# SEVEN REASONS CONVERTING COBOL TO JAVA TO SAVE CASH IS A RED HERRING

Before executing a COBOL to Java on z/OS conversion, there are several unrecognized pitfalls companies should be aware of that render possible savings to be of no consequence.

Companies are having discussions about reducing CPU costs through the wholesale conversion of COBOL applications to Java. Inviting as it seems at first glance, this option of reducing costs may be short lived and is a red herring because of unrealized issues and problems that occur when converting entire applications to Java.

## Seven Issues of Converting COBOL to Java

A while back, a company told us they were considering converting their huge base of COBOL Version 3 and 4 programs to Java. They admitted the code would run less efficiently, but thought the conversion would dramatically reduce costs because the code would run on a specialty processor like a zIIP.

Before executing a COBOL to Java on z/OS conversion, there are several unrecognized pitfalls companies should be aware of that render possible savings to be of no consequence.

## 1. Hardware Resource Problems

The initial premise to run as much as possible on zIIP processors to save in 4HRA costs will instead become a hardware resource problem with the allocation and resource sharing of zIIP processors. Not just Java applications, but other uses of zIIP engines may be impacted.

For example, batch workloads written in Java could end up using a large part of the zIIP capacity. DB2 is a large user of zIIP engines and is invoked indirectly by batch and transactional workloads.

A DB2 workload redirected to a zIIP is managed by DB2, not the user, and the amount varies with each different DB2 release. This means you would need to keep up to date on DB2 releases and maintenance because there would be a higher percentage offloaded to the zIIP for more recent DB2 releases.

Additionally, the need for DB2 usage of the zIIP would need to be balanced with other workloads like Java. To boot, transactional workloads increasing web usage in z/OS would impact Java usage, including:

- CICS – Java and web applications
- IMS – remote web workloads, through IMS connectors and servers
- WAS – Java web and server application

There is an excellent discussion about how IBM Db2 for z/OS uses the zIIP processors. William Favero's "It's time to ask 'Do I have an adequate zIIP capacity?'" is a must read.

## 2. Spillover to a GCP and Incurring MIPS Charges

Upfront planning would be necessary to minimize the possibility of spillover to a GCP occurring to avoid extra MIPS charges. You may need to isolate workloads on different LPARs to prevent it.

As the zIIP workload grows, zIIP engine shortages will occur. This would require constant monitoring and corrective actions. But there are hardware limitations to be considered. The zIIP:CP ratio is 2:1 for the zEC12 mainframe and up, as of 2013. For example, five GCP engines will allow up to 10 zIIP engines. The z13 and z14 maifntain this 2:1 zIIP:CP ratio.

As the zIIP workload grows, you'll need to plan for spikes that force the allocation of more zIIP capacity than you initially planned for. You'll also need an algorithm for determining workload capacity. Some shops continually evaluate and add zIIP engines when their overall usage hits 70 percent, but workloads are usually underestimated and will spillover to a GCP. Analysis of SMF Type 30 and Type 70 records can give you an estimate.

## 3. "zIPP-able" Only Means Eligible

Work that is zIIP-able is considered zIIP eligible, but just because code is zIIP eligible does not mean it's guaranteed to run on a zIIP, even if you have "sufficient capacity."

In his PowerPoint "zIIP Experiences," IBM's Adrian Burke provides plenty of information and examples of what work is considered zIIP-able, zIIP-ed (work that executed on a zIIP) and Un-zIIP-ed (eligible work that executed on a general CP). IBM states that its workloads are limited to how much can be offloaded to a zIIP processor. This is true for most workloads whether IBM or not. Burke suggests following percentages for these IBM workload types:

- XML Parsing – 98%
- CTG/CICS/ERWW – 40%
- IMS Connect ERRW – 40%

It's possible to force specific workloads to run only on zIIP processors. However, don't do this if the

applications have critical response time requirements (do this by setting IIPHONORPRIORITY=NO in PARMLIB member IEAOPTxx).

## 4. Intricacies of Converting COBOL to Java

Intricacies include things you should consider that could end up being problematic, costly or inefficient in a COBOL to Java migration, including:

- Loss of undocumented application knowledge of COBOL
- Retraining or hiring new developers that know Java
- Software and hardware resources are needed that you may not have (development and maintenance platform for Java)
- Software conversion issues
    - Won't be exact
    - Little consideration for performance
    - No accommodation for mainframe specific problem areas like decimal arithmetic and BCD conversions
    - Conversions tend to change the logic of COBOL programs due to the widespread use of many non-standard COBOL language tricks or undocumented features.
- How to tackle z/OS specific features like VSAM with Java

## 5. Performance Problems

The company considering converting COBOL to Java was correct in its assumption the code would run less efficiently. Ironically, while the company's aim was to reduce costs, the consequence of this migration would be dealing with CPU efficiency issues in the new Java code. Additionally, real-time response may suffer due to inadequate zIIP capacity.

## 6. Bad SCM for Java

Your current source control management system may not be a good choice for Java, which means you may need to consider implementing a new tool. Things you may need to consider are:

- How to integrate the new code into your current test/production promotion processes
- What backup and fallback capabilities are available
- Logging and audit facilities to meet corporate standards

## 7. Maintenance Methodologies

Do your programmers know Java? If so, how many? It's likely they have zero experience debugging Java, which will require toolset changes and providing education to change. Additionally, your current program change methodologies for COBOL won't apply to Java, so the time spent making changes or expediting fixes will be chaotic and lengthy.

# Migrate to COBOL Version 5 and 6 Instead

Of course, the issues involved in converting COBOL to Java are rooted much deeper than these seven summaries, but these alone are enough to start giving you a headache. It's best to avoid sacrificing performance for the red herring of saving cash.

Alternatively, why not just migrate your inefficient COBOL 3 and 4 programs to COBOL 5 and 6?