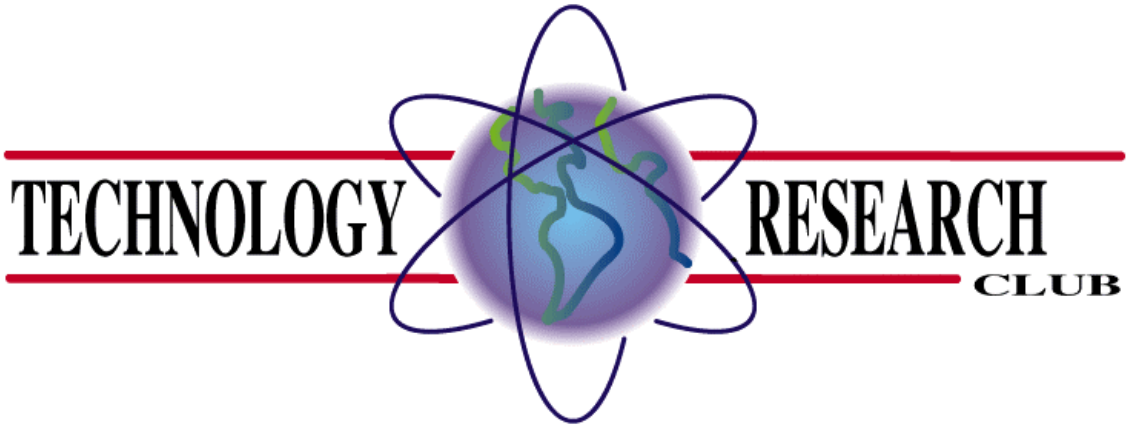


(Club de Investigación Tecnológica)

Legacy Transformation

Prepared by: Declan Good
August 2002

Since 1988 the Technology Research Club (Club de Investigación Tecnológica) has been carrying out applied research on the applications and implications of IT in organizations in Costa Rica. To date the Club has 60 members organizations, the most forward looking and successful private and public organizations in the country. This is the first report written in English and to be made publicly available, it is the 33rd research report. For more information on the Club, its members, the latest activities, as well as a complete list of research reports, visit www.cit.co.cr,

Edited and published by
Club de Investigación Tecnológica S.A.
Additional free copies from leda@cit.co.cr or www.artinsoft.com
San José, Costa Rica

About the Author

Declan Good has many years experience in maximising value from technology investment. At one time head of computer planning and research at the Canadian Customs and Excise, his subsequent career has focused on technology investment and technical architecture projects in both Canada and the United Kingdom.

He joined management consultants Woods Gordon in Ottawa (now Ernst & Young) after leaving the Canadian Government, and later worked with IT strategy consultants Butler Cox in the United Kingdom. He has been an Independent Consultant since 1992. Declan has degrees in engineering from Carleton University, Ottawa, and University College, Dublin. He can be reached at declan_good@compuserve.com

Acknowledgements

This research project was carried out in the United Kingdom for the Technology Research Club (Club de Investigación Tecnológica) with sponsorship from ArtinSoft.

I would like to thank Carlos Araya and Roberto Sasso of ArtinSoft for their help and encouragement in completing the research. Significant contributions were made by Donal Daly of Oracle, Rod McKenzie of ArtinSoft, and Martin Langham of Charteris. I should also like to thank the other individuals and companies who provided information and advice.

Contents

Synopsis

1. Introduction 3

Key issues addressed by the report 3

Organisation of the report 3

2. The Case for Legacy Transformation 4

3. The Transformation Process 6

Description of the process 6

Automating the transformation process 8

Overcoming inherent difficulties in transformation 9

Examples of tools 10

Implications 12

4. Developing a Strategy for Legacy Transformation 13

The charter of objectives 13

Portfolio assessment 14

Transform, replace, rewrite, or re-use? 15

Which target platform? 17

5. The Transformation Project 20

Outline of a project approach 20

Addressing the specific features of transformation projects 21

The need for an internal champion 23

6. Building the Business Case 24

"If it ain't broke, don't fix it" 24

Explaining software life-cycle investment 25

Cost, Value and Affordability 26

Tip of the iceberg 27

Bringing it all together.....29

7. The Supply Situation 30

Understanding the supply-side 30

8. Outlook for Legacy Transformation.....33

Further Reading

Glossary of Terms and Abbreviations

Synopsis

A legacy application is any application based on older technologies and hardware, such as mainframes, that continues to provide core services to an organisation. Legacy applications are frequently large, monolithic and difficult to modify, and scrapping or replacing them often means reengineering a organisation's business processes as well. Legacy transformation is about retaining and extending the value of the legacy investment through migration to new platforms.

Re-implementing applications on new platforms in this way can reduce operational costs, and the additional capabilities of new technologies can provide access to valuable functions such as Web Services and Integrated Development Environments. Once transformation is complete the applications can be aligned more closely to current and future business needs through the addition of new functionality to the transformed application.

In short, the legacy transformation process can be a cost-effective and accurate way to preserve legacy investments and thereby avoid the costs and business impact of migration to entirely new software. This report explains how transformation works and proposes a strategy for assessing the suitability of existing applications for migration to modern platforms such as J2EE and .NET.

The goal of legacy transformation is to retain the value of the legacy asset on the new platform. In practice this transformation can take several forms. For example, it might involve translation of the source code, or some level of re-use of existing code plus a Web-to-host capability to provide the customer access required by the business. If a rewrite is necessary, then the existing business rules can be extracted to form part of the statement of requirements for a rewrite.

The report takes the view that J2EE or .NET are suitable target platforms for transformation. The arguments in favour are based on technical and cost factors, on the fact that most automatic translation products target these platforms, on a growing skill-base in J2EE and .NET, making it easier to recruit staff, and on the availability of standard XML-based protocols for use by other applications, which facilitate the publication of application function to a network (usually referred to as 'Web Services').

Substantial automation of this transformation process is now feasible, making transformation an economically attractive proposition compared with rewriting or replacing the legacy application. The available tools cover all aspects of the process, although some manual intervention will be required. Using the tools in practice will depend on the scale of the task and whether automation is necessary or economic in every case. It is assumed of course that the existing applications are of sufficient quality and fit business needs well enough to make them worth transforming.

The tools available in the market are point solutions – they are applicable to specific scenarios and only handle part of the transformation process. Consequently there will be a need to buy in services to help design and execute the transformation as there are too many unknowns to be overcome without help from experts with previous experience. In-house resources will be needed to impart available knowledge about the legacy applications, and to build up the future knowledge base for maintaining the transformed applications and aligning them with business needs.

Transforming legacy applications is a task with both risks and rewards. It is easy to fall into the trap of relying on what seem like stable applications and hoping that they will be adequate to keep the business going, at least in the medium term. But these legacy applications are at the heart of today's operations and if they get too far out of step with business needs the impact will be substantial, and possibly catastrophic. The challenge for the CIO is to present the arguments for the legacy investment in the best possible light, but also to give management the full picture of these risks and rewards so that they can make a decision in full possession of the facts. Ultimately, legacy transformation is an 'enabling' project, that allows other things to happen, but has its own direct benefits as well.

In selecting a supplier or suppliers for a transformation project, it is best to strike a balance between the project-oriented players (who will take care of the transformation itself), and the infrastructure suppliers and in-house staff who have to make the end-result work every day. In today's state of the art, transformation expertise must be at the heart of the solution delivery. This can be done by appointing an independent project manager (in-house or contractor), and keeping functional changes and integration work separate from the tasks of code translation, data migration and associated testing.

In conclusion, the automated tools and techniques now available make legacy transformation technically and economically feasible. Replacement or rewrite are necessary in certain instances, but if the existing legacy application meets current business needs and the quality is good, then the chances are that the legacy asset can be effectively transformed to continue to meet the needs of the business in the future.

Conclusions

- It's not always necessary to scrap or replace legacy applications. Transformation is a feasible option if the current applications are of good quality and a reasonable fit to business needs.
- Automatic tools are available to migrate most data and code to modern platforms.
- If a total rewrite is required, there are tools available to help extract the existing business rules for use as input to the rewrite.
- Don't ignore legacy applications and hope they will go away. Carry out a regular audit on 'fitness for purpose'.
- Heads up on Web Services: Evolution of Web Services will put more pressure on to sort out legacy applications and make them accessible to customers and business partners. Start planning now.
- Transformation requires as much planning and business involvement as any other IT project.
- Involve in-house technical staff in the transformation project to ensure knowledge transfer – otherwise the organisation may be back where it started, with applications that no-one understands and are impossible to maintain.
- Don't expect in-house technical staff to beat the drum for transformation – most of them will prefer to rewrite the legacy application or replace it with the latest application package.
- Don't attempt transformation on your own first time out - choose a supplier with the appropriate transformation expertise to work with you.

1. Introduction

A legacy application may be defined as any application based on older technologies and hardware, such as mainframes, that continues to provide core services to an organisation. Legacy applications are frequently large, monolithic and difficult to modify, and scrapping or replacing a legacy application often means reengineering a organisation's business processes as well.

This report aims to explain the legacy transformation alternative, which maintains and extends the value of the legacy investment through migration to new platforms, and at the same time limits the need to reengineer existing business processes. It includes proposals for a legacy strategy and discusses transformation planning and cost justification issues. The report's intended audience include CIOs and their direct reports, systems integrators, and suppliers of commercial off-the-shelf application packages based on older technologies.

Key issues addressed by the report

1) What is legacy transformation?
2) How does transforming legacy applications help to meet business pressures for added functionality, responsiveness to change and improved cost-effectiveness?
3) How feasible is it, and is now the right time to do it?
4) What strategies should IS adopt for legacy applications?
5) What are the components of a legacy transformation project?
6) How should legacy transformation projects be managed?
7) How is legacy transformation to be financed?
8) Who should the purchaser look to for assistance?

