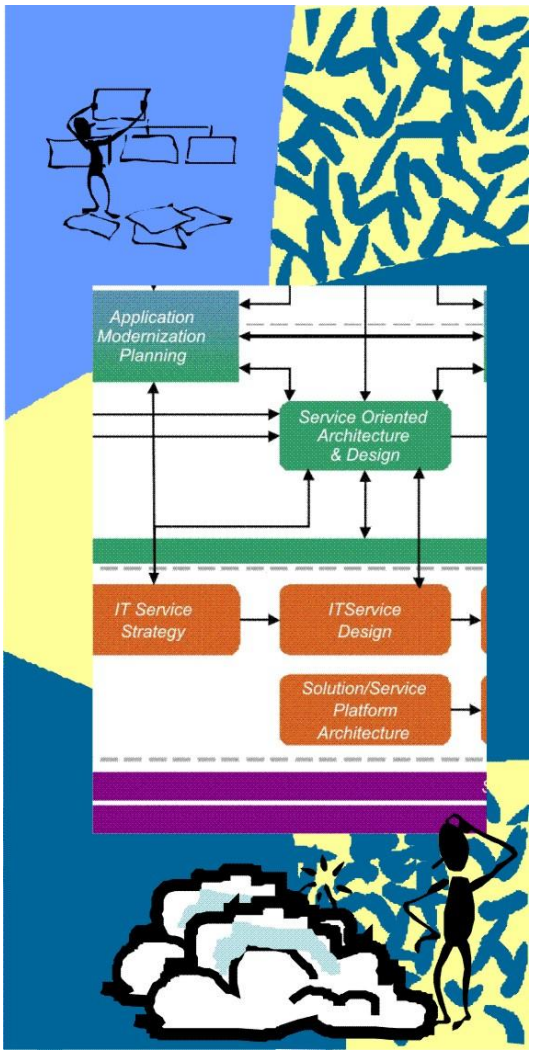
CBDI Journal



Independent Guidance *for* Service Architecture and Engineering

Case Study Report

Application Modernization Project - Portfolio Pathfinders

In this report, we narrate an actual modernization project undertaken by Everware-CBDI North America. The case study illustrates how the Agile Application Modernization approach is used in practice to deliver demonstrably modernized solutions that reflect future business needs. In this scenario we describe how we addressed specific application modernization project needs, while demonstrating how a limited portfolio assessment can significantly improve the outcome.

By Denzil Wasson

Originally published in the March 2010 edition of the CBDI Journal

Case Study Report: Application Modernization Project - Portfolio Pathfinders

In this report, we narrate an actual modernization project undertaken by Everware-CBDI North America. The case study illustrates how the Agile Application Modernization approach is used in practice to deliver demonstrably modernized solutions that reflect future business needs. In this scenario we describe how we addressed specific application modernization project needs, while demonstrating how a limited portfolio assessment can significantly improve the outcome.

Denzil Wasson

Introduction

In previous CBDI reports[1], we have described a framework and motivation for the adoption of Agile Application Modernization. While the framework is all encompassing, illustrates what is possible and in theory allows for a hyperspace jump from some tightly coupled anti-agile legacy quagmire to an agile business oriented solution portfolio, this may not be practical in all situations. This case study narrates a project that pragmatically makes a less than hyperspace jump in modernization, yet still leaves the organization on the right trajectory.

This case study is not intended to educate you on how to perform modernization transformations but rather to give you insight into what such a transformation looks like in the context of our modernization approach. We anticipate that most Clients will engage with modernization experts to perform the transient activities of modernization since that is likely the most efficient path to modernization and minimizes unnecessary investments in these transient skills and tools.

However if you want a strategic architectural outcome, then engaging with experts does not entail delegating all modernization responsibility. In this case study we collaborate with our Client in a business and architecture driven approach to modernization and we leave the Client in a superior architectural position rather than in the same architectural position expressed in a new technology.

Client Background

This large North American retail and logistics organization finds itself in a typical IT architecture situation. There is an agreed conceptual architectural vision of the future that is defined as less costly, less complex and more responsive to the business. However there is uncertainty as to what this means in terms of realization.

Scenarios range from outsourcing the management of the portfolio (parts of it are already outsourced), homogenizing the portfolio to facilitate interoperation and management (perhaps being able to outsource more since the outsource partner cannot support some of the legacy technologies), through to adopting SOA (although there is uncertainty as to what this implies even though they know they have duplication and congruence in the current portfolio). Each of these scenarios holds some challenge for

the Client so the stopgap tactic at this point is homogenization of the portfolio to facilitate further outsourcing.

In reality, legacy applications continue to be enhanced and maintained; new application stovepipes are being built in spite of the fact that the architecture group has yet to define and communicate the desired future state more formally.

In addition several legacy platforms are rapidly becoming burning platforms from both a Vendor support and skills perspective.

In a bid for some level of meaningful influence, the architecture team has defined the boundaries for new solution development in both very broad conceptual terms (difficult to govern?) and very fine-grained technical terms (low value but necessary?). Examples of these include:

- Minimize investment in legacy technology whenever possible. (A slippery slope in the face of powerful Business stakeholders?).
- Here is a big catch all QA gate prior to production. (A side effect of the outsourcing relationship?).
- Use industry standards such as Sun Java coding standards. (A pretty loose constraint?).
- Use Java. (The language is homogenous, what about design, what about architecture?).
- Use these specific development tools. (This is pretty common at most organizations).
- Use these specific java application frameworks. (Might this be an attempt to constrain architecture and design a little?).
- Here is a set of foundational classes providing common capabilities that you should use within your solution. (This is interesting and points to a desire for re-use and potential foundation for a future architectural state).
- Here is an automated code scan tool with configured rules that all code must pass through prior to QA. (Also pretty common, a good practice but provides marginal value when compared to governance and concomitant visibility of design, architecture and portfolio level decisions).

