Jessica Hall

CSD380

Chapter 13's case study shows the architectural changes made by numerous widely recognized DevOps companies, including Amazon, Google, eBay, and LinkedIn. These businesses first relied on monolithic designs, which worked well in the beginning but became detrimental to scalability, efficiency, and innovation as their operations expanded. A successful product or organization must constantly modify its architecture to satisfy shifting expectations, according to the evolutionary architecture principle, which is brought up in the study. Long-term success frequently requires moving from a monolithic structure to a more loosely linked, service-oriented architecture (like microservices), according to the case study. However, this shift needs to be gradual instead of sudden. To make this procedure easier, the Strangler Fig Application Pattern is presented. This method allows teams to develop new features in modern architectures while continuing to use the old system when necessary, instead of replacing older systems all at once. The legacy components are gradually phased out to ensure a less disruptive and more seamless changeover. These ideas are further supported by the case study on Amazon's architectural change. At first, Amazon used Obidos, a closely connected, two-tier monolithic application. Scaling and maintaining the system got challenging as its complexity increased. In order to overcome these obstacles, Amazon built a service-oriented architecture (SOA), which allowed teams to collaborate on various services separately while preserving system performance. Strict service orientation, the necessity of preventing clients from accessing databases directly, and the contribution of modular services to fostering creativity and efficiency are some of the lessons that can be drawn from this shift. The risks of closely connected architectures are also included in the paper. These risks include the potential for unnecessary dependencies, trouble testing and implementing modifications, and general inefficiency. Because of these limitations, bottlenecks are common in large organizations, making even minor adjustments dangerous. Businesses can increase developer productivity, testability, and code deployment confidence by implementing loosely linked architectures with clearly defined APIs.