Organisation of the report

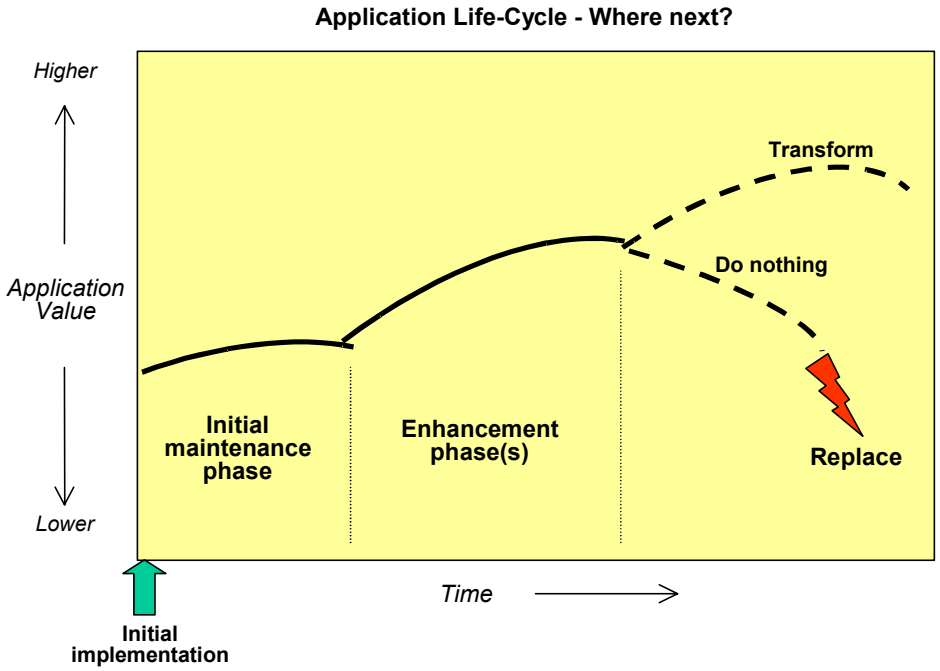
Chapter 2 considers the basic motivation for legacy transformation, and a description of the process follows in Chapter 3. Chapter 4 provides guidance on how to make a decision about when to go for transformation, and when to scrap or replace the legacy application (or in some cases, when to invest further). In particular, it reviews the difficult question of choosing a target platform. Chapters 5 and 6 deal with project planning and business case development, respectively. Chapter 7 discusses the supply-side options and makes some recommendations on choosing business partners to assist with transformation. Finally, Chapter 8 provides a general outlook for transformation. The Appendix contains a brief glossary of abbreviations and terms used in the report.

2. The Case for Legacy Transformation

Existing applications are the outcome of past capital investments. The value of the application investment tends to decline over time as the business and the technology context changes. Early in the life-cycle there will be enhancement investments to maintain close alignment with the business but eventually there will come a point where this becomes difficult. This can happen, for example, where the underpinning infrastructure is superseded, web access is required, or the weight of changes in the applications and lack of available know-how make it impossible to continue with enhancements.

Dissatisfaction with legacy centres on inflexibility (takes forever to make changes, can't make changes), maintainability (no documentation, no-one understands it, lack of skilled people), accessibility (can't make it available to customers, for example), cost of operation (runs on costly mainframe infrastructure, high license fees), and interdependency of application and infrastructure (can't update one without the other).

At this point we have a choice: Do we initiate a process of renovation and transformation, or do we write the application off and find a replacement?



Legacy transformation is about maintaining and extending the value of this legacy investment through migration to new platforms. Re-implementing applications on new platforms can have benefits through reduced operational costs, and through the additional capabilities of new technologies it provides access to valuable functions through more economical means. Migration to a new platform also provides an opportunity to align applications with current and future business needs through the addition of business functionality and through application restructuring.

Drivers for legacy transformation are operating cost reductions, mergers and acquisitions, internal reorganisation, new corporate infrastructure, need for Web-enablement, outdated

performance and functionality, data consolidation, and positioning for future changes such as B2B via XML and SOAP (Web Services).

Web Services may become a key factor in forcing change on legacy applications. Most organisations are in what might be referred to as the 'phase 1' stage of Web Services planning – perhaps running trials or simply assessing how important this concept will become in the future. Some are at 'phase 2', and are already exploiting the integration capabilities, often internally in their organisation. Realistically, Web Services could become a strategic issue in the near term, adding urgency to the need to take action on the legacy applications that will be at the heart of the Web Service infrastructure.

It is the position of this report that the tools and techniques to support automated transformation are now such that migration is both technically and economically feasible. Replacement and rewrite are necessary in certain instances, but if the existing legacy application meets current business needs, then the chances are that this legacy asset can be effectively transformed to continue to meet the needs of the business in the future.

When does an Application become a Legacy Application?

"A legacy application has been with the enterprise longer than the programmers who are now maintaining it, lacks good documentation, and has untouchable code" *Joe Celko,* IT Writer

"What's the definition of a legacy application? Answer: One that works." *Amey Stone,* Business Week

"Of course, the real definition of a legacy application is one that isn't Internet-dependent." *Amey Stone,* Business Week

"Legacy, in an IT context, is usually taken as referring to a mainframe application, although more recently even some client/server applications have been accorded this dubious accolade." *The Butler Group*

"People associate the term legacy with big iron and Big Blue, but the phrase is increasingly being used to include any and every application in existence before the birth of the Web." *Sarah L Roberts-Witt,* Writer on Internet infrastructure and services

"Although an information application may begin its life with a flexible architecture, repeated waves of hacking tend to *petrify* mature information applications…A application which has undergone petrification is termed a *legacy application*." *Anthony Lauder,* Consultant and *Stuart Kent,* University of Kent at Canterbury

"Old software still in use but which could benefit from re-engineering using more modern methods." *Princeton Internet Computer Dictionary*

3. The Transformation Process

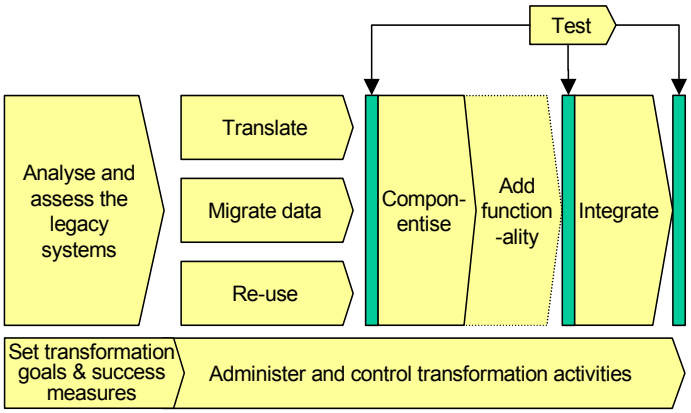
This section looks at current approaches to transformation and the tools available to help make it happen. It provides a model framework to assist in distinguishing between the wide range of products available in the market.

The basic requirement for a successful legacy transformation is to retain (and add to) the value of the legacy asset. In practice this transformation can take several forms, for example it might involve translation of the source code, or some level of re-use of existing code, with the addition of a Web-to-host capability to provide the customer access required by the business. It will be assumed throughout the discussion that the goal is to move to a commodity/open platform (such as J2EE or .NET) and that some additional functionality may be added in the process.

Description of the process

The diagram below is a model of the legacy transformation process including all the key activities involved. The sequence of activities can vary and there will be an iteration through the process when transforming a portfolio of applications, for example. It is usual practice to complete existing code translation, data migration and associated testing before adding new functionality. This is to test the end-result for equivalence with the existing application and to prove correctness of the translation and data migration before any changes are made to program logic and structure.

The input to the process is a legacy application and the outcome is a transformed application with most of the legacy asset intact, possibly enhanced, and integrated into the overall applications portfolio of the business. The process is subdivided into the several process steps necessary to bring about this transformation. It should be understood that, like most process models, there may be several 'paths' through these process steps, and there may be some iterations. For example, it may be decided to re-use some legacy code by 'wrapping' it in Java, while some other code is translated directly into Java – this involves different routes through the sub-processes. An example of iteration is when the transformation happens in stages: One part of the application is transformed to the point where it is integrated and goes live; then the process starts again to transform the next part of the application; more integration is required; and this process is repeated until all parts of the application have been transformed.



This model describes the process for transforming legacy applications – there are several possible routes through the process, and there may be some iterations before transformation is complete

This logic also applies to suites of applications that share the same data or deliver a common business function. Here it is best to transform the suite as a group, but if this is not feasible, then additional programming will be needed to provide the 'scaffolding' (adapters) required to keep the suite of applications operational during the transition phases. Typical activities making up these process elements are listed in the table below.

Sub-process	*Typical Activities*
Analyse and assess the legacy applications	▪ Inventory all application 'artifacts' – eg source code, copybooks, JCL, etc ▪ Application mining to extract business rules ▪ Map application elements ▪ Evaluate condition of source code ▪ Analyse database including, tables, views indexes, procedures and triggers, data profiling
Translate	▪ Refactoring (code structure improvement) ▪ Automatic code translation ▪ Manual adjustments
Migrate data	▪ Restructure ISAM to relational-type database schema ▪ Create relational environment ▪ Migrate all non-relational to relational according to a data model
Re-use	▪ Wrap COBOL code in Java (and expose only limited parts to Web processes) ▪ Extract business rules and objects and move into Java, XML ▪ Regenerate COBOL dialect to current standard
Componentise	▪ Extract components, align with business function ▪ Restructure into presentation, persistent data and business logic
Add functionality	▪ This is the stage where additional functionality can be added to meet specific business requirements not already met by the legacy application
Integrate	▪ Transfer executables, image files, Java etc to servers ▪ Web-enable ▪ Install data access link ▪ Performance tuning (as required) ▪ MQ and SOAP interfaces
Set transformation goals and success measures	▪ Agree goals with management ▪ Incorporate goals and measures into project plans ▪ Carry out periodic 'sanity checks' to ensure that fundamentals are being adhered to ▪ Measure after completion
Administer and control the transformation	▪ Version control and tracking ▪ Estimate effort required ▪ 'Due diligence' on feasibility of code conversion ▪ Establish metrics (eg of complexity) ▪ Document application (retro-document existing application and/or end result) ▪ Choose methodology to govern the transformation

Sub-process	*Typical Activities*
Test	▪ Use debugger ▪ Create test scripts ▪ Confidence testing ▪ Customer acceptance testing

Automating the transformation process

There is now a range of tools, products and standards available to assist with the transformation process as described above:

- Tools to help assess the complexities of the existing applications and to map/assess the code and interfaces.
- Tools to automatically translate code for leading languages.
- Tools to support conversion from legacy data structures to current relational databases.
- Products to help engineer the required B2B, Web and client-server structures.
- Standards such as XML and JCA (a standard for synchronous connection of J2EE to applications and transaction processors) that facilitate integration.
- Automatic analysis to help identify and develop standard components.

These tools, products and standards:

- *Cut time and costs* by reducing manual inputs and speeding up the transformation process.
- *Reduce risk* by using proven tools to ensure predictability and transparency of the process.
- *Preserve integrity of the legacy application* through acceptance criteria based on existing test cases.
- *Unify the existing applications* by migrating from several languages and/or platforms to one language and platform type.
- *Preserve legacy value* by transposing existing code to the new platform and ensuring that the conversion does not add new bugs.
- *Jump-start requirements definition* by providing an explicit requirements base-line for the existing applications on which to build the definition of new functional requirements.
- *Future-proof the solution* through targeting open platforms and improving the quality of the code and code structures so that the application can be readily adapted to fit changing business requirements.

