Legacy Information System Migration:

A Brief Review of Problems, Solutions and Research Issues

Jesús Bisbal, Deirdre Lawless, Bing Wu, Jane Grimson

Computer Science Department,

Trinity College, Dublin, Ireland

Abstract

The widespread use of computer technology over several decades has resulted in some large, complex systems that have evolved to a state where they significantly resist further modification and evolution. Although these Legacy Information Systems pose considerable problems (brittleness, inflexibility, isolation, non-extensibility, lack of openness, etc.), they may also be mission-critical: if one of these systems stops working the business may grind to a halt. Thus for many organisations, decommissioning is not an option. An alternative solution is Legacy System Migration that has recently become an important research and practical issue. Legacy System Migration is a relatively new field of research and few comprehensive methods or practical experiences have been reported. This paper provides a brief overview of existing research and practise when dealing with Legacy Information Systems, and in particular of the area of Legacy Information System Migration.

Keywords *- Legacy systems, legacy system migration, migration methodologies, software wrapping, software evolution, re-engineering, mission-critical systems*

1. Introduction

A Legacy Information System can be defined as “any information system that significantly resists modification and evolution” [Brod95]. Legacy information systems typically form the backbone of the information flow within an organisation and are the main vehicle for consolidating information about its business. A failure in one of these systems may have a serious business impact [Benn95]. These (mission critical) legacy information systems are currently posing numerous and important problems to their host organisations. The most serious of these problems are:

- these systems usually run on obsolete hardware which is slow and expensive to maintain;
- maintenance of software is generally expensive; tracing faults is costly and time consuming due to the lack of documentation and a general lack of understanding of the internal workings of the system;

- integration efforts are greatly hampered by the absence of clean interfaces;
- legacy systems are very difficult, if not impossible, to expand.

In response to these problems, several solutions have been proposed. They can be classified into the following three categories[1]: re-development, wrapping, and migration. Re-development involves rewriting existing applications (see section 2.1). Wrapping involves developing a software component called *wrapper* that allows an existing software component to be accessed by other components who need not be aware of its implementation (see section 2.2*). Legacy Information System Migration* (see section 2.3) *allows legacy systems to be moved to new environments that allow information systems to be easily maintained and adapted to new business requirements, while retaining functionality and data of the original legacy systems without having to completely redevelop them.*

A concept tightly related to legacy information systems, and to the processes of legacy re-development and legacy migration, is that of *Re-Engineering* (see section 2.1). This term is commonly used interchangeably with migration (e.g. [Snee96][Aebi97]) leading to a misunderstanding of these two concepts due to a lack of standardisation of the terminology to be used in a (relatively) new discipline. For the purposes of this paper, re-engineering is the examination [understanding] and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form [Chik90]. Therefore re-engineering ultimately leads to an (almost) complete re-implementation of the legacy system (e.g. [Gant95][Till96][Rena97a]). The resulting system may or may not run on a different environment. The distinctive aims of migration are to *avoid a complete re-development* of the legacy system, re-using as much of it (implementation, design, specification, requirements) as possible. In addition, the target system resulting from a migration process runs on a *different environment* (perhaps only a different programming language, or maybe a completely new architecture and technology). If most of the legacy system must be discarded (no re-usable components), the engineer will be facing a re-development project, not a migration project [Simo92]. In this paper, re-engineering is not seen as a solution to the legacy problem *per se*, but rather as a technology to be used in migration or re-development projects.

The remainder of the paper is organised as follows. The next section further discusses current approaches to the problems posed by legacy systems, as stated above, with special focus on legacy information system migration. Section 3 describes the main activities a general migration project must address, giving an overview of the migration process from a managerial point of view rather than from a technological point of view as done in the previous section. The concluding section presents a summary of findings and discusses a number of future research directions.

[1] *Maintenance* of the legacy information system has not been included as an additional category. See section 2 for a discussion.

