Abstract

A complex Enterprise Relationship Planning (ERP) data migration was in trouble: low quality, poor performance, instability, and an overworked team threatened project success. Facing failure, a small group of the original developers achieved a tremendous turnaround by applying Lean and Theory of Constraints (TOC) concepts directly to the data migration architecture. Their success shows how agile applies beyond software development, and can help others look for new ways to solve problems.

1. Introduction

If needed, a data migration is central and critical to a software implementation project. Every system and sub-team relies on quality data. Historical data is often critical to continuing business operations. Data Migrations are usually complex and difficult for multiple reasons:

- Companies perform migrations infrequently and are usually unprepared for the challenges
- Legacy systems knowledge is a scarcity
- A new system may replace multiple legacy systems
- Data quality issues accumulate over time

The paper describes the challenges and successes faced by the Data Migration team on a large Siebel implementation. It compares two different architectural approaches used on the project and demonstrates the benefits of using Lean concepts to improve a data migration.

2. Project Overview

In 2004-2005, a well-known software company implemented Siebel to manage customer service, software-licensing (assets), partner relations and order entry. The project organized around two phases: Phase-1 – Europe and the Americas regions, and Phase-2 – the Asia Pacific region.

The implementation involved migrating 14 legacy data sources to Siebel running on Oracle. Both phases faced similar levels of complexity. The migration was particularly complex due to the number of data inputs, complicated business rules, poor data quality, and large data volume (several hundred million records).

The data migration plan structured around a series of “practice” or full data migrations. Nine practice migrations were scheduled over a nine-month period. These practice migrations were also integration milestones for all project development teams and handoffs to QA teams for testing.

The Phase-1 Data Migration (DM) team had approximately 18 members. Team project goals were:

1) Minimize system downtime by keeping the blackout period as short as possible
2) Migrate all assets and related data per business requirements (leave no data behind)

2.1. Architecture of a migration

A data migration is essentially a long running batch process. Figure 1 outlines the main steps. In step-1, legacy systems data loads to a “source copy” database. Once the source is loaded, the migration moves data to a “staging” database in preparation for loading into the Siebel environment (step-2).

![Figure 1 Data migration overview](image-url)
The majority of the transformation work completes in the staging process. “Transformation” involves the merging and cleaning of data based on business and technical requirements. Additional business rules determine what data migrates to Siebel.

Next, data moves to the vendor staging tables (in this case, the Siebel Enterprise Integration Manager, or “EIM” tables) (step-3). Once the EIM tables are loaded, the EIM processes load the Siebel database (step-4). Finally, post data load processes (step-5) perform data cleanup not handled by the EIM load process.

2.2. Migration tools

The data migration used industry standard tools. Oracle was the target database platform so heavy use was made of SQL and PL/SQL. The data migration also used Informatica, the industry leading Extract, Transform, and Load (ETL) application. SQL views and PL/SQL code applied data transformations and business rules. Informatica handled data mapping and transport. Informatica also loaded all data from heterogeneous legacy systems to the source copy database (step-1).

3. Phase-1 Overview

The DM team experienced continual problems throughout the Phase-1 development and test cycles. Practice loads were proving it impossible to meet the goal of “minimize system downtime”. Full data loads were taking one week or even longer to complete due to multiple issues:

[Performance] Performance was a huge issue for the DM team. The data migration was a 5-step process, with several jobs per step. Initially, many job steps were taking several hours to complete; some as much 24 hours. The EIM load (step-4), which loads data to the Siebel schema, was taking days to finish. A practice load took from one to three weeks to complete.

[System crashes] System crashes were costly and severely affected successfully completing a fast, quality data migration. Multiple long-running job steps increased the chances of a system crash. The DM team experienced several types of crashes throughout the series of practice data migrations. The more common were the ORA-0600 error (Oracle kernel exception), rollback segment failure, and table index violations.

[Recoverability/restart] Two situations required restarting a load: 1) A job step crash, and 2) the discovery of a major defect. If a long running job crashed, it could delay the migration up to a day. Worse, a severe defect could delay the migration several days. Depending on the restart job step, several dependent multi-hour jobs might be required to correct the data.

[Quality] Initially, the DM team performed little testing other than some record counts, spot checks, and data reviews in the resulting system. Quality was poor. A separate QA team (Data QA) was responsible for fully validating the data migration results.