These properties are easy to motivate and communicate but unfortunately do not guarantee evolution towards the real desired future state. There is also a lack of visibility into the portfolio from an architecture perspective that is not being addressed through these guidelines.

Our Engagement Background

Everware-CBDI's expertise in consulting at the architecture level and performing various transformations ranging from tactical re-platforming through to enterprise wide architecture modernization led to our engagement for this project.

During the pre-sales effort, which in our mind forms the initial part of the assessment phase of the modernization, we discovered a larger problem.

The Client initially wanted to engage us on the re-platforming of a single legacy application as a pathfinder in moving from a burning legacy platform to a new Java

based platform. However we quickly came to understand that there are actually a number of legacy solutions that require some form of modernization. Since we believe that there is value in strategic approaches to modernization, we immediately suggested that the whole portfolio should be considered during at least the assessment and high-level planning stages of the modernization project.

We helped the Client understand that a portfolio level assessment would allow the modernization effort to achieve more benefits than a stovepipe-by-stovepipe re-platforming effort. However the need to provide a pathfinder project to Management precluded the portfolio level assessment from occurring at that time. As a pragmatic compromise we suggested that perhaps instead of simply re-platforming a single application, we would tackle a slice of two disparate applications (as a mini portfolio), thereby providing the following insights and benefits:

- The beneficial outcome of a 'wider than single stovepipe' perspective on modernization can be illustrated to management.
- We can illustrate the identification of similar / redundant functionality and the evolution of common capabilities into formalized encapsulations (these could vary from re-used libraries all the way through to formal services in a later SOA). The goal here is to show that through a strategic approach to modernization the Customer can achieve their goals of less complexity, lower cost of ownership and higher responsiveness to business change.
- We can demonstrate the accelerating pace of modernization that can be achieved by adopting a portfolio approach since each project does not have to re-invent various items each time and is able to re-use approaches, heuristics, knowledge, architecture and design patterns through to actual code or services from prior iterations.
- We introduce our repeatable methods to the Customer. This hybrid of agile and traditional methodologies, balances essential formal deliverables with actual code delivery. We do this by focusing heavily on the delta between the 'as-is' state and the 'to-be' state thereby minimizing the effort of transformation. At all costs we avoid going-back to first principles wherever possible but leverage the knowledge within the legacy application.
- The two applications have different legacy technical architectures so we are able to prove the modernization of both a GUI client/server application and a mainframe hosted terminal based (green screen) application.
- We are able to introduce the Customer to several important concepts with regard to our approach to modernization. Including:
    - Baseline and iteration vs. big bang – it is our experience that unless there is significant change in requirements, early iterations should replicate the current functionality (baseline) and then evolve the new solution (iteration) from there. This does not preclude restructuring and re-architecting towards SOA for example. This approach provides an important measurable checkpoint for the business in the modernization process and reduces the tendency of the transformation team to re-invent the wheel (revert to green-field / first principle development approaches).

- Pattern based analysis, architecture, design and transformation – We focus heavily on identifying patterns within the legacy portfolio. Most portfolios exhibit similar approaches to solving business problems. These could extend from data modeling practices through to technical solutions due to platform constraints. The objective of this approach is to leverage patterns in the transformation process. Similar to GoF design patterns, we are able to then specify things in terms of a pattern and that has immediate meaning to the transformation team without exhaustively re-specifying each instance of the pattern. As an example we may refer to a use case as a CRUD-Search pattern. This implies that the use case allows for the maintenance and searching of a particular major data element. An example of the use of this would be during assessment where we say "There are 23 instances of the CRUD-Search use case patterns that manage the following data elements: xx, yy, zz …". New pattern names may be introduced as required as long as they are defined. Note also that when a pattern is identified during the process we can re-use prior knowledge (heuristics) as to how that pattern is usually transformed and we may even leverage automation capabilities for common patterns.
- Functional equivalence vs. implementation equivalence – This is an important aspect of modernization that allows one to consider modernization as more than a simple re-platforming exercise. Note that we refer to equivalence rather than equality since equivalence has the same net effect but may not be the same. Consider that in most instances the Business needs to be able to at least perform the same function; however the implementation of that function can be vastly different ranging from a cheaper runtime platform through to leveraging some cloud based capabilities to achieve the function. That is to say the 'what' needs to be equivalent but the 'how' can vary significantly. This distinction affords the modernization effort many opportunities to optimize the transformation effort and the resulting solution.
- Functional redundancy/duplication vs. functional congruency – We often encounter resistance to SOA and other re-use strategies based on the fact that although the legacy portfolio may exhibit similarities in function, they are not identical and thus those functions cannot be factored out as re-usable. During modernization, we purposely look for similar (i.e. congruent) functionality rather than only identical. We find that many legacy portfolios exhibit prior attempts at re-use or 'clone and own' activity where once similar functionality has devolved into solution specific code. During modernization we have the opportunity to identify these similarities and encapsulate them as re-usable capabilities (candidate library functions or even services). Even if the initial versions must tactically support multiple interfaces, we have at least been successful in recognizing the capability and factoring it out of the code base. The identification of these kinds of functionality is an important benefit that can only be achieved by adopting a portfolio-based approach to modernization.

