Elsa M. Rodriguez Plaza

Student ID: 903453539

Professor Mark Moss

Software Architecture and Design (CS6310)

November 11th, 2018

Assignment 7: Design Debate Paper (v3)

The importance of understanding the hardware, respecting your team members and staying hands on

It was a Friday night and I was deploying a system that was supposed to be available for Acceptance testing by the beginning of following week. I started an automated deployment using a Jenkins server, everything worked as expected and after some time the deployment succeeded. After a while, I remembered that this deployment was critical for our stakeholders. For this reason I logged into the system that I had just deployed and to my surprise none of the new use cases we needed to demonstrate worked as expected. By the time I realized of that issue, it was already 5:00 pm and I still had sprint stories I need to finish up before the following Monday. At the time, we were working with engineers based out of Russia. I knew one of them had made these type of mistakes in the past, so I thought of checking his commits and voilà, three hours before he had committed code that I knew could impact the new use cases. I started trying to contact the developer; however, because it was already 11:00 pm in Russia, he was not available. So, I was left with a broken program and the person responsible for the issue unavailable.

Situations like the one mentioned before happen more often than we can imagine and as unreal as it seems most of the times whoever caused the problem will not be there to fix it. This is the reason why I totally agree with the quote "Commit-and-Run Is a Crime" (Nilsson).

As per my experiences, there are multiple problems that arise when someone commits and runs. First of all, most of the things never work the first time, there are multiple constraints that must be enforced before any code can be merged into the master branch. One must check for compilation errors, runtime errors, overall functionality, functional and nonfunctional requirements, and most importantly that the code does not break other team members' code. Consequently, the person who commits and runs skips running safety measures required for code to be fundamentally correct. So, figuratively speaking this programmer becomes a time thieve when he decides to save time for himself by stealing other developers' time. After all, when a situation like this happens, other programmers have to fix the broken code and make sure that everything works as expected after the fix is applied.

Another issue caused by carelessly committing code is that when a team is working on a critical release every single line of code will get deployed into the production environment. This means that the broken code will trigger a massive incident, which will cause the system to become severely impaired for hours until a solution is found. (VMware)

One other problem that might happen as a consequence of pushing bad code is one that in fact can be harder to detect. That is code, which does not apparently break anything, but it jeopardizes the security of an application. For instance, code that exposes access credentials for the company's managed resources on the cloud. These type of incidents can make a company go out of business. (Venezia)

It is impossible to control every programmer on a team. However, there are certain rules that can be implemented to prevent commit and run problems from happening. One of the rules, which we have enforced lately on my team, is the approved pull requests. This measure allows us to stop unreviewed commits from getting pushed to the master branch, which is especially useful for preventing severity two incidents resulting from broken code being pushed to the development environments. Another utility we rolled out recently as part of the Source Control System, is to block direct commits on the master branch. In addition to those, we also put in place automated tests to run against the source code, and automated notifications alerting the test results. Finally, we scheduled internal demos days before an important demo, so we can spot problems and solve them in a timely manner.

Software Engineering is a discipline that continuously evolves, it can be hard to stay up to date, especially when there a large number of technologies for solving the same problems. However, architects are supposed to be the technical leaders everyone looks up to, they should be able to see the "big picture" and help the technical team accomplish great things. An architect can easily become a manager if he spends his time in meetings with management, rather than innovating or rolling out new technologies. This is essentially what the quote "Architects Must Be Hands On" (Davies) states.

One of the reason why I think the quote is correct is because some time ago, I worked on developing "Project X", which turned out to be the most frustrating and difficult project of my entire career. There are multiple reasons why I stated that. Firstly, our architect knew a lot of theory, but he had not written a single line of code in over five years, nor he was interested in doing so. Secondly, whenever the team had a questions on the system's requirements, he would explain the problems in an such an abstract manner no one could understand how to transform his instructions into a functional piece of software. Thirly, because of the profound confusion he caused on all the members of team, we ended up working in silos, nothing ever got fully done.

