

Oracle IT Modernization Series
The Types of Modernization

An Oracle White Paper
September 2008

Oracle IT Modernization Series
The Types of Modernization

Introduction	3
Modernization Target Based on SOA.....	4
The Modernization Process.....	5
Application Portfolio Analysis	6
APA Business Analysis	7
APA Macro Analysis	7
APA Micro Analysis	8
Application Portfolio Management.....	8
Modernization Types	8
COTS Replacement.....	9
SOA Integration.....	10
Re-architecting	11
Automated Migration.....	15
Re-hosting.....	19
Summary	22

Oracle IT Modernization Series

The Types of Modernization

INTRODUCTION

Today's IT organizations are looking to reduce costs, increase their ability to react quickly to ongoing business demands, and reduce the risk of reliance on legacy skill sets while also adhering to ever-increasing compliance demands. In an effort to carry out this difficult task, organizations are turning to IT modernization as a solution. With IT modernization, organizations are looking to move to an open systems modernization architecture that combines a high-powered infrastructure—consisting of a grid of computer hardware, operating system, and database and application servers—with an orchestrated service-oriented components and a User Interface and Business Intelligence layer that provides access form of portals, mobile devices, business-to-business (B-to-B) communications as well as other access mechanisms that are required in today's dynamic business world. In its complete form, the target platform is crucial in order to reduce total cost of ownership, increase agility, eliminate reliance on legacy skill sets, and satisfy compliance requirements.

In order to move into a more modern environment, organizations have to modernize their existing applications in a manner that preserves the business content of these applications as they are transformed to the new environment. This modernization process requires an assessment of an organization's current environment as well as the employment of a number of different modernization techniques in an effort to determine the best approach for each application to be modernized.

MODERNIZATION TARGET BASED ON SOA

An important aspect of any modernization or migration exercise is the platform to which you are modernizing.

The ultimate goal of IT modernization is a reduction in total cost of ownership (TCO), an increase in agility when reacting to ongoing business requests, a lowering of the risk of reliance on legacy skill sets, and the ability to meet compliance demands. To accomplish this, a new target environment is required—an environment based on SOA.

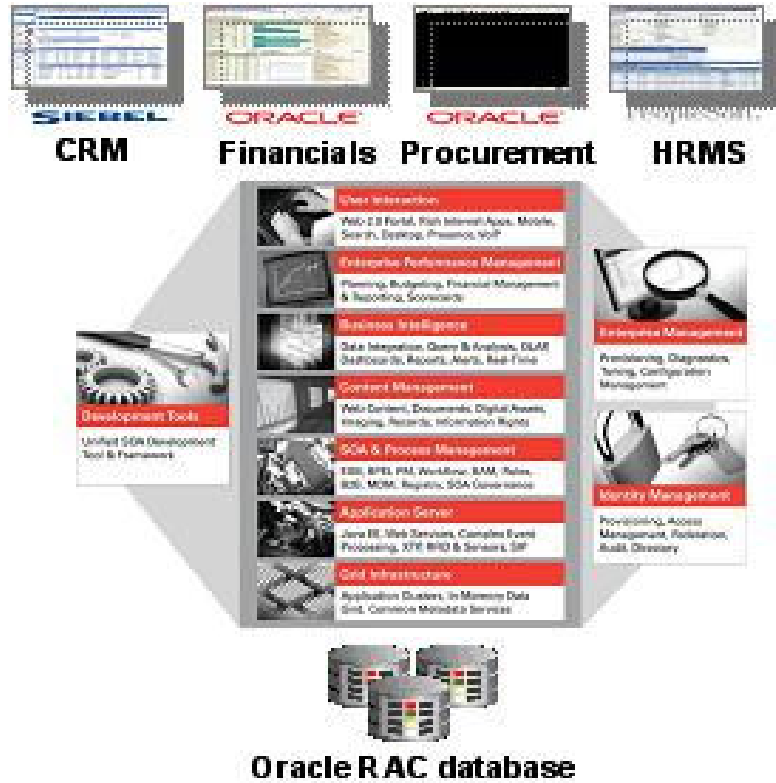


Figure 1 – Modernization Target based on SOA

A modernization environment based on SOA provides a mainframe-like quality of service grid-computing platform, thus allowing individual application components to be accessed as services, to be located and accessed only when needed—at execution time—and to be executed on different platforms as the need arises, increasing flexibility and lowering execution cost by ensuring maximum use of underlying grid resources. It further incorporates the concept of modern User Interaction and Business Intelligence where the various application components can be accessed via a number of external mechanisms including the web, mobile devices and business-to-business (B-to-B) communication. Such a modernization target needs to be “hot pluggable”, allowing it to be implemented via both Oracle and non-Oracle products.

The use of SOA application components orchestrated by an orchestration engine allows the creation of applications that more closely reflect the process flow and

the business procedures dictated by the organization. Such process-driven applications driving new components, re-hosted components or commercial-off-the-shelf (COTS) components are more closely aligned with business processes and procedures, and thus are easier to enhance and maintain. Concepts that users perceive as “easier to change”—such as process and work flow changes—are removed from the individual SOA components and incorporated into easier-to-change orchestration flows instead.

But the question then remains: How do you get to the SOA Modernization Architecture? And more importantly, how do you get there within acceptable parameters of time, cost, and risk? To answer this, we need to examine the concept of IT modernization.

THE MODERNIZATION PROCESS

The modernization process allows an organization to get from its current legacy environment(s) to a next generation IT environment. In contrast to a “rip and replace” strategy—effectively a new development effort with all the risks that go with it—modernization focuses on retaining existing application assets by transforming them to the new languages, databases, services, and environments that make up the new environment. The use of modernization techniques rather than new development techniques is especially important when an organization finds that business knowledge encapsulated within its legacy applications is unknown outside of those applications. This is not as uncommon as one might think. Many organizations are finding that one of the greatest risks of replacing an existing application via a new development effort that does not make use of existing application knowledge is that the new application lacks functionality that was contained in the original application.