Overcoming inherent difficulties in transformation

In considering the feasibility of automated tools for transformation, it is useful to consider this list of obstacles:

- The legacy application is monolithic. Client tier, persistent storage and application logic are intertwined.
- The legacy application is tied into operating-application-specific facilities, hindering portability to other platforms.
- The task at hand involves transforming several applications that are 'stove-piped', and possibly written in different languages, at different times, and by people that didn't speak to one another.
- The code is the only place where the business rules are documented, the rules are spread around the code, and there's no-one around who understands the application.
- The boundaries of the application are ill-defined - the source code is only part of the application. Its operation may depend on JCL for example, or in the case of 4GLs, on a run-time engine.
- Duplicate code; un-executed code; multiple copies of interfaces: Which is the 'correct' code? Are there hidden dependencies?
- Code quality is poor.

Specifically, this is how the obstacles listed above are addressed:

Obstacle	*How this is addressed*
Monolithic applications	Modelling the architecture of legacy applications separates out the client-tier, application logic and persistent data. Once migration of the code is complete, componentisation can begin, to align the components with business function. Representation of applications and data as components enables rapid and easy assembly of new business functions. Current state-of-the art limits componentisation to business functions as a general rule, and may be limited by the structure of the original code.
Application tied into OS-specific facilities	Translators include automatic translation of OS-specific program calls into the equivalent on the target platform. Where there is an unrecognised call, this is labelled as an exception, to be manually translated. If there are many occurrences a solution can be retrofitted into the tool by the supplier. An approach found in some translators is an 'analysis' run on sample code before translation commences, to identify the likely number of exceptions. If these fall into a small number of categories, the translator can be adjusted to deal with them automatically.
Incompatible stove-piped applications	The great advantage of the modern open application platform is its ability to integrate diverse applications, once the initial conversion of the code has taken place to the target platform. The evolutionary development approach adopted by most

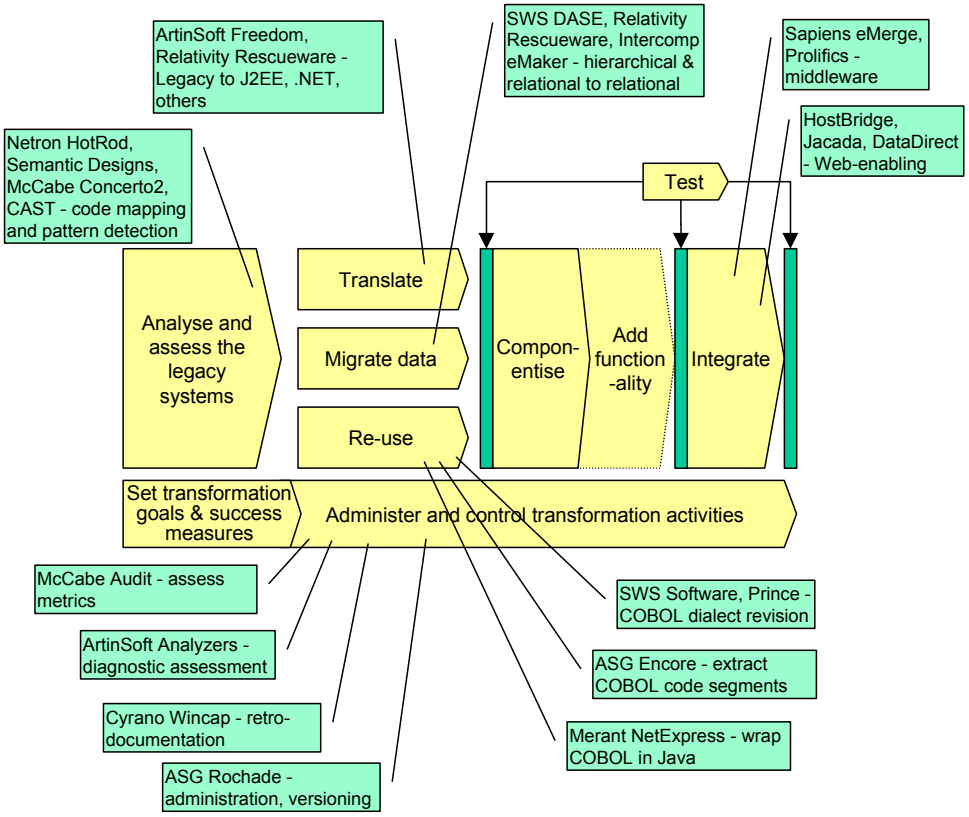
Obstacle	*How this is addressed*
	businesses means that transformation will happen in conjunction with existing applications, data and infrastructure. Transformed applications will have to coexist and interoperate with those applications.
Code holds the business rules and no-one knows them	Application mining can separate interface code, flow control, IO, and so on, from the small proportion of the code that represents business logic, thus isolating the business logic. The search for business logic can be narrowed down by automated searches for code constructs that typically embody business rules. This is followed by code-inspection workshops with the code maintainers. A complementary approach starts with the data whose value reflects the execution of a rule and trace the code that sets the value (this is most useful in Y2K, Euro type situations but can be generalised.) From these the implicit processes are mapped and ready for input to the definition of requirements or new functionality. In parallel with this effort, a know-building process is undertaken (more about this later in the report).
Where are the real boundaries of the application?	A combination of application mining and manual inspection enables the boundaries to be mapped. Specific steps are taken in 4GL translation, for example, to deal with run-time libraries and similar infrastructure. Transformation methodologies specify that to ensure functional equivalence, all code, interfaces, job control and data need to be considered. This preserves functional integrity and reduces testing.
Code issues – duplication, un-executed code, identifying the 'right' version of an interface, and so on	Sorting this out is a by-product of the application mining referred to above – recognising however that there are situations where the requirement is to simply move the code from an older platform, without any attempt to establish the business logic. Automated tools for persistent data conversion can produce data element definitions, determine which source records are used for data in tables in the target database and designate the source for the data in each table.
Code quality issues	Translation tools can applied to a sample of code to assess it. If the issue is fundamental than that, for example, the application does not function correctly, then it may be necessary to drop it as a candidate for transformation and look to extract the business rules for a rewrite. Automated inspection tools exist (such as Advantage) to assess structure).

Examples of tools

The diagram below shows the process model with examples of current products mapped onto it. This is a representative selection only and there are other products available and other suppliers operating in this space. The list does demonstrate that available tools cover the whole process in one way or another. It should be noted that most of these products are 'point solutions'. That is, they typically deal with one or two specific programming languages, and they are restricted to a small number of target

environments. For example, the translators are constructed to handle specific languages, although they can be extended fairly readily to handle other languages as required by the market.

Further, the products are generally limited to one part of the transformation process. This is an inevitable consequence of the structure of the transformation process, because although the transformation process is presented as a monolithic transformation, in fact the elements are quite different from one another, and the state changes are not at all consistent. For example, the Translate activities change code from one language to another, while the Analyse and Assess the Legacy Application activities change a state of (relative) 'ignorance' to one of 'knowledge'. Thus it is reasonable to suppose that the tools that support automation of these separate process elements will remain distinct (even when they are accessed through a common interface, for example).



This is not necessarily a bad thing, but it does mean that more than one tool is likely to be needed and some mixing and matching will be necessary - with sufficient expertise and good project management, a transformation project will succeed. However, it does suggest that some scepticism is required when a supplier talks about a solution that "will take care of everything".

There are limits on automatic decomposition of applications into components that could become building blocks for new applications. In general, current transformation techniques take the applications to a new platform, not to a new architecture. It is not easy to manually re-structure the transformed application to create components below the business function level. A feasible and realistic goal might be a target architecture

consisting of business objects that encapsulate the business logic of a single entity and the data particular to that entity. Ultimately, the feasibility of re-use depends on the structure of the original legacy application – well-shaped structured code lends itself more readily to some componentisation, whereas linear, monolithic code needs wrapping in total.

Implications

- Substantial automation of the process is feasible, making transformation an economically attractive proposition compared with rewriting or replacing the legacy application. This does assume that the legacy application is 'fit for purpose' in the first place.

- The available tools cover all aspects of the process, although some manual intervention will be required. Using the tools in practice will depend on the scale of the task and whether automation is necessary or economic in every case.

- Each of the tools available in the market only handles part of the transformation process. Consequently there will be a need in most cases to buy in services to help design and execute the transformation as there are too many unknowns to be overcome without help from experts with previous experience. Tools provision can vary – in some cases the code can be sent away to be translated, in others the tools are licensed on a per use basis. Some tools require operation by specialist staff.

- In-house resources must be involved throughout, at the beginning to impart available knowledge about the legacy applications, then at the testing and acceptance stage, and building up the future knowledge base throughout.

A strategy for legacy transformation is discussed in the next chapter.

4. Developing a Strategy for Legacy Transformation

Developments in legacy transformation tools and techniques provide an opportunity for the business to review its legacy portfolio. To transform or replace, to scrap of re-invest? These are the questions to be addressed by a legacy strategy.

When the requirement is to transform a single legacy application, then the choice of which way to go will be decided by the quality of the application, and whether or not there are off-the-shelf products readily available to replace it, if the quality is poor. In most other situations the CIO needs to begin with more fundamental business questions.

The charter of objectives

There are four common drivers for considering a transformation project:

- Reducing operating costs and the maintenance and overheads of older applications.
- Improving maintainability of the application in situations where no-one knows the application anymore, or where there is high staff turnover, lack of suitably qualified resources, or limited/outdated documentation.
- Improving access to the legacy application(s), often to provide Web access to customers and business partners, or as a result of a merger or organisational restructuring, when there will be new users and products to be added.
- Positioning for future projects (such as Web Services, or expected business change).

In practice some of these drivers may be combined. These drivers can be seen as a kind of spectrum, going from 'push' factors (which mean that the legacy application must react to fit a new environment) to 'pull' factors (the business wants to seize an opportunity to grow/reach new customers/add products and services).

