Application Modernization, Service By Microservice

Incremental Application Modernization Succeeds Where Wholesale Rearchitecting And Replatforming Failed

by Kurt Bittner and Randy Heffner

November 20, 2015 | Updated: December 28, 2015

Why Read This Report

Application development and delivery (AD&D) leaders face a frustrating balancing act: deliver new applications ever faster while keeping the older ones technologically relevant. Rewriting older applications is financially and pragmatically impossible, yet delivering new capabilities often requires organizations to wring new life from older applications. Modularizing and incrementally modernizing older applications provides organizations with the time and the means to keep older applications fresh and adapted to new purposes.

Key Takeaways

Work Incrementally, But Plan Strategically

Modernizing applications is imperative for organizations to adapt and grow. Ultimately, modernization has to be continuous, but getting there takes time and patience. Building a little modernization into every project and every release is the key.

Incentivize Developers To Work Differently

Meeting immediate release goals is important but can myopically lead to monolithic applications. Incentivizing modularity and meeting immediate needs means balancing short- and long-term goals.

Treat The Service Platform As A Strategic Asset

Common services need a home, shelter, and regular care. Project-dominated funding models often neglect common services. Shift cost-savings from systems of engagement (SoE) efforts to support the services that enable them to deliver faster.

Application Modernization, Service By Microservice

Incremental Application Modernization Succeeds Where Wholesale Rearchitecting And Replatforming Failed

by Kurt Bittner and Randy Heffner
with Christopher Mines, John R. Rymer, Derek Nahabedian, and Ian McPherson

November 20, 2015 | Updated: December 28, 2015

Table Of Contents

Notes & Resources

Forrester interviewed 14 vendor and user companies, including 3Pillar Global, Accenture, Alliance Global Services (an EPAM Systems company), Capital One, Cast, Deutsche Bank, Disney, Fidelity Investments, GE Capital, IBM, Skytap, Target Brands, ThoughtWorks, Torry Harris Business Solutions, Travelers Indemnity, and Under Armour.

Related Research Documents

Brief: Software Innovation Requires A Loosely-Coupled Application Architecture

A Developer's Guide To Forrester's Strategies For API Success

How To Manage APIs For Customer Engagement

FORRESTER®

Forrester Research, Inc., 60 Acorn Park Drive, Cambridge, MA 02140 USA
+1 617-613-6000 | Fax: +1 617-613-5000 | forrester.com

Aging Applications Put Modern Capabilities At Risk

DevOps and continuous delivery practices help organizations deliver better customer and user experiences faster and with higher quality. Great results have been achieved, mostly by delivering better systems of engagement (largely mobile and cloud applications). Yet these applications don't stand alone; much of the data and functionality that make them possible comes from vintage, legacy applications, or would, if developers could access them. As organizations seek to deliver new and better experiences faster, they run into a problem: They can't evolve the old applications as fast as they need to. AD&D leaders seeking simpler solutions run into a variety of problems:

- **Legacy applications are not a single kind of thing.** The term "legacy application" is a broad catch-all that includes 40-year-old assembler applications, 30-year-old COBOL/CICS applications, 15-year-old Cold Fusion applications, and 10-year-old PHP applications — not to mention both off-the-shelf and custom applications. Even applications delivered yesterday are legacy when their architectures are poorly conceived. Each type of application has its own challenges and opportunities. No one technique works to modernize all, but there are some common principles. The central theme of modernization is increasing modularity and improving APIs. But decades of neglect cannot be remedied all at once.
- **It's not always clear which applications should be modernized.** Application portfolio rationalization reduces the size of the modernization problem by ensuring that not all applications need to be modernized. Decades of mergers, acquisitions, and disparate departmental IT strategies have left organizations with many applications that are either no longer needed or need to be consolidated. Before modernizing, organizations reduce their application portfolios, but teams don't always agree on which apps do or don't need modernization.[1]
- **Wholesale change to an application's architecture is challenging and risky.** Decades ago, applications were monolithic, depending only on the scant resources provided by the operating system (OS). Over time, new layers were added. Today, most applications rely on a rich ecosystem of relatively independent components. Modernizing applications requires organizations to break apart monolithic code, reduce dependencies on aging middleware, and replace much of it with modular services, cramming decades of application evolution into short modernization projects. It is challenging work that requires teams to understand the old while evolving the application toward the new.
- **Legacy applications lack APIs to support mobile and cloud applications.** Mobile and cloud applications thrive on representational state transfer (REST) APIs, but few legacy apps provide them. Thus, modernization becomes imperative when new applications require data and functions locked in legacy systems. Organizations don't modernize simply to reduce cost or prevent hypothetical future outages; they are driven by critical customer needs.
- **Old code, lost skills, and brittle app architectures amplify importance and challenge.** Many cases require the modification of vintage systems anyway. Even if their core business logic is still valid, 40-year-old batch applications and 30-year-old green-screen applications don't typically