"If a Software Architect does not code as part of her day-to-day activities, is a matter of time when she becomes obsolete" (Franco Capo). According to Carlos Franco Capo, who is a director of technology at Expedia, staying up to date if one of the duties of a Software Architect. "Project X's" architect was only holding a generic title that can be often be assigned to a vast majority of roles within a company. The "Architect" whom I am referring to had long ago stopped growing his capabilities. Consequently, he had no credibility as a technical leader, the team knew that all she could do was draw "boxes and arrows" (Franco Capo) based on erroneous technological assumptions and obsolete ideas.

Nowadays, Software Architects are expected to know every single aspect of the software design and architecture. Cloud computing has become the preferred approach for deploying software products in the last decade. Cloud providers offer numerous services to ease continuous integration and delivery (CI/CD), provisioning, security, and many other aspects needed in software development, including Infrastructure as Code. Because everything is "facilitated" by the Cloud providers, the business expects the architects to be these magical creatures capable of accomplishing all things when it comes to delivering a system.

Presently, I am developing a solution that involves implementing a Web Application in React JS (GUI), a Backend composed of multiple microservices using Flask (Python microframework), the Redshift Data Warehouse, and a Postgres Relational Database. Currently, I am the only engineer participating in this effort, so I have also assumed the role of architect.

Not only do I need to design, develop, and test the whole system, but I also need to make sure we have a CI/CD pipeline capable checking out the code when there are commits, run the automated tests, and deploy to Amazon Web Services Cloud (AWS).

The most complex step in the process of developing this system is the implementation of the deployment strategy. The reason being is that I come from a Software Development background. Therefore, it supposes a great challenge to comprehend which is the most optimal hardware configuration needed to run the system as efficiently and cheaply as possible on AWS. For this reason, I consider the quote "You Have to Understand Hardware, Too" (Wickramanayake) to be fundamentally correct.

Fortunately, I came across AWS Cloudformation, which is AWS's managed service specially designed to provision stack of resources on the Cloud. Cloudformation's most attractive feature is that by writing a single JavaScript Object Notation file (JSON), one can spin up an entire infrastructure with the configurations you required by the Software Product. Behind the scenes the Cloud provider maintains the infrastructure, installs software updates, replaces defective hardware as needed, etc. (Amazon). Nonetheless, to be able to write Cloudformation scripts one needs to know about the type of services those servers have to run, the storage, memory and processing capacity (CPU) of the cloud nodes, the communication strategy, the security requirements imposed by the business, among many other implications that need to be addressed along the way.

"In my own experience I found that those software engineers who knew their hardware far out-performed those who didn't – in the quality of work, in the speed of work, in their presentations, in their visibility, in their confidence, and in their growth." (Buoyanci). According to Buoyanci, engineers should also know how a computer internally works to be able to design efficient code. The reason why this is stated, is that if a Software Architect understands the fundamentals of memory management, how multiple processors can be used to run multithreaded applications, how to distribute the load when a system is overloaded, how to autoscale based on CPU or memory utilization, storage types that can be used, etcetera, she will end up architecting efficient systems both in terms of cost and performance.

Understanding the hardware needs of a system, together with the best way to design efficient software are key aspects when it comes to architecting a successful product. After all, users are very impatient people, who resent having to wait for a screen to load.

Works Cited

Monson-Haefel, Richard, Davies, John, Nilsson, Niclas, and Wickramanayake, Kamal "97 Things Every Software Architect Should Know." - O'Reilly Media, 5 Feb. 2009, shop.oreilly.com/product/9780596522704.do. 2 Nov. 2018

Franco Capo, Carlos. "Why a Software Architect Should Be 'Hands on.' LinkedIn, 31 Oct. 2016, www.linkedin.com/pulse/why-software-architect-should-hands-carlos-franco-capo/.

"Severity Definitions & Response Targets – VMware Support." VMWare, 24 Oct. 2018, www.vmware.com/support/policies/severity.html.

Venezia, Paul. "Murder in the Amazon Cloud." InfoWorld, InfoWorld, 23 June 2014, www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html.

"AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning." Amazon, Amazon, aws.amazon.com/cloudformation/.

"What Every Software Engineer Should Know about Hardware." Buoyanci, 6 May 2015, buoyanci.com/what-every-software-engineer-should-know-about-hardware/.