

LEGACY SOFTWARE MODERNIZATION

A practical guide for enterprises looking
to transform outdated applications



Ed Price
Director of Technology

**ARE
YOUR
APPS
ON THE
BRINK OF
EXTINCTION?**

01

Legacy systems endanger enterprises

02

Identify risks

03

Determine the approach

04

Learn from failure

05

Case study: Transforming an acquired compliance product

06

Modernize: The action plan

07

Invest in products, not projects

LEGACY SYSTEMS ENDANGER ENTERPRISES

In today's digitally driven world, enterprises need software to survive. These systems are the foundation on which our whole economy relies. No matter the industry, all businesses are burdened by and rely upon software that—without proper oversight—become outdated.

Letting software systems languish puts organizations at risk of losing functionality or roadblocking enhancements. Yet, many organizations hesitate to modernize a legacy system. They continue to tolerate an outdated platform because it provides a core operational function—fearing that one false move could take down an entire system (or end careers).

Software doesn't always age well. Old code lacks responsive functionality. Dated UX gives users a bad experience. An absence of accessibility and current features further alienates user groups. Meanwhile, competitors are shipping new product to market, snagging market share.

Legacy systems are not defined by years in operation. They expire because of an inability to meet organizational needs, incompatibility integrating with newer products, or they have been sunsetted by the vendor. Older mainframe and distributed systems often lack modern functionality, code, or interfaces and struggle to sync with contemporary offerings that both customers and employees expect. Software systems are inherently complex—legacy systems more so with years of accumulated debt from unaddressed issues.

Enterprise CEOs charge IT senior management with handling system maintenance and resolving technical debt. While it's common to find workarounds to achieve the desired functionality, the sum of ad-hoc fixes and integrations add layers of complexity and risk to these applications. The cost of maintenance continues to rise over time, and the extra work doesn't address any of the applications underlying limitations. With limited IT resources, a backlog of critical maintenance for legacy systems quickly impedes the ability to deliver enhancements or new products for the business.

Capturing scope to remedy the issues is difficult with decrepit business logic buried deep in dated code. The people with domain expertise are often gone, crippling the company's ability to move quickly to resolve critical operational issues effectively. Just keeping software viable can require more effort and investment than replacing the outdated components. Legacy systems need maintenance (e.g., complex patching and alterations) to sustain functionality.

Replacing a monolithic application at any enterprise is a highly complex undertaking, one that is often misunderstood, underfunded, and at odds with the priorities of the business.

Fifty percent of digital leaders struggle because legacy systems grow in technical debt. Years of operating businesses successfully require changes. Ultimately changes, hacks, and lack of support prevent enterprises from quickly creating business value for their customers and keeping operational expenses in check.