In order to make use of modernization, the first step an organization must take is to carry out an application portfolio analysis (APA) of the current applications and their environment. Once the APA has been completed, the organization will understand the current application state and will have the information it needs to decide which applications are the best candidates for modernization. Additionally, the organization will have the information necessary to choose among various Modernization techniques, such as SOA Integration, Re-hosting, Automated Migration, Package Replacement or Re-Architecting.

Modernization focuses on how to achieve a more modern technology environment using various modernization techniques.

You cannot modernize what you do not understand.

APPLICATION PORTFOLIO ANALYSIS

Application portfolio analysis (APA) helps an organization better understand its current application environment. In particular, the APA reveals two important aspects of each application: its business value and its technical quality. The measure of technical and business quality in turn can be used to determine which types of modernization are likely to yield the most value.

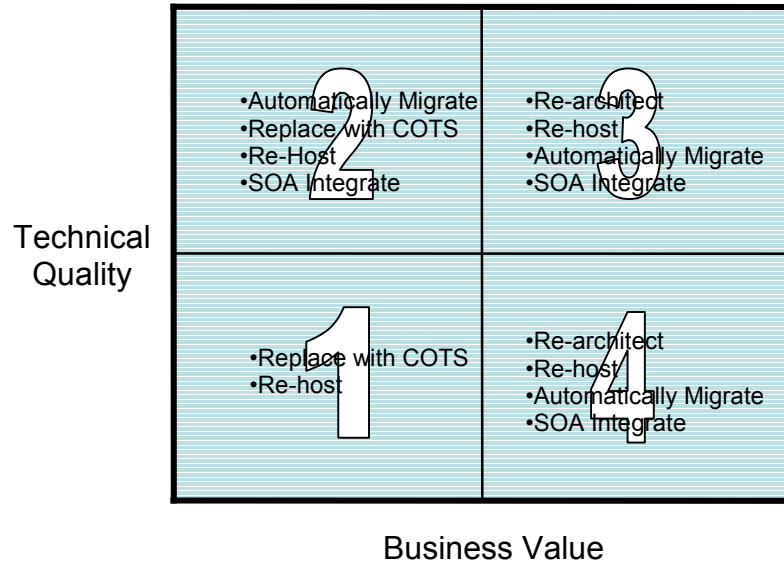


Figure 2 – Application Portfolio Analysis Measures

Business value is a measure of an application’s importance to the business, and is determined by such things as contribution to profit, level of usage, user satisfaction, and the value of the information maintained by the system. Technical quality is a measure of the goodness of the application against a defined set of technical criteria, such as maintenance cost, language used, vendor support, training availability, and hardware required. The importance of each of these criteria varies from organization to organization, and as a result each organization will categorize its applications differently with respect to business value and technical quality.

When used as part of a modernization process, application portfolio analysis is actually carried out on different levels, depending on the purpose of the assessment and the stage of the modernization process. These levels are APA business analysis, APA macro analysis, and APA micro analysis. A macro analysis and micro analysis may not be needed, depending on the particular modernization project, but a business analysis is always needed.

APA Business Analysis

An APA business analysis determines the value of the application in the context of other applications and the business as a whole.

The most abstract form of application portfolio analysis is the APA business analysis. At this level, the application is examined in its entirety in the context of the overall organization. Information is gathered regarding the value of the application to the business, the long-term need for the application, the degree to which the application is filling a commodity need rather than a unique business need, the potential for replacing the application with a commercial off-the-shelf (COTS) application, the overall cost of maintaining the application, the potential cost saving that can be achieved through modernization, and the risk involved in leaving the application within its current application environment.

This information is then used to prioritize which applications should be modernized first, as well as which type and/or types of modernization are best applied.

APA Macro Analysis

An APA macro analysis determines the value of the various application artifacts in the context of other application artifacts and the application environment.

The second form of application portfolio Analysis is the APA macro analysis. This analysis examines the current technical environment at a macro level to determine the artifacts that make up the applications and their environment, such as programs, data sources, data fields, batch jobs, teleprocessing monitor accesses, operating system accesses, and third-party utilities used. The information gathered during macro analysis can also be used in conjunction with an APA business analysis to help determine the type of modernization to be used and provide an inventory of the artifacts that will need to be modernized.

A macro analysis not only identifies the artifacts themselves, but also the links among the artifacts, and therefore can also be used to group applications into tightly connected sets of artifacts, allowing an organization to break down the total modernization problem into manageable, lower-risk phases. A macro analysis also helps determine any potential future integration points between a modernized application and any other applications that remain in the legacy environment due to a phased modernization approach.

A macro analysis looks at an application in its totality, crossing technologies such as languages and databases. At the same time, a macro analysis provides an inventory of the number of languages, databases, and other technologies that made up the legacy environment, along with the number of artifacts that use these technologies. This is very useful when determining the size of a modernization project. For example, when modernizing IBM mainframe applications, it is very useful to know the number and size of assembler programs that are part of the applications, as these typically have to be treated differently than other programs.

Although an APA macro analysis always includes looking at the application itself, it also allows for information to be gathered from other sources. For example, the existing workflow or use cases of a current application are useful sources of legacy information. However, this information may not be part of any computer system. It may be necessary to go “outside” the current application to the user community in order to determine links between such things as CICS transactions and the human process flows that use them.

An APA micro analysis determines the code structure of an application.

APA Micro Analysis

The third form of application portfolio analysis is an APA micro analysis. At this level, program code is examined at a detailed level in order to determine the exact internal structure of each program. This level of analysis typically occurs during the modernization process as part of an actual translation of code from one form to another. However, an APA micro analysis can also be used to determine code quality, which in turn helps determine the degree to which automated modernization techniques can be used (since these techniques typically cannot change the quality of the code).