- Meet in the middle service growth - In our experience building enterprise services can actually be inhibited by adhering too strongly to one of two common approaches; 1) Top-Down – the scope and difficulty of getting a service specified, designed and implemented correctly in a top down fashion is difficult to motivate let alone achieve in the real world due to many factors including scope, dependencies and timing (ivory tower services). 2) Bottom-Up – a recipe for service anarchy since the provisioner's perspective is only a single project and no evolutionary path beyond that exists. In contrast our approach to modernization aims to grow services in a meet in the middle approach. Candidate services are initially identified using a top down approach, but are then specified, designed and built (or enhanced) during iteration, with the requirements of that iteration in mind but guided by the strategic architectural vision. They are then evolved as we iterate through the portfolio. The goal of this approach is to pragmatically realize real world services that are already reused and can be evolved as required. Taking a portfolio approach that allows proper sequencing of the modernization is critical to ensure the success of the meet in the middle service growth approach.

The Client agreed and selected a representative slice of two discrete applications as the scope of the pathfinder. The applications were both stable legacy applications that maintain data that is core to the enterprise. The first is a location (retail store and distribution center) configuration data repository called Corporate Directory while the second is a distribution center capacity planning application called DC Capacity.

We commenced the project by assigning a principal modernization and service architect to conduct the assessment.

Assessment Phase

Following our own published AM methodology, we commenced the engagement with a rapid portfolio (in this case mini-portfolio) assessment. The goal of the assessment phase was to understand any new business requirements, assess the current system and understand the Client's vision for the future state of this functionality. The assessment comprised assessing the current system assets, conducting structured interviews with relevant Client staff and facilitating structured sessions to establish a common future state vision.

The tooling for assessing the current systems assets was a combination of commercially available and Everware-CBDI proprietary tooling. For this legacy environment, the Jumar Solutions Project Phoenix[2] tools allowed us to parse and extract information from the existing code base, which we then converted into a common format loosely based around KDM type packages that we use for all assessments. We were then able to view this data from several perspectives to fulfill our assessment needs. As an example, we were able to conduct object counts and complexity analyses using certain properties and calculations that have been developed by Everware-CBDI.

The assessment activities were guided by the need to complete our modernization assessment template. This assessment data store represents the starting point of our knowledge base for the modernization effort and establishes the outline view of the AS-IS solution/portfolio and TO-BE solution/portfolio, proposes a viable approach to

modernization, establishes the scope of the modernization and provides high-level estimates of the effort. It is important to note that this information is gathered at a high-level and we are looking at major aspects of the scope under assessment. Further, some of the information may already exist at the Client, in which case we merely reference that information within the assessment.

Table 1 below conveys the main content of the assessment template.

Perspective	View	Content
AS-IS	Business (Process view, Data view)	Major business functions supported Major Use Cases Major Data elements Use Case patterns Data structure / modeling patterns Redundant / congruent process elements (within or across solutions) Redundant / congruent data elements (within or across solutions) Affinity and process / data lifecycle matrices
AS-IS	Architecture (Logical view)	Abstract architecture into a logical view Identify architectural pattern(s) Identify integration points
AS-IS	Design Aspects (Implementation view)	Identify design standards / patterns used Identify design solutions implemented for: • Security (authentication & authorization) • Exception handling • Auditing / Logging • Code/Reference/Look up tables • Identifiers • Concurrency control • Integration • Batch • UI • Reporting • Persistence • Session state • Re-use • Other

Perspective	View	Content
AS-IS	Quality	Reference to any QA standards or test assets
AS-IS	Non-Functional	Service levels Sizing and performance
AS-IS	Metrics	Counts and complexities including mapping of perceived patterns to pattern instances Metrics for planning purposes
TO-BE	Business (Process view, Data view)	Business motivation for modernization Business agility requirement High-level business delta between AS-IS and TO-BE - Gap Analysis. This may be None, New requirements, Business Process Optimization, Data Consolidation or mapping current capability to a re-usable asset or service. High level Use Case and Data models (delta only)
TO-BE	Architecture (Logical view)	Architectural vision and principles summary. Identify TO-BE architecture patterns and high level mapping from AS-IS architecture
TO-BE	Design (Implementation View)	Identify TO-BE design standards Identify TO-BE design patterns Map AS-IS design solutions to TO-BE design Identify capabilities (may be utility, underlying and core services) to service common functional requirements in the solution/portfolio
TO-BE	Deployment View	High level deployment platform description
TO-BE	Quality	Reference to any desired QA standards or test assets
TO-BE	Non-Functional	Delta to AS-IS Non-functional
Project	Strategy	Modernization strategy recommendation
Project	Planning (Metrics)	High-level estimate for modernization effort

Table 1 – Assessment Template Content

In this instance where we were working with a portfolio of solutions rather than a single solution, we performed the assessment activity on both solutions prior to moving

forward with other iterations. This allowed us to realize and demonstrate the benefits of the portfolio-based approach.

Common assessment findings that apply to both applications in scope are shown in Table-2 below. These findings can be re-used for these two applications and most other legacy applications within this Client. This is an important aspect of our approach that allows subsequent modernization assessment iterations to be more productive even if we are assessing new legacy assets.

