IT'S AN OLDER (COBOL) CODE, BUT IT CHECKS OUT

Overview: Updating existing programs is easier and more cost-effective than replacing them. But older code needs to be reviewed periodically to prevent stagnation. Prioritizing and optimizing your highest-value code can be simplified with the use of automated testing and quality checks.

“It's an older code, sir, but it checks out.” This classic line is from Star Wars Episode VI: Return of the Jedi. A strike team posing as an engineering crew gives a stolen authorization code as they attempt to pass through a security checkpoint. The antagonist, Darth Vader, asks if the crew has provided a "code clearance," to which Admiral Piett replies, "It's an older code, sir, but it checks out."

This same line could almost be uttered by any one of us when looking over our legacy applications. After all, we all have older code — maybe not a security code, but older code — in our applications. Do we review our portfolios, and how do we know if this code “checks out”?

The following steps may seem daunting at first but, as with everything, prioritization and automation will help.

By first cleaning up your portfolio you will be able to focus on the programs with true value:

• Is the code used? You can use SMF records to see if programs have been run and how frequently. Have they been run in the past month, quarter, year? If not, then you may look to remove it from your active portfolio.

• What code is used most often? Programs that are run daily could have more value than those that are run less frequently. Ranking programs by use will provide focus in your efforts to improve your

portfolio.

Next, optimize code of true value:

• Does the code perform well? Have you used Strobe to see if there are inefficiencies you can easily correct? Have you reviewed the database access? Small, low-risk changes can sometimes yield big improvements in performance.

• Could it benefit with the improvements in new compilers? Recompiling your portfolio with the latest compilers can offer long-term, low-cost improvements in performance.

• Has your code been compiled with a modern compiler like COBOL 6.2? Use a tool like the File-AID/MVS Library utility to determine if you have old code that has not been recompiled with a supported compiler release.

• The Db2 precompiler can point out aspects of the SQL that may not perform well due to old features. As with a current source compiler, a current Db2 precompiler should be used in the precompile phase.

• Products like File-AID for Db2 have an SQL Analysis feature that can be used to explain SQL from source programs or from Db2 packages/collections. This will provide valuable performance information.

•You may be able to improve your application performance when using VSAM files by converting them to IAM, which provides improved performance.

• Find and eliminate the wasteful SQL statements that are slowing you down. Even seemingly small changes can add up to big savings.

• Is there newer technology that would make it better? Just like with taking advantage of the newer compilers, look for advancements that would improve access to your logic. Do you have code which could be opened up via a REST API?

• Add code quality checks with SonarQube. You want to be sure that changes aren't inadvertently introducing dead code and logic flaws, not only by adding access to SonarLint for each developer, but by automating these checks — before compiles are done. This will enforce quality checks which will improve your codebase with every change.

• Automate testing to make sure you haven't introduced errors when you update your code. Implementing automated testing will increase quality and free up development time.

Ongoing improvements may be the easiest to implement. You can handle programs as they come and this will naturally lead you to those that are being enhanced the most often. Keeping your existing applications going will be more cost-effective and less disruptive than replacing them. To prevent their stagnation, you need to keep verifying that they still "check out."