Application Portfolio Management

“Application Portfolio Management Initiatives will support IT Governance and Investment in 40% of Global 2000 enterprises and large government IT shops within the next two years”

Gartner Predicts, 2006

Application portfolio analysis is needed as part of any modernization project, but it also has value all on its own—especially when it can be repeated on an ongoing basis. When portfolio analysis in general—and a macro analysis in particular—is carried out on a regular basis, then application portfolio analysis becomes known as application portfolio management (APM).

Since APM means that analysis occurs on a regular basis instead of just once, dynamic information such as ongoing program CPU utilization can also be collected. This information can be combined with static information such as application structure to provide an even more insightful set of information for making both modernization as well as daily operational decisions.

Since APM involves a repeated activity, an APM product must provide not only the information recovery capabilities required for a one-time analysis, but a repository in which to store the results for ongoing analysis. For a business analysis, the APM product can be used to provide additional information such as CPU utilization and failure rates over time to help prioritize modernizations. At the macro level, an APM system can be used to optimize current environments—for example, by determining which programs are never invoked even though they’ve been identified as being in some program call chain. At the micro level, an APM system can be used to perform code analysis to assist in code maintenance and code restructuring.

MODERNIZATION TYPES

After carrying out an APA business analysis and determining that an application is a candidate for modernization (and also possibly using an APA macro analysis to perform an initial analysis of the application itself), an organization must decide which modernization technique to use to take the first step toward the SOA Modernization Architecture. There are five main modernization techniques: Commercial off-the-shelf (COTS) applications replacement, SOA Integration, Re-architecting, Automated Migration and Re-hosting.

COTS Replacement

Provided the package exists, COTS Replacement based on SOA-based packages is highly cost-effective.

One Modernization option that is nearly always considered is replacing the legacy application with a commercial off-the-shelf (COTS) application.

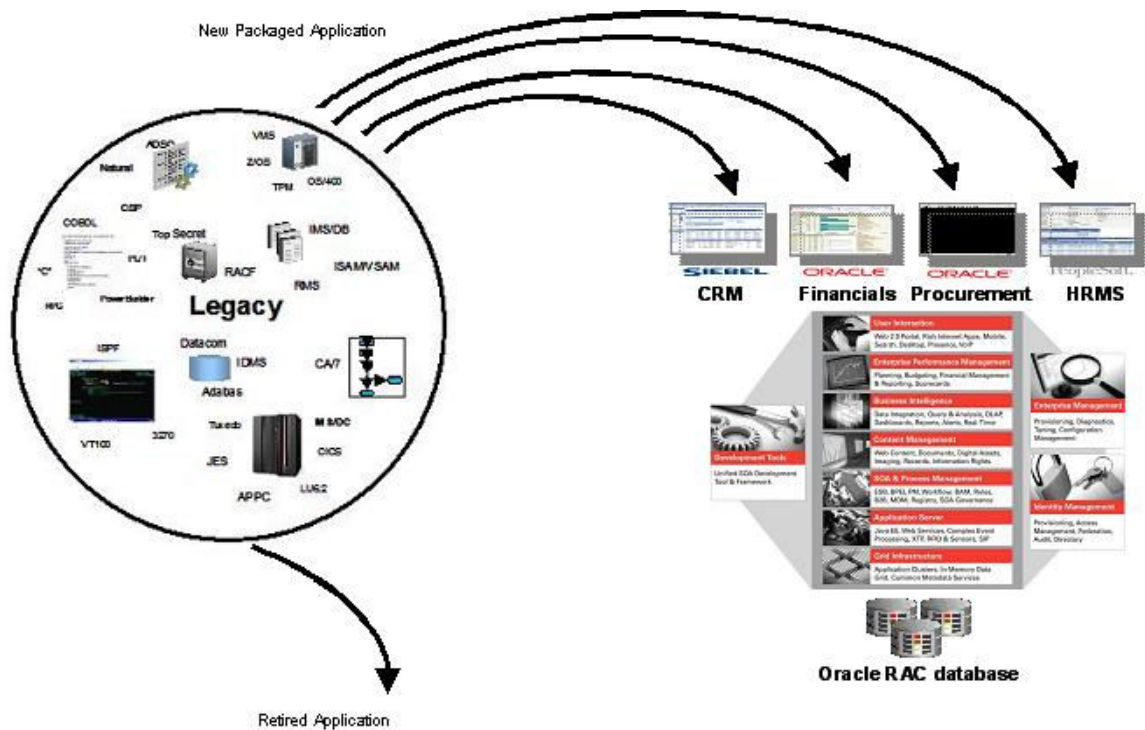


Figure 3 – COTS Replacement

Of course, a legacy application can be replaced with a COTS application only when such an application exists. But for many utility systems—such as payroll and accounting—and increasingly for other systems—such as manufacturing and customer relationship management (CRM)—there are packaged solutions available.

Use of a COTS solution assumes that an organization is willing to adapt its operations to the operational characteristics of the COTS application. In some instances, this is an acceptable trade-off, but this trade-off is also why COTS replacement is typically not used for applications that incorporate unique aspects of the business, and thus have very high business value.

When COTS Replacement targets the same environment as other types of modernization, maximum agility is achieved by a COTS application that is made up of SOA components that make use of capabilities such as SOA component orchestration. These SOA COTS components can then be mixed with other modernized components—such as re-architected components, re-hosted components, and automatically migrated components—using an SOA orchestration engine. This maximizes the agility of the complete application by viewing the COTS application as a set of reusable components rather than as an application “silo.”

SOA Integration

SOA Integration provides a quick start for SOA—but it does not solve a number of legacy issues.

The simplest way to begin to address IT modernization is to “wrap” existing application interfaces using SOA wrappers, thus creating SOA services that can be registered to an SOA management facility on a new platform, but are implemented via the existing legacy code.

