great Britain.

ROSSAK, W. 1991 "Some thoughts on System Integration: A Conceptual Framework", Journal of System Integration, USA.

ROYCE, W.W., 1970, "Managing the development of Large Software Systems: Concepts and techniques" August Proceedings WESCON

RUMBAUGH, J., M. BLAHA, W. PREMERLANI, F. EDDY, W. LORNESEN 1991 Object-oriented modeling and design, Englewood Cliffs, NJ: Prentice Hall.

SACHDEVA, S. 1995 "Metadata: Guiding users through disparate data layers" Application Development Trends, also available at http://www.inquiry.com/publication/adt/html/dec95/fe1205/ADT19951201FE1205.html

SCHULTE, R. 1996 "System Software Architecture Scenario" Gartner Group Symposium 96 November Presentation [CD-ROM]

SEBPC, 1998 SEBPC links www.dur.ac.uk/CSM/SABA/sebpc.html

SHNEIDERMAN, 1987 B., Designing the User Interface, Addison-Wesley.

SIMPSON, D., 1995 "Downsizing: Pull the Plug Slowly" Datamation July Newton MA, USA.

SLEE, C., C. SLOVIN, 1997 "Legacy Asset Management", Winter Information Systems Management, pp. 12-2.

SNEED, H., 1995 "Planning the reengineering of legacy systems" January IEEE Software.

SPICE CONSORTIUM 1994, The process improvement guide, issue .05. Spice consortium.

STEPHENS, W., G. MYERS, L. CONSTANTINE, 1974, “Structured Design”, IBM Systems Journal, vol 13, no. 2. pp. 115-139.

SYSTEM TECHNIQUES INC. 1995 (1), “Wrapping Legacy Systems for Reuse: Repackaging vs Rebuilding” http://www.systecinc.com/white/wplist.html

SYSTEM TECHNIQUES INC. 1995 (2), “Existing Systems Management, Repository Role in Legacy Migration”, Points of View White Paper, available at: http://www.systecinc.com/whit/whiteadt.htm, 12/05/1997.

TUCKER, M.J. 1997 “Bridge your legacy systems to the Web” Datamation, March pp114-121.

VAN HAELST, W., K. JANSEN, 1997 “Control and Audit of SAP R/3 Logical Access Security” IS Audit and Control Journal, Volume III, pp37-44.

VAN MULLIGAN, E., R. CORNET, T. TIMMERS 1995 “Integrating Legacy Systems in a Client-Server Environment”, Proceedings of AMICE, USA.

VON BERTALANFFY, L, , 1968 General System Theory, Braziller, New York.

WALSHAM G., 1993 Interpreting Information Systems in Organizations. Wiley, Chichester.p5, p166.

WARD, P. T., S.J. MELLOR, 1985 “Structured Development for Real-Time Systems” three volumes, Yourdon Press

WASSERMAN, A. 1980 “Principles of Systematic Data Design and Inplementation” in Software Design Techniques (Ed P. Freeman, A. Wasserman). IEEE Computer Society Press, pp. 287-293.

WATERS, R., E. CHIKOFSKY, 1994 "Reverse Engineering" Communications of the ACM, May Vol 37, No. 5, p23, Chicago

WHITESIDE, J., J. BENNETT, K. HOLTZBLATT, 1988 "Usability Engineering: Our Experience and evolution" Handbook of Human Computer Interaction, M. Helander (Ed.) NY.

WIRTH, N., 1971 "Program Development by stepwise Refinement", CACM, vol. 14, no. 4 pp 221-227, Chicago.

WU, B., D. LAWLESS, J. BISBAL, J. GROMSON, V. WADE 1997 "The Butterfly Methodology – A Gateway free approach for Migrating Legacy Information Systems" Third IEEE International Conference on Engineering of Complex Computer Systems. Proceedings pp 200-205.

YOUNG-GUL, K. 1997 "Improving Legacy Systems Maintainability", Winter Information Systems Management, pp 7-11.

YOURDON, E.N., L.L. CONSTANTINE, 1978 Structured design, Yourdon Press.

Bibliography

MOWBRAY, T.J., R. ZAHAWI, 1995 The essential CORBA – Systems Integration using distributed objects. John Wiley & Sons, USA.

SESSIONS, R. 1997 Microsoft's Vision for Distributed Objects. John Wiley & Sons, N.Y.

PRESSMAN, R., 1997 Software Engineering A practitioner's approach. McGraw Hill, USA