The amount of effect that goes into software development can be a large undertaking. But every software engineer needs to find a way to tackle these difficult tasks. One company may require to overhaul and recreate their entire system to make it run more efficiently as in the way of the caste Study: Evolutionary Architecture at Amazon. Or they have a difficult task in procuring a way for people to send instant messages daily while needing to run at the performance level of the browser and not some stand alone software like Facebook did in 2008 with the case study: Dark Launch of Facebook Chat.

In the case study involving Amazon, they had an issue with their program Obidos in 1996. Obidos was designed to store “business logic, all the display logic and all the functionality that Amazon eventually became famous for” (Kim et al., 2016). Their software was so large and “too tangled” (Kim et al., 2016) and complex that that the “individual pieces could not be scaled.” (Kim et al., 2016) Amazon explains that there were a lot of things that needed to be done but couldn’t. The software had too many complex pieces of code that couldn’t be combined into one system. (Kim et al., 2016) Amazon would need to go through and create “a fully distributed, decentralized services platform serving many different applications.” Stated by Werner Vogels (Kim et al., 2016).

During the case study the authors of our book The DevOps Handbook suggest there were three main lessons learned from this study. First lesson to note was that “strict service orientation is an excellent technique.” (Kim et al., 2016). This technique would give them more control over their software. The second lesson they learned was to limit the direct access from their clients to their database, the software would become more reliable and easier to scale. (Kim et al., 2016) Last lesson learned was that changing to “Service orientation” (Kim et al., 2016) would enhance immensely the processes for developing and operating processes. (Kim et al., 2016) Which led them to an incredible feat of “136,000 deployments per day.” By the time it was 2015. (Kim et al., 2016)

This case study involving amazon stated that Facebook was going through a giant problem for their software engineers. This fear would be a huge undertaking that would take about a year to finish. (Kim et al., 2016) Facebook is one of the most popular social media websites, and for ten years was the most popular website that in 2008 saw over seventy million users daily. (Kim et al., 2016) The biggest challenge they faced was creating a chat system. A software engineer that assisted in the development of the chat box system named Eugene Letuchy states that “The most resource-intensive operation performed in a chat system is not sending messages. It is rather keeping each online user aware of the online-idle-offline states of their friends.” (Kim et al., 2016) This online/offline system is a crucial part in allowing users to send messages back and forth. (Kim et al., 2016)

To code this chat system Facebook would need to use C++, JavaScript, PHP and other technologies as well. (Kim et al., 2016) During this process, the engineers would deploy their code once daily which began only to be seen by the team using their GateKeeper service that toggles services. (Kim et al., 2016) They called this their “dark launch process” where they would test the chat when it was hidden from the UI but could still send tests to the backend service. (Kim et al., 2016) Which allowed them to figure out performance issues. They deployed the final chat system slowly, the Facebook employees got it first, which then rolled it out to “1% of the customer population.” (Kim et al., 2016) Then later to 5% where they continued the deployment gradually. (Kim et al., 2016)

Both these case studies seem to be different in nature. However, both represent a problem that the engineers were able to solve. Amazon solved it by recreating their service which is still being used. Whereas Facebook’s chat system took a lot of time to create due to the complexity and performance it required.

References:

Kim, G., Debois, P., Willis, J. O., & Humble, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. https://dl.acm.org/citation.cfm?id=3044729

Shivang. (2023, June 9). Facebook Real-time Chat Architecture Scaling With Over Multi-Billion Messages Daily - Scaleyourapp. *Scaleyourapp*. Retrieved June 14, 2023, from https://scaleyourapp.com/facebook-real-time-chat-architecture-scaling-with-over-multi-billion-messages-daily/