- Harvard Business Review

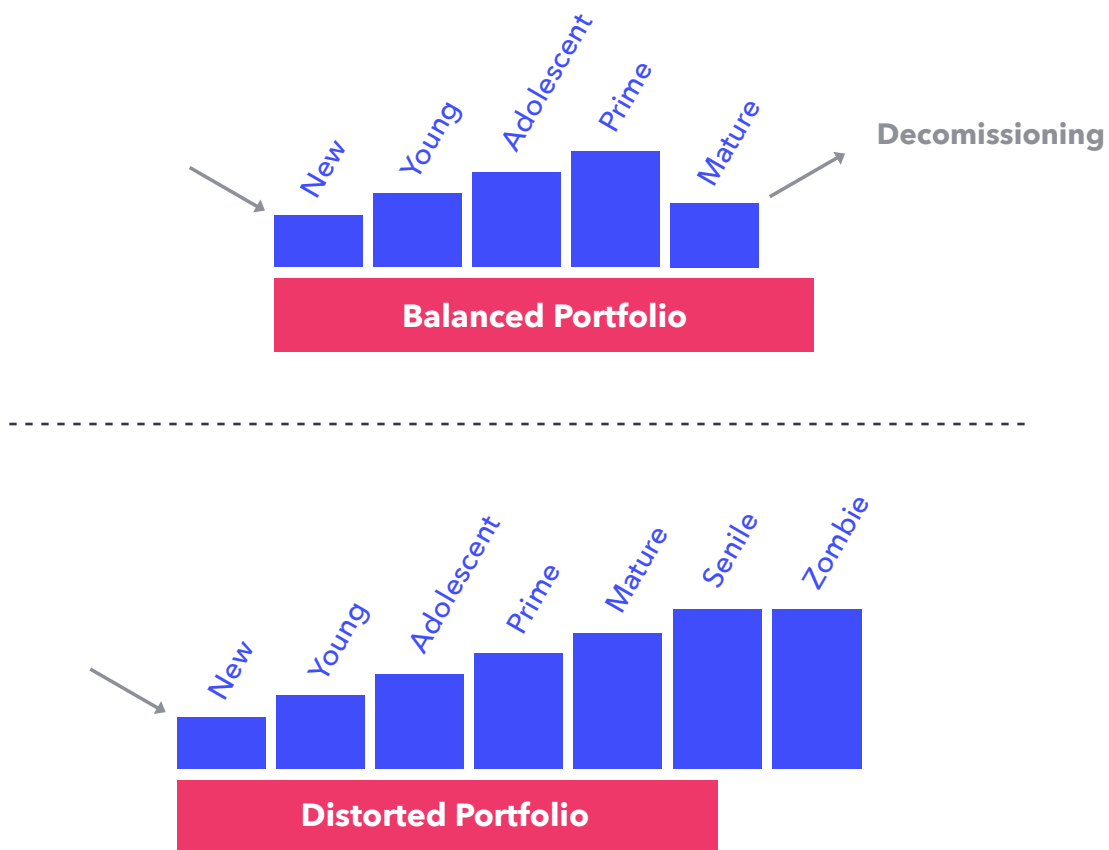
Despite mainframe green screens, brownfield dated code, or out-of-support systems, companies marred by legacy issues can still compete. Fixing aging systems takes courage, calculated risks, and an iterative approach in planning, funding, and execution. Untangling the issues of an aged system and modernizing can significantly shift the trajectory of organizations for the better.

With tech continually advancing, companies need to keep close watch over systems to ensure they operate effectively. Systems require continuous funding and support to remain viable. Reliable teams need to be in place to maintain the product, release new features, and implement integrations. This white paper covers the inherent risks of legacy applications, varied approaches for transformation initiatives, and the step-by-step action plan for companies to modernize legacy systems.



IDENTIFY RISKS

Most global 2000 companies have a somewhat distorted view of the applications in their portfolios. As shown below, Gartner's report, "Decommissioning Applications: The Emerging Role of the Application Undertaker," notes that the distorted application portfolio contains many applications that should have been decommissioned. These risk-averse companies downplay the products in a senile or zombie state. Rather than live in denial, take time to understand the legacy system issues and the corresponding risks influencing modernization efforts.




SOFTWARE ROT

Like a fresh bowl of fruit, software rots the longer it sits dormant. Technical debt (e.g., inflexible, proprietary, buggy, unsupported, closed systems) left ignored causes products to decay. Many organizations fumble along with incompatible hardware, operating systems, or integrations. Consequently, the company's operations experience a loss in productivity. This instability rips through the enterprise forcing many to pay a premium for adequate support to fix escalating and compounding issues.

HIGH MAINTENANCE COSTS

Maintenance, manual testing, environment management, and deployments eat away a large part of IT spend. Legacy software becomes more of a cost center vs. a revenue generator. Some companies allocate ample IT funds to merely keeping the system viable, rather than investing in modernizing to meet the evolving business needs. This approach to funding affects business performance and revenue adversely.



In the typical agency, between seventy-five percent and eighty percent of the IT budget was spent on operations and maintenance of legacy systems that are rapidly becoming obsolete.

SLOW SPEED TO MARKET

Quarterly features/releases are no longer the industry standard. Companies slower to respond and repair inadequate products are especially vulnerable to competitors. Older software lacking modern amenities with poor usability and user experience further jeopardize enterprise operations and customer retention. Businesses need to remain nimble to respond to market pressures, and ever-changing high customer demands fast.

COUPLING TOO TIGHTLY

Pre-cloud, legacy systems were built as a large, tightly coupled, self-contained, monolithic applications. Today, modern tech uses a service-based architecture with loose coupling and cloud-hosted container-based microservices as the gold standard. Organizations with older, self-contained products need to create less dependencies and more autonomy in systems or components of systems using microservices, multi-cloud containers, and self-healing.