reflect the real-time and highly engaging ways that business must work today. The organizations may not actually even know how the applications work or whether they are still essential, and the people who could answer these questions are often no longer available. Organizations seeking to modernize applications sometimes find the work resembles software archaeology as much as software engineering.

Organizations seeking to modernize applications sometimes find the work resembles software archaeology as much as software engineering.

"A large insurer's 'aha moment' was when they realized that the developers supporting their mainframe COBOL applications were retiring over the next decade. The old batch applications were no longer meeting their needs and were going to become almost impossible to evolve if they did not act soon." (Sreekanth Singaraju, VP of solutions, Alliance Global Services, recently acquired by EPAM Systems)

Horror Stories Abound In Major Application Rewrites

The siren song of modern technology platforms seduces organizations saddled with old applications they no longer fully understand, that depend on aging technology, and rely on expertise they no longer have. Many organizations have stories like these:

- **A manufacturer's warranty management systems consolidation overran budget.** Every country had its own warranty management system, fragmenting the global supply chain and complicating operating processes. As the company simplified its product line, it wanted to do the same for warranty systems. The company chose a rules-based platform, and after six months, it demonstrated a working prototype. While the results were promising, the prototype illustrated that the variations in local processes would make application availability later and more costly than expected. The project was canceled.
- **An insurer spent years and millions but failed to replatform claims management.** The insurer was consolidating claims processing from multiple regional centers to a single central site. They were also consolidating multiple existing claims systems that had grown up with acquired companies. Managing the complexity of many different state rules and stakeholders proved more complex than expected. After over-running the initially projected end date by over a year, a larger insurer acquired the company and the project was canceled.
- **A large government agency struggled to consolidate disparate state systems.** Each state had different requirements, processes, and stakeholders. Lacking a mandate to simplify everything with a single common process, the project became bloated with new requirements and special processing. Cost overruns and delays eventually caused a project reset.

In each case, the organization underestimated the complexity of the rewrite, and the productivity of new technology was overrated. Requirements tended to multiply along with unforeseen technical issues. But it's really not about technology; the main culprit is often simply size.

> "We did an analysis of hundreds of projects over a multiyear period. The ones that delivered in less than a quarter succeeded about 80% of the time, while the ones that lasted more than a year failed at about the same rate. We're simply not very good at large efforts." (AD&D leader, large financial institution)

Incrementally Modernizing Applications Succeeds Where Rewrites Fail

Out of these failures, successful patterns have emerged and organizations have learned a more incremental and continuous approach to modernizing applications. No organization has the time to stop what it is doing to rewrite a business-critical application, and if it's not business critical, why bother? Instead, organizations have found they can incrementally evolve existing applications without rewriting them. Over the past 10 to 15 years, service-oriented architecture (SOA) best practices have helped organizations extend the life and capabilities of their legacy investments, and today's modern DevOps delivery approaches further such success by letting teams work more incrementally.

New Technologies Can Simplify Apps, But Not If One Has To Rewrite The Whole Application

Decades ago, operating systems were rudimentary, databases did not exist, and file access methods like VSAM dominated application architectures. Over time, middleware database and application servers took over data access responsibilities. Java frameworks like Hibernate and Spring further abstracted database and UI interaction, respectively, and applications focused more on solving business problems and less on technology. Over time, increasing levels of abstraction have enabled developers to more quickly deliver applications because they write less code by using supporting frameworks and existing services (see Figure 1). The problem is that using these new technologies in older applications often came with replatforming's steep price tag.

