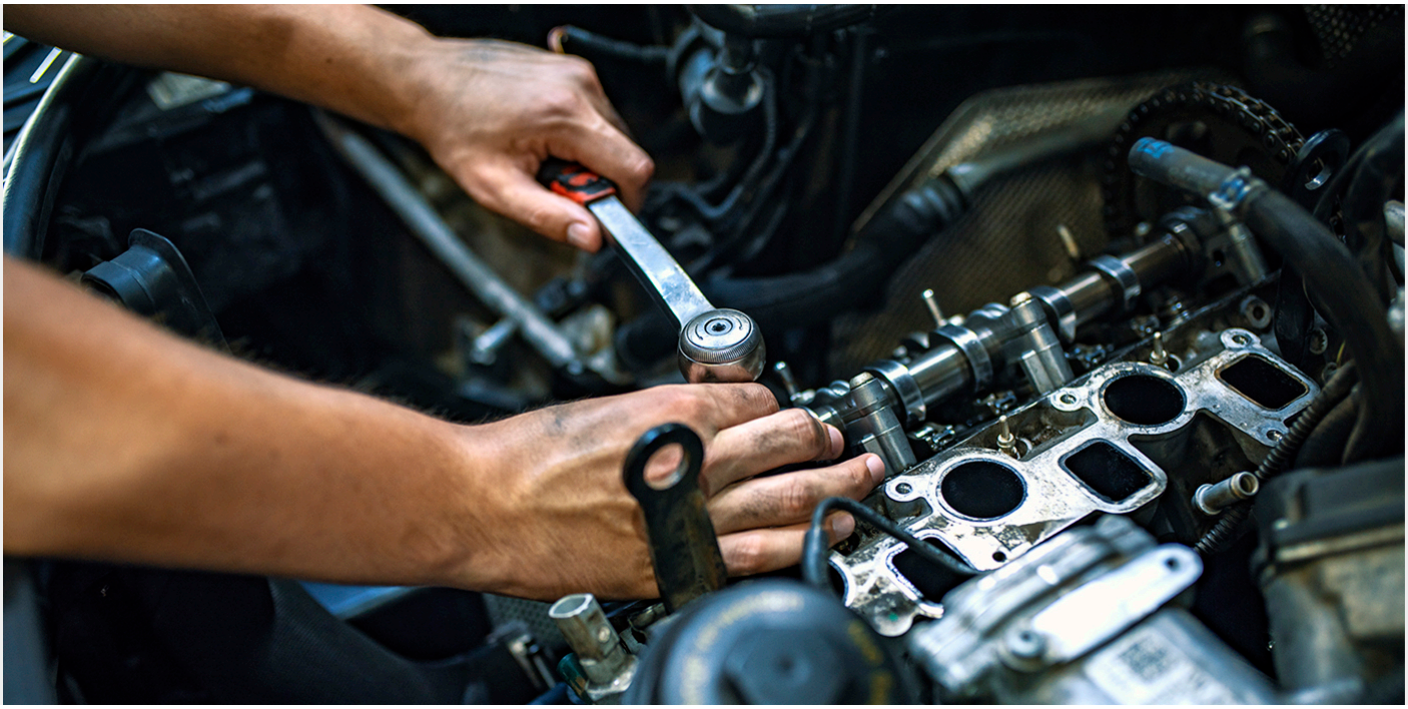
DESIGNED FOR OTHERS: HOW TO PLAN FOR FUTURE MAINTENANCE



We've both spent a lot of time working on cars, especially Robert, who is a former auto mechanic. Ask any mechanic about cars they hate to work with, and you will come up with many examples where it appears that the designers didn't take maintenance into consideration. The designers are focused on quick assembly, which makes sense because easier, faster assembly keeps costs down. There are times where a part is put into an assembly, then that assembly is attached to the car, which makes it difficult to now gain access to the part.

As an example, many vehicles have a crossmember support right next to the oil drain plug. In most cases this will just cause the oil to splash in all directions, but in a few cases, it makes it difficult to remove the drain plug. This is a design that easily could have been prevented. Other maintenance items are not as forgiving. Changing the spark plugs on a late '90s Camaro with a V8 engine is almost impossible without removing the engine. The car was designed to have a V6 engine, but V8 engines were installed in them, as well, making the engine a tight fit and making maintenance extremely difficult. One final example: When performing maintenance on an F250 Super Duty newer than 2015, it is sometimes necessary to lift the entire cab off of these trucks to do just about any work on them.

Robert thought he had left that (and dirty fingernails) behind when he moved into software development. The surprise is that it is still there. In software there are many examples where the initial design may have been quick to build, but difficult to maintain.

This does have consequences. Ignoring future maintainability adds to Tech Debt, which affects your ability to add new features in the future. Maintaining anything, whether it be a car or software, can

be tedious and time-consuming. In automotive, the technicians who maintain the vehicles do not get to design them, but when a feature gets added to software it is usually the same programmers who maintain it. Debugging through code that was written years ago can take a lot of time and brainpower to fix a simple issue. That time could have been cut in half, if not more, if the feature was not rushed and had been designed properly from the start.

What we need to do is the same thing that mechanics ask auto designers to do: Don't consider only ease of assembly, but look to the future and ask, "*How can this be maintained?*"

We recommend the following practices to help you write code that is easily maintainable:

- **Identify** code which may need to be updated in the future, perhaps where there might be changes in legislation or priorities which would require us to have to change this logic. Make this part of your code review.
- **Document** not only what the code does, but what the intent was, and explain how to update it.
- **Collaborate**. Don't be afraid to collaborate with team members on how the code should look. All team members will be maintaining this code, so why not ask them if they have a better way to structure the code? There is more than one way to write a solution and you are not always going to come up with the best one.

You should also consider these technical aspects:

- **Use tables**, but make them obvious and easy to update, and make them callable in a database.
- **Use comments** in the code to explain how to update in the future.
- **Create as many methods** as you need to make the code readable. A method should be limited on what it does. The more a method does, the harder it will be to reuse that method. Reusing methods will prevent duplicate code and make the code easier to read.

When designing and coding, we do have to produce things quickly—that need doesn't go away—but take the time to think of how your code will be maintained in the future.