2. Coping with Legacy Information Systems

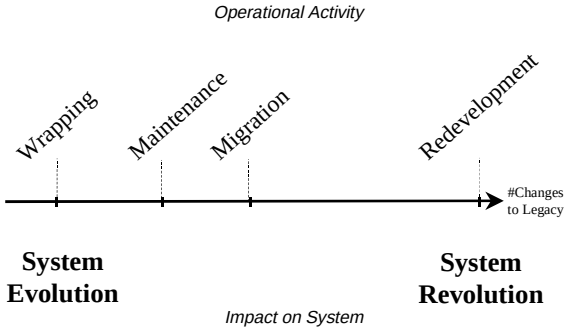

Figure 1. Approaches to Cope with Legacy Information Systems

Figure 1 shows the approaches, as given in section 1, commonly used to cope with legacy systems. *Maintenance* has been added only for completeness, since it is part of every system's life-cycle, be it legacy or not. In fact, if a software system can be maintained within an acceptable budget it is usually not considered a legacy system [Weid97]. Figure 1 illustrates two different issues regarding these approaches. Firstly, their impact (w.r.t. number of changes, with the cost and risk they involve) on the legacy system, with re-development being the option that leads to most important changes (system revolution). Secondly, the fact that they form a continuous spectrum of approaches, rather than four completely separate alternatives. Given a concrete legacy system problem, it is not always possible to decide whether the solution being applied belongs to only one of the four alternatives of Figure 1, and not to the others. For example, wrapping could be seen a maintenance activity which aims at making components oblivious to changes in the wrapped component; migration could involve wrapping part of the system, maintaining some other part, re-developing others, and migrating others, etc. It must be noted that although these approaches are described as being applied at a system level, in reality it is likely that they would be applied at a component level. Therefore, different *operational activities* (as named in Figure 1) could be applied to different system's components.

The approaches presented in this section can be divided into two categories. Firstly, those that require the legacy system to be shut down for a period of time while developing or cutting-over to its replacement target system (section 2.1). And lastly, those approaches that recognise that mission-critical legacy systems cannot be out of operation for any significant amount of time and provide a mechanism for the legacy system to remain operable throughout migration (sections 2.2 and 2.3). Within the second category, the way in which this mission-critical support is provided varies (e.g. gateway-based vs. gateway-free approaches), according to a compromise between complexity and flexibility of the approach.

2.1. Legacy Information System Re-development

The Big Bang approach, also referred to as Cold Turkey [Brod95], involves re-developing a legacy system from scratch using a modern architecture, tools and databases, running on a new hardware platform. For any reasonably

sized system it is clear that this is a huge undertaking.

Renaissance is an ongoing research project that aims to develop a systematic method for system evolution and re-engineering[2] [Rena97a]. It defines a set of activities and tasks to support an overall re-engineering project and the flow of control between the identified activities, which drives the co-operation among tasks. Renaissance identifies generic activities that should be specialised for the organisation applying it and the system to which it is being applied.

Tilley proposes a high-level framework for legacy system re-engineering from several perspectives [Till96]. From each perspective, the problem of re-engineering is divided into phases and the issues to be addressed in each phase are identified. Although this approach does offer some guidance, it is far too high-level to be applied in practice. Ganti and Brayman [Gant95] propose general guidelines for migrating from a centralised legacy environment to a distributed environment. Selected business processes are re-engineered as required and then linked to legacy systems which have data and business logic of value in the new target environment. New applications are then developed to fit these processes. This approach recognises that legacy system migration should cause as little disruption to the current business environment as possible. However, it is not clear how the cut-over to the new, separately developed, target system will be handled.

In reality, the risk of failure is usually too great for a re-development approach to be seriously contemplated. Apart from the failure risks, another very real concern arises from the constantly changing technology and business requirements [Grah94]. Thus, organisations could find themselves in a position where the re-developed system no longer meets their business needs and the technology used is considered obsolete when finally finished.

2.2. Wrapping Legacy Information Systems

For many organisations, the complete re-development of legacy systems is not an option and they are forced to seek alternative ways of dealing with their legacy systems. Current practical solutions mainly adopt what is referred to as *Wrapping* [Grah94]. Wrapping involves surrounding existing data, individual programs, application systems, and interfaces to give a legacy system a 'new and improved' look or improve operations ([Weid97][Wins95]). The wrapped component acts as a server, performing some function required by an external client, which does not need to know how the service is implemented [Snee96]. Wrapping permits re-using well-tested components, trusted by the

[2] Re-engineering has been included in this (re-development) section because most approaches to re-engineering, as those described here, ultimately propose that the legacy system be completely re-implemented. For this reason re-engineering is seen as being closer to re-development than to migration, which aims at avoiding the long, costly and risky process of implementing a complete legacy system. In addition, although a re-development process would involve developing the system from scratch (requirements analysis, design, etc.), in practice it also involves the understanding of the existing system, and therefore involving much of re-

organisation, and leveraging the massive investment done in the legacy system for several years.

The most popular implementation of wrapping is *Screen Scraping*. Screen Scraping is the process of replacing the character based front ends of legacy systems with a client based graphical user interface [Merl95]. Putting a graphical user interface onto a legacy system is a cheap and effective method of leveraging legacy data onto the desktop. Users are then free to use graphical data manipulation and input tools common on the desktop to input data and process the system output. Despite the commercial success of screen scraping it is still very much a short-term solution. Placing a graphical interface onto a legacy system does not address many of the serious problems faced by legacy systems. Problems of overloading, inability to evolve to provide new functions and inordinately high maintenance costs are all ignored. In many cases screen scraping actually compounds an organisations maintenance problems. Wrapping legacy system access with screen scraping software adds a functionally superfluous layer to the legacy system which will itself have to be maintained.

