Experiences in Migrations of Legacy Systems

Bill Wood, Mike Gagliardi, and Phil Bianco

Software Solutions Conference 2015
November 16–18, 2015
Background

An agency wanted to migrate two paired and tightly coupled systems
  • Both are 24/7/365 and support disaster recovery at multiple sites
  • Both have many classes of users
1. Service-based system: 15% functionality (going to 40%)
   • Java, Relational DB, COTS tools, service-based, J2EE, SAP, Informatica
2. Legacy system: 85% of functionality (going to 60%)
   • COBOL, hierarchical DB, mainframe

Migrate to a well-defined target reference architecture (TRA) as a basis for a common platform infrastructure (CPI): developmental, operational, test

Briefing is focused on the legacy migration
Horse Shoe Model

Longer Journey / Greater Impact

Existing Solution  Target Solution

Shorter Journey / Lesser Impact

Business Architecture

Application and Data Architecture

Technical Architecture
Phases of Our Legacy Migration Activities

- Analysis of 13 RFIs
- Recommended they “lift and shift” the legacy to their target architecture
- Recommended to start an immediate “Discovery and Analysis” task
## RFI Analysis per Response

<table>
<thead>
<tr>
<th>RFI-Specific Questions</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 Migrate Hierarchical DB DB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2 Migrate Application Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3 Integration &amp; Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4 Application Maintenance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5 Acquisition &amp; Contracting</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6 Past Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Technical Approach Analysis Framework

<table>
<thead>
<tr>
<th>F1 Design / Analysis</th>
<th>Code</th>
<th>Data</th>
<th>Docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2 Migration Code, Data, Infra (Q1 &amp; Q2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3 Integ &amp; Test (Q3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4 Sync &amp; Cutover</td>
<td>Sync</td>
<td>Cut</td>
<td></td>
</tr>
<tr>
<td>F5 App Maintenance (Q4)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Approach 1-4

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>4</th>
<th>4</th>
</tr>
</thead>
</table>

### Cost and Timeframe ROMS

| Cost (ROM) $ Mil | 16 | 12 | 30 | 5-10 | 28 | 30 | 5-10 | 24 |
| Timeframe (ROM) Months | 24 | 18 | 24 | 24-48 | 24 | 24 | 12 | 9-12 | 24 |

- **Green**: Explicit
- **Yellow**: Implicit
- **Red**: Ignored
### Pros and Cons of Top 4 Alternatives

<table>
<thead>
<tr>
<th>Description</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Explanation for Yellow</th>
</tr>
</thead>
<tbody>
<tr>
<td># of data of record systems</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>More than one authoritative data source is a synchronization and fault management challenge</td>
</tr>
<tr>
<td>Move over users to the new system incrementally</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>The confidence from the use of the functionality is delayed until everything is cut over</td>
</tr>
<tr>
<td>Initialize new database with old data</td>
<td>Incremental</td>
<td>Big Bang late</td>
<td>Big Bang early</td>
<td>Big Bang late</td>
<td>Big Bang Early requires long-term synchronization</td>
</tr>
<tr>
<td>Streaming data between systems during phases and over months</td>
<td>Forward and back</td>
<td>None</td>
<td>Forward</td>
<td>None</td>
<td>Subset of 1 (above)</td>
</tr>
<tr>
<td>Fallback due to new system failure during development</td>
<td>Recover new system and re-synch</td>
<td>Not needed</td>
<td>Multiple ways</td>
<td>Not needed</td>
<td>Subset of 1 (above)</td>
</tr>
<tr>
<td>Fallback due to new system failure after cutover</td>
<td>Recover new system and re-synch</td>
<td>None</td>
<td>Defined ways</td>
<td>None</td>
<td>Problems will always arise after cutover. Need to design for fallback.</td>
</tr>
<tr>
<td>Queries executed during development</td>
<td>Hierarchical new DBMS</td>
<td>Hierarchical</td>
<td>Hierarchical</td>
<td>Hierarchical</td>
<td>Increased complexity for query management.</td>
</tr>
<tr>
<td>Benchmarking for performance</td>
<td>By testing and operation</td>
<td>By testing</td>
<td>By testing</td>
<td>By testing</td>
<td>You can’t operationally benchmark until cutover; performance issues will be discovered late.</td>
</tr>
<tr>
<td>Data/code coupling</td>
<td>Separable into segments</td>
<td>Separable into segments</td>
<td>Too tightly coupled for separability</td>
<td>N/A</td>
<td>If the code and data coupling is overly complex, will be difficult to segment.</td>
</tr>
<tr>
<td>Legally mandated updates during development before cutover</td>
<td>In both code and data if moved; in one if unmoved</td>
<td>In both code and data if moved; in one if unmoved</td>
<td>In both code and data if moved; in code if one is unmoved</td>
<td>Code and data in both</td>
<td>No response; handled this very well.</td>
</tr>
</tbody>
</table>