SECURITY BREACHES & DATA LOSSES

Lack of proper evergreen investment in systems with years of focusing on maintenance and not keeping up with current versions, using patches, and running unsupported software creates security holes in software negatively impacting enterprises and customers. Open source software, purchased packages, unsupported/EOL tech stacks like .NET 2.0 or Java 6 present vulnerabilities. For example, Experian and Capital One both experienced breaches because each lacked current versions, causing the enterprises irreparable reputational damage and monetary losses. With new GDPR in EU and CCPA in California, businesses face stiff penalties for not adhering to data protection and privacy standards.

INVESTING IN NON-DIFFERENTIATORS

Systems or services that do not influence critical business operations, add significant value to customers, or drive a main source of revenue are all non-differentiators. Companies, especially those trailing in performance, put themselves at even more risk by investing heavily in non-differentiating activities vs. modernizing core products. Prioritize modernizing core systems before considering an innovative business-style investment. For example, PNC Bank is now poised to react to customer needs and market drivers quickly because of a year-long BIAN.org Open Banking API initiative.

TALENT DETERRENT

Outdated tech with green screen mainframes or Internet Explorer 7 or 11 handicap businesses from attracting and retaining talent. The best and brightest people expect to work for forward-thinking companies using top-level, innovative systems. Modern tech enables employees to focus on creating business value instead of decoding three-character acronyms, being locked to a desktop computer, and using antiquated technology stacks.

INSUFFICIENT DOMAIN EXPERTISE

Working with legacy systems requires domain knowledge from workers. As time goes on, key employees with this tribal knowledge leave the company or retire. In response, the organization recruits and onboards new resources to resolve longstanding IT

problems. The new crop of talent lacks proper documentation chronicling the older system and attempt workarounds or hacks made to evolve the system over time. The team burns time, energy, and budget to understand the legacy application and implications of modifications before starting to resolve issues.

REDUCED CAPABILITIES

A failure to upgrade to most recent versions of libraries, products, and integrations creates technical debt exponentially. The technical debt grows causing security vulnerabilities, slower performance, dependencies that don't need to exist, and the inability to take advantage of newer software. Workarounds barely keep old software afloat with custom integrations and patching—bleeding the IT budget dry, generating limited results, and burning out resources.



DETERMINE THE APPROACH

Many organizations and applications suffer from one or more of the risks previously mentioned. When faced with multiple risks, prioritize the biggest barriers above others. From there, figure out the best strategy to resolve the legacy issues.

The US government spends a reported \$80 billion, nearly three-quarters of the annual federal IT spend, maintaining and operating legacy systems each year. Concerned about NASA's burgeoning obsolete tech, in 2016, Congress asked the space agency to detail plans for modernizing mission-critical systems, including:

- Identify the top three critical legacy systems.
- Share detailed replacement plans.
- Monetize the projected spend to run the systems.
- State the year it began using the oldest programming languages.
- List the size of legacy codebase.
- Quantify the staff fluent in each code language.

Two systems noted in the response dated back to the 1950s. As of 2020, NASA is still executing the plan.

Learning from the NASA story, take inventory of what's outdated and identify the top three critical legacy systems. Then, determine which systems offer the least risk or the fewest blockers. Based on our experience, here are five ways to approach modernizing.

CLOUD MIGRATION

Many IT leader's and team's workload involves migrating an existing application to the cloud. Does migrating to the cloud add value? Absolutely. Is it the key to fixing legacy issues?

No.

Typically, moving to the cloud involves a large number of resources struggling to manage problematic on-prem environment maintenance with workarounds, which causes slow testing and delivery mechanics. Offering cloud alone does not resolve core legacy issues of an inflexible architecture, inaccessible business functions, or need to scale. The application, flaws and all, just relocated.

The execution: Treat cloud migration as the first step in a multi-phase effort to:

- Provide cloud at scale to an otherwise unscalable application in the short-term.
- Allot time for a proper refactoring effort or replacement software.
- Open the opportunity for a new tech stack and support options.

IN-PLACE UPGRADES

Systems not running the current version risk security holes (e.g., CVEs), support, and overall system fragility issues. Legacy systems older than N-2 or N-3 require dependency software such as Java 6 or .NET 3.5 to be loaded on workstations or hosts. Older software applications with dependencies chance conflicting with newer versions or operating systems and polluting other applications in the organization's ecosystem. Getting an older system updated to a supported version is an important step to address compatibility and usability issues of legacy applications.