> Organizations modularize for a variety of reasons, but the common theme is faster response to rapidly changing markets, empowered customers, and nimble competitors.

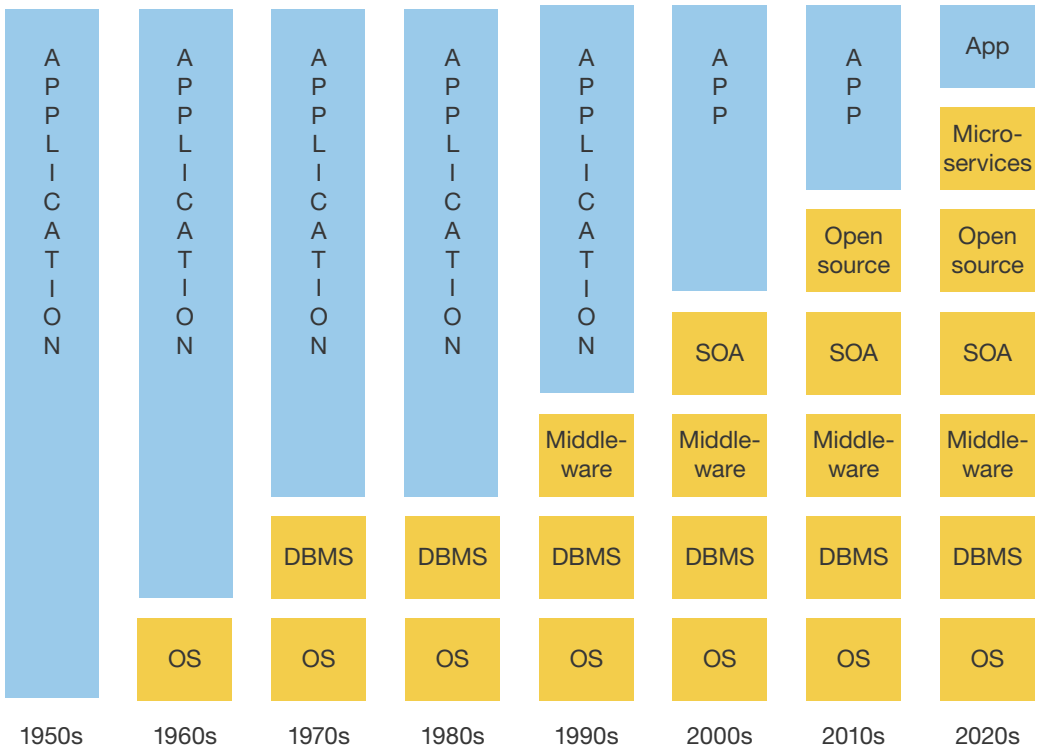
Over time, a variety of forces have aligned to change the possibilities for modernization:

- **SOA laid the foundation for incremental, "virtual" modularization.** Even as monolithic vintage applications remained monolithic, organizations have used SOA to modularize access to their data and functions. SOA services also provided a place to layer in adaptations to vintage business rules

and to provide new capabilities that combined data and functions from multiple vintage applications. Such best practices with SOA have enabled organizations to decouple applications and improve their maintainability ability to deliver and test applications at much higher delivery speeds (while SOA worst practices have caused some to mistakenly think of SOA as a thing of the past).

- **SOAP and, even more so, REST have enabled platform and language interoperability.** Over the years, organizations have primarily used simple object access protocol (SOAP) and REST messaging styles to implement SOA principles. Both provide platform and language interoperability, although REST's interoperability extends further, practically speaking, into JavaScript and other environments important for modern applications.[2] Widespread adoption of these technologies, implemented over the web, has given birth to large numbers of independent, reusable services and APIs.[3] Within organizations, these technologies have increased modularity and flexibility.
- **Microservices increase modularity and incremental deployment for APIs and services.** Microservices benefit Agile and continuous delivery practices by reducing the size of deployment packages, which enables smaller, targeted changes to move to production more frequently — even via multiple intraday deployments.[4] Growing use of microservices — including APIs for mobile and cloud applications — shows the way to greater agility and implementation flexibility for building and deploying APIs and services.
- **Cloud services and open source software offer new code replacement alternatives.** Open source projects and other cloud service providers such as Amazon, Google, Microsoft, and Salesforce have made large numbers of high quality components and services available that facilitate microservice implementation and enable organizations to reduce the amount of code they need to write. This has opened up new possibilities for replacing custom code within applications.