Perspective/ View	Content
AS-IS / Business (common)	Significant Location data and processes are perceived within both applications Major conceptual Use Case patterns: Find Object and then Manage Object Major Conceptual Data patterns: For location – Core entity with tightly coupled property entities For code tables – multiple individual tables that exhibit code / description / and one or two other properties (e.g. short description) Common major data elements – Location, Country, Location Type, Employee Business wants consistency of implementations so that Users' application suite behaves consistently. Business Users are happy to use the new application for some Use Cases and the old application for other Use Cases with the understanding that over time all Use Cases will move into the new application suite.
AS-IS / Architecture (common)	Stove-pipe architecture(s) Common database Integration via common database Integration via TP monitor level calls (e.g. CICS transtion to transaction calls)
AS-IS / Design Aspects (common)	See individual application assessments
AS-IS / Quality (common)	No prior formal QA assets.
AS-IS / Non-Functional	See individual application assessments
AS-IS / Metrics	See individual application assessments

Perspective/ View	Content
TO-BE / Business (common)	Modernization required to facilitate homogenization for outsourcing Burning legacy platform Move towards common services The applications in scope are stable. Thus the ability to perform rapid changes is not crucial however it is desired that duplicate or similar functionality be factored out of the applications to optimize future enhancement efforts. No significant Use Case or Data model changes are anticipated since there are no new requirements. Other applications currently couple directly to some of the underlying tables so the schema need to (at least initially) remain stable
TO-BE / Architecture (common)	Homogenous technical architecture Model View Controller application patterns Web applications using Spring MVC Encapsulated, separately managed re-use in anticipation of services
TO-BE / Design (common)	Standard libraries java, jsp, jstl, spring, displaytag and custom (in house developed) POJO classes (minimize third party dependencies) Dependency injection (Spring) so that dependencies are made explicit Indirection to common services (consume services via delegates) to minimize evolution overhead when adopting a pure SOA Data Access Object pattern for persistence using JDBC Standard (IBM) RAD project structures Minimize web application session data to facilitate load balancing and fail over Common capabilities (re-usable future services): Code tables / reference data (cached) Authentication & Authorization Logging Auditing
TO-BE / Deployment (common)	Tomcat web servers (multi-node load balanced), Websphere application server (multi-node, load balanced), DB2 database
TO-BE / Quality (common)	New QA team is starting to document manual test cases (scripts) that will be leveraged against the transformed code. Re-useable services should have automated regression tests to facilitate certification and trust.

Perspective/ View	Content
TO-BE / Non-Functional (common)	See individual application assessments
Project Strategy (common)	In the absence of significant stakeholder resistance we would like to baseline 'what we have today' and then iterate improvements within the new technology platform. Re-use is desired and re-usable elements should be encapsulated in a foundational toolkit (libraries) for a potential future SOA deployment. Identification, extraction and standardization of common functions are thus important. Several of the applications in this portfolio share a common database (directly rather than via some data access pattern). This affords a good opportunity to potentially introduce some core services.
Project Planning (common)	See individual application assessments

Table 2 – Common Assessment Findings

Corporate Directory assessment findings (summarized):

Perspective/ View	Content
AS-IS / Business (Corp Dir)	Major function: Management of various location types and associated properties. The application essentially represents a master data store for Location data. Major Use patterns - management of master data and associated properties Major Data patterns – parent and child property structures, typed data, denormalized reference data / code table data (hidden associations) Major process elements – Location Management, Location layout Management, Location Footage Management, Location type management Major data elements – Location, Location Group, Location Type, Location Square Footage
AS-IS / Architecture (Corp Dir)	Classic mainframe stovepipe Some common logic factored in CICS NCAL modules Database and transaction level coupling – applications that require Location data read the database directly.

Perspective/ View	Content
AS-IS / Design (Corp Dir)	A COBOL mainframe terminal based application executing in CICS and persisting via SQL to DB2. List detail patterns with prompt flows for code table / reference values. User initiated paging over large datasets. • Authentication via mainframe userid / password • No role based authorization • Basic timestamp / userid auditing. No Logging. • Separate modules for code tables (prompting) • Data re-use through database level coupling. No perceived process re-use. • Integration at database level. Hard coded transaction-to-transaction flows. • No batch in scope • Natural business Identifiers • Consistent 3270 UI with UI standards • No reporting in scope • Persistence via embedded SQL • Session state passed around between modules.
AS-IS / Quality (Corp Dir)	See common
AS-IS / Non-Functional (Corp Dir)	Business hours availability. No formal performance service level.
AS-IS / Metrics (Corp Dir)	High level counts and complexities (within scope of slice not whole application): 5 DB2 data tables with average attribute density average of 14 and average association density of 2.1 (higher densities usually imply more complexity). 4 Reference / code tables. 5 CICS transactions 2 of simple and 3 of medium complexity (based on our complexity measures)
TO-BE / Business (Corp Dir)	Fundamental business is unchanged. Workflow changes (optimization) are anticipated since the functionality is moving from the restrictive terminal based UI to the richer web UI.
TO-BE / Architecture (Corp Dir)	See Common

Perspective/ View	Content
TO-BE / Design (Corp Dir)	See Common
TO-BE / Deployment (Corp Dir)	See Common
TO-BE / Quality (Corp Dir)	See Common
TO-BE / Non-Functional (Corp Dir)	No change
Project Strategy (Corp Dir)	See Common
Project Planning (Corp Dir)	The scope in question disregarding any re-use appears to be a six-week problem for two developers.