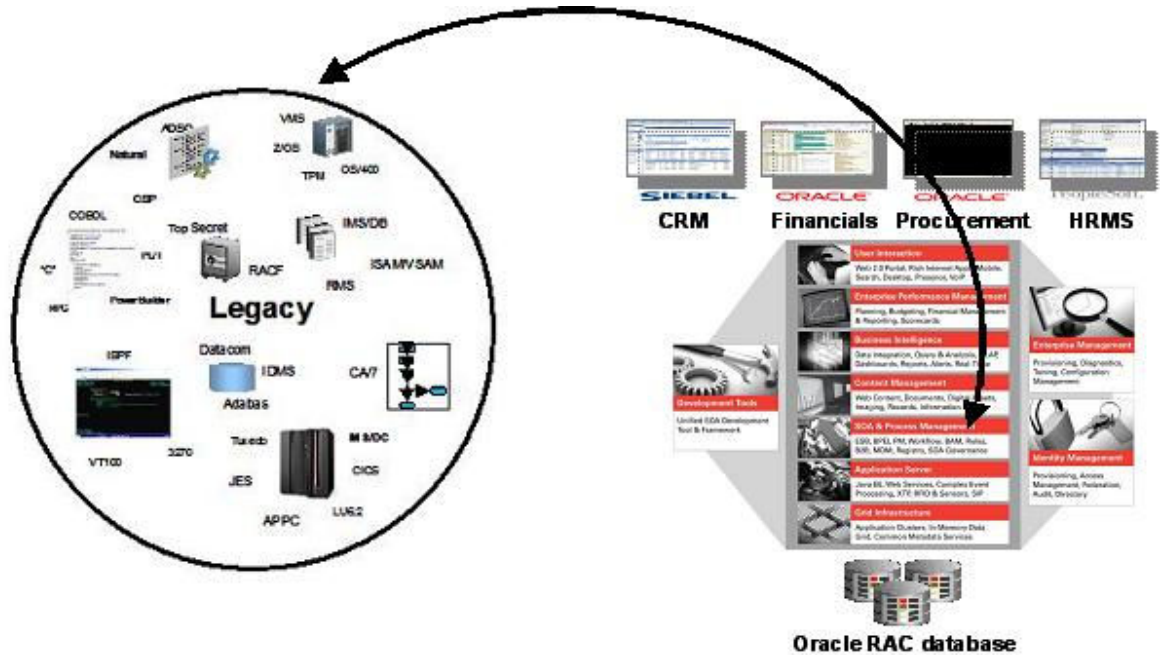


Figure 4 – SOA Integration

The registered SOA services can then be reused and combined with the results of other more invasive modernization techniques such as re-architecting.

Using SOA Integration, an organization can begin to make use of SOA concepts, including the orchestration of SOA services into business processes, leaving the legacy application intact. Of course, the appropriate interfaces into the legacy application must exist and the code behind these interfaces must perform useful functions that can be packaged as services.

SOA Integration typically takes advantage of three legacy application interface points:

- 1) Presentation screens. Here a legacy screen or group of screens is replaced with an SOA service that drives the underlying program in the same manner as the original screen(s). Presentation screens are often good candidates for SOA Integration since many legacy applications use a screen or set of screens to drive a single transaction, for example the adding of a customer or the approving of a purchase order.
- 2) Functional calls. In this case, the “call” to a legacy procedure or program is replaced with an SOA service that issues the same call. Legacy callable

procedures or programs may have been written as reusable components and thus are candidates for reusable services.

- 3) Database calls. In this case, a “call” to a legacy database or file system is replaced with an SOA service that issues the same native call and returns the requested data. Since the target environment will use relational databases for handling data, in some cases this call can be further modernized by allowing an SQL call to be issued, even though the data is not stored in a relational database environment.

SOA Integration often involves communicating with the existing application on its current platform. However, applications can also be re-hosted (see “Re-hosting,” below) so it is also possible to SOA integrate an application after re-hosting. SOA integration in this manner is useful because the SOA integrated re-hosted application components; new Java components, COTS components, and the orchestration engine that brings them together all now reside on the same platform, avoiding inter-machine communication.

Since SOA Integration is noninvasive to the legacy application, it allows legacy components to be used as part of an SOA infrastructure very quickly and with little risk. As such it is definitely a first step toward a completely modern environment.

However, since the code that implements the services remains unchanged, SOA Integration does not solve the problems associated with maintaining a legacy environment including retaining the need for legacy skill sets and therefore provides only partial benefit in reducing TCO, increasing application agility, reducing dependence on legacy skill sets, and achieving compliance.

