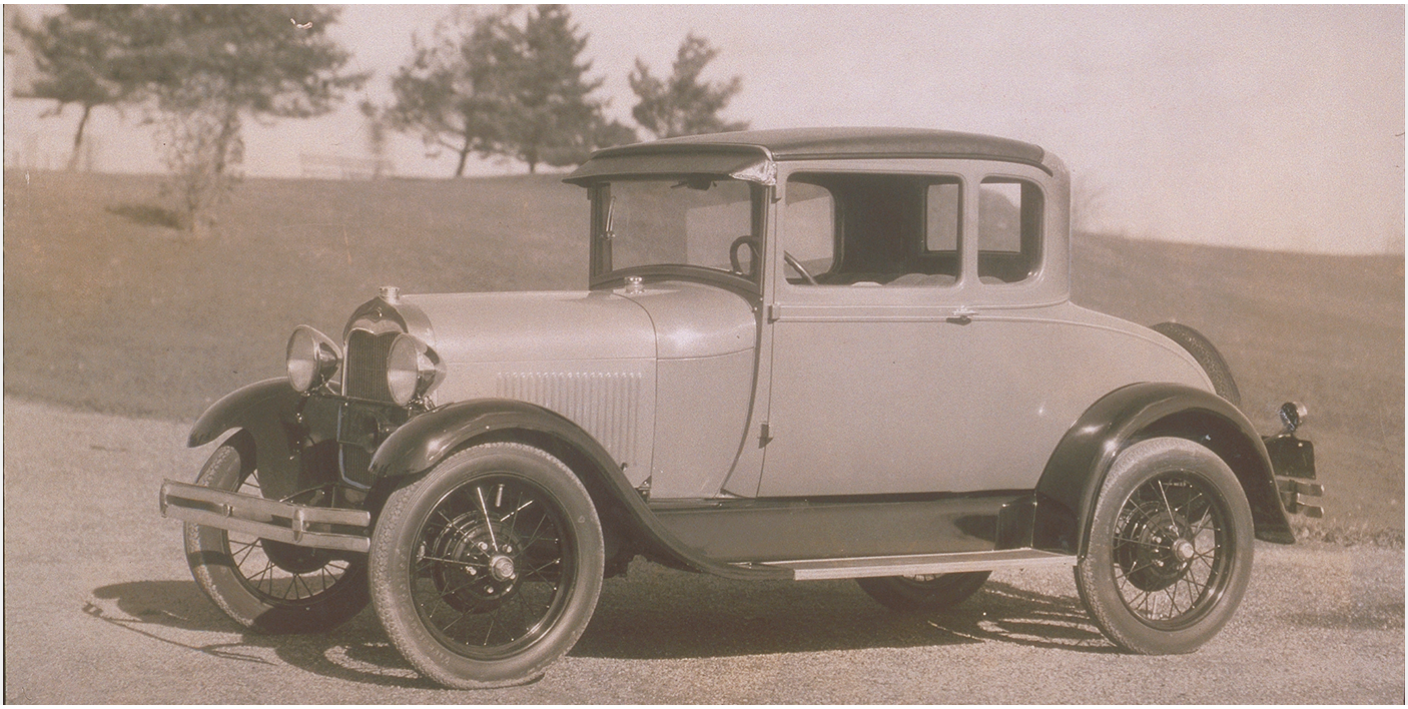
# “MAKE IT JUST ONE BOLT!” THE CASE FOR GREATER SIMPLICITY IN MAINFRAME CODE



Back in 1927, the Ford Motor Company introduced the Model A to replace the legendary Model T which had been in production for close to 20 years with few changes in design. This was a chance for Ford to start over and design a much more modern car. But in doing so, it didn't ignore the simplicity which made the Model T such a success.

As an example, the original design for the carburetor on the new engine had 14 little screws. That was judged to be too complex, not only for assembly but for future maintenance and repair. When Zenith, the company contracted to design and manufacture the new carburetor, was told to redesign it for simplicity, it came back with a version that only had two bolts holding the carburetor together. According to a well-known story, when presented with the new design Henry Ford said, “Two is too many—make it just one bolt!” In the end, millions of Ford Model As were produced with carburetors using only one bolt. Many are still running today.

That simplicity helped in manufacturing then, and for maintenance thereafter. When I look at software development, I'm sometimes reminded of this example. As I tell developers, “**Simple is hard**, but most often worth it.” How often do we code 100 lines when ten will do? Why does this matter? For the same reasons it did with the Model A: It's simpler to manufacture and simpler to maintain.

Here are five ways to keep code simple:

- Requirements—Start with what is necessary to serve the user and solve that problem. It can be tempting to do more while you are there; to solve multiple problems by building in logic for a possible future expansion or to offer multiple options for that solution. Granted, you do need to

plan for the future, but you only need to solve the *one* problem today.

- Structure—I asked developer Zyad Alyashae about writing simpler code. His response? "My answer to simple code is comprised of one word— If the structure is right, the code falls right into place. The same way the one bolt would fit in place if the carburetor was created so that one bolt is good enough." Structure your code so that the simpler solution is sufficient to address the problem at hand.

  Alyashae continues, "As an example, be sure to compartmentalize. That means breaking the code into simple blocks or compartments which are assembled to build the logic. Each block should be easy to understand on its own; the function is clear. I think of it like a shape-sorter toy for kids—the existing code system infrastructure is the box and the new code being added is like the shapes. If the structure is set up correctly, then the shapes fit just in place with where they belong, almost conceptually like the toy." Avoid making code too long and too complex. You may think it's easier that way, and it may be when you are writing it, but it won't be over the long-term.

- Graphical analysis—View your code with a graphical analysis tool like BMC AMI DevX Code Insights. With this tool, you can clearly see how your code flows, giving you great insight into potentially tangled logic. If it appears the logic looks like spaghetti in the chart, can you imagine how hard it is to follow the code without the graphical aid?
- Documentation—If you feel the need to write extensive documentation for your logic, let that be a warning sign. Simple code is easier to understand and needs less documentation. The goal should be to require as little documentation as possible for a good understanding.
- Code review—In code reviews, always ask, "Is this the simplest way? How difficult will this be to maintain?" Knowing that you could be the next person to work on this code, how do you feel about it?

I'll say it again—*Simple is hard.* Developers need to take steps to deliver simplicity if we want to deliver code that can be easily maintained in the future.