CICS COBOL TO JAVA CASE STUDY AND SUCCESS STORY

GOVERNMENT APPLICATION - COBOL WITH CICS TO JAVA

SoftwareMining usually licenses our COBOL To Java Translation Tool to System-Integrators and clients to enable the work to be performed onsite. However, in a recent project we effectively became part of the delivery/testing team. The project involved translation of COBOL/CICS/VSAM code to Java.
Being involved in testing phase, it soon became very clear that much of the project success depends on the legibility of the generated code. If the code is easy to understand by java developers, and the developers do not require large amount of training to understand the legacy issues (REDEFINE, DEPENDING on, or CICS API's - for example what on earth did EXEC BIF DEEDIT do again ??) then the project has a much better chance of succeeding. So first thing to emphasise, emphasise and emphasise again is to use a translator/code generator which can yield the best quality code, and which can remove as much dependency on legacy architecture as possible. Bear in mind it is not always possible to remove all legacy dependencies. For example something like 75% of GOTO statements can be removed by simply restructuring the code. However removal of the remaining 25% may result in spaghetti code – it maybe better leaving them alone.
The following were the important stages and lessons from this project:

VSAM DATA MIGRATION

Before VSAM structures could be migrated to SQL tables, a manual process was required to identifying which structures are the same. For example the structures File-Definitions for CLIENT and CUSTOMER were actually referring to same data. Once they were identified, the information was fed to the translation-tool in order to generate code addressing same table in both cases.
The important lesson here is - this type of analysis is best performed by COBOL developers who are familiar with the system. Do this before last guy leaves the organization!
Oracle was selected as the target database, and Oracle utilities were then used for migration of VSAM data files to the database.

DATA ACCESS PERFORMANCE – VSAM AND SQL ARE DIFFERENT ANIMALS

The process of moving from VSAM to SQL was a lot more simple than in rehosting projects where VSAM api are manually replaced by custom made SQL statements. This is because the new object-relational java-classes generate the SQL at runtime automatically/dynamically. For example to read the customer record – the system merely writes: *customer.read()*. In this case, all SQL statements are automatically generated by the java framework - similar to Java HIBERNATE framework.

However, the big change in architecture (VSAM to SQL) caused a few performance issues which needed addressing. The issues were caused because when COBOL reads a data-file using an index, the system merely places the file–pointer to the beginning of that index. To achieve the same thing in a database, an SQL statement has to be issued to read all records which match the index / selection-criteria. Potentially millions and millions of record could match. This could create a huge load on the database. The solution was to provide sets of API's to developers – allowing them to effectively reduce the number of records matching the search result. Four different solutions were evaluated and developed within the Java-framework layer. Two of them were used successfully to overcome the performance issues. The selected solutions did not require developers to rewrite SQL statements – but merely use new object-relational API's to reduce the size of the search space.

FROM BMS TO HTML

Managing the end-user expectation become very important when dealing with new screens. For example, the original application was designed to utilize F1 or F5 keys for a particular function. Unfortunately Internet-Explorer launches the help screens when F1 key is pressed, and F5 always refreshes the page. The users have to *click* on F1 and F5 buttons rather than *pressing* the keyboard keys …. Of course there are ways of disabling the default functionality of the browsers – but this had some other implications.
Luckily, the translated/delivered code has provided an upgrade path: the new screens (manually redesigned JSP) can be plugged into the application without needing to change a single line of business-logic code. This work has been schedule for a next stage of the project.

KEEPING ORIGINAL DEVELOPERS INVOLVED

During the course of this project - the importance of the original COBOL developers became very clear. The testing phase (where most of the time/money was spent) went a lot more smooth when one the original developers got involved. The knowledge of how the application is suppose to behave, the intention of a particular piece of code, and whether it is still being utilised turned out to be extremely helpful.
Our experience showed it would be very beneficial to re-train the existing developers in Java, and to allow them to continue their involvement with the application. The moral of the story is – do the migration before they all move to more exciting jobs and roles.

FINALLY

The above is a very brief summary of the experiences gained. For further information please contact www.softwaremining.com or watch COBOL to Java/C# Project Planning and Management - 10 min Guide with Embedded SQL on Youtube.