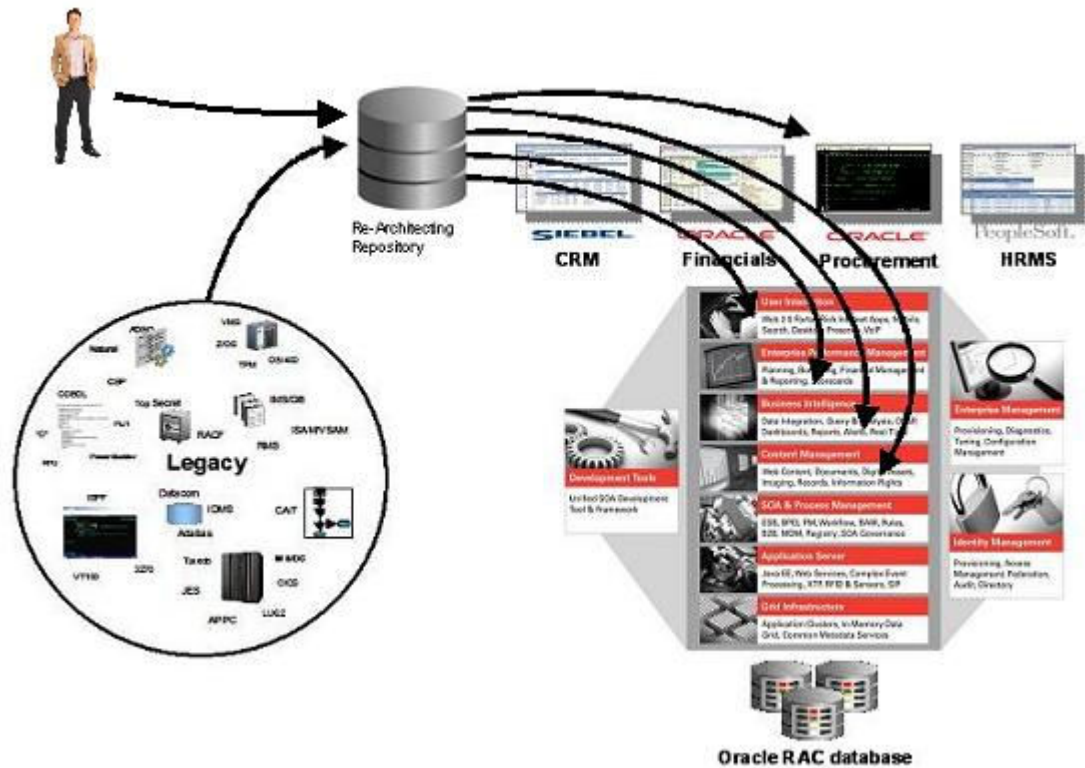
Re-architecting

The modernization approach that maximizes the benefit of SOA and new technology capabilities is Re-architecting.

Re-architecting is based on the concept that all legacy applications are a mix of business-relevant code and technical code that exists only to implement legacy technical support capabilities. Since the new environment provides much of this support functionality in other ways, the original technical code is no longer needed. Thus, Re-architecting focuses on recovering and reassembling the business-relevant code from a legacy application while eliminating as much of the technology-specific code as possible.

Re-architecting is typically used to handle modernizations that involve changes in architecture, such as the introduction of object orientation and process-driven services. In addition, Re-architecting recognizes that there is information other than application code that is useful as source for the re-architecting process, such as application process interaction, data models, and workflow. Re-architecting may also step outside of the legacy application to incorporate concepts such as workflow that were never part of the legacy application.

Re-architecting yields the highest modernization benefit—but is also the costliest approach.



Re-architecting is typically accomplished in four phases:

1) Phase 1: Recovery

Recovery extracts a present case model from the legacy environment.

In the recovery phase, the original application is analyzed using APA macro analysis and micro analysis techniques to create a clear understanding of both the application and the models behind it. These models are typically in the form of legacy modeling techniques, such as information engineering models. The sources for this information will typically include gathering information “top down” from user walkthroughs of the existing system, existing workflow and corporate process models as well as “bottom up” existing data models, data dictionaries, and the legacy application code itself.

The results of this analysis are typically placed in a repository in preparation for the later Re-factoring phase. In this manner the recovery step typically provides a platform-independent present case model of the current application design along with the analyzed code that implements the present case model. The code may be represented in the form of an abstract syntax or its original syntax.

Even though the complete re-architecting process will require human intervention, the recovery phase can often be highly automated—especially when it gathers information from existing computer sources.

Redesign defines a future case model using more modern modeling techniques.

2) Phase 2: Redesign

In the redesign phase, a future case model is developed based on SOA principles and new modeling techniques more conducive to producing applications that will execute on the SOA Modernization Architecture. New workflows are developed as necessary to cover the content of the present case model and these from the “use cases” of the new model. New business process steps are developed using techniques that align the new processes more completely with the business processes, and new graphic, system, and business-to-business interfaces are defined to cover the functionality of previous interfaces and/or add new interfaces.

An important aspect of the re-design phase is to note that re-design will often combine with COTS replacement, If COTS content exists that can be used to replace all or part of the legacy applications, then the COTS components will form part of the re-designed architecture.

In some cases, components of the future case model may correspond closely to the present case model, but in other cases re-architected changes or desired changes in business operation may mean the two models differ substantially. For example, the redesign phase may examine batch processes to determine which processes are “inherently” batch due to business requirements, and which are batch due to legacy technology constraints (such as CPU availability) and thus might be re-factored into more online processes.

In a number of cases, design concepts that were implemented via code in the legacy environment may also be replaced with functionality made available natively by the new environment. For example, reports may be replaced by data warehouses and reporting tools, batch job control may be replaced with orchestration flow, data edits may be replaced with database stored procedures, and some business rules may be mapped into a business rules engine. This type of mapping allows for a reduction in the number of lines of code in the new system, resulting in a more agile system that costs less to maintain.

Ideally, both the present case model and the future case model are stored in the same repository, setting the stage for the refactoring phase.

3) Phase 3: Refactoring

In the refactoring phase, the business logic of the recovered present case model is examined to first determine what can be mapped to the COTS components that are part of the new design architecture. Functionality for which COTS components exist can be mapped to the new COTS functionality. In this way, re-architecting provides a way to achieve “gap analysis” that is based on comparing the mined legacy content to the new application functionality, thus insuring a much more complete form of gap analysis than can be achieved by just talking to end users.

Refactoring maps the current business logic code to business logic code that supports the present case model.

For functionality that does not map to COTS or to new capabilities provided natively by the new technology environment, that functionality is transformed and reassembled into new SOA J2EE components based on the future case model. For example, a set of individual procedural routines to Add, Update, Approve, Pay, or Delete an Invoice would become methods on an Invoice Object used by a Change Invoice process, and then might be further re-factored to eliminate and reconcile duplicate edits that were previously repeated in a number of places due to “cut and paste” development.

Due to the nature of refactoring, it can never be completely automated. However, it can be partially automated using tools that analyze the recovered legacy content in order to find “candidates” for mapping to the COTS environment or for reassembly into new components. For example, a file or set of legacy files can be identified as candidates for a new business object. Once this has happened, code analysis can be used to “trace back” from any file updates to determine the code that impacts the field values that make up the file records. The actions performed by the code then become candidates for COTS capabilities or for the methods on a new business object.

The completion of re-factoring yields a future case application defined at a platform-independent level and designed to maximize the use of COTS, SOA and new environment capabilities. For content that cannot be re-factored to COTS, it is possible to re-factor directly to a target language such as Java - however it is often useful to target a higher-level abstract representation, such as an Application Development Framework, that makes use of generation techniques.