	Business Driver			
	Internal focus ←		→ External focus	
	Deteriorating system	Economy	eBusiness	Get ready for change
Key business objective	Survival, operational continuity	Reduce operating costs	Extend reach inside business and/or external to business	Position the business for the future
Value dimensions	• Improved maintainability (documentation, easier to fix) • Access to support • Lower operating costs • Access to new customers (package supplier) • More adaptable system	• Reduced operating costs, licence costs, back-up/disaster recovery costs • Opportunity to outsource • Reduced complexity • More adaptable system	• Increased revenues • New customers/users • Better service to existing, new customers/users • Reduced customer acquisition costs • Brand enhancement	• Adaptable system • Closer integration with business partners • Extended services to customers • Web Services option • Re-use of components • Future-proofing

These drivers are contrasted in the diagram, suggesting that there can be quite a variation in the value that the business will be looking for from a transformation. The following points need to be considered:

- Drivers to the left of the model emphasise cost. The justification for the transformation project is based on survival and cost savings. This eliminates the possibility of scope creep from the transformation project (for example). The likelihood is that the underlying issues are the platform, languages and data structures – in other words, the architecture.

- Drivers to the right are about business opportunity, and the issues will be about functionality and future capability.

- There are different levels of business involvement. Obsolescent platforms and economy drivers are likely to be controlled by the IT department, and be internal projects for the business. E-business and re-positioning drivers will need to involve marketing, sales, product development and so on, as well as business partners and customers in some cases, and so have an external focus.

- ….this may also determine who the real customer is – IT or the business.

- Where a third-party package is contributing to the deteriorating application situation there will be business involvement in the investment decision – the need to decide on target platform for example.

- Where cost savings are the primary driver, then outsourcing may be an added consideration, combined with transformation of the legacy application.

Portfolio assessment

In any business there is the obvious distinction between financial, sales order processing, human resources and manufacturing applications, for example, but of course within these categories there are often several hundred individual applications with millions of lines of code between them. A pre-assessment of the portfolio is desirable before proceeding to make individual decisions about transformation methods.

The emphasis of the pre-assessment is on *business* value (one of the aims of transformation is to address technical quality, so this question is postponed until later in this process). The goal is to streamline the applications portfolio by reviewing the existing applications to see whether they continue to provide business value. If applications can be identified for decommissioning this will result in immediate savings. This streamlining will require commitment from the business. Here is a starter list of questions to ask about each application:

- Does it directly impact customers or business partners? If this application stopped running tomorrow, would it have any impact on the business?

- When was the last time the users of this application were asked about its value? What would be their answer if they were asked today?

- Is there more than one application capable of providing the information?

Transform, replace, rewrite, or re-use?

There are alternatives to total transformation of the legacy application, and at this point in the development of the transformation strategy these alternatives should be reviewed. The main decision factors are (I) the quality of the legacy application, and (ii) the availability of replacement packages. 'Quality' in this case is a subjective term applied to the application itself, regardless of the platform it runs on or the code in use, and should be assessed in terms of such parameters as:

- Current effectiveness (eg errors generated, number of workarounds, level of support needed).

- Stability of core business rules – will the application logic stay much the same in the medium-term? There is an underlying assumption in legacy transformation that the current software asset is a valuable one. If the business model is going to change substantially then this assumption has to be called into question. Having said this, in practice the code is often the only repository of business rules and these are scattered throughout the code. Thus any attempt to 'start from scratch' needs to re-construct and document the requirements captured in the legacy code and take these requirements as the starting point for the negotiation of new requirements (using application mining).

- Gaps in the functionality.

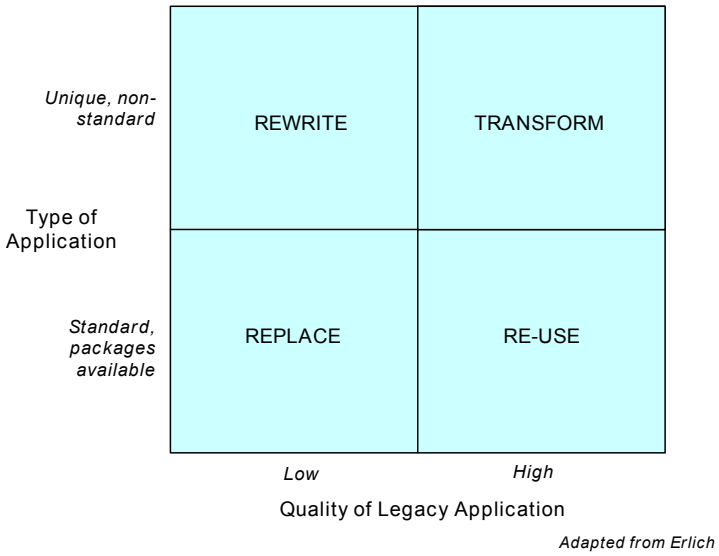
- Stage of the legacy life-cycle – in the earlier stages of the life-cycle, a 'legacy' application will likely map closely to functionality requirements, although the platform is obsolescent.

In summary, the 'quality' assessment is about the suitability of the legacy application in business and technical terms.

The availability of a replacement package is the other important consideration and this will depend on the uniqueness of the current application. If the quality of the legacy application is poor and there is comparable functionality available in a third-party software package , it makes sense to replace it.

There are four broad options: transform, re-use, rewrite or replace. As explained earlier, some or all the elements of the transformation process apply to the first three. The diagram[1] overleaf shows the combinations of factors that might lead to these four options.

[1] Diagram adapted from a presentation by Len Erlikh of Relativity Technologies, Inc



Transform – Apply the transformation process, adding functionality and business reach as required.

Re-use – There are two possibilities here, one where the legacy application is centred on a third-party package/DBMS already, and the other where the business has developed its own application from scratch. If the legacy application portfolio is largely centred around a third-party package then the best way forward may be to upgrade to the latest version and use wrapping techniques to provide the required reach and other functionality improvements. For in-house applications consider wrapping the application. To provide direct access to data by end-users without going through the legacy application will usually mean adding a back-end data warehouse as well. The drawback of this approach is that it adds more elements to be maintained and two sets of data to be kept synchronised.

Rewrite – The key asset here is the business rules and data structures – the application is the problem. Application mining and analysis of code logic and data structures is required to provide the starting point for the rewrite.

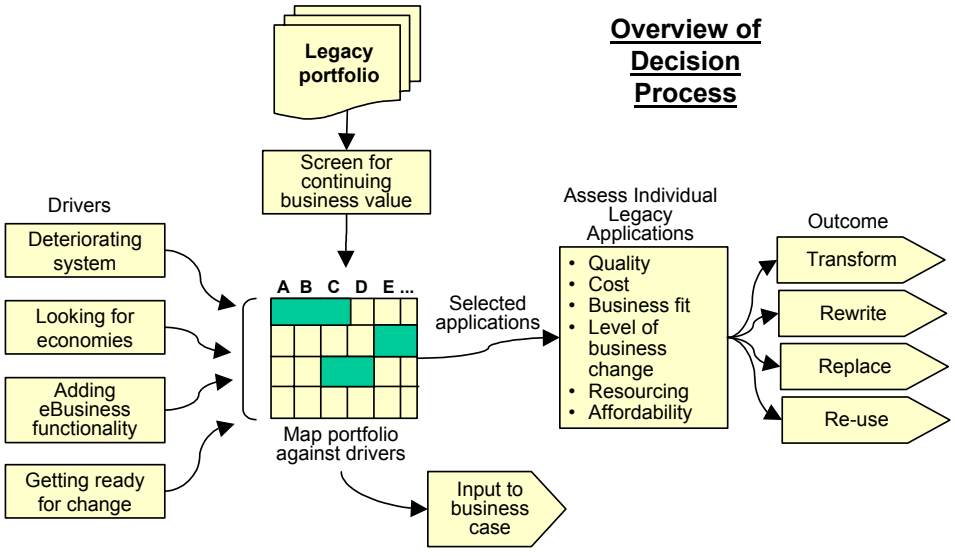
Replace – Look for a suitable package or outsource. Be prepared to make changes to the business model to meet the package half-way.

Checklist for choosing legacy transformation

- Good fit of existing application with business needs
- Moderate functionality changes needed in existing application
- Significant functionality to be added in a new application and close integration with existing required
- High operational costs of existing application
- Need to migrate to J2EE or .NET for strategic reasons
- Future vision includes Web Services
- Adding Web access
- Difficult to find resources to maintain amend applications on existing platform

Checklist for choosing to re-use
▪ Business rules satisfactory
▪ Low operational costs of existing application
▪ Difficult to separate logic from persistent data and presentation layers
▪ Simple Web access required, allowing a wrapping solution
▪ Have resources to keep core legacy maintained
▪ Off-the-shelf software central to existing, rely on 3rd party to support, maintain
Checklist for choosing to rewrite
▪ Business rules satisfactory but needs extensive functionality added
▪ No off-the-shelf solution comes close to meeting needs
▪ Poor quality code in existing, with high maintenance costs
▪ Can afford time, cost and disruption involved
Checklist for choosing to replace
▪ Application significantly out of line with business needs
▪ Willing to make changes to business model to fit off-the-shelf solution
▪ Can afford time, cost and disruption involved

The diagram below summarises the overall decision process in graphic form.



Which target platform?

Transformation projects are broadly aimed at converting code or modules to Java, C++ (or C#), to a relational database environment, or to a HTML architecture – or indeed some combination of these. The majority of *new* enterprise-level development in the foreseeable future will take place on one of two platforms: Microsoft's .NET platform or the multi-vendor Java 2 Platform Enterprise Edition (J2EE). This is not the same thing as

saying that all legacy transformation should target one of these two platforms, but there are sound arguments in favour of doing just that:

- There are technical advantages to these platforms. For example, they provide for re-use, scalability, and wide access to related products and services. Re-use derives from the component aspects of these frameworks. Scalability is inherent in the architectures, and every significant supplier is behind one or the other of .NET and J2EE. Integrated Development Environments (IDEs) make the development and maintenance task easier. Application containers (runtime environments) provide the qualities of service necessary for enterprise applications such as transaction handling, security and persistence services. These factors (plus market competition for the supply of platforms) ultimately reduce operating costs.

- Most automatic translation products target these platforms, although with a current emphasis on Java (and therefore tending to favour J2EE). COBOL dialect translation is a notable exception.

- There is a growing skill-base, making it easier to recruit staff, and because these are perceived to be leading-edge technology, they are attractive to existing staff who want to extend their own skills and advance their careers.

- They facilitate the publication of application function to a network using standard XML-based protocols for use by other applications (usually referred to as 'Web Services').

The very factors that make these platforms suitable for Web Services are of course of great interest in transforming legacy applications, because of the way integration can be facilitated. Most legacy applications form part of an existing portfolio of interrelated and interdependent applications and Web Services are a next step in the evolution of application integration. Drawbacks include the continuing evolution of standards for Web Services and resolution of security issues.