FIGURE 1 Application Modularity Is Increasing



Motivation For Modularizing Varies, With A Business Responsiveness Theme

Organizations modularize for a variety of reasons, but the common theme is faster response to rapidly changing markets, empowered customers, and nimble competitors. Examples illustrate the range of motivations and responses:

- **Aging systems could not support new businesses.** A mortgage insurer found that it could not replace everything all at once because of the size and complexity of the effort but was able to make progress line of business by line of business. Over a period of years, the insurer was able to release in small increments, componentizing common services, standardizing interfaces, then replacing underlying implementations. An ESB connects components as they gradually modernize the application.
- **Major product expansion triggers modular replatforming.** A major multiline insurer was expanding one of its lines of business with new products that the old application could not support. Faced with a significant application overhaul anyway, they plunged in. They reduced the amount of COBOL application code by 70% by moving hard-coded rules into a rules engine where they were

easier to understand and maintain. From there they implemented the new functionality as services in Java, still running the mainframe, but called from the COBOL applications. Now developers use these services in other applications as well. Enterprise architects and developers work together to discover services at the code level then refactor them into separate services. This refactoring reflects a general trend toward more modular applications composed of more specialized services (see Figure 2).

"The main goal was not to achieve reuse. The main goal was to improve the modularity and maintainability of the application. Reuse, if it happens, is a bonus." (Companywide chief architect, insurance company)

› **Acquisitions trigger modularization.** A major US-based manufacturer and retailer adopted an incremental modulation strategy when they acquired four mobile applications companies and found their existing aging, monolithic Cold Fusion application unable to cope with the changes. In its place, they began evolving toward a four-tier SOA application. Similarly, UK-based telco systems provider EE's modularization push was initiated when the company was formed by merging the UK operations of T-Mobile and Orange.[5]

FIGURE 2 Application Architecture Characteristics Have Shifted

Characteristic	Pre-1980	1980-2000	2000-2010	2010+
User interaction	Batch, online	Client-server	Multichannel	
Code structure	Single language, tightly coupled design, single-purpose	Different languages for client and server, tightly coupled design, single-purpose	Different languages for components/ services, loosely coupled designs, decoupled services	Plethora of languages, loosely coupled microservice-based designs, service independence
Data architecture	File system, hierarchical DBMS	RDBMS	RDBMS, non-SQL databases	
Business rules	Hard-coded	Hard-coded plus stored procedures	Rules engine	Encapsulated in services and rules engines

Organizations Incrementally Modernize Their Applications On An SOA Foundation

SOA and API investments are paying dividends by enabling organizations to modernize applications incrementally. Even when a particularly painful legacy application or platform caused heated discussion about how to get rid of it, it's not uncommon for clients to tell Forrester that, after a bit of incremental modernization, the pain and heat subsided. The option of replacing it was taken off the table because the organization found too much value and priority in building atop the legacy to worry with replacing

it. SOA and APIs do not require implementing an ESB — an ESB may help, but instead, the real need is for development teams to embrace service-oriented principles of encapsulation and API-enabled interoperability. AD&D leaders have learned that:

- **Service platforms enable organizations to go faster but require strategic investment.** Organizations don't create a common service platform as an accidental byproduct of some other initiative; project teams have short-term objectives and cannot afford the time or work needed to build assets for broader benefit. Service platforms eliminate waste caused by redundancy, duplication, and inconsistency, and they help new initiatives deliver faster by building atop what's already done. To counter short-term pressures that prevent long-term progress, leaders must sponsor, support, and nurture the platform's formative years.

> SOA and API investments are paying dividends by enabling organizations to modernize applications incrementally.

