Kody Pope

6-30-24

Assignment 6.2

Case Study: Strangler Pattern at Blackboard Learn (2011)

What took place at Blackboard before 2011 is an excellent example of what we now know as a monolithic architecture. And here is the idea that one big giant thing did everything, and because of this, if an error ever occurred, then that meant it was a massive error for everyone and could be a headache to solve. But have no fear there is a solution to this problem, and the solution is called the strangler pattern.

(Myerscough), shares with us what the strangler pattern is. First what is the strangler pattern? Before I go further, let me first share why it is called the strangler pattern so that we have a better understanding of what is happening. (Myerscough), shares that the strangler pattern is modeled after the strangler fig pattern. This is a plant from the fig family that attaches to a tree and then slowly takes its place. I did hear that it is replacing the tree, but bit by bit, leaf by leaf and branch by branch, rather than trying to remove the tree entirely and then plant itself, it will start in one small spot and then work its way down the branch, and then to another branch eventually taking over the entire tree.

Hopefully, this is what we are doing with our monolith architecture. (Myerscough) tells us just how we can use the strangler pattern. You may wonder why we would not want to remove the entire “tree” and start over from scratch. Well, the answer is straightforward. To redo a whole system using new technologies would take way too much time and resources. And we honestly might risk losing out on market share trying to do so. Especially when we do not even need to do so. We can just use the stringer pattern to implement changes that customers would not even notice. All they notice is that our “tree” is getting better, so here is how to do it.

First, we would take one small part of our architecture we wish to improve, create a façade, or in this case, a replica of what was there before, and use APIs to direct customer traffic. In one of the replicas, we will make our changes. After the changes are made, we will direct our customers to that new API; if all goes well, the customer will never notice. They are experiencing a different technology than what they have been before. However, if there are errors, it is easy to redirect them quickly. Any new incoming customers and those currently there to the old branch that we still have running at a separate API.

But if there were no errors, we would have a branch of code that could replace the old one, and we would remove the old API as this new API with the changes would now act as our parent branch. Then, it is time to move on to another part of our architecture and follow the same steps again. We will create a replica. using API, make the changes, redirect the current customer to the new API. If all goes well, keep the new code and then delete the other API with the old code. The fun does not just stop there. Not only can we use this method to replace the entire architecture, but from now on, any changes made to other parts of the codes, or even parts we have already updated, can be used to implement changes as a way to test to make sure nothing fails as if there is now a quote-unquote, older version to revert to.

This is precisely what Blackboard did with their monolith; these APIs were internally known as building blocks. What a great term for such a process. The idea here was to create a building block that already had a starting point (the old code), and then they would make changes and replace the old “unchanged building block” with the new one. If there was an issue, they could switch back the blocks. But the first old building block would get thrown away if there were no issues.

Works Cited

Myerscough, Tim. “Strangler Pattern.” Mechanical Rock Blogs, 4 May 2020, blog.mechanicalrock.io/2020/05/04/strangler-pattern.html. Accessed 1 July 2024.