4) Phase 4: Regeneration

Regeneration creates platform-specific code using code generation techniques.

The final step in Re-architecting is re-generation. This step maps any non-COTS platform-independent re-factored models and code abstraction into a platform-dependent form. To carry out this mapping, the regeneration process can take advantage of application development frameworks and developer IDEs. These frameworks and tools make the creation of the final application simpler by providing both design templates and reusable implementation components that make maximum use of the new environment, while helping to hide any remaining platform specifics from the application code itself—thus increasing flexibility and lowering maintenance cost.

Although re-generation can be a manual process, in most cases the use of application frameworks and design patterns will facilitate automation. The regeneration techniques can also be used to incorporate platform specifics such as the final choices for specific Java technical architectures.

Since modernization via Re-architecting takes maximum advantage of the knowledge contained in the existing application, it is much less costly and less risky

than developing an application from scratch. At the same time, since it does involve a high degree of change, it has its risks.

As a result, a number of other modernization techniques have been developed that do not achieve all of the benefits obtained via re-architecting, but nonetheless can be steps on the overall path to gaining the full benefits of the new environment. These other techniques allow an organization to break the process of reaching the optimal modernization target into a series of steps, which—although more costly in total—may be less risky to carry out.

Automated Migration

Automated Migration is fast—but it only works when the “delta” between legacy and new architectures is minimized.

Some legacy applications make use of technology that is possible to migrate to a new technology in an automated fashion. Such a migration is considered automated only if at least 80 percent of the migration or transformation can be handled via technology rather than manually.

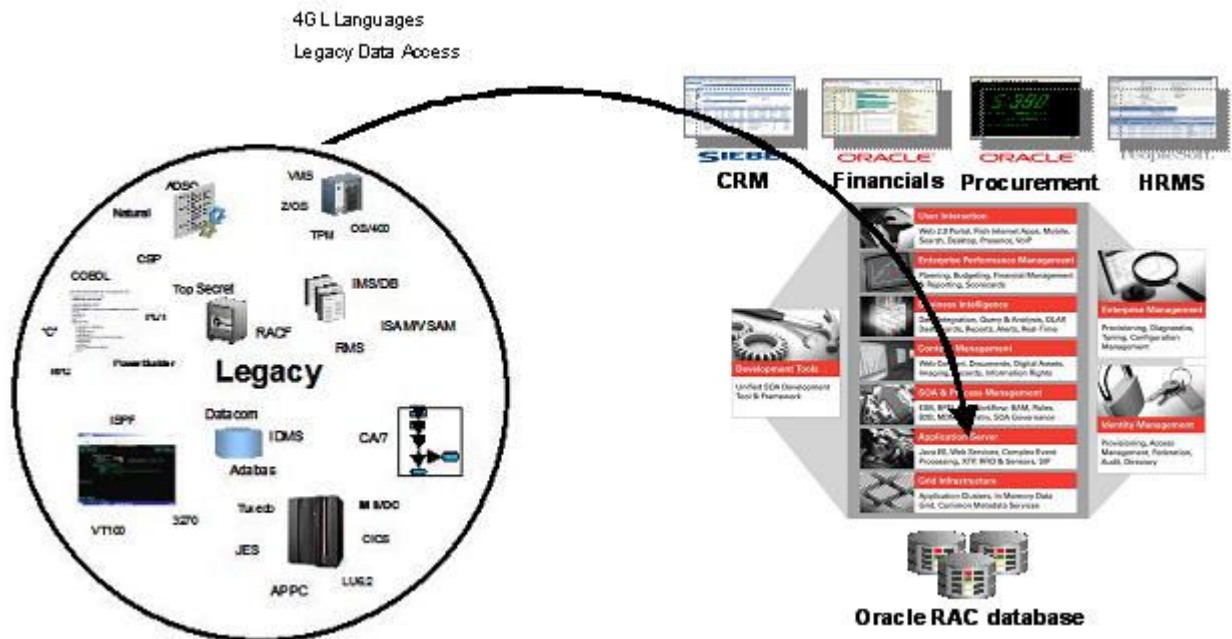


Figure 6 – Automated Migration

In order to reach this degree of automation, the migration or transformation process must be “algorithmic” in nature and not require the injection of human intelligence into the transformation process (as is needed for Re-architecting).

Automated Migration does not change the design of an application.

As a general rule, such migrations are most successful when there is a well-defined mapping between the source and target architectures. The easiest way to understand this is to note some examples where such a mapping does not exist.

1) Transforming from a procedural design to an object-oriented design.

Although it is possible to transform procedural code such as COBOL into an object-oriented (OO) language such as Java, it is not possible to map the procedural design that surrounds COBOL programs into the OO design that surrounds good Java programming. This is because the fundamental design concepts of OO—such as the class and its behavior,—are architectural concepts that require human intelligence to design. The designer of a truly OO application will use these techniques in new ways that cannot be recovered from an application developed using non-OO techniques. This is equally true when attempting to automatically transform legacy procedural flow into more modern modeling approaches such as Unified Modeling Language (UML). It can be done, but the resulting UML will be present case UML and not be the same UML that would be designed for more workflow driven application.

Thus, creating an OO design from a legacy application cannot be automated to a high level.

2) Transforming from pseudo-conversational code to conversational code

One of the most interesting concepts in the history of computing was the introduction of pseudo-conversational programming by IBM in the 1960s as part of their CICS and IMS/DC products. This concept, introduced at a time when machines were expensive and machine space at a premium, forces the dropping of context information, including such things as variable content and transactional content across each screen I/O.

Although somewhat similar in nature to the loss of context caused by a “thin client” HTML application, pseudo-conversational is still not the same as HTML when it comes to transaction handling, and it is very different from a conversational application, in which application context is always retained across any application display or event. It requires human intelligence to map a set of pseudo-conversational screens to conversational code, - thus transforming pseudo-conversational code to conversational code cannot be automated to a high degree.