The choice of .NET versus J2EE is the subject of much debate. Both have evolved from existing application server technology. J2EE is quite mature and is already running large-scale enterprise applications. .NET is the newcomer, but definitely here to stay. Today, Microsoft-based solutions are limited nearly entirely to Wintel class platforms – in other words, the choice of .NET implicitly chooses the platform, middleware and operating system and it is arguable that it takes more effort to scale to several hundred concurrent users that a similar J2EE implementation. Conversely, the smaller organisation may choose one platform as their near-universal standard, and will go with .NET for its low cost of entry and focus on rapid application development. Both J2EE and .NET are being repositioned to deliver the Web Services vision.

The strategic arguments for each side are familiar: Platform portability versus vendor lock-in, cost advantages of a bundled supplier versus dealing with several suppliers, a better architecture versus the risk of instability while the architecture is implemented.

The bottom line is that either platform is a suitable target for legacy transformation and the choice between them is best made on the basis of business and technical strategies overall.

There may be considerations that lead to other target options. Some possibilities include:

- Migrate persistent data level: Moving from mainframe hierarchical, networked or relational databases to an RDBMS environment. Some corresponding code changes will be needed in the application.

- Move to browser access, while at the same time reengineering the application to support Web-enablement: Add Web self-service functionality, to cover activities normally undertaken by in-house agents; remove capabilities that should only be available in-house – eg change discounts; add data, capabilities necessary for Web – eg email address input; migrate to shared databases; create data warehouse to support Web access (for example, legacy database may be non-relational or non-ODBC compliant); separate out presentation logic.

- Built complementary solutions around a CICS legacy: CICS TS V2 provides an EJB execution environment, enabling use of CICS support for EJB session beans to provide client access to CICS transactions, programs and resources via IIOP. (The CICS IIOP server provides the run-time environment in which the container and, in turn, the enterprise beans execute and from which they may interact with other CICS services and resources.) Possibilities include HTML or XML client-side presentation, with execution of servlets/beans in WebSphere environment, and Java method invocations then flowing over IIOP to execute enterprise beans running under CICS. Java beans under CICS in effect 'wrap' the existing applications.

- Migration to client-server: This is likely to be confined to situations where a high-function user interface is required. Client-server can result in higher support costs for distributed computing environments coupled the more complex client/server application environments. It does not always make sense today with J2EE and .NET available.

- Migrate to COBOL: Translators exist for certain 4GLs to COBOL, which may meet the minimum requirement to exit a particular obsolescent platform. Specific examples include 4GL to COBOL conversion (e.g., CA-Easytrieve Plus, DYL-260/280, DataAnalyzer, etc.) — also BAL to COBOL conversion, CSP to COBOL, and OS/VS and VS COBOL II to COBOL for OS/390. Other translators identified include PL/I to Cobol, Natural to Cobol, also Cobol II, 74, or 85 to OS/390 Cobol, OS/390 Cobol to Client/Server Cobol.

5. The Transformation Project

A transformation project exhibits many of the characteristics of traditional development projects such as objectives setting, user involvement, testing, scheduling, monitoring, and so on. There are however some factors that differentiate a transformation project:

- *The solution is built on the foundations of a legacy asset*, rather than starting with a discovery of business requirements. Of course there may be additional functional requirements to be added, but the usual procedure is to add this functionality *after* the transformation is complete.

- *Enabling technologies need to be procured for translation, data migration, and re-use, or a suitable partner identified to provide the technologies.*

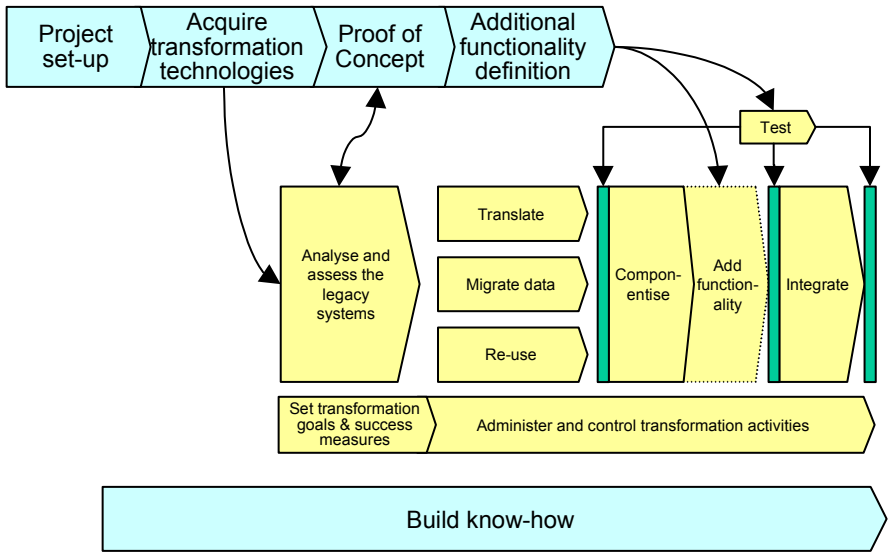
- Because the legacy application is already part of today's business operations, *a smooth transition is vital.*

- *Know-how needs to be built up over the course of the project* so that support capabilities are in place on completion. This is the case to some extent in any development project but because of (usually) outdated documentation and limited understanding of the legacy application, coupled with the move to a new architecture, specific attention is needed to create know-how and make it accessible.

- *Adjustments will be needed to existing development methodologies* to ensure that the work is structured to fit the needs of a transformation project and delivers to business and technical objectives, schedule, and budget.

Outline of a project approach

The classic development project follows a four-stage pattern of Plan, Analyse, Design, and Implement. The greatest risk is that the transformation project will be seen as implementation only, with little need for plan, analysing, or design. One way of understanding the totality of the work involved is illustrated in the diagram below.



This is a logical expression of the tasks and activities involved, not a definitive way to complete the project. Any organisation undertaking a transformation project should check its plans against this diagram to ensure that nothing has been forgotten. The diagram illustrates the point that initial work is needed before progressing the transformation project. For example, the available transformation technologies may have to be sourced from multiple suppliers, and it is likely that consulting expertise will be needed, at least for a first transformation project. Secondly, when there is uncertainty about the quality of the legacy application it can be worthwhile to run a proof of concept mini-project to confirm feasibility and (perhaps equally important) to firm up the total project costs. Subsequently the requirements for additional functionality can be acquired, following standard development procedures, with the proviso that this functionality is being added to an existing application, so the business users are not starting with a blank sheet.

Building know-how is emphasised here because of the peculiarities of the legacy asset. Although not always the case, legacy applications are typically lacking in documentation, business users are unfamiliar with the precise nature of the business rules built into their systems, and the technical expertise is confined to perhaps one or two people in the IS department. The *Build Know-How* activities might include the following:

- Confirm the business rules, and make them accessible via documentation or HTML access. Application mining is a useful technology for homing in on the parts of the legacy code that are likely to contain the rules.
- Identify any gaps in the rules - this is especially significant where the application access is to be extended (eg via Web-enablement) – and work towards filling these gaps.
- Assemble a 'user model' that can be used as the basis for acceptance of the transformed application. The 'user model' contains descriptions of the functionality to be exhibited by the application, the constraints that it must satisfy (eg browser version, hardware), and properties that the application must possess, such as portability, maintainability, security and so on. Review available test scripts and extend these to ensure that the 'user model' can be verified.
- Get development and support staff up to speed on the target technologies.
- Ensure transfer of relevant know-how from consultants throughout, and particularly when it is a first-time transformation project for the business.

Addressing the specific features of transformation projects

The 'method' described in the previous paragraph described a framework for transformation. The specific features listed earlier are partly addressed by this framework, and by additional specific activities:

Building on the foundations of a legacy asset, rather than starting with a discovery of business requirements – The transformation process described earlier includes an assessment and analysis phase. For situations where the business rules in the legacy are uncertain, they can be clarified in this phase – or where a rewrite is called for, they can form the basis of requirements for development. Application mining tools can be useful in this process. The inclusion of activities to build know-how are complementary to this work and ensure that the legacy asset is well understood.

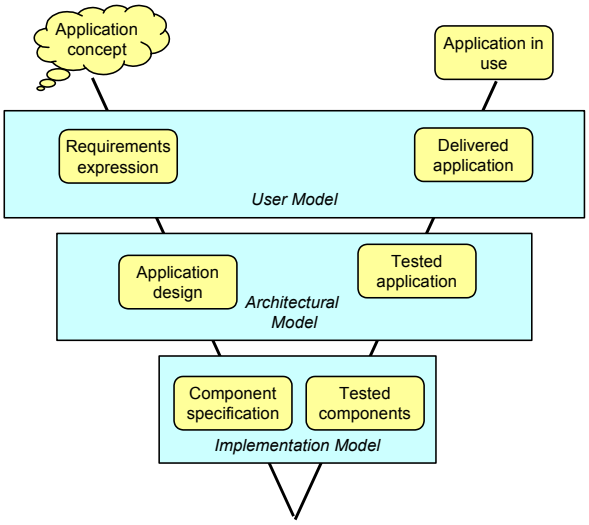
Procuring enabling technologies for translation, data migration, and re-use, or a suitable partner identified to provide the technologies – The framework above provides for this.

Achieving a smooth transition – There is no simple recipe for making the transition comfortably. One approach that has been used successfully to migrate groups of business function into the live environment, using middleware to integrate with the function that has yet to be migrated. This happens over four or five iterations until the migration of all the business function is complete. The data will need to be synchronised throughout. With data, one approach is to have logical legacy and new databases which are kept synchronised using a product such as DC Metalink. Alternatively middleware can be used to provide direct access from legacy. The direct access approach may require some modification to the legacy application code and is most suitable when there is no requirement to migrate the legacy data store.

Building up know-how over the course of the project – The concept here is that the project plan needs to gradually build up the knowledge about the existing and target applications, and to create the knowledge to support it longer-term – this could be in documentation and in people's heads. The difference between this and the usual approach is the explicit tasks identified to make this happen. Note that there are different kinds of know-how – customer (user), programmers (for on-going support), and support staff (eg help desk).

It is advisable to involve a consulting partner, at least for the first project – someone who will anticipate the many 'gotchas' and who knows their way around the target environment. Here's a checklist: Do they have a methodology? Have they access to the necessary tools? What is their track record? Do they know their way around .NET, J2EE or whatever target platform you plan to adopt?

Making adjustments to existing development methodologies **–** Transformation projects require structure and deliverables like any other project. Any methodology based on the V model will work effectively, although some modification to procedures may be necessary to follow the principles of user, architectural, and implementation model testing.



The so-called V model describes application development as a progression of stages from requirements expression, through design and build to acceptance. The deliverables from the later, upward-pointing phases are shown, through testing, to be implementations of the matching specifications on the other side of the V.