2.3. Legacy Information System Migration

In situations where re-development is unacceptably risky and wrapping does not represent a suitable alternative due to its limitations, migrating the legacy system to an open environment should be considered. This represents a much more complex undertaking but, if successful, the resulting long term benefits (see section 3.1) are also greater (e.g. more flexibility, better understanding of the system, ease of maintenance, reduced costs).

Although legacy information system migration is a major research and business issue, few comprehensive approaches to migration have been developed. Given the bewildering array of legacy systems in operation and the problems that they pose it seems unlikely that a generic migration method suitable for all legacy systems is possible. However, a set of comprehensive guidelines to guide migration is essential. Before embarking on a migration project, an intensive study needs to be undertaken to find the most appropriate approach of solving the problems the legacy system poses. To the best of authors' knowledge, no successful, practical experiences of real migration projects, following a comprehensive migration approach have yet been reported. The few (successful) migration-like reports found in the literature ([Ocal96][Aebi97]) describe ad-hoc solutions to the problem at hand.

This section first discusses important practical issues to be addressed during any migration project. It then analyses two proposed methodologies that achieve a different compromise between flexibility and complexity.

---

engineering activities.

2.3.1. Legacy Information System Migration Issues

The essence of legacy system migration is to move an existing, operational system to a new platform, retaining the functionality of the legacy system while causing as little disruption to the existing operational and business environment as possible. This is a significant challenge that could quite legitimately encompass a large number of different areas of software engineering: program understanding, database understanding, system development, testing, etc.

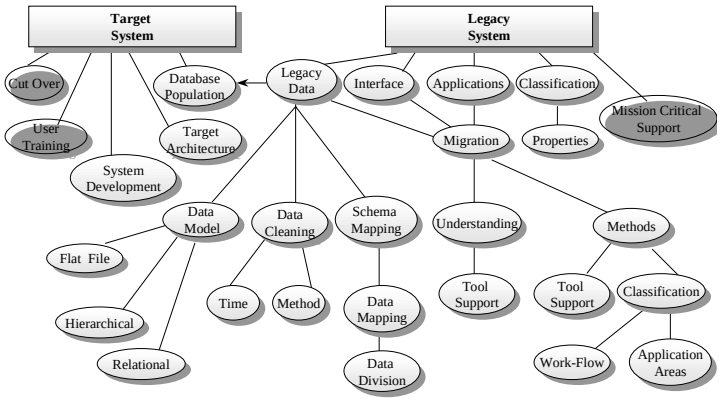

Figure 2. Classification of Legacy Information System Migration Issues

In legacy system migration some issues are common to any software engineering project (e.g. target system development, testing, and database model selection), and have been extensively researched and are widely supported. Other issues, however, are specific to migration (e.g. target system cut-over with mission-critical support, target database population) and have not been the focus of much research as yet. Figure 2 shows a classification of the most important practical issues that must be addressed during a migration project, and will be briefly discussed in this section. These issues can be divided into two broad categories: those that concern the legacy system and those that concern the target system.

Before any attempt is made at developing a target system, the legacy system must be understood. This involves *understanding* the *legacy data*, *interfaces* and *applications* for which tool support is required (see section 3.2). Although there is some tool support available, often tools may have to be developed to fit individual legacy and target systems [Aebi97]. A classification of the type of system being migrated could be investigated, *properties* identified and migration guidelines for each type of legacy system developed (e.g. [Stev98]).

The target *database* must be *populated* using *legacy data*. Firstly the legacy *schema* must be *mapped* onto the target schema ([Bati86][March90]), and the required transformation must be worked out. *Data mapping*, at instance level ([Davi97][Fong97]), must also be performed before data is migrated. The possibility of migrating the data in separate steps by *dividing* the *data* into independent chunks is also a challenge (see section 3.5). Legacy data is poor quality data ([Fox94][Aebi94]), therefore a migration project may aim to perform also some *data cleaning* [Aebi97]. The

method (e.g. [Kuki92]) used to clean the legacy data, and also the *time* at which the data is cleaned (i.e. before, during, or after migration) must be decided.

The *cut-over* of the target system represents the last step of a migration project. When migrating a mission-critical, large sized legacy system, this step cannot involve a simple switching off the legacy and turning on the target (see section 3.5). The cut-over process must minimise failure risks and maybe include the possibility of returning to the legacy as a failsafe option.

Some solutions to the legacy system problem assume that a legacy system can be taken out of operation during migration (see section 2.1). Others, however, recognise that some legacy systems are mission-critical and thus cannot be shut down for any significant amount of time (see sections 2.3.2 and 2.3.3). Providing *mission-critical support* is also an issue to be considered when facing legacy migration.

2.3.2. Gateway-based Approach: Chicken Little Strategy

Brodie and Stonebraker have proposed the *Chicken Little* strategy which aims to provide support for mission-critical legacy systems by allowing the legacy and target systems to interoperate during migration. This interoperability is provided by a module known, in general, as a *Gateway*, "a software module introduced between operation software components to mediate between them" [Brod95].