"Ultimately it was a strategic decision by the CIO that we needed to change. We had the same business logic on five to 10 applications; if a change was missed, it caused a lot of problems." (Todd Piehler, VP of architecture and data services, Fidelity Investments)

- **"Encapsulate, isolate, and renovate" enables incremental replacement of aging apps.** A major manufacturer with retail operations had a 15-year-old application that no longer met its needs. Unwilling and unable to replace it all at once because of the risks to its business, the company isolated the parts of the old application back-end behind REST APIs while it moved the application front-end to node.js. With the monolithic server application encapsulated, the company was able to replace server components at a pace driven by business needs. With increased modularization it is now pushing into greater container use to give them more deployment flexibility. Third parties and vendors are now using the APIs, with the possibility of opening them up further in the future.

"Now we have a team making the centralized decisions about the set of REST services offered by the platform. As part of this, containerized solutions with an API have become a key criteria for purchasing." (AD&D leader, manufacturer/retailer)

- **Hybrid applications are here to stay.** Hybrid applications are composed of a combination of custom code and cloud-based resource, some open and some proprietary. Examples include an insurance underwriting application that makes use of a cloud-based risk rating service or a hotel reservation system that makes use of a mapping service like Google Maps. Applications of all types are becoming hybrid, and the trend will continue over time as organizations add new capabilities or replace existing proprietary capabilities with superior capabilities from third-party cloud providers.

"A hybrid application spans multiple infrastructure and cloud locations, making use of resources and services from each. Components of the application may run on-premises or in the cloud." (Brad Schick, CTO, Skytap)[6]

› **API management enables service use to scale.** Organizations need to treat APIs and the services that realize them like strategic assets; they need owners, budgets, documentation, and agreements about how and when they will evolve and change and regression tests to make sure that changes don't break them. They need curated "stores" where developers can find high quality services that they can try and "buy." "API management is a big deal" says Todd Piehler, VP of architecture and data services at Fidelity Investments.[7]

Organizations need to treat APIs and the services that realize them like strategic assets.

› **Developers have to learn new habits.** Short-term habits die hard. AD&D leaders need to take the long view to encourage developers to reuse before they write new.[8] A major telecom provider does this by measuring developer productivity, using static code analysis that favors using a service over writing new code. Reusing services greatly increases quality and reduces cost and time-to-market once developers adopt new habits.

"It is more work to design and create services than to simply hard-code the behavior. The costs of long-term maintenance are not born by the original development team. It's now paying lots of dividends, but there was a lot of initial pushback. It does slow things down in the short run." (Todd Piehler, VP of architecture and data services, Fidelity Investments)

Recommendations

Fund Application Modernization With New App Delivery

Demand for customer-facing applications tends to crowd-out important but not urgent application modernization. Catching up on long-overdue modernization work means tying it to strategic business objectives. To make progress, AD&D leaders need to:

› **Incorporate incremental modernization into strategic customer-facing work.** Legacy systems enable organizations to serve-up the enterprise data and transactions that bring life to business processes and enable winning customer experiences. Every improvement to a system of engagement is an opportunity to improve the stability and modularity of systems of record. Conversely, every such opportunity that is squandered digs the organization into an ever deeper hole of legacy entrapment.

› **Turn APIs and SOA — especially business services — into strategic assets.** Some APIs are technical building blocks or part of an applications UX layer (e.g., for synchronizing mobile app data), while others provide business building blocks (e.g., submit an order) for consistent customer interaction across touchpoints, processes, and ecosystems. As APIs become the *lingua franca* of

business, they increasingly connect organizations with their customers and partners and enable deeper customer connections. APIs can even open new angles into business strategy and new ways to execute on existing business strategies.[9]

- **Incrementally provision and expand platforms for APIs and services.** Done right, service platforms enable organizations to deliver faster, improve quality, lower cost, and increase application reliability. These platforms can be incrementally built and expanded over time as an organization's maturity and scope of API deployment grows.[10]
- **Replace custom, non-core functionality with standard, open services.** Commercial vendors and open source projects provide organizations with opportunities to replace custom code they have to maintain with world-class alternatives, especially in areas like mapping/GIS, search, credit rating, and financial rating. In addition, available public APIs also provide these and a wide range of other capabilities.
- **Solve scaling problems and close functional gaps with services.** When applications fail to make the grade, don't compound problems with hasty patches. When application remodeling is needed, factor-in time to refactor and improve the application by breaking it into modular services that will be easier to scale and maintain going forward.