Waterfall methodologies fit the V pattern as do most 'lightweight' methodologies.

Lightweight methodologies have superseded the traditional waterfall approaches in many development shops. These lightweight (agile) methods are adaptive and cope well with change, and are people-oriented. They are suitable for legacy transformation because of the way transformation projects are approached (it is necessary to get through the conversion phase of at least part of the existing portfolio before it can be confidently

predicted what the introduction of new functionality will cost), the low effort required in design (the requirements are largely inherent in the legacy), and the proportionately higher effort required for testing and cutover. If a development department already has a documented lightweight methodology in place, some adjustments may be needed to fit a transformation project. For example, the scope and definition of the earlier phases and deliverables will need to be adjusted, and activities such as testing will need to be changed.

The need for an internal champion

There is a common perception in business that legacy systems are yesterday's solutions. The legacy application is not seen as a basis for going forward, but as outdated, difficult to maintain, and possibly lacking a good fit with business needs. A champion is essential to promote the transformation option and put forward the business case.

This will not be easy, as there will be interests opposed to transformation and favouring other options. In-house technical staff have little incentive to keep the legacy application but will lean to a rewrite or package replacement options because these provide new development experience and the opportunity to learn new skills. Existing platform suppliers will want to keep the status quo. Senior managers will be at the receiving end of marketing campaigns from ERP and package suppliers, all promising benefits through replacement and process redesign. End-users may be more positive because the legacy application does the job they expect of it today, even though the cost may higher than they would like and it lacks certain capabilities (for example, no Web access). However, they will find it difficult to assess the technical advantages and risks of options that are quite different from one another (rewrite versus replacement by a package versus transformation).

Because of these considerations, it must fall to the CIO to develop the case for transformation and to explain the pros and cons of the various options to business managers. The CIO will have to push in-house analysts to look at all the options. Experience shows that they will prefer the design of a new solution over re-use of the old and this tendency needs to be counter-balanced by the CIO. Alternatively, a third party may have to be involved to see that an objective analysis takes place.

Note however that business managers need to champion the transformation project overall. It may appear to be a like-for-like replacement of the existing application, but there will be a need to accept the end-result as equivalent, and to be involved in any functional changes. And when there are problems or delays (as there always are), there needs to be someone to remind everyone of why the work is being done and the benefits that will accrue to the business as a result.

6. Building the Business Case

These are hard times when it comes to any form of business investment. The situation is made worse for legacy investments because many of top management's background assumptions and time-honoured business models are inadequate to understand what is going on. The key is to find the right way to present the issue to top management, spelling out the downside of not investing, and contrasting the risks and benefits of the various alternatives.

What makes justifying a legacy transformation project so difficult? First of all, managers are most comfortable with the idea of spending money to get something new and at first sight transforming a legacy application seems like a project to 'fix' something that already works. Second, the legacy project is often an 'enabling' investment – that is, it positions the business to achieve something else. For example, one of the benefits might be to make it easier to add new functionality, implying that the functionality could be added some other, possibly less costly way. Indirect benefits of this kind are always more difficult to quantify and justify. Third, the costs of the transformation project have to be spelled out, while the true cost of today's legacy (disruptions, maintenance issues, costly operations, and so on) is hidden in other budgets. These are the issues covered in this section.

"If it ain't broke, don't fix it"

This is likely to be first reaction of a manager to a request to spend money on a legacy application. Like other situations in life, people become used to their legacy applications, the awkward interfaces, the workarounds, the time required to make changes, the cost of maintenance and operation, and so on. Every user is overworked and understaffed and people don't have the time to think about potential fixes and improvements. They may grumble and complain but they're too busy to do anything about it. So unless there's a crisis, they would just as soon get on with the job, thank you very much.

Management hears the grumbles and complaints but unless there's a demand from customers or moves by a competitor they are unlikely to react. Management is used to the size of the maintenance budget and it seems just as easy to approve the same budget for next year, without getting into an argument about it. Any proposal to transform a legacy application is therefore likely to get a negative response.

Overcoming this inertia is the first step in moving the legacy transformation proposal forward and to do this it is necessary to explain the risks and opportunity costs of doing nothing. For example, the risk posed by key maintenance staff leaving: Legacy applications by their nature are opaque, not-so-easy to work with, because of the layers of changes made over the years, and lack of clarity in documentation. (Documentation isn't usually created to help maintainers, it's there to help users, and to explain how and why the application was built that way in the first place.) So it takes long and close acquaintance to get to know the ins and outs of these applications, and hence the dependence on the people who maintain and enhance them. Of course people can be replaced. But this takes time, there's a learning curve involved, and if the programming language is relatively obscure, it may prove difficult to hire in the skills needed. Other potential risks include withdrawal of support by platform component suppliers and lack of replacement and upgrades for hardware and software.

At the business level there may be risks posed to future business plans still on the drawing board. Can the legacy application cope with those 50% extra customers? Will the people in Pre-Sales be able to cope with that many quotations, given the state of the application?

An example of opportunity costs is the staff time that could be released by moving to up-to-date tools. The current maintenance staff are tied up maintaining (with some difficulty) an ageing application - this work could take less time and effort on a transformed application due to the availability of tools and their inherent productivity. Changes and enhancements would be executed more quickly and efficiently, and the staff in question may be able to free up time for new development.

These arguments based on risk and opportunity costs (the costs of doing nothing) must be complemented by an explanation of specific benefits and how the legacy investment advances the organisation's business goals. While these will be context-dependent, typical benefits can include long-term cost reductions, time-to-market improvements, and extended business reach, just to pick three.

Explaining software life-cycle investment

After the investment push on legacy applications leading up to Y2K, management may have concerns about further spend on the same applications. The same reaction is likely to a transformation request for a two- or three-year-old 'legacy' application. (This situation can arise, for example, when the build of the new application started some years back, but because of delays and time taken to roll out across the company, is already showing it age.)

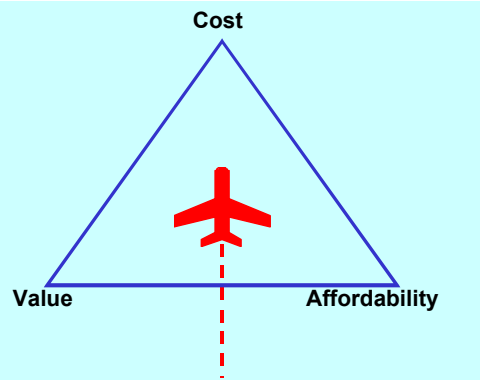
In some ways this is a similar issue to the one in the previous section, but with an added dimension, as it implies there's a history to this legacy question. The basic issue here is that management has not been given a view of the future. In other words, their expectations have not been properly managed. This has its origins in the way we have treated large projects in the past, as relatively isolated, one-off events, rather than part of an on-going business programme. Businesses have acquired the habit of thinking of a applications 'project' as something that is finished when the application is delivered, the build team disperses, the consultants leave, and the users get on with the job.

The justification argument needs to focus on the treatment of software investment and get across the notion that software is (or should be) a non-perishable asset. Businesses have built up substantial investments in legacy applications and their effective operation is core to running today's businesses. It makes sense to keep a legacy application running if it continues to serve business needs. Yet the forces of business change, technology obsolescence and decay in internal know-how put applications at risk over time.

This is why business invests in maintenance and improvement. But is the legacy investment like a car, to be replaced by the latest model every so often, or is it more like a house, requiring regular attention and renovation, and an extension added if and when the family grows? Recent developments in tools and platform technology make it practical and sensible to take the latter view. Looked at in this light, legacy transformation becomes a way of prolonging the software investment and quite possibly a way to deliver better data and added flexibility for future expansion upgrades – as well as costing less.

Cost, Value and Affordability

Not every legacy justification starts from the same point. It is necessary to understand where this starting point lies in what might be described as the 'Bermuda Triangle' of Cost, Value and Affordability. The common confusion and difficulty that IS finds itself in justifying investments in 'enabling' projects (and this is what legacy transformation is, first and foremost) lies in this uncertainty. The reason behind this difficulty is that once the objections on the grounds of cost are addressed, then the ground shifts to questions about value (is it worth it?), and when *that* question is dealt with, the question of affordability (can we afford it?) is raised. After that, the discussion shifts to why it costs so much in the first place. And so the argument goes around and around.



The Bermuda Triangle illustrates the potential trap and shifting nature of objections raised to a legacy investment proposal. The way out of the triangle is to identify the current position and then work systematically to achieve the right balance of arguments.

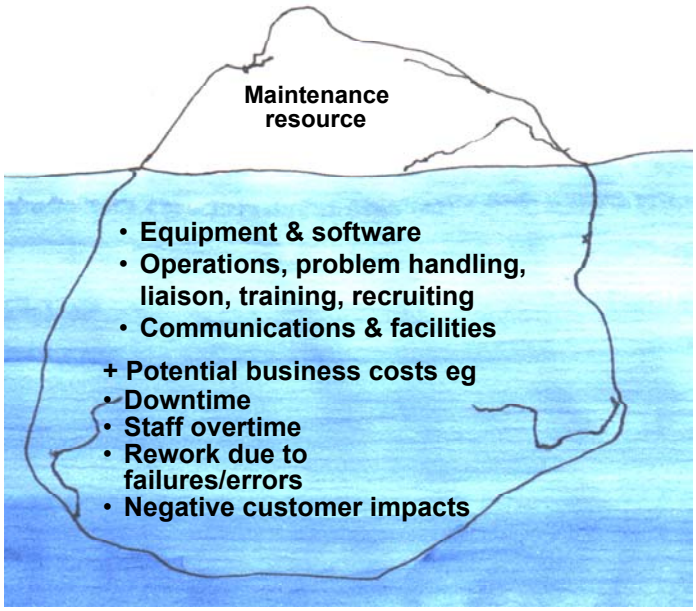
The way out of the triangle is to identify the current position and then work systematically to achieve the right balance of arguments. At the 'cost' apex, the central discussion is about how much to spend, what the options are, and which choice is the right one. At the 'value' apex on the other hand, the focus is on how worthwhile the investment is – at its most basic, the objection goes back to "if it ain't broke, then don't fix it". Finally, at the affordability apex, a favourable value and cost balance has been established, and the objections centre around priorities and the other opportunities to spend this money in the business.