The *Chicken Little strategy* outlined in proposes an 11 step plan to be followed in any migration project:

Step 1: Incrementally analyse the legacy information system

Step 2: Incrementally decompose the legacy information system structure

Step 3: Incrementally design the target interfaces

Step 4: Incrementally design the target applications

Step 5: Incrementally design the target database

Step 6: Incrementally install the target environment

Step 7: Incrementally create and install the necessary gateways

Step 8: Incrementally migrate the legacy databases

Step 9: Incrementally migrate the legacy applications

Step 10: Incrementally migrate the legacy interfaces

Step 11: Incrementally cut-over to the target information system.

Chicken Little is a refinement of the Composite Database Approach [Brod93] (previously called *Stepwise Migration* in [Simo92]). Using this strategy, legacy applications are gradually rebuilt on the target platform using modern tools and technology. Initially the target system will be quite small but will grow as the migration progresses. Eventually the target system should perform all the functionality of the legacy system, which can then be retired. During migration, the old legacy and its target system form a composite information system, as shown in Figure 3.

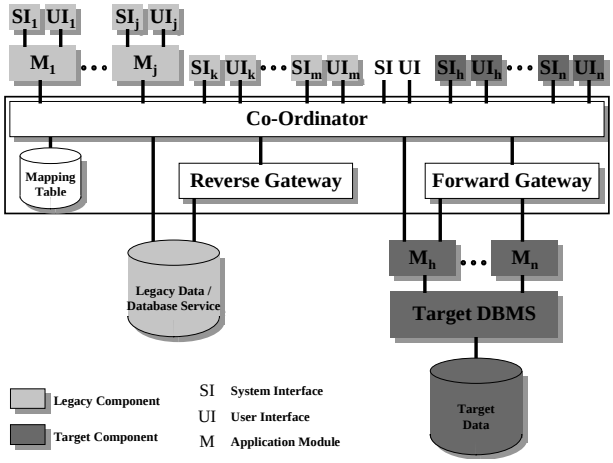

Figure 3. Composite Database Approach

A Forward Gateway is used to translate and redirect calls forward to the target database service and translate results returned by the target database for use by legacy applications. A Reverse Gateway is responsible for mapping the target data to the legacy database. This mapping can be complex and slow, which will affect the new applications. Also many of the complex features found in relational databases (integrity, consistency constraints, triggers etc.), may not be found in the archaic legacy database, and hence cannot be exploited by the new application.

The approach may involve data duplicated across both the legacy and the target databases. To maintain data integrity and consistency a Transaction Co-ordinator is employed which intercepts all update requests from legacy or target applications and processes them to identify whether they refer to data replicated in both databases. If they do, the update is propagated to both databases using a two-phase commit protocol as in a distributed database [Bell92]. As Brodie and Stonebraker themselves point out: "update consistency across heterogeneous information systems is a much more complex technical problem with no general solution yet advised, and it is still an open research challenge" [Brod95]. Thus, it seems that applying the *Chicken Little* approach would pose a big challenge to the migration engineer.

2.3.3. Gateway-Free Approach: Butterfly Migration Methodology

The Butterfly methodology [Wu97] assumes that while the legacy system must remain operable throughout migration, it is not necessary for the legacy and target systems interoperate during this process. This assumption leads to the elimination of gateways, avoiding the massive complexity they involve.

It is commonly agreed that the successful migration of the data management service from the legacy data to the

target is the key to overcoming many of the problems posed by legacy information systems ([Brod95][Aebi94]). Thus Butterfly methodology gives particular focus to the migration of legacy data in a mission-critical environment. The development of the target system is completely separate to the data migration.

Once data migration commences, the legacy data store is "frozen" to be read-only. Manipulations of legacy data are redirected by the Data-Access-Allocator (DAA), Figure 4, and the results stored to a series of auxiliary datastores named TempStore(s) (TS). When legacy applications access data, DAA retrieves data from the correct source, i.e. the legacy data or the correct TempStore.

