THESE ARE MAINFRAME DEVELOPMENT'S GOOD NEW DAYS

Sometimes, old-time mainframers get together and talk about the good old days. I was there; those days weren't so good. I started out in college learning to be very proficient with a key punch machine; I was ecstatic when I was able to get access to an IBM® 3270 and enter my code and have it stored on disk. It was the first example I had of not accepting the status quo and being open to new technologies that would make my job easier. Over time I have been open to the many improvements in the developer experience.

With that, I can say that I feel that today is the golden age for the mainframe. This is the best time to be a developer working on mainframe applications. Here are some of the areas in which today's mainframe outshines "the good old days."

- **Communications**—The ability to access systems and work remotely is a huge improvement over the way we used to have to drive into an office to fix errors, then wait while they were confirmed. We may think that our always-on culture is disruptive, and it can be, but contrast that with having to stop what you are enjoying and drive to the office for an unknown amount of time. Working on the mainframe means you are working on a critical system, so there will always be disruptions; the difference is that now there are far fewer. It has also enabled "work from home," which used to be impossible, or at least very difficult. We can now work from anywhere.
- **A choice of interfaces**—In addition to the "classic" Interactive System Productivity Facility (ISPF), there are also now Eclipse and Virtual Studio (VS) Code solutions, among others. The ability to have multiple displays on multiple monitors and not have to remember keywords

makes it so much easier to work. Sure, you could, over time, be proficient in ISPF, but that's it—it took time, and developers should not have to go through an initiation process to be productive. It's much better to start out day one with a modern, familiar interface that is easily upgraded and configured.

- **New perspectives**—I learned from those who came before me, and many of them insisted things were "better than before," and, in fact, "just perfect." There was a tendency to not change because change was risky. Now, with new developers coming to mainframe development, we see that things have changed, there are new ideas. I think some of these ideas were there before, but because of limitations in technology, they couldn't be implemented. Now, with a fresh look, we are seeing that the ideas were sound, and the technology has caught up so we can implement them. Where these ideas and practices were previously discarded because we looked at them and they wouldn't work, we are now benefiting from them. It is great to have this new perspective to spur us on to greater improvements.
- **Automation, and automation of automation**—Automation is a good concept, but it has been the victim of what I mentioned above. We either tried it and it never worked, or we implemented it with a complex process and don't really want to touch it because it is fragile. Now is the time to reevaluate and benefit from new technologies. I know that when I look at what is available with webhooks and REST APIs, I am amazed at the possibilities. It is so much easier to set up automation now than it was in the past. Where I might have given up previously, I am now eagerly looking to new automations, putting something into place and then a few months later, adding to it.

  This also applies to automating your automation. By this, I mean that you may have already automated test scripts. If so, great! But if you don't automate them, the alerts, and, really, the whole process, then you are missing out on some great benefits. In short, look at everything—including code reviews—and see if there is some way that manual tasks can be automated and supplemented.

- **Graphical displays**—Going from a card deck to seeing the code on a 24x80 display was a huge advancement. I worked in that box for, well, a long time, before graphical displays became possible. Having my program charted out—automatically—was a game-changer. I am a visual learner and to see my program—and the connections of my program with others—right there on my monitor was amazing. My days of "playing computer" or interactively debugging for analysis were over. I could see the structure of the program, drill down into a paragraph, and see the flow of data, of the field from where it entered, how it was processed, and then left to another program, file, or database. This graphical display meant I could take on any new program, understanding it quickly, and have confidence in my changes. It was an instant improvement in productivity.
- **Agile scrum**—I was taught to write new programs in a modularized way—stub out sections, test, then fill in the logic. This was a good practice I followed and was a way to catch errors early. However, this was done for multi-month waterfall projects. We didn't take the best practice and move it to the next level. The game changer came when we were able to implement a source code management (SCM) solution, BMC Compuware ISPW (now BMC AMI Code Pipeline), which enabled multiple developers to work on code at the same time. That helped us keep track of versions and enabled an Agile scrum framework for mainframe development, which delivered huge increases in productivity and quality. I've found that taking an open approach with our SCM and build and deploy, through the use APIs and webhooks,

ensures that development teams can continuously improve their agility.
- **Code coverage**—I remember when testing was guesswork and sometimes bluffing. You ran your new code through as much data as you could find with the assumption that it should hit your changes, then you could verify they were right and that you didn't break anything. But that was just it, you had to assume, and that is where the bluffing came in for a code review. The testing took a lot of time to run and was not very effective. Code coverage provided proof that the lines I changed were tested. No more guesswork, no more bluffing. It also meant that I could test with much less data than I had been using, so I could test sooner and more often. I could save the massive datasets for final testing.

I am more excited about mainframe development than I have ever been—I envy developers today with everything they have available. The applications they will be able to envision, the connections, and the things they will be able to accomplish to keep the mainframe going in the decades to come will be amazing. All of this will be done faster and with much greater quality than we could have ever imagined back in the slow old days. These are truly the good new days for the mainframe, and I am glad I had a small part in building it.