Retail Banker
International

Comment

Why translating legacy code is a modernisation trap

Srikumar Ramanathan highlights the practical dangers of turning to automated tools to convert legacy code into modern languages

April 23, 2026

Share

Srikumar Ramanathan warns that automated tools may offer speed but often create a false sense of progress, leaving underlying complexity untouched and limiting future innovation; image credit: Digitala World/shutterstock

It was a few decades ago when I was part of the professional services arm of a large multinational technology firm, when I was asked to work with a customer who had some mission critical applications written in COBOL. The code was running on an obsolete proprietary hardware platform quickly going out of support. The need of the hour was to migrate these applications to a more modern platform so that the customer can not only continue without worrying about obsolescence but also grow their business and be agile with their customer needs.

Being a ‘smart and eager’ technologist, my estimate for doing that work was purely based on the preliminary study of the number of different programs and their complexity levels, assuming that I should be able to get my team to translate these into three-tier architecture, which was considered state-of-the-art at that time. We won the business and I was asked to lead this project.

Only upon closer scrutiny did I understand the complexity involved in making this transformation. The COBOL programs were very tightly coupled with data file definitions, processing logic and even formatting of outputs, all in one program. They had no intermediary storage of results and no logical separation between data and logic. What was one program in COBOL often translates into multiple programs and functions, besides the need to define data models to support those.

Often, for performance purposes, some ‘capabilities’ are embedded in multiple programs

Extracting these capabilities and modularising them is essential to create a more maintainable code. A typical example would be pricing or ‘payments. These are often repeated in monolithic cores.

This obviously meant that my initial estimations were out of whack, I could either recreate the functionality in a new environment, or I could own up to my simplistic estimation and explain why it is worth investing the extra effort in modernising it the right way. I chose the second option, at the risk of getting beaten down by both the customer and my managers! It was the right thing to do! The scars that this program left on me still sting as I recall the whole process of transformation!

However, more than the scars, the feeling that I still recall with pride, is the ease with which we could implement newer features on the rearchitected system and the pace at which we could implement new products and services. The interesting story is that the President of the firm, offered me the CIO job at the end of the program and having accepted that, I reaped the benefit of all the hard work too!

The reason I recall this story today is because of late, there have been many announcements of products that can translate COBOL code into Java, Python etc. This prompted me to wonder if I could have had an easier job implementing that project if I had this tool then? My answer is a definite no! I might have been able to give the customer a working system in a shorter timeframe, but the customer would have spent a lot more time implementing new features and probably even failed as a business struggling to implement features that were required for them to stay relevant.

Translations from one language to another is not modernisation

Translations from one language to another is not modernisation and will never give you the desired

outcome.

Fast forward to today. Having worked on similar projects across large enterprises since then, the lessons have only deepened. The temptation to reach for the nearest tool, whether it was a three-tier architecture decades ago or an AI-powered code translator today, is always strong, and always carries the same risk: optimising for speed of delivery rather than longevity of outcome.

What became clear, project after project, is that the most critical and most underestimated activity in any modernisation effort is extracting the *business logic* from the legacy code, not just the program logic. These are not the same thing. Program logic tells you what the system does. Business logic tells you why it does it, and under what conditions, and for whom. The gap between those two things is where most modernisation projects quietly fail.

Consider a recent engagement involving a global financial services firm running a cards platform that was over four decades old. It processed billions of transactions annually and had to respond in milliseconds. It worked. But it was written in COBOL and Assembler, woven together over generations of developers, and virtually no one alive fully understood why it did what it did.

Translating that code would have been an act of archaeology performed blindfolded. The only viable path was to first understand the business, map its capabilities, and then engineer forward from that understanding rather than backward from the code. By rebuilding around business capabilities rather than translating existing code, the firm achieved overall

efficiency improvements of 40 to 60% and ended up with an architecture that could actually evolve.

There is another humbling lesson worth naming: It is not just LLMs but subject matter experts hallucinate too. Domain knowledge that seems settled and shared often turns out to be fragmented, contested, or simply wrong when held up to scrutiny. One of the most valuable things a structured approach to business logic extraction does is surface those disagreements early, while there is still time to resolve them, rather than after they have been encoded into the new system.

Conclusion

The conclusion I keep arriving at is the same one I reached on that first project all those years ago: the shortcut is rarely short. Translating COBOL to Java, or any legacy language to a modern one, may produce a working system. But a working system is not the goal. The goal is a system that the business can evolve, one that makes adding a new product or responding to a new regulation a matter of days, not months. That outcome only comes from modernisation done at the level of meaning, not syntax.

The code is the symptom. The business logic is the cure, once it has been properly understood and liberated.

Srikumar Ramanathan, Chief Solutions Officer, Mphasis