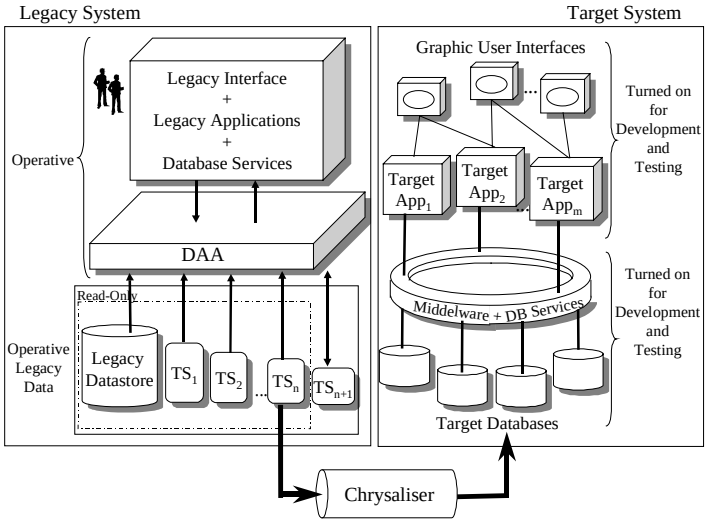

Figure 4. Butterfly Methodology, Migrating the Data in TempStore TS_n

A Data-Transformer, Chrysaliser, is employed to migrate the data in the legacy system as well as in the TempStores to the target system. When Chrysaliser is migrating the legacy data all manipulations are stored in TS_1; when migrating the data in TS_1, all manipulations are stored in TS_2; and so on. If the time taken to migrate a TempStore is faster than that taken to build the next one, the size of each TS will decrease at each iteration.

Figure 4 shows a scenario during the legacy data migration. The combination of DAA and Chrysaliser serves as a data migration engine for the legacy data migration. The legacy system operates normally throughout migration until the size of the last TempStore has reached a pre-determined threshold value, so that the amount of time necessary to migrate this last TempStore is sufficiently small to allow the legacy system to be brought down without causing any serious inconvenience to the core business. Using the Butterfly methodology, the legacy system will never be inaccessible for a significant amount of time and at no time does the legacy system need to inter-operate with the target system.

3. Legacy Information System Migration Process

This section provides an overview of the migration process from a managerial point of view, rather than from a technological point of view as done in section 2.3.

For the purposes of this paper, the following phases for a generic migration process have been identified:

Phase 1: Justification

Phase 2: Legacy System Understanding

Phase 3: Target System Development

Phase 4: Testing

Phase 5: Migration

Each of these phases will be discussed, providing a general description of the objectives and expected outputs of each phase.

3.1. Justification

Legacy system migration is a very expensive procedure that carries a definite risk of failure. Justifying a migration is more difficult than for an ordinary application development as, in many cases, migration may be viewed as a last resort. Consequently before any decision to migrate is taken, an intensive study should be undertaken to quantify the risks and benefits, and fully justify the development of the legacy system involved [Snee95]. The projected target environment needs to be considered carefully in terms of the legacy system functionality and legacy environment to identify expected benefits in both the long and short term. The costs involved in the actual migration should be weighed against these benefits to provide a significant cost benefit analysis and an estimation of the possibility of failure. A complete planning of the whole project is imperative if it is to succeed ([Benn95][Simo92]), and will also support its justification.

The RENAISSANCE project [Rena97a] aims to provide support for justification, assessing cost, risks and benefits of a software engineering project. It is mainly concerned with re-engineering rather than migration.

3.2. Legacy System Understanding

The system resulting from a migration must meet some business/user requirements but as the legacy system already partially meets these requirements, it is essential to the success of the migration that the functionality of the legacy system and how it interacts with its domain be understood. A poor understanding of the legacy system will lead to an incorrect specification of requirements for the target system and ultimately a failed migration project.

As many legacy systems have poor, if any, documentation ([Benn95][Simo92]), many research projects have chosen to focus on the area of recreating documentation from legacy code (e.g. [Bigg94][Wong95][Chin96][Rich97]).

Although tools can considerably reduce the amount of time needed to understand a migrating system, the automated understanding of a system's structure is still far from being achieved [Bigg89]. In order to recover a relevant design and documentation, much interaction from a system expert is required. Once the design and documentation have been constructed, they still have to be understood. Reusable components and redundancies have to be identified before the requirements for the target system can be produced. In addition, interactions between the legacy system and other information systems and resources must be identified. Failure to do so can result in expensive failures in related systems. This interaction is often unclear from the code or documentation, adding another source of complexity to this phase.

As well as understanding legacy applications, the structure of the legacy data must also be uncovered ([Bach88][Samp93][Hain96][Anti96]). Many different data models have been proposed and implemented over the years including the hierarchical, network, relational, extended relational and, most recently, the object-oriented data model. The database understanding becomes much more complex in case of the still very prevalent standard file format, for which there is no centralised description of the data structures, but they are buried in the application code. Individual source programs must be examined in order to detect partial structures of files.

As with applications, once the legacy data structure is extracted it still has to be understood. A general lack of standardisation with regard to naming and structuring of data elements combined with implementation specific optimisation components make the understanding process very difficult to automate. Once the structure has been understood redundancies have to be identified so that only necessary data will be migrated [Aebi97].