Location	*Central Issue*	*Symptoms*	*Way forward*
Cost apex	Choice of solution	• "Is this the right amount to pay?" • Indecision about the choice of option to go with: "Let's replace the legacy application" "No, let's rewrite it" • Uncertainty about how much legacy transformation will cost	• Compare the various options • Include the 'do nothing' option • Include operating costs, not just project costs • Get a fixed-price quotation
Value apex	Benefits	• Uncertainty about what it's worth • "If it ain't broke, don't fix it" • "There are lots of assumptions here – we don't know what the future business will look like"	• Explain the risks and opportunity costs of doing nothing • Ask business users to put a value of flexibility, reliability and other anticipated improvements • Consider the opportunities to leverage the transformation

Location	*Central Issue*	*Symptoms*	*Way forward*
			results across the business
Affordability apex	Priorities	• "It's a good idea, but is this the best way to spend the money?" • "We have other priorities this year" • "There isn't enough in the budget" • "We're spending enough already"	• Adopt a portfolio management perspective – score projects on contribution to business, current level of technical/cost satisfaction to demonstrate relative benefits • Show the savings • Offset the capital spend spike by tying the transformation project into something that everyone *is* in favour of (eg Web access by customers)

Tip of the iceberg

A common thread running through the previous discussion has been the cost of doing nothing. Much of the cost of today's legacy is hidden in what might be called the 'iceberg effect' . On the surface, the only costs are the maintenance resource. But the real costs of a legacy application include equipment, software, personnel (operations, problem handling, liaison, maintenance, training, recruiting, and so on), communications (tariffs, support), and facilities (space, moves and changes, power, cooling). Business costs may include disruption to the business, downtime, staff overtime, rework due to failures/errors, or negative customer impacts.



Management is often unaware of the substantial costs of today's legacy applications – including the business impact.

The reason this is important to bring out in the open is because the costs of the legacy investment proposal will be visible and could look inordinately expensive if compared with the visible costs of the today's application. It is necessary to compare like with like.

According to Gartner, 60 to 80 percent of the typical IT budget is spent on maintaining mainframe applications and the applications that run on them. If transforming legacy applications can dent this figure by even 10 percent, the impact will be substantial. Further reductions can come from consolidating servers and storage – Gartner have pointed out that hardware expenses account for 18 percent of the typical budget (and the same amount again for operations staff). Consolidation can lead to 20 percent reduction in these costs (due to the improvements in utilisation and load sharing). Thus it is important to show what the comparable figure is for the existing legacy applications.

The text-book definition of cost refers to "both the measurable and hard-to-measure resources for making goods and delivering services…the full cost of any cost object….is the cost of resources used directly for that object plus a share of the cost of resources used in common in making all objects"[2]. In this instance the costs are being collected for the purpose of making a decision so the precise allocation of resources is secondary. Allocating operating costs in this way can be a complex exercise, but the idea is to show the scale of the costs involved, not to tease out every detail. The resource allocation needs to focus on the areas where costs would vary if the transformation project goes ahead.

At the risk of stating the obvious, it will be management judgement that ultimately comes into play, and the figures are there to inform this decision.

This issue of visible/invisible costs applies equally to other options than legacy transformation. A case in point is legacy replacement: For example, the analysts looking at other options may have asked an ERP supplier for a quotation to replace the legacy application(s). It is necessary to go beyond this quotation, to consider the other (often substantial) costs involved, which are often well above and beyond the supplier's quoted price to do the work. These are another form of 'iceberg' effect, although the 80/20 rule is less likely to apply:

- Escalating training and change management costs due to changes in roles and responsibilities brought on by the new business models implicit in the ERP package.
- Data conversion is not like-for-like, thereby adding to the effort required and diverting operational staff from their day-to-day tasks.
- User retraining.
- The data in the new package will not map one-to-one with today's legacy coverage, and extensive analysis may be needed to deal with the gaps. Adjustments may be necessary to other in-house applications.
- Package configuration and testing, especially of interfaces to in-house applications, is largely unpredictable. Contingency allowances need to be made.
- The consulting costs quoted are for the work as described. Apart from the obvious 'scope creep' factor, failure to transfer know-how to in-house staff can mean long-term reliance on the consultants and escalating consulting costs.

[2] CH Brandon, RE Drtina *Management Accounting* McGraw-Hill (1997)

Bringing it all together

Transforming legacy applications is a task with both risks and rewards. It is easier to rely on what seem like stable applications and hope that they will be adequate to keep the business going, at least in the medium term. But these legacy applications are at the heart of today's operations and if they get too far out of step with business needs the impact will be substantial, and possibly catastrophic. The challenge for the CIO is to present the arguments for the legacy investment in the best possible light, but also to give management the full picture of these risks and rewards so that they can make a decision in full possession of the facts. Ultimately, legacy transformation is an 'enabling' project, that allows other things to happen, but it has its own direct benefits as well.

- To sell a large-scale transformation project successfully to management, it is necessary to present decisive evidence that the project will save money and strengthen the business.
- Spending money to keep legacy applications going 'as is' can be a mistake. Making business plans based on these legacy applications is another mistake. The costs and risks associated with the 'no change' option need to be spelled out.
- The transformation project's expected ROI will come in part from the projected decrease in maintenance and running costs of the new application compared with the legacy application.
- The most important justification for beginning this kind of project is, however, the business need.
- Appropriate cost, value and affordability arguments should be marshalled in advance in order to overcome potential objections.
- Applications are not finished when the 'project' is finished. Application life cycles require on-going spend to ensure business and technical effectiveness. Legacy transformation should be presented as part of normal asset maintenance and renewal, not as a 'one-off', unusual rescue mission, so to speak.
- Update the business case as the work progresses to ensure visibility to stakeholders (management, users, project team, suppliers) and to enable periodic 'sanity checks'.

7. The Supply Situation

The discussion so far has concentrated on transformation technology and projects. In this section the discussion turns to the supply-side and where project managers should look to buy transformation technology and associated services. The concept of a 'legacy transformation value chain' is introduced as a way of understanding the roles of different players on the supply-side. This is a way of thinking about the role that each type of player takes on in the value chain – from this the potential purchaser can take a view of how to go about selecting suitable supply partners.

Understanding the supply-side

The work needed to deliver transformation has several dimensions – requirements definition, program translation, testing, change management, installation of new platforms and so on. As explained earlier, the current state of development of the market means that there are few situations where the purchaser can rely on finding one supplier to satisfy all these needs, and the specialised know-how needed usually makes it impractical to undertake the work in-house. The purchaser will be looking to buy in some (or all) of the products and services needed, but will not want to deal with too many suppliers, to avoid the risks of split responsibilities and confusion over who does what. So who should the purchaser deal with?

A good starting point for understanding the supply-side is the 'legacy transformation value chain'. The diagram overleaf shows such a legacy transformation value chain. This value chain shows the different type of player - the providers of platforms and enabling projects, providers of transformation products and services, and the consumers – the IS department and the end-user (and possibly outsourcers).

The categories are somewhat arbitrary but essentially they reflect the source of supplier revenues. Note that a single supplier may take on more than one role. For example, selling consulting days as well as software licences.

The best way to categorise the suppliers is to consider their revenue drivers. Each player in the value chain has different revenue drivers. The players upstream in the value chain (to the left of the diagram) are looking for on-going revenues, from licences and upgrades, hardware and operating systems, training and support, and so on. The mid-stream players are mainly involved in the transformation project itself, so they will achieve what are essentially one-off revenues. Outsourcers and the in-house IS department for their part are concerned with the cost of operations and what revenues they derive from end-users and the business.

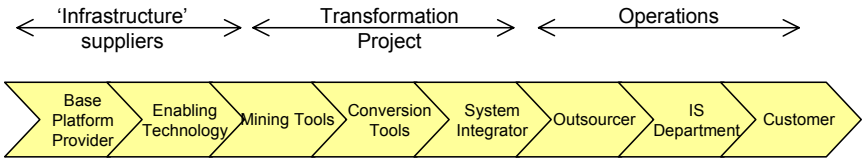
The value chain has three implications for supply-side behaviour:

- *Position in the value chain is the driver for supplier strategies.* For example, it is in the interest of suppliers at the 'infrastructure' end to give away migration tools 'free' since their revenues are from software licenses, product upgrades and support generally – all recurring revenues over the life-cycle of the transformed applications. Likewise, integrators may find it profitable to do the same, as their revenues come from the services provided. This can result in a squeeze on transformation toolset suppliers. The drawback for the purchaser is that the choice of technologies can be determined by existing business relationships rather than the best tool for the task at hand.

- *Partnerships in the value chain.* Players upstream in the value chain will build partnerships with players closer to the purchaser such as integrators, who essentially 'own' the customer relationship. The integrator handling the transformation project will tend to make key decisions on tools, infrastructure and so on. The reverse may sometimes be the case, for example where the infrastructure sale has been made, and the infrastructure vendor will be looking to help the purchaser move all applications across with the help of an integrator partner.

- *Automation is not the primary interest of infrastructure suppliers and integrators,* but less automation equals greater cost and increased likelihood of errors being introduced: Infrastructure suppliers and integrators will tend to a decision that favours their revenue position. For example, the integrator makes more money from extended manual effort (since revenue is a function of man-days expended), but this costs more for the purchaser, and manual intervention can lead to the introduction of new bugs in the application. Infrastructure suppliers have limited application know-how and their professional services fees are expensive, and geared to their products. Either way, the purchaser stands to lose if the best use is not made of the transformation tools and technologies.

Legacy Transformation Supply Value Chain



Examples	Microsoft Oracle Sun SAP IBM Jacada	IBM Websphere Acucorp Oracle9iAS BEA	Relativity Netron Cyrano Allen Systems SEEC CAST	ArtinSoft Tominy SWS S/W Services	CMG Progeni Adpac SEER Computing LegacyJ Modis Solution NIIT S/W Solns	EDS Capita		
Revenue driver (examples)	No. seats Upgrades Size of installation	Web traffic expected No. seats Upgrades	No. project team seats	Lines of code converted Days worked on project	Days worked on project	Transaction volumes processed	Budget based on on-going cost of operation	
Revenue duration	On-going	On-going	Project	Project	Project	Contract		
Competitive strength	Marketing Technology owner	Limits degree of change required	Specialist with proprietary techniques	Specialist, promises quick results	People skills, experience	Scale economies	Gatekeeper	
…and weakness	Far from customer	Point solution, doesn't address basic transformation issues	Requires highly-skilled people Still leaves much of the work to be done	Only part of the solution required Some manual intervention needed	Track record rarely fits precisely, therefore limited credibility	No or very limited track record	Limited resources (numbers, skills) No experience	

Implications for purchasers

The best strategy is to strike a balance between the project-oriented players (who will take care of the transformation), and the infrastructure suppliers and in-house staff who have to make the end-result work every day, and to ensure that the transformation expertise is at the heart of the solution delivery.

At the same time, the purchaser will want to avoid dealing directly with a multiplicity of vendors and the risk that suppliers may 'pass the buck' when difficulties arise.