3) Transforming from legacy procedures to business services

Like SOA Integration and Re-hosting, automated migration does not change core application structure. Therefore the ability to use migrated code as services carries they same restrictions as SOA Integration and re-hosting in that the legacy interfaces have to exist. If they do, they can be used but in many cases, “ideal” services do not exist in legacy applications; they simply were not designed that way.

Thus, although technical SOA interfaces may be created as part of an automated migration, business services that more closely follow the

business process are much more difficult to create automatically unless the legacy components needed to support the services already exist.

4) Transforming to a business process driven environment

To achieve the maximum benefit of the new environment, the modernization process must transform legacy applications to SOA services that can be orchestrated by an orchestration engine. In this case, a business process automation (BPA) tool is used to define both human and computer processes that are driven by a business process management (BPM) engine at runtime.

Unfortunately, most traditional applications either have no workflow concept outside of batch job control scripts, or if they do, the workflow is embedded deep within the code. Transforming these applications to accept the concept that workflow should be at the top of the architectural stack and drive the rest of the application is not something that can typically be automated. In fact, determining the current workflow may require interviewing current users, as all or part of the current workflow may not even be automated.

Thus, creating applications that make use of orchestrated SOA services that align with business processes cannot typically be done automatically.

5) Transforming from batch to online processing

Many legacy applications make extensive use of the batch-programming concept, where online transactions are recorded and then processed later, often overnight. The batch concept does not disappear in the new environment, as there are clearly some transactions that must continue to be processed in this manner. For example, stock trades may not be able to be valued until the stock market closes, and therefore final processing cannot occur until the end of the day.

At the same time, there are many traditional batch processes that can be wholly or partially modernized into online processes. Instead of waiting, incoming transactions can be processed immediately, resulting in more up-to-date information. The transformation of these batch processes to the correct forms of online processing requires human thought and cannot be highly automated.

In addition to architectural changes, the main additional concern of any Automated Migration process is the quality of the source code. If the source code is of poor quality, then transforming it automatically to another language or environment will not improve its quality.

All of that said, there are situations where the above issues can be either avoided or eliminated, thus making Automated Migration a suitable alternative. These include:

- 1) The migration of legacy applications and data that use legacy databases and file systems to a relational database

If the source and target design are similar enough, Automated Migration achieves results quickly and consistently.

Legacy applications written in 3GL languages such as COBOL and PL/1 that make use of legacy database and file systems such as VSAM, IMS/DB, ADABAS, IDMS and Datacom retrieve data by issuing calls embedded within the program code. Given a data model mapping from the legacy database or file formats to relational tables it is possible to use Automated Migration to automatically remove the calls and replace them with SQL – leaving the rest of the code alone. Additionally, the same data model mapping can be used to migrate the actual data from the legacy database or file system to a relational database.

- 2) The migration of applications written in fourth-generation languages to Java or re-hosted COBOL

Fourth-generation language (4GL) environments—such as NATURAL, IDEAL, ADSO, and PowerBuilder—typically consist of both a language and a runtime environment that provides additional capabilities in support of the executing programs. These fourth-generation languages were created in an attempt to move away from the complexity of third-generation languages and environments such as COBOL, PL/1, CICS, and IMS/DC by using a runtime environment to “abstract away” from legacy environments.

As a result, these 4GL environments are somewhat more modern architecturally than 3GL environments. This means that they do not suffer as much from some of the architectural issues that make automated migration virtually impossible for 3GL environments such as COBOL/CICS.

For example, it is possible to automatically migrate a NATURAL/ADABAS application to Java/Oracle. The resulting application will not have a completely object-oriented design, since the original NATURAL application was procedural - but because NATURAL applications are coded at an abstract level that makes use of a conversational programming style, the procedural Java code created will be conversational and can make use of object-oriented libraries that supply the needed environmental functionality. This results in a “mixed mode” application that still takes advantage of some new capabilities, is still reasonably maintainable, and whose code can be further modernized in order to attain the full benefit of the new environment.

Additionally, it is possible to automatically migrate legacy mainframe 4GL languages to COBOL and then re-host them. Although this may seem like a bit of a step backward, the fact that these 4GL languages are typically procedural in nature and use mainframe concepts such as mainframe data typing means that migrating them to COBOL is relatively straightforward. As part of the migration to COBOL, the legacy databases typically associated with these 4GLs can also be eliminated with the result being re-hosted on a relational database. This technique is particularly useful with the legacy 4GL application is intertwined with legacy COBOL.

3) Program restructuring

In some cases it may be useful to clean up code prior to carrying out a specific modernization exercise. Automated migration techniques can be used to perform this cleanup. In this case, the source and target language may well be the same. Program restructuring can be done to carry out dead code elimination, GOTO elimination, or loop restructuring—activities that can be useful in preparing the code for other types of modernization.

The clear advantages of automated migration are speed and consistency. Since a computer carries out the modernization process, it can be done quickly and consistently, and can even be repeated on a more recent copy of the source code in order to include ongoing changes. Modernization will be done the same way every time, so although automated migration is more invasive than either SOA Integration or Re-hosting, the degree of testing and risk will be considerably lower than with any manual effort.

The disadvantage of automated migration is that only algorithmic transformations can be made. If the goal of a modernization effort is to make major change to architecture and application design, automated migration will not work.

Re-hosting

Re-hosting involves migrating an application to another platform while leaving the application largely untouched. Many legacy applications execute within an environment that provides facilities such as terminal handling and file services for the application. One of the most common of these environments is IBM's CICS, which is a very complex environment with its own programming style.

Additionally, many of these CICS-based applications also make use of IBM's early file systems such as VSAM.

In order to facilitate modernizing these complex legacy applications into a new environment, it may be possible to migrate the applications to a re-hosting environment that supports the same environment as the legacy platform.