The execution: Migrate to a more current version still supported by the vendor and compatible with any dependent products involved in the solution. Understanding the implications and roadblocks of performing in-place upgrades to current tools helps IT leaders map out the next move with minimal disruption to the organization.

REARCHITECT

Rearchitecting is a lot like renovating an older home. The builder decides to tear the home down to the studs, repurpose the foundation or some existing elements with long-term value, and rebuild the remainder of the home using contemporary measures. Similarly, rearchitecting software repurposes good patterns when possible and rebuilds using modern standards. This approach enables enterprises to exploit the enhanced capabilities of a new tech stack.

The execution: Rewrite code and functionality, shifting to a new tech stack. Rearchitecting requires an entirely new blueprint to show the builders how to approach modernizing the system. Like renovating a home, refine the work required by targeting specific areas to address.

REFACTOR

This technique alters the existing code structure while keeping the application's behavior intact. A common pitfall of refactoring is stakeholders expecting immediate feature parity. The desire for full parity often results in failure. The scope gets too wide, the work takes too long and costs too much—leaving business partners unsatisfied. Manage refactoring work closely with strong teams at the helm to effectively resolve technical debt, improve features, and scalability. Build alignment from the cross-functional team and stakeholders to determine if the features being rewritten are still valid and necessary before executing. Prioritize the timing of feature delivery and develop a clear roadmap to manage expectations and results.

The execution: Create a service blueprint to map out the current product and what needs improvement. Take the existing application and break it down into bite-size increments, then optimizing existing code without changing external functionality. Slowly refactor the code aspect until complete.

REBUILD OR REPLACE

When an application contains elements worth salvaging, rebuild and rewrite code, while still preserving scope and specs. In other words, rebuild a pre-existing solution, leveraging the knowledge gained from the legacy software. If nothing's worth saving, retire and replace the system—either building or buying something entirely new with clearly defined requirements. Both initiatives include additional barriers and complexities impacting the decision (e.g., team composition, processes, technology choices, and scale of investment).

The execution: Decide whether to build new, buy on off-the-shelf, or customize an existing product. Each option relies on deep knowledge of business processes. Use story mapping to define requirements of the new product and understand the outcomes the business wants. Add a service blueprint to document any viable elements worth repurposing.

Don't risk losing critical information. Include legacy data migration as part of any change.

04

LEARN FROM FAILURE

Companies make mistakes. Projects fail. Carnegie Mellon offers the top ten reasons why modernization efforts fall short in "Why engineering projects fail." To avoid repeating mistakes already made learn from them.

- 1 The organization adopts a flawed or incomplete strategy.
- 2 The organization relies too heavily on outside consultants.
- 3 The team is tied down to old technologies and inadequate training.
- 4 The organization falsely thinks a legacy system is under control (when it isn't).
- 5 The organization's needs are oversimplified.
- 6 The overall software architecture isn't given enough consideration.
- 7 There is no defined application modernization process.
- 8 There is inadequate planning and follow-through.
- 9 Leaders lacks long-term commitment to the strategy.
- 10 Leaders pre-determine technical decisions.

TRANSFORMING AN ACQUIRED COMPLIANCE PRODUCT

THE COMPANY

A global organization with a longstanding history as one of the world's leading B2B data and content providers is delivering solutions to banks and businesses for over 180 years. As the company continues to expand, they acquire multiple companies in the compliance and financial services industry. The effective integration of these new businesses requires a specific skill set. The product maturity and technical debt from these acquisitions introduce significant risks.