3.3. Target System Development

The main goal in a migration project is to build a fully operative, functionally equivalent (see section 3.4) system in an open environment. Once the legacy system is understood a requirements specification can be prepared. A target system developed according to this specification will have the same functionality as the legacy system. Decisions have to be made with regard to the architecture which should be chosen for the target system. A primary design intention of this architecture should be to facilitate maintenance and extension in the future, so that the developing target system is not the legacy system of the future. Currently, it is believed that client/server architectures lead to open applications which will not suffer the same deficiencies that current legacy applications (see section 1), making these the desired target architectures. Also, due to the mission-critical nature of many legacy systems, transaction support will be a basic requirement for the target, enterprise-wide applications. These applications seek a secure and distributed environment,

where many users can access simultaneously and efficiently a diversity of data sources and applications [Bell92]. Finally, giving the growing importance that the World Wide Web (WWW) has in the way companies do business, the target architecture should facilitate WWW's integration within the enterprise-wide information system [Perr95].

An extensive literature exists regarding possible architectures and criteria to select between them ([Berg98][Rena97b][Orfa96][Orfa94]). This issue is outside the scope of this paper and will not be further addressed here.

3.4. Testing

Testing is an ongoing process throughout the migration of a legacy system. Up to eighty percent of a migration engineer's time could quite legitimately be spent testing [Dede95]. Due to the mission critical nature of the legacy information systems, it is imperative that there are no inconsistencies between the output of a legacy system and that of its replacement system. It is not advisable to introduce new functionality to the target system as part of the migration project ([Beiz90][Simo92][Snee95]). By keeping the same functionality, a direct comparison of outputs is sufficient to determine the validity of the target system.

A practical issue related to the justification of a migration project is that in order to justify the expense and risk of a migration project, it is likely that the target system will be required to offer new functionality. In this case, the legacy system should be migrated first. New functionality can be incorporated into the target system after the initial migration has been performed [Snee95] (such an approach has already been successfully implemented and presented in [Beiz90]). Even when the functionality is unchanged, factors such as a change in the hardware, operating system, or compiler might result in inconsistent output for the same source code. All of these factors have to be addressed in the testing phase.

A large number of tools can be found to support testing [Ovum96] in a migration process, since those are needed in a generic software engineering project.

3.5. Migration

The migration phase is concerned with the cut-over from the legacy system to the target system. When dealing with mission-critical legacy systems, this process must cause as little disruption to the business environment as possible. Three different transition strategies have been proposed [Simo92]:

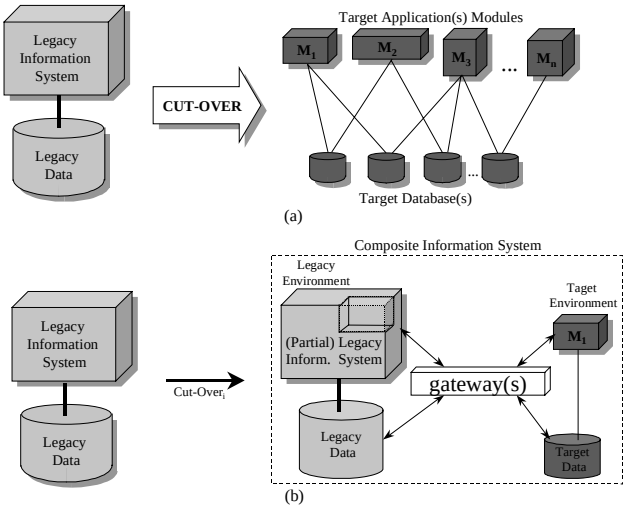

Figure 5. Cutting-Over the Legacy Information System

1. Cut-and-Run Strategy: it consists of switching off a legacy system and turning on a new feature-rich replacement, as shown in Figure 5(a). This is, in many cases, not a realistic option ([Brod95][Simo92]). Cutting-over to the target system in one single step represents too high a risk for many organisations as it would result in the whole information flow being managed by an untried and thus untrusted system.
2. Phased Interoperability: to reduce the risk involved in this phase, the cut-over should ideally be performed in small, incremental steps. Each step should result in the replacement of only a few legacy components (applications or data) by corresponding target components. The example illustrated in Figure 5(b) represents a step where only a small part of the legacy functionality has been cut-over to the target, the rest remains in the legacy. The migration approach presented in section 2.3.2 was developed with this transition strategy in mind.
3. Parallel Operations: both systems, legacy and target, can be kept in operation for some time. If so, all operations are performed in both systems, considering that the legacy system works correctly the target would be under continuous testing. The legacy system would be retired only when the target system is trusted by the organisation.

It must be noted that a concrete transition strategy designed for a particular migration project would probably involve a combination of these approaches, applied to different components of the legacy system.

The incremental migration described above (Phased Interoperability) is a potentially highly complex process. For this method to be successful, it must be possible to split the legacy applications in functionally separate modules. This is not always possible for legacy systems, the vast majority of which are badly structured. The same problem arises if the legacy data is also incrementally migrated. It could be difficult, if not impossible, to discover which portions of the data can be migrated independently. In addition, the management of the whole process would also be a challenge as it

may involve dealing with heterogeneous environments, distributed applications and distributed databases ([Bell92][March90]). Each of these fields is still an open research area and results may not be mature enough to be used in a mission-critical environment.