The Data QA team used a “black box” testing approach. They compared the source copy data to the migrated data while applying their interpretation of the data transformations. Data QA built a test harness in PL/SQL to perform the comparisons. Feedback was slow. It took a week after the full data load for QA to code their tests and another week to run the tests.

[Resource conflicts] Development and testing environments are often scarce on large development projects and teams often share hardware. The DM team shared a development server with Analytics. Both teams relied on Informatica. When both were running Informatica processes, the database server slowed to a crawl. Multiple times the DM team had to halt their processing to avoid resource conflicts with Analytics.

[Repeatable process] Performance and stability were affecting the ability to automate migration processes. This caused an unnecessary reliance on manually starting job steps. Manual handoffs caused several issues including missed steps, handoff latency, extra labor spent running the migration, and off-hours work.

[Sub-optimization] Sub-optimization occurred when attempting to make every job step run as fast as possible. Optimization techniques like dropping and rebuilding indexes, or Informatica “partitioning” tricks usually led to disappointing results. The problem is most optimizations add systemic complexity, which negatively affects quality. Optimizations can also overload a database affecting stability.

[Overworked team] Due to the described problems, stress levels were high. The DM team required overtime to keep up with the workload.

3.2. Assessment

After several practice data loads, the major issues were apparent. First, quality was poor due to the employed testing processes. Feedback from Data QA took two to four weeks after a load completed, too slow to be useful. The “black box” testing approach made it difficult to determine what and where defects existed in the process. Testing needed to occur for every transformation at every job step to be effective.
Data migration processes were too labor intensive. Manual handoffs were slow, mistake prone, and wasting developer time. A solid, repeatable load process would relieve workload and improve quality.

Informatica was a “development bottleneck”. The tool was a significant contributor to many of the Phase-1 data migration issues. Additionally, the Informatica developers were overworked. They ran most data load processes, performed development, and researched many of the defects. Removing the reliance on Informatica in key areas of the migration would help solve many problems.

Finally, and most important, many of the data migration problems resulted from processing data in one large batch. Processing all data at each step before moving to the next step was the root cause of performance issues. A logical solution would be a method that processed data in smaller batches.

### 3.3. Phase-1 Outcome

Towards the later development stages, some members of the DM team proposed changing to a batch processing architecture, and even developed a prototype that worked exceptionally well. While the batch framework looked like a strong solution, Management and the DM team felt it was risky to switch late in the development cycle.

The main process improvements made during Phase-1 involved testing. After a few data loads, the DM team realized the testing process was not working and they themselves had to take full responsibility. The DM team developed a suite of validation tests utilizing the utPLSQL unit-testing framework [1]. By practice load 5, every job step had near 100% coverage on data transformations.

Additionally, the DM team created a “mini-load” process made up of a subset of the total data. The mini-load simulated the full load process and ran in few hours instead of days. The DM team ran the mini-load and unit tests a few times a week. The full utPLSQL validation tests combined with the mini-load greatly improved quality.

The Phase-1 migration was a limited success. In the end, the team did achieve enough improvements to meet the negotiated downtime goal of 72 hours.

### 4. Phase-2 – A new approach

Management approved two developers to work on the batching framework in parallel to the standing architecture for Phase-2. The new framework quickly proved superior and the DM team abandoned the Phase-1 data migration approach.

Lean and TOC principles heavily influenced the new framework design. Table 1 outlines the key principles guiding the development of the Phase-2 framework.

<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-inventory</td>
<td>Small batch size, overlap processing</td>
</tr>
<tr>
<td>Eliminate waste</td>
<td>Defects, delays, handoffs, unnecessary complexity</td>
</tr>
<tr>
<td>Build quality in</td>
<td>Shorten feedback, unit testing, test each step, repeatable</td>
</tr>
<tr>
<td>Optimize the whole</td>
<td>Pull system, average cycle time, limit to capacity, fast as slowest step</td>
</tr>
</tbody>
</table>

**Low-inventory** – Lean concepts and TOC teach that large batch sizes causes longer product lead times, poor performance, lower quality, higher costs, overtime work, missed deadlines, and system instability [2a]. It seemed logical that processing data in small batches would see the same benefits manufacturers achieve by lowering inventory. Agile methodologies like XP and Scrum also advocate “low-inventory” systems by developing functionality in small increments.