Supplemental Material

Companies Interviewed For This Report

3Pillar Global

Accenture

Alliance Global Services (an EPAM Systems company)

Capital One

CAST

Deutsche Bank

Disney

Fidelity Investments

GE Capital

IBM

Skytap

Target Brands

ThoughtWorks

Torry Harris Business Solutions

Travelers Indemnity

Under Armour

Endnotes

[1] For more information on the shifts going on in business applications, see the "Digital Innovation Reshapes The Future Of Business Applications" Forrester report and see the "Don't Just Maintain Business Applications — Raise Business Responsiveness" Forrester report.

[2] For more information, see the "The Dawn Of Enterprise JavaScript" Forrester report.

[3] Effective strategies for SOA and APIs require maturity on multiple fronts. Application development teams and solution architects are challenged to draw a clear relationship between SOA and APIs, and it's even more challenging to craft a strategy for obtaining the highest business value from both. Use Forrester's model of eight central maturity areas — ranging from portfolio management and design strategies to development life cycles and funding — as a foundation for evolving your organization's SOA and API strategies toward greater business agility and value for money. See the "Drive Business Agility And Value By Increasing Your API And SOA Maturity" Forrester report.

[4] With increasing frequency, microservices are appearing in enterprises' efforts to achieve better software architectures, especially among those operating at scale, such as eBay, Google, Netflix, and Wal-Mart. Forrester provides our take on what microservices are and how AD&D pros should take advantage of them in modern applications, their solution architecture, and their integration strategy. See the "Microservices Have An Important Role In The Future Of Solution Architecture" Forrester report.

[5] For more information on EE's journey, see the "SOA Plays An Important Role In A Telco Merger" Forrester report.

[6] Source: Jason English, "Brad Schick on Evolving Complex Enterprise Applications to the Cloud," Skytap Blog, June 8, 2015 (https://www.skytap.com/blog/brad-schick-on-evolving-complex-enterprise-applications-to-the-cloud/).

[7] With its critical role of managing relationships between API users and API providers, API management is one of three major management products to foster mature enterprise API success. See the "How To Manage APIs For Customer Engagement" Forrester report and see the "The API Management Solutions Market Heats Up" Forrester report.

[8] This type of culture change for developers is one aspect of the move to agile-plus-architecture helping organizations achieve sustainable business agility. See the "Best Practices For Agile-Plus-Architecture" Forrester report.

[9] APIs allow an enterprise to go beyond its traditional offerings to pursue new markets and customers by creating new products and services from its assets, data, or processes. APIs can also create new go-to-market strategies and new value for existing offerings, such as targeting customers through influencers rather than targeting customers directly. See the "How APIs Reframe Business Strategy" Forrester report.

[10] Forrester calls out five major elements of a comprehensive platform for APIs and services. See the "A Developer's Guide To Forrester's Strategies For API Success" Forrester report.

FORRESTER®
CHALLENGE THINKING. LEAD CHANGE.

We work with business and technology leaders to develop customer-obsessed strategies that drive growth.

PRODUCTS AND SERVICES

- Core research and tools
- Data and analytics
- Peer collaboration
- Analyst engagement
- Consulting
- Events

Forrester's research and insights are tailored to your role and critical business initiatives.

ROLES WE SERVE

Marketing & Strategy Professionals

CMO
B2B Marketing
B2C Marketing
Customer Experience
Customer Insights
eBusiness & Channel Strategy

Technology Management Professionals

CIO
- Application Development & Delivery

Enterprise Architecture
Infrastructure & Operations
Security & Risk
Sourcing & Vendor Management

Technology Industry Professionals

Analyst Relations

CLIENT SUPPORT

For information on hard-copy or electronic reprints, please contact Client Support at +1 866-367-7378, +1 617-613-5730, or clientsupport@forrester.com. We offer quantity discounts and special pricing for academic and nonprofit institutions.

122550