4 Common Application Modernization Failures to Avoid

Application modernization can be hard. We'll take a look at some of the anti-patterns of application modernization to avoid in order to succeed with your strategy.

1 Inadequate requirements for microservices goals

A team can set out to do 'just enough' to modernize their application suite by simply lifting-and-shifting it from on-premises or co-located servers to cloud infrastructure.

The brute-force migration of code and data that were never intended for a future of elastic capacity and microservices agility would fail to deliver the expected benefits of cloud modernization.

Only making cosmetic changes to a user interface isn't really modernizing at all.

Resolving and refactoring memory and CPU usage, synchronization and data states can untangle knots that accelerate the modernization push.

2 Misguided value expectations

An application modernization initiative should transform the entire digital backbone to support and grow the organization far into the future.

Teams drive down labor and service costs incurred through constantly maintaining software estates, reducing issue resolution costs, SLA or regulatory penalties, labor spent managing disruptive upgrades, and paying capex to expand and reserve ever-increasing infrastructure to meet usage demands.

Cost-based valuation scenarios will limit the scope of available improvements to whatever is most expedient.

Most companies still value top-line revenue growth over cost-cutting.

Modernization efforts and tooling can be prioritized for quick wins on the revenue and productivity side that also contribute to faster release cycles and better long-term results.

4 Common Application Modernization Failures to Avoid

3 Dependencies and technical debt

Technical debt robs the business of agility and throws sand in the gears of any application modernization effort.

The day a piece of code is promoted to production, it becomes legacy code. Fast forward 10 years and you find that the technology stacks the code was written for encountered generational shifts and most of the team members that wrote the software will have already moved on.

"Lift and shift" is seldom a good option for dealing with such dependeties due to ongoing business activity. It is incumbent upon the IT team to generate quick wins by decoupling the software suite at a more granular level and service-enabling one function at a time.

Extracting valuable intellectual property-in the form of business logic and processes from existing systems-also allows the business to realize continued value from that IP after modernization.

Once you scratch the surface on any complex critical application, there are far too many moving parts for humans to perceive and address at once. Correlating the threads of an existing Java Struts application with its big-iron backend to say, an API-driven Spring Boot or Kubernetes architecture that talks to a cloud data lake requires factory-level automation.

4 Scars From Past Failures

Failure is the only intergenerational constant in software development and integration. When 79% of application modernization projects fail at a cost of $1.5m and 16 months of work days, why try harder?

Initiating an application modernization initiative without intelligent tools for assessing and automating refactoring inevitably leads to burnout and people leaving the project or company. Employees bear the scars of past modernization trials.

All of the appropriate stakeholders of modernization within the organization and its partners should be aligned around a common source of truth, progressive delivery, and setting a regular cadence of quicker functional wins.

Rather than setting a big-bang replatforming requirement that may seem too daunting to reach, many companies instead opt for service level objectives (or SLOS) that improve fidelity and performance over time.

The counterintuitive secret of application modernization success?

Even the best-performing teams will inevitably encounter some failures on their way to a future state of a modern, scalable and agile application estate. But teams that use a data-driven and automated strategy for application modernization will be in a better position to understand and manage the risk and iterate much more intelligently and quickly.

And that brings us full circle. The modernization journey of a thousand apps starts with just one service.

vFunction

Request a Demo ▸

About vFunction

vFunction is the first and only AI-driven platform for architects and developers and architects that intelligently and automatically transforms complex monolithic Java applications into microservices, restoring engineering velocity and optimizing the benefits of the cloud. Designed to eliminate the time, risk and cost constraints of manually modernizing business applications, vFunction delivers a scalable, repeatable factory model purpose-built for cloud native modernization. With vFunction, leading companies around the world are accelerating the journey to cloud-native architecture and gaining a competitive edge. vFunction is headquartered in Palo Alto, CA, with offices in Israel. To learn more, visit vFunction.com.