Table 3 – Corporate Directory Assessment Findings

DC Capacity assessment findings (summarized):

Perspective/ View	Content
AS-IS / Business (DC Cap.)	Major function: Management of distribution capacity planning and capacity exceptions. See common
AS-IS / Architecture (DC Cap.)	Stovepipe Client / Server architecture Some common server logic factored in CICS NCAL modules. Some common client logic factored into utility classes. Database and transaction level coupling

Perspective/ View	Content
AS-IS / Design (DC Cap.)	This is a fat client/server application where the C++ clients execute on Windows and communicate with CICS COBOL servers via SNA. The servers persist via SQL to DB2. • Authentication via client OS for access to clients. Authentication via mainframe userid / password for servers • No role based authorization • Basic timestamp / userid auditing. No Logging • Drop downs for code tables – not factored out to common • Data re-use through database level coupling. No perceived process re-use. • Integration at database level. Transaction to transaction flows. • No batch in scope • Natural business Identifiers • Consistent GUI with GUI standards • No reporting in scope • Persistence via embedded SQL within servers • Session state passed around between modules.
AS-IS / Quality (DC Cap.)	See common
AS-IS / Non-Functional (DC Cap.)	Business hours availability. No formal performance service level.
AS-IS / Metrics (DC Cap.)	High level counts and complexities (within scope of slice not whole application): 5 DB2 data tables with average attribute density average of 12 and average association density of 1.4 (higher densities usually imply more complexity). 3 Reference / code tables. 6 client/server transaction pairs 3 of simple, 1of medium and 2 of complex complexity (based on our complexity measures)
TO-BE / Business (DC Cap.)	Fundamental business is unchanged. Workflow changes (optimization) are anticipated from the older style GUI to the web UI however these should be intuitive enough to avoid significant User training.
TO-BE / Architecture (DC Cap.)	See Common
TO-BE / Design (DC Cap.)	See Common
TO-BE / Deployment (DC Cap.)	See Common

Perspective/ View	Content
TO-BE / Quality (DC Cap.)	See Common
TO-BE / Non-Functional (DC Cap.)	No Change
Project Strategy (DC Cap.)	See Common
Project Planning (DC Cap.)	The scope in question disregarding any re-use appears to be a five-week problem for two developers.

Table 4 – DC Capacity Assessment Findings

The assessment results brought the Client's most important needs to the forefront and show that a consistent, mature approach to assessing the portfolio will pay off for the modernization effort.

Planning Phase

The planning phase produces plans and outline deliverables ranging from business improvement through to platform modernization. The assessment revealed that there were several aspects of planning that we need not concern ourselves with, such as:

- Plan solution platform – the Client has rolled out the TO-BE platform and associated management capabilities.
- Business improvement - The applications in scope are stable with few changes anticipated so no significant business improvement is anticipated. There may be workflow optimizations as a side effect of modernization.
- Produce application modernization / SOA reference framework plan – This pathfinder is an exploratory effort that the Client is using to help shape what such a plan would look like. Therefore this is likely to be produced in a subsequent iteration after the results of the pathfinder have been evaluated.
- Provision application modernization / SOA reference framework – This pathfinder project will be used as input to the definition of such a framework.

Relevant planning focus areas are:

- Prepare and evolve application modernization plan – We needed to prepare the plan for the modernization of these two application slices. This will also provide a pathfinder work breakdown structure tailored to this Client. Using our modernization work break down structure and estimating heuristics we produce the modernization plan. The first iteration of modernization was estimated at six weeks long for the first application (Corporate Directory) merely three weeks for the second application. The timeframe is halved due to the re-use of approach, standards, design elements, implementation artifacts

and utility and core services. The anticipated team is the modernization architect and 1.5 senior technologists (designer/programmer level).

- Identify Candidate components for modernization – Using the assessment business knowledge, actual effects analysis and affinity matrices from assessment, we identify portions of each solution for modernization. We find components that provide important use case functionality, provide representative technical complexity and are cohesive enough to be a useful iteration. We also identify several base components to provide infrastructure level capabilities that all iterations will leverage. We also superimpose these selection criteria from both applications in order to identify components that have some functional congruence to promote our re-use objectives. We identify three candidate services at this juncture, being a core Location service, a utility Code Table service that will offer generalized capabilities for all reference / code table capabilities, a utility service that provides for authorization capabilities.
- Evolve service portfolio plan – While we were not strictly producing a formal service portfolio plan, we still wanted to identify and plan the delivery of reuseable functionality. Although they may not be implemented as services, we wanted to be aware of their availability, lifecycle status and all dependencies to promote their re-use and potential evolution to true services.
- Produce service implementation architecture – We planned to produce an implementation architecture for the re-useable capabilities that will facilitate evolution to services. At the broad (planning level) this includes principles such as encapsulation in 'black box' components to encourage re-use as is, re-use via delegates (indirection) to minimize coupling to consuming applications and minimize dependencies.
- Produce service specification architecture – The client is unlikely to invest in true rich service specifications at this point but we are able to specify the service functionalities within javadoc comments and the service dependencies are made explicit through the configuration files used by the Spring framework (although this does require some governance to ensure that all invocations of services are on injected classes).
- Propose solution architecture – We defined an overarching Model / Model-View-Controller architecture as the client has desired. However within this outline architecture we defined the architecture strategies for the re-use of the identified services. We propose the adoption of standard interfaces to these services that will facilitate their consumption. These interfaces include java interfaces as well as custom JSP tags that consume the authorization services.

During planning we utilized a collection of tools that included metrics extracted from the code base; sizing, complexity and effort calculations; common architecture patterns; legacy architecture mapping heuristics; template work break down structures, estimating worksheets and the TO-BE state that was defined during assessment.

Once we had the plans in place and the deliverables outlined for this iteration, we sought stakeholder approval to proceed with the Analysis phase.

Analysis Phase

During analysis we drill down into enough detail on the deliverables to enable the deliver phase. Since each project is unique, we selected the relevant elements from our modernization framework.