Although in theory any number of legacy technology environments could be supported in this manner, the most commonly used environment for re-hosting consists of an online CICS and batch JCL environment executing COBOL or PL/1 programs accessing VSAM files.

“One estimate says that COBOL/CICS applications account for 60% of all the applications that are currently in operation, and another estimate states that these applications process 85% of all the transactions that are processed”

Database Journal

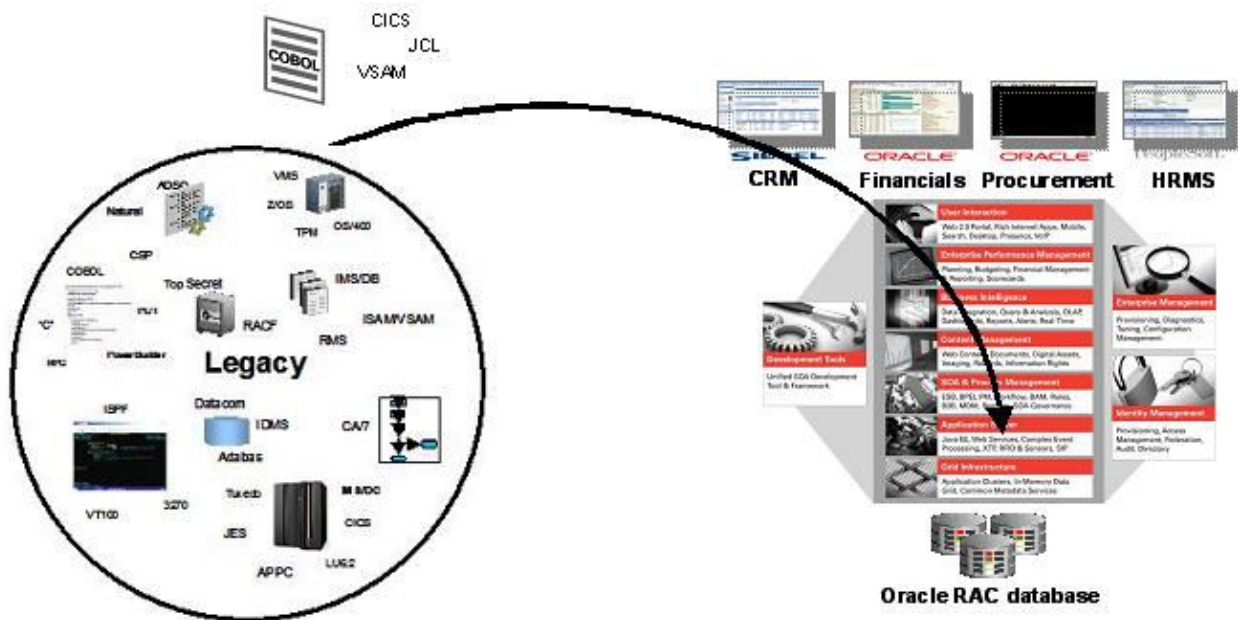


Figure 5 – Re-hosting

Although re-hosting environments do not cover all legacy technologies, re-hosting can still be used in combination with other modernization techniques such as automated migration. For example, an organization can eliminate a third-party legacy database such as ADABAS, IDMS or Datacom from their COBOL code during the re-hosting process by transforming the legacy data calls to access an Oracle Database. Similarly, legacy code using other IBM teleprocessing environments such as IMS/DC can be automatically migrated to CICS and then re-hosted.

Re-hosting provides the ability to move some applications to another platform without changing the core application—although some consideration must be given to any external environmental functions used by the application that are not part of the re-hosting environment. Since the application itself does not change, the clear advantage to this approach occurs when the new platform has a lower TCO than the original one, as occurs when moving from a mainframe platform to an open systems platform such as UNIX or Linux.

Re-hosting in combination with SOA Integration can be a further step toward a better environment that the re-hosted COBOL components can be SOA-integrated, with the resulting SOA services managed and orchestrated by the process orchestration mechanism of the new environment. In fact, SOA-integrating a re-hosted application can be preferable to SOA-integrating an application in its original environment because the re-hosted SOA-integrated services can execute on the same platform as other SOA services, eliminating the need for communication among disparate environments. Having the re-hosted

code on the same platform also makes it easier to further modernize the implementation of individual SOA services to Java.

The disadvantage of Re-hosting, similar to SOA Integration, is that although re-hosting can reduce total cost of ownership, it retains much of the legacy architecture and programming languages. This means that the services implementation itself could be cumbersome and the legacy implementation means a continued reliance on legacy skill sets. Thus not all of the potential reduction in TCO is realized.

SUMMARY

Legacy applications are an increasing business problem. They carry a high cost of ownership, they can't be changed easily to react quickly to ongoing business demands, maintaining them requires a legacy skill set that fewer and fewer people hold, and they do not meet compliance demands. To address these issues, organizations must modernize their applications by moving to an SOA environment that provides User Interaction, Business Intelligence, Integration, Process Management, Application Server and Grid Infrastructure capabilities.

Although it is possible to develop applications from scratch that fully utilize new technology concepts, this is an expensive and highly risky approach. A better approach is to make use of Modernization.

Modernization allows organizations to maximize the use of their existing application assets as they move toward better technology environments. Maximum use of these environments is achieved through Re-architecting or COTS Replacement, but several other modernization approaches—SOA Integration, Re-hosting, and Automated Migration—can also be used as steps in the path to achieving the desired final goals.

With computer systems being an integral part of today's business, achieving the benefits provided by modern technology is no longer optional. Organizations that do not begin the move today will find their business lagging behind those who are aligning their IT strategy with business goals in a cost-effective manner by using modernization to reduce costs, increase agility, reduce reliance on legacy skill sets and improve compliance.



The Oracle IT Modernization Series: The Types of Modernization

September 2008

Author: Ted Venema

Contact: oracle-modernization_us@oracle.com

For more information: www.oracle.com/technologies/modernization

Oracle Corporation

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

oracle.com

Copyright © 2008, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.