Jake Vollkommer

The Architecture of Selenium WebDriver - An Open Source Library

Selenium WebDriver is an open-source tool used for automation in web browsers. It was primarily created to run test cases, but it is capable of running limitless automated process within various browsers. The idea in mind was that humans are not good for this. Humans are very error prone, especially when carrying out mundane and repetitive tasks for long periods of time. Also, no human wants to continuously do these mundane and repetitive tasks. So, the Selenium library offers developers a relatively simple way to record, write, and execute end-to-end test cases which simulate a user in web browsers, which actually isn’t a very simple task.

The Selenium library is relatively new in terms of the state of computing at the time. However they constantly have to keep up with browsers being updated, like the release of HTML 5 in certain browsers a few years ago for example.

Selenium is implemented as an API, with a backbone is javascript. It has very specific but simple architecture themes which are adhered to. The general idea around all the architecture themes is that it is important that the service works well, is robust, and it not too difficult to use. The way this is kept true relies heavily on their architectural design of course, heavily focusing on having sympathy for Javascript implementation. This is because nearly every browser has javascript support.

There are two extremes of complexity when writing a library for developers. The author, on one extreme, can keep all the complexity hidden from the developer so they do not have to understand how any of it works. The other extreme is that the author spreads the complexity out very thin, and requires the developer to understand how everything works to implement any of it. Obviously both are poor solutions, and a balance is needed. However, the authors of Selenium have decided to keep the complexity mostly compacted and hidden from the developer, enough so that they don't need to understand everything but also don't have to understand everything. This makes the user experience much better. It is very transparent that user experience is of top priority for this open source library.

Developers across the map all contain very different skill sets. The architecture has to keep this in mind. In order to keep the best user experience, the architecture must allow developers to remain comfortable, and be able to code in whatever language they are comfortable in. Because of this, Selenium offers a wide variety of languages at the expense of more code to maintain. However, the maintenance is made easier after making all the browsers look identical to the language bindings. This is a big job, but again the user experience is at focus, and the pros weigh out the cons with this in mind.

The physical organization of functionality is highlighted very well by an approach of layers built on top of each other. The base of the structure is Google Closure - a library which allows source files to be kept small and focused while offering primitives and modularization mechanisms. On top of this, a utility library lies containing what Selenium likes to call “atoms”. These atoms carry out all the basic functionality of simulating a user in a web driver. They include tasks such as getting values of attributes or simulating a click. On top of the atoms exists two adjacent layers: the WebDriver and Core, which offer all the methods you see when implementing the library. This structure is simply and beautifully displayed by an image contained in the article: *Layers of Selenium Javascript Library*.

The atoms are a perfect demonstration of the architecture’s robust and simple structure. On one hand, they enforce the goal of having a Javascript-friendly implementation. What’s more important for the library as a whole is the simplicity of fixing potential bugs. Because the atoms are the foundation of higher methods, instead of having to fix bugs across multiple implementations across the codebase, it is possible to fix the bug in one place. This also means that the cost of change is dramatically reduced and does not sacrifice must stability nor effectiveness.

Selenium practices the Open-Closed principle well, in the sense that the library is open for extension but closed for modification. This is displayed perfectly through the use of atoms and the layered structure of Selenium’s functionality. As far as Object-Oriented design principles go, a lot of them don’t necessarily apply to Selenium, because although it makes a point to have an Object-Oriented design, it couldn’t be implemented entirely as such.

An interesting SOLID principle to look at in the context of Selenium is the Dependency Inversion Principle, which states that high-level modules should not depend upon low-level modules. Both should depend on abstraction. It is clear that this is well practiced just by looking at the layer’s diagram. WebDriver and Selenium Core only depend on atoms, which can be abstracted across tons of implementations.

It is sometimes thought that Object-Oriented design equates to good design. This is not always true, especially in bad practice. Selenium having Javascript-sympathetic implementation shows that a lot of good design principles can be kept without a strict standard Object-Oriented design.

I feel that Selenium’s architecture is really solid. A good architecture offers good maintainability, extensibility, and modularity. The use of Selenium’s atoms make it easy to see how each of these points are important to Selenium’s architecture. While reading the article, these three things kept popping out to me and I was fairly satisfied by the end that I didn't have many complaints. It is clear that the authors of Selenium care a lot about maintainability, extensibility, and modularity. A lot of pivots in their product have revolved around these points, always with the user experience in mind.