# HOW TO PROPERLY TEST AND DEBUG COBOL PROGRAMS YOU'RE MIGRATING

A large percentage of COBOL migration problems are data related, so thorough testing is crucial to discovering them. The testing you do now will save you from countless hours of debugging, potentially losing revenue and disappointing your customers.

It's inevitable you will encounter challenges when migrating to COBOL Version 5.2 or 6.1. They don't preclude the benefits of migrating, but they exist. According to IBM, a large percentage of these migration problems are data related, so thorough testing is crucial to discovering them. The testing you do now will save you from countless hours of debugging, potentially losing revenue and disappointing your customers.

Keep in mind there are changes to COBOL Versions 5.2 and 6.1 that affect how debuggers operate, so your toolset must support those changes. There are some small external changes to the 5.2 and 6.1 languages that COBOL programmers deal with every day; however, the internal changes are more extensive.

The "back-end" of COBOL Versions 5.2 and 6.1 has radically changed and keeps changing, even daily. It's imperative to use a toolset that both is enabled to and can continuously deliver support for frequent changes made to these new versions.

Here are some **best practices to follow** to properly test and debug COBOL programs you're migrating:

## 1. Prepare Your Testing Environment

Brush off your testing suites and introduce a strict regimen for testing. As you move through your applications, you'll become more experienced and will encounter problems that are familiar and repeatable. Categorize the problems you find and build a test suite that finds those common problems related to both data and poor coding practices. Likewise, optimize your testing by ensuring you have the latest IBM compiler and Language Environment maintenance up to date. Refer to the IBM COBOL fix list for this information. A systematic plan must be adopted to keep the compiler and Language Environment update to date with monthly fixes. Testing with outdated maintenance would be wasted time and effort.

## 2. Fix Your COBOL Programs

Before you start nonchalantly migrating COBOL programs, you should analyze each one and determine where fixing is required. Our colleague Glenn Everitt wrote about the importance of this a while back. His conclusion: if you neglect fixing your COBOL programs early on, your migration efforts will be more difficult than you can imagine. We agree. When testing each application lined up for migration, verify that each provides the correct results, performs according specifications and scales to expected norms. Remove all exceptions and abend conditions too.

## 3. Start Unit Testing

Unit testing of mainframe applications is also important. This process allows developers to test the small parts of an application to find and fix low-level bugs before moving into testing processes that involve larger parts.

However, unit testing is one of the weakest spots in the mainframe development life cycle. Because it's a tedious manual process that sucks time and energy away from other development tasks, which must be accomplished for code to move forward, unit testing often disregarded or left incomplete.

Fortunately, the dearth of unit testing in mainframe environments is gradually changing with the advent of automated unit testing tools that allow developers to:

- Validate code changes immediately
- Eliminate dependency on specialized mainframe knowledge
- Automatically collect and keep test data with unit tests using tools like Compuware
- Automatically generate unit tests and test assets

Automated unit testing makes it easier to deliver quality with velocity.

The idea of automation and testing applies to migrating your programs to COBOL Versions 5.2 or 6.1 too. Unit testing would ensure program quality early on, preventing future migration issues, and automation would accelerate that testing in the already time-consuming migration process.

## 4. Invest in Source Debuggers

As mentioned earlier, changes to COBOL Versions 5.2 and 6.1 affect how debuggers operate, so you should invest in source debuggers that support these newer versions. Modern debuggers provide the best experience and direct access to data from the COBOL programmer perspective. Avoid using a source tool unintended for 5.2 and 6.1. That's like stepping through machine code and will

severely hinder debugging. We've worked with many customers to successfully migrate their applications to COBOL Versions 5.2 and 6.1 using a variety of Compuware's application debugging, fault and failure management, and performance analysis tools, including:

- Xpediter for visual, interactive debugging, even if programs are compiled OPT(2)
- Abend-AID for abend analysis of 5.2 and 6.1 programs, including optimized programs
- Strobe for profiling of 5.2 and 6.1 programs, including optimized programs

## 5. Utilize Compile Options

Utilizing compile options will help you identify testing issues prior to moving programs into production. There are several compile options that may be beneficial to identifying issues in tests, but these are the core options worth noting:

  - **SSRANGE** identifies out-of-range table issues, generating an IGZ006S message when an index or subscript points to storage beyond the bounds of a table. The use of SSRANGE should be restricted to testing, as IBM has indicated it can increase CPU time by an average of 18 percent.

  - **ZONECHECK** either generates warnings or causes a program to abend if the zone bits are not numeric. It will insert IF NUMERIC class tests in front of any zoned decimal variables acting as sending fields.

  - **DIAGTRUNC** generates warning messages during the compilation to alert the programmer that, during a move statement, the receiving numeric variable contains fewer integers than the sending field. This can help ensure truncation will not create an issue.

  - **RULES(NOEVENPACK)** generates a warning message when the packed data contains an even number of digits. According to the Enterprise COBOL for z/OS Programmer Guide V5.2 (or V6.1), packed data should always have an odd number of digits to prevent truncated data. However, systems programmers can change the warning message to an error, forcing a coding change.

- **RULES(NOLAXPERF)** will issue warnings for inefficient coding practices, as well as compiler options that can impact performance.

Ensuring you properly test and debug COBOL programs early in the process will prevent bigger issues from inhibiting your progress through the ensuing steps of a migration. The benefits will be noticeable in the next phase of your plan, application optimization.