Migrating a legacy system in an incremental fashion is designed to reduce the risk of the migration phase. However, its inherent complexity carries with it a high risk why may actually result in increasing the risk involved in migration. These two sources of risk must be balanced if the migration phase is to succeed. The migration phase is central to every migration project and much research is still required in this area.

4. Conclusions and Future Research

The problems posed by legacy systems are a roadblock to progress for many organisations. The challenges they provide must be addressed. In this paper some of the most important issues involved in dealing with legacy systems have been briefly outlined, and the most significant of the currently proposed approaches have been discussed, given special attention to legacy migration.

Given the scale, complexity and risk of failure of legacy system migration projects, it is clear that a well-defined, detailed methodology that can be easily implemented is essential to their success. Although legacy information system migration is a major research issue, there are few comprehensive migration methodologies available or even an agreed approach to the migration problem. To date few approaches to address the legacy problem have been offered and those that have been proposed are either too high-level or have yet to be applied in practice ([Till95][Rena97][Wu97]). Partial solutions such as ‘wrapping’ are widely adopted in real practice. Whilst such solutions may meet requirements for a period of time, they may actually cause more problems in the future by making the legacy system more complicated and, consequently, harder to manage.

A generic migration process proposing five stages which every approach for migration must address: justification, legacy system understanding, target system development, testing and migration, has been described. A brief outline of the major activities to be addressed during each stage has been given.

In summary, as the area of legacy system migration has not been paid serious attention by the research community until relatively recently [Benn95], research is very much needed into all aspects of migration. More specifically, apart from much needed research in legacy system understanding and constrained target system development, a great effort should, in the authors’ view, be devoted to:

- developing a sound migration process model

- developing a comprehensive migration methodology
- identifying the lower level migration activities and their workflow
- identifying and developing tools to support migration
- developing generic mechanisms for data migration engine
- case studies for migration based on currently available methods
- gaining experiences of real migration practices

5. References

[Aebi94] D. Aebi, R. Largo, "Methods and Tools for Data Value Re-Engineering", Proceedings International Conference on Applications of Databases. Lecture Notes in Computer Science 819, Springer-Verlag 1994, pp. 499-411.

[Aebi97] D. Aebi, "Data Re-engineering - A Case Study", Proceedings 1st East-European Symposium on Advances in Databases and Information Systems (ADBIS'97), September 1997, Springer-Verlag electronic Workshops in Computing, Ed.: C.J.van Rijsbergen.

[Anti96] J. M. Antis, S. G. Eick, J.D. Pyrce, "Visualising The Structure of Large Relational Databases", IEEE Software, 13(1), January 1996, pp. 72-79

[Bach88] Bachmann, "A CASE for Reverse Engineering", Datamation, July 1988, pp. 49-56

[Bati86] C. Batini, M. Lenzerini, S. B. Navathe, "A comparative analysis of methodologies for database schema integration", ACM Computing Surveys, 18(4), December 1986, pp. 323-364

[Beiz90] B. Beizer, "Software Testing Techniques", Second Edition, Van Nostrand Reinhold, 1990

[Bell92] D. Bell, J. Grimson, "Distributed Database Systems", Addisson-Wesley, 1992.

[Benn95] K. Bennett, "Legacy Systems: Coping with Success", IEEE Software, 12(1), Jan 1995, pp.19-23

[Berg98] J. K. Bergey, "System Evolution Checklists based on an Enterprise Framework", Software Engineering Institute, Carnegie Mellon University, February 1998

[Bigg89] T. J. Biggerstaff, "Design Recovery for Maintenance and Reuse", IEEE Computer, 22(4), July 1989, pp. 36-49

[Bigg94] T.J. Biggerstaff, B.G. Mitbander, and D.E. Webster, "Program Understanding and the Concept Assignment Problem", Communication of the ACM, 37(5), May 1994, pp. 72-82

[Brod93] M. Brodie, M. Stonebraker, "DARWIN: On the Incremental Migration of Legacy Information

Systems', Technical Report TR-022-10-92-165 GTE Labs Inc.

[Brod95] M. Brodie, M. Stonebraker, 'Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach', Morgan Kaufmann Publishers, Inc. USA, 1995

[Chik90] E. J. Chikofsty and J. H.Cross II, 'Reverse Engineering and Design Recovery: A Taxonomy', IEEE Software, 7(1), January 1990, pp. 13-17

[Chin96] D. N. Chin, A. Quilici, 'DECODE: A Co-operative Program Understanding Environment', Journal of Software Maintenance, 8(1), 1996, pp. 3-34

[Davi97] S. B. Davidson, A. S. Kosky, 'WOL: A Language for Database Transformations and Constraints', Proceedings 13th International Conference on Data Engineering, April 1997, pp. 55-65

[Dede95] G. Dedene, J. De Vreese, 'Realities of Off-Shore Reengineering', IEEE Software, 12(1), Jan 1995, pp. 35-45

[Fong97] J. Fong, 'Converting Relational to Object-Oriented Databases', SIGMOD Record, 26(1), March 1997, pp. 53-64

[Fox94] C. Fox, A. Levitin, T. Redman, 'The Notion of Data and Its Quality Dimensions', Information Processing and Management, 30(1), January 1994, pp. 9-19.