The best strategy for the purchaser will look like this:

- Keep the project management activity independent. Appoint an in-house project manager or employ a contractor or trusted consultancy.

- In planning the project, group the code translation, data migration and associated testing, and keep the rest of the work separate. This will clarify where responsibility lies and make best use of the specialised skills available.

- Research the most suitable tools for the translation, data migration and re-use activities, based on the purchaser's specific situation and either find a suitably-qualified integrator, or deal directly with the suppliers involved if it is a sizeable transformation project. For example, major translation work can sometimes benefit from custom additions to the translation algorithms, and the supplier is in the best position to do this.

- Select an integration partner who will involve in-house staff in the changes and subsequent integration so that there is transfer of know-how. Use the integrator to sub-contract services from infrastructure suppliers, both to manage them more effectively, and to share the risk of cost overruns with the integrator.

- Consider running a 'proof of concept' project on a smaller or less business-critical application in order to assess the performance of the suppliers, and to get the in-house team up the learning curve in a low-risk environment. This will also give a better indication of the costs.

8. Outlook for Legacy Transformation

Legacy applications serve a vital role in the landscape of commerce.. The service they provide is critical to the day-to-day function of industry as a whole. For example to conceive of e-commerce solutions without exploiting legacy applications is to ignore the one key to making end-to-end e-commerce a reality.

The most common complaint about legacy applications is that they are too old, too inflexible and too outdated to add real value to the evolution of industry system solutions. While an ideal world would discard last year's technology each year to replace it with the latest in system design and functionality, reality and prudence prevent this. In the case of mission-critical business systems, this is neither practical nor prudent. It is necessary to rethink how the life-cycle of legacy applications is managed. This report has argued that transformation is feasible, is becoming easier, and with the appropriate choice of strategy and project management, presents a real alternative to replacement or rewriting applications from scratch.

Transformation cannot happen overnight because it involves the collaboration of so many constituents with so much valuable business information and functionality at stake. However the availability of tools and processes can greatly simplify and speed up the process and, as even a brief consideration of the factors will reveal, the alternatives can be even more costly and time-consuming and risky for the business.

What is the future outlook? Current vendors' strategies are narrowly focused and suppliers have a proprietary interest in furthering their own tools and services. Without more co-operative effort across the value chain, this situation will continue to slow down the wider acceptance of transformation as a legitimate and effective way of moving forward.

On the other hand, the demand for access to the benefits of the present development tools available for the Java and .NET environments may drive transformation forward and increase the demand for transformation products and services. These environments include code generators, intelligent editors with built-in wizards, logic-tracing tools and debugging facilities. Analysts can create design models that feed specifications into development products. These tools are part of integrated development environments that synchronise business models, specifications and program logic across the development cycle. Changed business rules in a design model are reflected in the system source code and a coding change will also be reflected within a design model. Such model-driven development and maintenance is a very effective and efficient way to evolve systems over the long term.

Another factor is the lack of skilled resources: There are too many critical legacy applications and too few skilled technicians to work on them. Transformation provides the opportunity to increase the effectiveness of the legacy programmers through better analysis, development, upgrade, debugging and componentisation tools. It makes sense to migrate legacy applications to these environments rather than look for the tools to be retrofitted to legacy environments.

Putting these factors in perspective, legacy transformation is currently at the early adopter stage, but if the level of interest in J2EE and .NET keeps increasing and users are looking to take the benefits of these platforms, then the future of the technology looks positive.

Further reading

The following articles and white papers have helped to shape my views:

Anon *Application Mining – what* 24th August 2001 http://www.it-analysis.com/article_pf.php?id=1575

John Bergey, Liam O'Brien, Dennis Smith *DoD Software Migration Planning* Technical Note CMU/SEI-2001-TN-012 Carnegie Mellon University, August 2001

A Bainbridge, J Colgrave, A Colyer, G Normington *CICS and Enterprise Java Beans* IBM SYSTEMS JOURNAL, VOL 40, NO 1, 2001 http://www.ibm.com

Michael L Brodie, Michael Stonebraker *DARWIN: On the incremental migration of legacy information systems* Technical memorandum TR-0222-10-92-165 Electronics Research Laboratory, College of Engineering, University of California, Berkeley, March 1993

Joe Celko *Breaking tradition* Intelligent Enterprise, February 21, 2002 http://www.intelligententerprise.com

Arie van Deursen, Paul Klint, Chris Verhoef *Research Issues in the Renovation of Legacy Systems* ETAPS Conference, 1999

Jim Duggan *Graceful Retirement: Your Applications, Not You* AV-15-4789 Gartner Group, February 2002-08-15 http://www.gartner.com

Len Erlikh *Unlock the power of your legacy systems* Relativity Technologies Inc, 2000 http://www.relativity.com

Len Erlikh, Mike Ferris *Business-Rule Extraction from Cobol to Java* 1998 http://www.devx.com/premier/mgznarch/javapro/1998/JP_junjul_98/le0698/le0698.asp

Franck Gonzales *What's ahead for client/server architecture?* Owendo, France 2001 http://www.owendo.com

Simone Kaplan *Despite the sluggish economy and uncertain business climate, right now is the perfect time to tear down your legacy applications and start over.* CIO Magazine March 15 2002 http://www.cio.com/archive/031502/infrastructure_content.html?printversion=yes

Anthony Lauder, Stuart Kent *Legacy system anti-patterns and a pattern-oriented migration response* Computing Laboratory, University of Kent at Canterbury, 2000 http://www.ukc.ac.uk/research/publications/2000/compsci.pdf

Jerry Loza *Is a switch to Java a good move for you?* June 27, 2001 http://www.techrepublic.com/article.jhtml?id=r00820010627ggp01.htm&src=bc

Amey Stone *Keeping legacy software alive* Business Week, June 14, 2001 http://www.businessweek.com

Scott Tilley *The Net Effects of Product Lines* SEI Interactive, September 1999 http://interactive.sei.cmu.edu

Chad Vawter, Ed Roman *J2EE vs Microsoft.NET: A comparison of building XML-based web services* The Middleware Company, 2001 http://www.middleware-company.com

D. Vecchio, J. Sinur, J. Duggan *Legacy Evolution: Not as Black and White as It Seems* TU-10-1478 Gartner Group, 24 February 2000 http://www.gartner.com

Richard Veryard *Identifying Web Services* Interact, CBDi Forum, February 2002 http://www.cbdiforum.com

Ben Wilson *Chickens and turkeys migrate, but not necessarily in IT* ANUBEX, February 2002 http://www.falconsoft.be

Legacy Value Legacy Value Restoration Cognizant Technology Solutions, White Paper, November 2001 www.cognizant.com

Parallel Operation of Software: Is it a Desirable Transition Technique? Joint Financial Management Improvement Program, July, 2001 http://www.jfmip.gov

Web Services Triple Tree Spotlight Report, February 15, 2002 http://www.triple-tree.com

WebIT™ Web-Enabling of Legacy Applications Intercomp White Paper, August 2001 http://www.intercomp.com

Glossary of terms and abbreviations

Application container	Components under the J2EE standard are grouped into and run inside application containers. (Components are files of code that are accessed by the application at runtime). A container provides runtime support for the components and provides a unified view of the J2EE services. Each type of component (e.g., EJB, JSP, servlet, applet, or client application) has container specific to that component type. Under .NET, components run within the context of the Common Language Runtime (CLR). The CLR provides many of the same functions as the J2EE container.
Application mining	Application mining is used to extract business rules from legacy code.
B2B	Business to Business.
Copybook	A common piece of source code designed to be copied into many source programs. Commonly used by COBOL programmers. Referred to as 'libraries' in IBM OS, and implemented as 'partitioned data sets'.
EJB	Enterprise Server Beans. Beans are individual J2EE components that provide business functionality. They can manage their own persistence or delegate this function to its container. There are 3 types: Session beans, entity beans, and message beans. The EJB container (frequently referred to as the application server) provides transaction management, security, remoting (i.e., calling distributed objects), object life cycle, and connection pooling services to the beans. Beans are automatically pooled by the container when appropriate, eliminating the need for constant creating and destroying.
IDE	Integrated Development Environment. Visual Studio .NET is a notable example - it eliminates the most time-consuming and difficult tasks from development. A variety of IDEs support the J2EE platform. IBM's Visual Age and BEA's WebGain Studio are two IDEs, and Borland JBuilder is also used.
IIOP	Internet Inter-ORB Protocol. Protocol that enables CORBA over the Internet. Enables browsers and servers to exchange integers, arrays and more complex objects, unlike HTTP, which only supports text transmission.
IO	Input-output.
ISAM	Indexed Sequential Access Method.
Java	A set of technologies for creating and running software programs. A trade mark of Sun.

JSP	Java Server Page.
JCA	Java Connector Architecture. It is designed to connect J2EE environments synchronously to applications and transaction processors. It is an adaptation of IBM's Common Connector Framework.
JCL	Job Control Language. A set of statements controlling the execution of a series of jobs.
J2EE	Java 2 Platform, Enterprise Edition. It enables large-scale Java applications to be created consistently in a distributed environment. J2EE provides a container that manages components, a set of specifications for how these components operate, and a set of standard interfaces.
Legacy application	A legacy application may be defined as any application based on older technologies and hardware, such as mainframes, that continues to provide core services to an organisation.
Logic-tracing	A function of an IDE that tracks the execution of program code, used for debugging.
MQ	Message Queue. IBM's WebSphere MQ (formerly MQSeries) and Microsoft MSMQ allow messages to be sent from one application or service to another with guaranteed delivery.
.NET	.NET is not an acronym, but is usually capitalised. It refers to Microsoft's set of tools for applications development as well as standards and a Web-based philosophy for all Microsoft's products and services. Current .NET implementations run on Windows platforms.
Refactoring	Restructuring program code to improve its maintainability, readability, and so on.
SOAP	Simplified Object Access Protocol. Web Services communication protocol providing an XML-based format for transporting messages and invoking services over the Internet.
XML	Extensible Mark-up Language – Data format that can be read by people and machines. Data values and meta-data are both included in the data, to provide a self-describing syntax. Standard published by World Wide Web Consortium.
Web Services	A collection of business services or capabilities taken from single or multiple applications that can be published to a network using XML-based protocols, for access by other applications.
Wizard	A utility in a program that outlines a series of sequential tasks to set up a portion of the program.
Wrapping or wrappers	Wrapping or wrappers allows access to legacy function from an object-oriented environment. The object wrapper operates as a called method of an object, and then uses a traditional procedure call to execute an existing application function. It s a black box approach that isolates the internal complexities of the legacy programs. It can create longer-term maintenance problems.