LESSONS LEARNED FROM WATERFALL AND MY FIRST TASTE OF AGILE

Overview: Learn how one of BMC AMI DevX's expert application developers discovered the benefits of an agile approach to projects and the risks of waterfall early in his career.

Having been a mainframe programmer for over 50 years, it should come as no surprise that I've been party to the siloed, top-down practices of waterfall. But I've also practiced what we call Agile development and DevOps today. In fact, the teams I've worked on over the past five decades have sometimes hovered closer to the best practices of these modern philosophies than to waterfall, though not always completely.

My earliest big project is a good example. Back then, we didn't know about waterfall or Agile or DevOps, we just tried to get the job done. Nevertheless, we began this project in a very waterfall fashion, and quickly ran into problems. It wasn't until we began using more agile methods of work that things began to improve.

If you're skeptical of Agile development and DevOps and are going take anything from this story, I hope it's this:

> *Don't balk at these modern methodologies—which are necessary in a digital economy—just because you've been doing mainframe application development one way for decades.*

Regardless of your background, you can adapt your experience to these Agile and DevOps and actually leverage them to improve as a mainframe professional with years of expertise—because, chances are, you've leveraged some of these practices before without realizing it.

The Project in Waterfall

The project was for a manufacturing plant in a small Midwest town. I worked first on a manufacturing bill of materials system so that we could more accurately acquire labor standards and calculate manufacturing costs.

This led to building a revamped order entry system/material requirements planning system that was used to feed work to the plant floor. It was 1980 or so, when online systems were very expensive and beyond our budget; when waterfall thrived.

There were primarily two of us working on the project, with one or two others who would temporarily assist. At any time, there were two or three people working on design and coding.

I distinctly remember being eager to start coding, but everyone else wanted things mapped out first. I don't remember how much time we spent designing the system, but let's say it was three to four months.

This was waterfall for sure, in that most of the design was done before coding started. But it was not a highly detailed design. There were a good number of details left to judgement to be used at coding time.

Still, we worked on development for probably four months and extended it by at least another two as we found more work to do near the end. Things weren't going according to the plan we had spent months creating.

Shifting Speeds to Agile/DevOps

It was at this point that we began a natural shift to more agile work. We kept finding things that we hadn't thought of and had to iterate on-the-fly.

On-the-Fly Iteration

For example, it was always assumed that we would do the maximum amount of data validation feasible. (I pictured myself as very particular about data validation and not just that numeric fields were numeric. I was always on the lookout for how we could validate aspects of the input.)

But when we tried to run sample data through the new system, bugs reared their ugly heads. There was always a need to make edit routines better. If data was rejected, it was done in a way that did not destroy integrity.

New Responsibilities

Somewhere in this time, the lead programmer left, and I was put in charge. I worked on debugging and training data entry people (keypunchers) and users (the input data had to follow different rules). Finally, though there were still rough edges, I decided that a full parallel operation was needed.

Due to the number of remaining problems, I became the computer operator and switched to

working a third shift after the normal evening work was complete. I would run the new system as far as it could go. If there were fatal errors, I had to diagnose the data, correct it if necessary and rerun it.

In many cases, code had to be changed to either correct the logic or to add validation logic so that bad data got rejected. In some cases, redesign of some routines was required. I spent considerable time talking with data entry people and end users to get control of data errors and improve quality of the data.

Cross-Team Collaboration

My time as operator and instant fixer was painful in some ways, but invaluable in getting things worked out in a reasonable time frame through quick response. I had to spend time every day coordinating with the day shift operations and with other programmers and with users and with keypunchers. This was kind of like a scrum.

I remember times I dreaded having to talk to the order entry supervisor who had a bit of reputation as being hard-nosed and wanted no delays in getting information to the plant floor. But things usually turned out well even when I had to apologize for what seemed like too many errors.

I remember having to go to the production control manager who had a real reputation for being tough. I was shocked at his level of understanding when I pointed out the problems and the fixes that I was working on.

After four months or so, progress was being made and at a much faster pace. Through these efforts, it all worked out, and rather well, with the system becoming the backbone of at least three additional plants.

What About Waterfall?

In theory, waterfall may seem logical, but only if it's supporting the kind of work that doesn't require fast-paced iteration and innovation. You can indeed get predictable results—but results that take forever to complete and that leave users unhappy in today's digital economy. And by the time software changes are delivered, the users' needs have changed.

It is not that you blindly start on a journey without any idea of the destination, as many who come from waterfall development assume of Agile and DevOps. You might even want a rough-route map. But you do need to adjust to things you encounter as you encounter them. Having a rigid plan can spoil a trip.

> *Quitting the waterfall mentality is pretty much like quitting smoking.*

There are some serious challenges. There are habits to break. But quitting is worth it, because waterfall is not nearly as logical as many still assume. Even early in my career, before the dawn of the digital economy, waterfall failed us. It was an agile approach that saved the project.

There are so many benefits to be gained from achieving true mainframe agility. But understand that achieving those benefits is the result of a journey to which you've committed.