[Gant95] N. Ganti, W. Brayman, 'Transition of Legacy Systems to a Distributed Architecture', John Wiley & Sons Inc, 1995

[Grah94] I. Graham, 'Migrating to Object Technology', Addison-Wesley, 1994

[Hain96] J-L. Hainaut, J. Henrard, J-M. Hick, D. Roland, V. Englebert, 'Database Design Recovery', Proceedings 8th Conference on Advanced Information Systems Engineering (CAiSE'96), May 1996, pp. 272-300

[Kuki92] K. Kukich, 'Techniques for Automatically Correcting Words in Text', ACM Computer Surveys, 24(4), December 1992, pp. 377-439

[March90] S. T. March (Ed.), Special Issue on Heterogeneous Databases, ACM Computing Surveys, 22(3), September 1990

[Merl95] E. Merlo, P-Y Gagne, J-F. Girard, K. Kontagiannis, P. Panangaden, 'Reengineering User Interfaces', IEEE Software, 12(1), January 1995, pp. 64-73

[Ocal96] A. J. O'Callaghan (Ed.), 'Practical Experiences of Object Technology', Cheltenham : Stanley Thornes in association with UNICOM, 1996

[Orfa94] R. Orfali, D. Harkey, J. Edwards, 'Essential Client/Server Survival Guide', John Wiley and Sons, 1994

[Orfa96] R. Orfali, D. Harkey, J. Edwards. 'The Essential Distributed Objects Survival Guide', John Wiley and Sons, 1996.

[Ovum96] Ovum Ltd., 'Ovum evaluates. Software testing tools', London : Ovum Ltd 1996.

[Perr95] L. Perrochon, R. Fischer, 'IDLE: Unified W3-access to interactive information servers', Computer Networks and ISDN Systems, 27(6), 1995, pp. 927-938

[Rena97a] ESPRIT Project - Lancaster University, 'RENAISSANCE Project - Methods & Tools for the evolution and reengineering of legacy systems', http://www.comp.lancs.ac.uk/computing/research/cseg/projects/renaissance, December 1997

[Rena97b] RENAISSANCE Project, ESPRIT project Lancaster University, 'D5.1c Technology selection', http://www.comp.lancs.ac.uk/computing/research/cseg/projects/renaissance/D5.1C_introduction.htm, December 1997

[Rich97] R. Richardson, D. O'Sullivan, B. Wu, J. Grimson, D. Lawless, J. Bisbal , 'Application of Case Based Reasoning to Legacy Systems Migration', Proceedings 5th German Workshop on Case-Based Reasoning - Foundations, Systems, and Applications, March 1997, pp. 225-234

[Samp93] T. Sample, T. Hill, 'The Architecture of a Reverse engineering Data Model Discovery process', EDS Technical Journal, 7(1), 1993

[Simo92] A. R. Simon, 'Systems Migration - A Complete Reference', Van Nostrand Reinhold, 1992.

[Snee95] H. M. Sneed, 'Planning the Re-engineering of Legacy Systems', IEEE Software, 12(1), January 1995, pp. 24-34

[Snee96] H. M. Sneed, 'Encapsulating Legacy Software for Use in Client/Server Systems', Proceedings 3rd Working Conference on Reverse Engineering, November 1996, pp. 104-119

[Stev98] P. Stevens and R. Pooley, 'Software Reengineering Patterns', Proceedings of SIGSOFT'98 6th International Symposium on Foundations of Software Engineering, 1998.

[Till96] S. R. Tilley, D. B. Smith, 'Perspectives on Legacy System Reengineering', Software Engineering Institute, Carnegie Mellon University, 1996.

[Weid97] N. Weiderman, L. Northrop, D. Smith, S. Tilley, K. Wallnau, 'Implications of Distributed Object Technology for Reengineering', Technical Report CMU/SEI-97-TR-005, Carnegie Mellon University, June 1997

[Wins95] P. Winsberg, 'Legacy Code: Don't Bag it, Wrap it', Datamation, 41(9), May 1995, pp. 36-41

[Wong95] K. Wong, S. Tilley, H. Muller, M. Storey, 'Structural Redocumentation: A Case Study', IEEE Software, 12(1), January 1995, pp. 46-53

[Wu97] B. Wu, D. Lawless, J. Bisbal, J. Grimson, R. Richardson, 'The Butterfly Methodology: A Gateway-free Approach for Migrating Legacy Information Systems', Proceedings 3rd IEEE Conference on Engineering of Complex Computer Systems (ICECCS '97), September 1997, pp. 200-205