**Eliminate waste** – Eliminate waste is a key Lean concept. A simple definition of “Waste” is “anything that does not add customer value [3]”. Special attention focused on eliminating defects, handoffs, delays, and complexity. Automation played a key role in eliminating waste.

**Build quality in** – Quality is an important concept in both Lean and TOC. “Quality control should be used to check the process, not the product [2b]” best summarizes the changes made to the testing processes. Quality came from testing each process step, shortening feedback, unit-testing code, creating a repeatable process through automation, and reducing cycle time, not “black box” testing.

**Optimize the whole** – Work should be limited to capacity. A pull system processes work as it arrives at the rate of the step’s capacity. Average cycle time is a process measurement that measures quality, speed, productivity, and cost [3].
4.1. From push to pull

The new framework made minimal changes to the originally developed code, and simply changed the way data flowed through the system. Instead of forcing all data through each step, the new framework emulated a “pull” system.

A batch control table orchestrated the new approach (Figure 2). The table contained a batch number field, data range fields, and one column per process step. Each step column acted as a simple queue, informing step servers when a batch was available for processing.

<table>
<thead>
<tr>
<th>BATCH_NUM</th>
<th>SRC</th>
<th>FIRST_ID</th>
<th>LAST_ID</th>
<th>DMSTG</th>
<th>DMEIM</th>
<th>DMBASE</th>
<th>DMPOSTSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SAPJP</td>
<td>374952</td>
<td>384982</td>
<td>COMPLETE</td>
<td>COMPLETE</td>
<td>COMPLETE</td>
<td>READY</td>
</tr>
<tr>
<td>2</td>
<td>SAPJP</td>
<td>364938</td>
<td>374951</td>
<td>COMPLETE</td>
<td>COMPLETE</td>
<td>IN_PROGRESS</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SAPJP</td>
<td>354916</td>
<td>364937</td>
<td>COMPLETE</td>
<td>COMPLETE</td>
<td>IN_PROGRESS</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SAPJP</td>
<td>344888</td>
<td>354915</td>
<td>COMPLETE</td>
<td>COMPLETE</td>
<td>READY</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SAPJP</td>
<td>324811</td>
<td>334847</td>
<td>COMPLETE</td>
<td>COMPLETE</td>
<td>READY</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SAPJP</td>
<td>314752</td>
<td>324810</td>
<td>COMPLETE</td>
<td>IN_PROGRESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SAPJP</td>
<td>294617</td>
<td>304656</td>
<td>COMPLETE</td>
<td>IN_PROGRESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>SAPJP</td>
<td>284570</td>
<td>294616</td>
<td>COMPLETE</td>
<td>IN_PROGRESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SAPJP</td>
<td>274509</td>
<td>284569</td>
<td>COMPLETE</td>
<td>READY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>SAPJP</td>
<td>264435</td>
<td>274507</td>
<td>COMPLETE</td>
<td>READY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>SAPJP</td>
<td>254171</td>
<td>264434</td>
<td>COMPLETE</td>
<td>READY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>SAPJP</td>
<td>241422</td>
<td>254170</td>
<td>IN_PROGRESS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SAPJP</td>
<td>226359</td>
<td>241417</td>
<td>IN_PROGRESS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>SAPJP</td>
<td>216284</td>
<td>226358</td>
<td>READY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>SAPJP</td>
<td>206243</td>
<td>216283</td>
<td>READY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>SAPJP</td>
<td>196207</td>
<td>206242</td>
<td>READY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>SAPJP</td>
<td>186025</td>
<td>196206</td>
<td>READY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>SAPJP</td>
<td>175983</td>
<td>186024</td>
<td>READY</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2 Batch control example

Step servers simply polled the batch control for work, processed a ready batch, and set the next step’s status. The number of step servers started balanced processing without overloading hardware. For example, a typical run configuration might be three DMSTG servers, two DMEIM, four DMBASE, and one DMPOSTSQL. Differing batch size experiments found that a size of 20,000 assets worked best for throughput.

The framework implemented several quality focused features. A logging feature recorded timings and record counts for every batch and process step, which greatly aided in troubleshooting issues. A “stop-the-line” feature suspended all processing for the entire system if a batch failed at any step. The utPLSQL validation tests were modified to test at the batch level and made to run within the batching framework.