- Specify TO-BE Business – Since there were no new requirements and the applications were stable, there was no new business requirements analysis and modeling. However we did still produce UML models by mining the legacy assets and producing outline data, structural and process models. We also specified the changes to User workflows via prototyping.
- Analyze Current System - We analyzed the data, structure and process models that we were able to automatically derive from the legacy code base. We used our pattern-based approach to select portions of the system to analyze. For example we perceived that one of the applications had a repeated pattern of separate modules that would create, modify and copy certain items – the congruence of these functions discovered during our analysis led to us combining these functions within a single module thereby actually reducing the size of the resultant code base with only a marginal increase in complexity. We also used our complexity numbers and Client subject matter experts' input to point us to the likely location of unique business rules that need to be analyzed. During this analysis we were also able to confirm and refine our initial list of service and service operations.
- Prepare Current System Portfolio Assessment – The detailed gap analysis performed showed us that there was no business gap but that there was a user interaction gap and a structural gap that would need to be addressed. We also identified a gap in the data structures that would need to be addressed in the longer term. This gap revolved around the proliferation and duplication of code / reference tables and associated lookup (read) and management functionality. Essentially in the longer term we would like to consolidate this functionality and data within a single generalized utility service. In the short term we can combine the functionality but not the data. In the longer term we plan to combine the data, once all direct reference to the tables have been modernized. Figure 1 below illustrates the planned evolution of code tables. Note that over time the spaghetti is removed and the result is a lot more agile since we:
  - will easily be able to introduce new code tables
  - will be confident that all code tables are consistent
  - will be able to easily control the reading and management of code tables
  - easily understand the dependencies on code tables
  - and we have significantly reduced redundant code table management and lookup functionality.

  We are also able to evolve towards the strategic picture without introducing massive change into the remainder of the portfolio right now. This strategy can be brought to bear for any functionality that exhibits structural similarity and is appropriate to generalize.

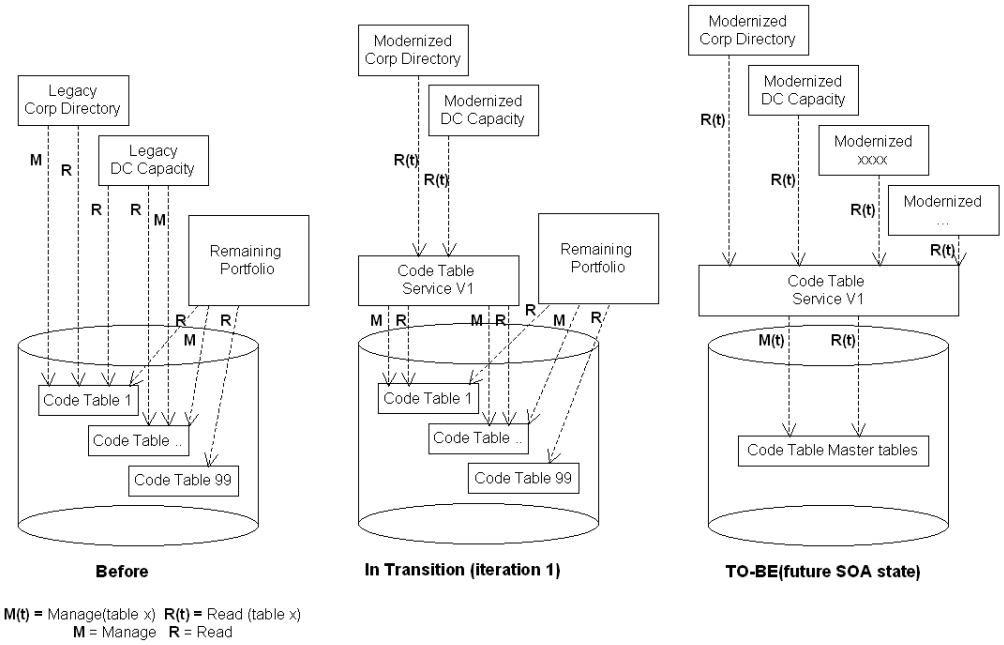


Figure 1 – Pragmatic Evolution of code tables

- Evolve modernization plan – We reviewed the modernization strategy in more detail to ensure that the proposed strategy was workable. For example is it realistic to consolidate the code table functionality? We reviewed the planning estimates against the most simple and most complex actual units of work to ensure that we didn't significantly under or overestimate the work due to the unique characteristics of the work unit. This essentially amounted to a manual sampling exercise to ensure that we provided the most accurate plan possible. These two perspectives allowed us to produce the final increment plan.
- Evolve the service portfolio plan – We detailed the previously identified services with specifications of their interfaces, and details of their implementation and subsequent management. This focus on management after the fact is crucial to ensure that the work done in this iteration does not devolve over time into the previous practice of 'clone and own'.
- Propose solution architecture – We finalized the solution architecture including details of the overarching architecture and the mapping of the legacy architecture to the new architecture. We also focused on the details of lower level architecture considerations including: How the services would be encapsulated and shared as 'black boxes'; what use case functionalities would be combined; detailing short term and longer term structural changes; specifying the required quality attributes and the QA architecture. We also conducted an architectural proof of concept that we felt was required to prove the proposed architecture. Figure 2 is a layered dependency diagram that illustrates the before and after architectures at a high level. The packages that have been factored out from the Corp Directory and DC Capacity code base are re-usable and represent a step towards re-use and eventual services.