Green: Preferred  
Yellow: Not Preferred
It’s Complicated

Understanding the Legacy System Architecture
• Infrastructure tools
• Application component relationships (data and code)
• Business process threads (BPTs)

Understanding the Target System
• Target reference architecture (TRA)
  • SOA; layered infrastructure
• Designing services on top of TRA
• Business process threads

Mapping Between Legacy and Target in Phases
• Architecture mismatches (development, operational, certification, sustainment, COTS)
• Operating with dual authoritative data systems, cutover, synchronization
• Relationships between application components (legacy vs. TRA, COTS)
• Ineffective BPTs
It’s Worrisome

- Is there too much code/data coupling and spaghetti code to partition for migration?
- Are the current business processes and screens appropriate?
- Is the CPI stable? Is the TRA stable?
- Are COBOL-to-Java transformation tools up to the job?
- Can we operate with two systems overlapping authoritative data?
- Do we have sufficient technology expertise in legacy system, TRA, discovery and analysis tools, transformation tools?
- Is the business logic only understandable in the legacy code?
- How can we overcome the lack of architectural documentation?
- Will the entire testing and certification process change?
- Will I create a maintenance nightmare?
- It’s a 24/7/365 operation – no downtime!
Experiences

Customer
- Rush to use this data to get started removing legacy
- Different and misplaced emphases between people

SEI Team
- Strong differences of opinion during the analysis
- Schedule driven; a small subset completed it
- Proposed alternatives and a migration plan
Phases of Our Legacy Migration Activities

- Analysis of 13 RFIs
  - Recommended they “lift and shift” the legacy to their target architecture

- Task Order for D&A
  - Recommended to start an immediate “Discovery and Analysis” (D&A) task

- ROI modernize

- New Chief Architect
  - Chief Architect left; lost our White Knight

© 2015 Carnegie Mellon University
Distribution Statement A: Approved for Public Release; Distribution is Unlimited
Experiences

Customer

- Built a task order with costs for Discovery and Analysis (D&A)
- Forced to build a return on investment (ROI) for complete modernization
- They were not able to get money for the D&A

SEI Team

- Built the task order
- Supported the ROI effort
- Frustrated by lack of $
Phases of Our Legacy Migration Activities

- Analysis of 13 RFIs
  - Recommended they “lift and shift” the legacy to their target architecture

- Task Order for D&A
  - Recommended to start an immediate “Discovery and Analysis” task
  - New Chief Architect

- We also worked on evaluation of the “sister” program for modernization
  - ROI modernize
  - Performed various explorations of alternatives
  - None were completed

- Performed an AoA for Legacy Modernization
  - Program Manager left

- Chief Architect left
Don’t Live in a Dreamworld

- Big-bang changes usually fail
- Conducting transactions across networks and keeping response times satisfied!
- Transforming spaghetti code automatically
  - Separating presentation from business processing from data access!
  - Moving from green screens to windows
- Making multiple types of changes simultaneously!
- Edict: No changes to the legacy system!
- The TRA is a good start, but an application architecture with data modeling is needed
- Operating with multiple sources of authoritative data is a concern
Target Reference Architecture

This is a good start
But it is not enough

Need a system and application software architecture
• End state
• Each release
Migration Approaches

Options considered based on RFI proposals

1. Do nothing (baseline)
2. L&S (baseline): switch to relational DB and new platform
3. L&S then modernize
4. L&S then re-engineer
5. Re-engineer
6. Hybrid: L&S, modernize, re-engineer
Migration Process

1. List the options
2. Develop evaluation criteria
3. Score the options
4. Make a selection

- Goals for sequencing
- Constraints on phasing
- Approach to migration
  - Data, code, user, business processes ordering
  - Quality attribute considerations
  - Migration tooling
- Define phases
  - Groups
    - Legacy: functionality, code, data BP, users
    - Tiers/layers
    - Transient code in legacy and TA
  - Throwaway
  - Align with infrastructure roadmap

- Determine and score options
- Explore implementation alternatives for options
- Build an end-state architecture
- Build a roadmap
Evaluation Factors

Cost

• How much will the migration labor cost?
• How much recovery of investment cost after cancellation?
• How early will cost savings happen? License and support during migration?
• How much will the ongoing legacy license and/or support cost post migration?

We discover

- Merged

<table>
<thead>
<tr>
<th>W</th>
<th>Factor</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>How is data organized wrt TA: relational, normalized, distributed, partitioned</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

Performance

- How is data organized wrt TA: relational, normalized, distributed, partitioned

Risk

- How modernized are the user screens wrt the TRA?
Summary

Technical
- Included OMB guidelines in evaluation factors
- Fit quality attributes in evaluation factors
- Plan – but don’t overdo it

Management
- Changing political environment is hostile to technical work
- Need to have PoC at management decision-making level
- Drift and redirection can become a way of life
- Management happy with the summary sheet
  - Don’t overdo the supporting details
- Did the system engineering in an Agile manner