The DM team built the framework in PL/SQL using test-driven development techniques. UNIX shell scripts and Windows command files controlled starting and stopping step servers. PL/SQL also replaced Informatica for data transport and mapping. Removing Informatica eliminated a tremendous amount of system complexity and bottlenecks.

Issues experienced in Phase-1 of the migration disappeared:

- **Performance, Sub-optimization**: The data migration dropped from taking days to 8-10 hours. Automation made it simple to run the load process unattended overnight.

  Performance became a secondary issue. Efforts focused on improving quality and load repeatability. When performance issues arose, logging information made it easy to identify bottlenecks and focus tuning efforts.

- **System crashes, Recoverability/restart**: The new architecture increased throughput while reducing server load. System crashes from overload disappeared.

  Occasional index violations crashed job steps, but with the new architecture, a crash affected one batch only. In a few minutes, the issue could be diagnosed, corrected, and the migration restarted at the failure point.

- **Resource conflicts**: Server resource conflicts became manageable. Even with Analytics running Informatica, the full data migration would slow from 8 hours to around 12 hours.

- **Quality, Repeatable process**: Automating most data migration processes made regularly running the migration simple. The DM team ran 3+ practice loads a week for 8 weeks, creating a refined and stable process. Running multiple data loads weekly made quality extremely high. The DM team was ready a month before the scheduled Phase-2 go-live.

5. Conclusions

The Lean/TOC principles that influenced the new migration approach all reinforce each other and helped make the Phase-2 migration a success. Lowering inventory enabled process automation by making the data manageable. Automation in turn eliminated waste and helped build quality into the system.

The small batch size allowed step-processing overlap, an extremely important optimization. Processing was limited to capacity by designing the framework as a pull system. The average cycle or load time dropped significantly, shortening the development and test feedback loop further improving quality.

The performance improvements and automation allowed a significant increase in the number of practice data migrations with minimal impact on developer workload. The team created a quality, fast and repeatable migration process in a short time period. Table 2 shows some important comparisons between the Phase-1 and Phase-2 systems.
Table 2 Migration phase comparisons

<table>
<thead>
<tr>
<th>Phase-1</th>
<th>Phase-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 large batch – process all data per job step</td>
<td>Process data in small batches, overlapped steps</td>
</tr>
<tr>
<td>Process data as fast as possible at each step</td>
<td>Process data only as fast as the slowest step consumed</td>
</tr>
<tr>
<td>Load processing time 1+ weeks (some loads 10-15 days)</td>
<td>Turnover full load under 24 hours</td>
</tr>
<tr>
<td>9 practice loads in 9 months</td>
<td>25+ practice loads in 2 months</td>
</tr>
<tr>
<td>Load process extremely labor intensive (~200 manual steps)</td>
<td>Automated, very little labor required to run data load</td>
</tr>
<tr>
<td>Unrepeatable process – no two loads ever the same</td>
<td>Automated and fully repeatable</td>
</tr>
</tbody>
</table>

Was changing the architecture for the Phase-2 migration worth the risk? Would Phase-2 have been successful without making the changes?

At the start of Phase-2, pressure on the DM team increased. Not only did they have to complete development for the Phase-2 migration, they had to support production system changes. Phase-2 only had two months before go-live, a small window for development. In addition, the DM team lost several team members due to budget constraints.

The new approach removed a tremendous amount of workload on the team. They were able to support both the Phase-1 production support while developing for Phase-2.

Continuing with the Phase-1 architecture would have been extremely risky under the Phase-2 constraints. Applying Lean and TOC to the data migration architecture certainly contributed positively to the success of the ERP implementation.

6. Acknowledgement

Mary Poppendieck encouraged me to submit this experience report. In addition, a colleague on the Data Migration team and I took Mary’s Lean Practitioners Course at the time the events occurred. Course principles influenced the Phase-2 outcome. William Grosso and Clarke Ching provided valuable input, guidance and advice in writing this experience report.

7. References

[1] Steve Feuerstein, utPLSQL, open source unit testing framework for PL/SQL
   http://utplsql.sourceforge.net/

   a. Pages 20-68 offers excellent information on the how inventory impacts systems
   b. P. 42, Goldratt paraphrasing Dr. W. Edward Deming