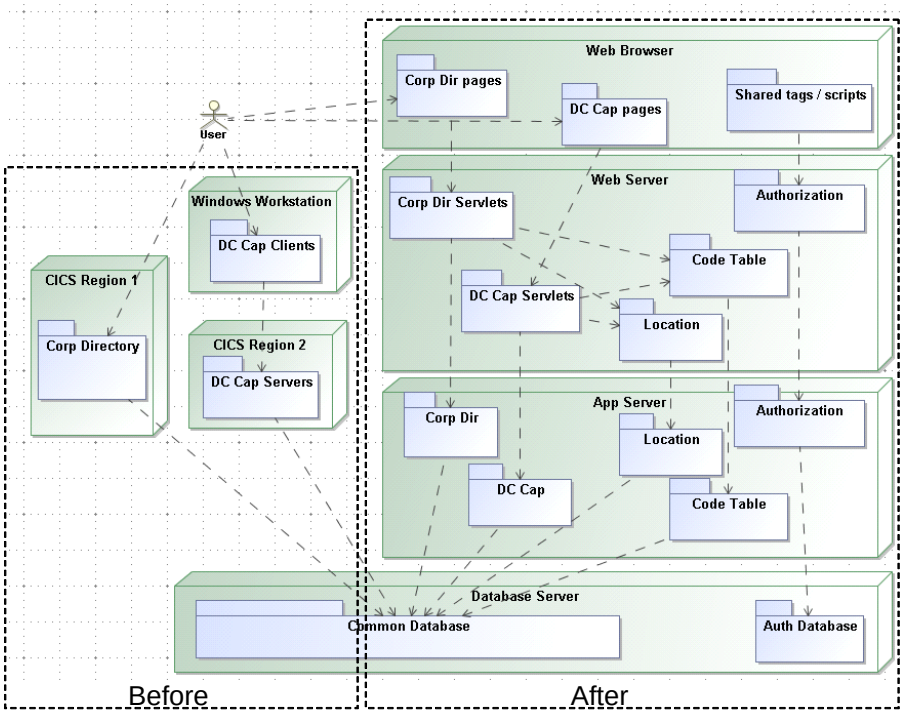


Figure 2 – Before and after architecture and dependency overview

- Design the solution – Since we did not need to design new business functionality, we were focused on mapping the existing solution to the new design. Specific design activities included the specifying of use case realizations for each use case pattern perceived and the mapping of work units to a specific realization pattern. Note the use of patterns again. This allows us to encapsulate the design of the future solution in the minimum number of realizations. We also prototyped new or significantly changed work flows (UIs) on a pattern basis, again using patterns so that we could minimize the prototyping effort. Even though there may be one-off type functionalities within the application, we generally find that we do not need to design every element of the future solution but rather design the relevant patterns and map the legacy functionality to the appropriate pattern. We also designed our service interfaces. The Client QA area took responsibility for the design of solution tests at the UI level and we produced test designs for testing the service interfaces.
- Plan Service and Solution provisioning – We sequenced the provisioning so that the re-used services would be provisioned first, then we would provision the required data access object (DAO) layer and then the remaining functionality would be provisioned in a top-down fashion.

For the analysis phase we leveraged the Jumar and Everware-CBDI tools to mine the existing code base and produce UML representations of data, structure and process. We then utilized the UML models (in MagicDraw) to perform the required design mapping, changes, refactoring, realizations and any other required elaborations using tags within UML profiles to communicate information to the delivery phase). We used simple html editors for prototyping the UIs. Everware-CBDI's rich service

specification template was used to guide the specification of services, and Everware-CBDI's system architecture template was used to drive activities and capture information not in the models and specifications.

These deliverables together with remote access to the legacy application formed the input to the delivery phase of the project.

Deliver Phase

The delivery phase took place within Everware-CBDI North America's onshore (USA) software factory. We applied Model Driven Development techniques that allowed us to deliver rapid transformations that consistently adhered to requirements, architectural prescriptions, design standards and patterns and produced a consistent code base. Our software factory recognizes that not all Clients will continue to evolve the modernized code base using Model Drive Development approaches and as such produce a code base that can be maintained using traditional non-model based approaches.

For this project, the Client intended to outsource the maintenance of transformed code and was interested in the produced models from a technical documentation perspective.

There is not much to discuss about the delivery phase. Our Clients perceive this as a service based black box. However I will highlight some principles that we adopt in our transformation factory:

- The transformation of the modernization assets produced thus far into a new solution is a technical formality. If we are unable to perform this in an offsite mode then the earlier stages of the project are deficient.
- Although we have a base transformation stack, we utilize various tools per engagement based on the particular (from and to) technologies. We use these tools for knowledge discovery (detailed) and to re-factor and/or convert that knowledge into new structures. We will engage in project specific partnerships to achieve this.
- We use a template and pattern based code generation approach so that we can leverage the pattern based, analysis and design activities that were applied during assessment and analysis.
- We pragmatically apply automation where it has a beneficial effect. Some legacy artifacts are esoteric enough and exist in small enough volume to warrant a manual process of detailed knowledge discovery.

On this project the delivery phase produced code that was compatible with the Client's chosen development and execution environment. The deliverables were packaged as two distinct EAR files (one per application) and a set of shared libraries to represent the re-usable services that were developed. The re-used services are thus re-used by projects as black boxes to prevent the practice of 'clone and own'. These services will be maintained as their own projects and will be managed by the architecture group.

As part of their QA responsibility the client will produce user interface based test scripts for the applications. We supplied JUnit based regression tests for the service interfaces.

The final stages of delivery are the physical deployment of the application into the Clients development environment. We then conducted a technical handover and

provided support in deployment and during the initial stages of QA to ensure that the Client staff that would be maintaining the code base had a good grasp of what we had delivered.

Evolve Phase

Since the evolution phase is separate from subsequent modernization iterations, it represents the phase within which the Client continues to maintain and enhance the modernized applications. The Client's architecture group has a vested interest in the evolve phase to ensure that the architecture retains some of the strategic properties that will facilitate the transition to SOA when desired.

Lessons

We would recommend that enterprises engage with a partner that specializes in architecture driven practices to guide and facilitate the modernization effort. It is unlikely that the enterprise would need to develop modernization skills since these are transient and of no further use once the transformation is complete. What the enterprise does need to do is to work with the partner to define the appropriate To-Be business, architecture and design.

The use of an architecture driven and coordinated approach will ensure that the various modernization activities (that may be performed by discreet suppliers and in parallel) contribute to your future state architectural vision.

Use simple tactics to achieve a long-term strategy. In this project some of those tactics included:

- Always look at more than a single stovepipe – even breaking the portfolio into a series of mini-portfolios will yield a better result than the stovepipe view. There are a variety of portfolio slicing and dicing approaches other than traditional stovepipe boundaries that can be brought to bear depending on your circumstances.
- Target a homogenous technical and application architecture in spite of the fact that the legacy technologies were different.
- Look for duplicate or similar functionality that can be extracted and encapsulated as re-usable assets. On this project these were tactically packaged, as jars for inclusion in projects but strategically these could become hosted services. The tactical tradeoff of additional management overhead in version control of these jars was deemed worthwhile to move towards the strategy.
- Mitigate risk by engaging in pathfinder activities such as this Client did. The net effect of which was a validated modernization approach, increased confidence and actual delivery of the first iterations of two applications. Be sure to select portions of the portfolio that are representative if engaging in pathfinder activities; too simple or too complex will lead to extrapolating invalid data when considering the remainder of the transformation.
- Engage your Business User's and be honest about the time it will take to realize the ultimate benefits of modernization. In this project, the Users understood that the modernization would occur over several iterations, however they also saw immediate benefits of the approach given that the

modernization of the second application slice was faster and cost less. The Users also understood that during transition, they would be 'living in the renovation' – that is to say that they would be engaging some of their business in the modernized applications and some in the legacy application.

- Be wary of changing things to be 'better' simply because you are 'making changes anyway'. In our experience this is a common mistake that leads to scope creep and introduces new dependencies and unanticipated consequences. For example in this project we purposefully avoided any database schema change since that would have too much impact on the remainder of the portfolio. However this doesn't imply no changes; for example we introduced several structural and workflow changes that optimized the resulting code base and the User experience but did not introduce scope creep or unanticipated consequences. That is not to say that desired database schema changes will never be made, we just need to schedule them at an appropriate time. For example the consolidation of the various physical code tables will be more easily achieved at a later time once we have iteratively removed the dependencies that currently prevent that change.

Conclusions

At first blush it seemed to the Client like this project was merely a replatforming problem. However through this project we were able to demonstrate to the Client that taking an architectural approach with a portfolio level perspective improves the modernization project outcomes and lays the foundations for greater strategic ROI. In this instance the Client has established the groundwork for a workable modernization approach that achieves the short-term benefit of technical homogenization but also places them on a strategic path towards a more agile service based implementation. This approach to modernization places the Client in a good position to adopt SOA when they are ready.

---

[1] The Agile Application Modernization Project. CBDI Journal Jan 2010
[2]Jumar-Solutions Project Phoenix, http://solutions.jumar-solutions.com/jumar-solutions/project-phoenix.php

About CBDI

CBDI Forum is the Everware-CBDI research capability and portal providing independent guidance on best practice in service oriented architecture and application modernization.

Working with F5000 enterprises and governments the CBDI Research Team is progressively developing structured methodology and reference architecture for all aspects of SOA.

CBDI Products

The CBDI Journal is freely available to registered members. Published quarterly, it provides in-depth treatment of key practice issues for all roles and disciplines involved in planning, architecting, managing and delivering business solutions.

Visit www.cbdiforum.com to register.

Platinum subscription – A CBDI Forum subscription provides an enterprise or individual with access to the CBDI-SAE Knowledgebase for SOA delivering ongoing practice research, guidance materials, eLearning, tools, templates and other resources.

Everware-CBDI Services

At Everware-CBDI we enable large enterprises and governments to become more agile by modernizing their business systems. We have repeatable processes, resources, tools and knowledge-based products that enable enterprises to transform their current applications in an efficient, low risk manner, into an optimized service-based solutions portfolio that supports continuous, rapid and low cost evolution. Our consulting services range from providing practices and independent governance to architecture development, solution delivery and service engineering.

Contact

To find out more, and to discuss your requirements visit www.everware-cbdi.com or call

USA and Americas: 703-246-0000 or 888-383-7927 (USA)

Europe, Middle East, Africa, Asia, and Australasia: Telephone: +353 (0)28 38073 (Ireland)

www.everware-cbdi.com

IMPORTANT NOTICE: The information available in CBDI publications and services, irrespective of delivery channel or media is given in good faith and is believed to be reliable. Everware-CBDI Inc. expressly excludes any representation or warranty (express or implied) about the suitability of materials for any particular purpose and excludes to the fullest extent possible any liability in contract, tort or howsoever for implementation of, or reliance upon, the information provided. All trademarks and copyrights are recognized and acknowledged. The CBDI Journal may be distributed internally within customer enterprises that have current corporate subscriptions. Otherwise CBDI Journals may not be copied or distributed without written permission from Everware-CBDI.