THE CHALLENGE

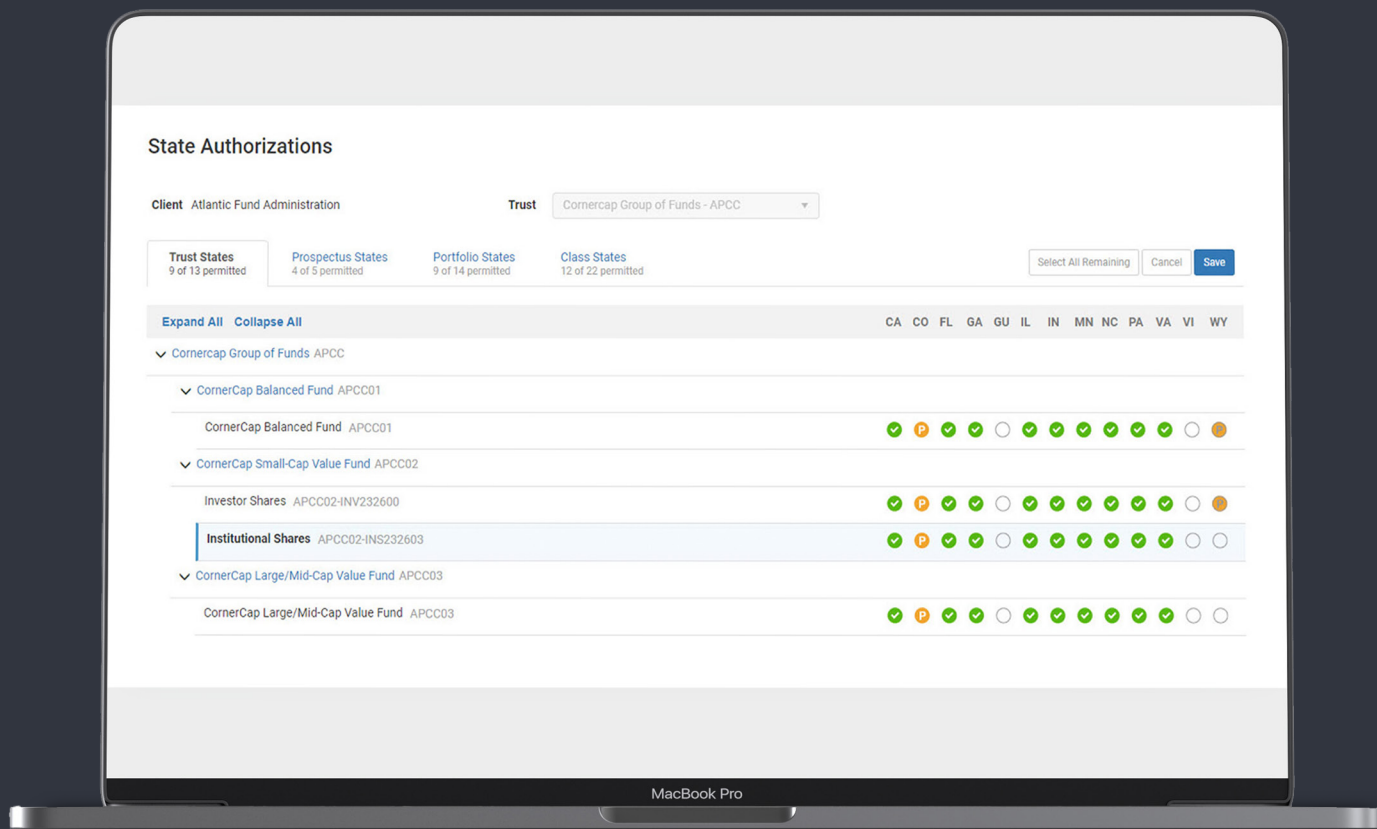
Devbridge was called to recover a struggling product rebuild effort and deliver to market—stemming from an acquisition of a company in the compliance space. The challenge at hand...save a 20-year-old desktop app that had begun a SaaS transformation previously handled by internal teams, independent contractors, and an offshore agency. Mounting complexity and nuanced per-state filing requirements resulted in unmaintainable business logic and four years of investment without working software. With frustrated executives, exhausting budget, and severely delayed timeline, the client was anxious to demonstrate immediate results and justify the necessary spend to productize the new platform.

THE APPROACH

We came in with a laser focus on getting the product into users' hands fast. We used an integrated approach, blending our teams with the client's globally distributed teams. Quarterly roadmap planning sessions rotated through all key stakeholder locations with engineering, design, and product management attending in person.

Scrum rituals were strategically scheduled to account for different time zones and facilitated in person—as well as virtually. To effectively reduce risk, the first scrum teams were blended Devbridge and client-side engineers. At scale, three Devbridge teams, the client team, and a pool of domain experts used dual-track scrum to parallelize the workstreams and gather just-in-time requirements within each sprint.

Our team of product managers and product designers extracted user stories from the business through intense daily discovery sessions with client SMEs and users. Our engineering team evaluated and refined technical architecture. A healthy CI/CD pipeline, automated testing, and clear acceptance criteria allowed the team to accelerate UAT before making a release. The first milestone was to release the product to an internal audience as a pilot.



THE RESULTS

We recovered the failed build and delivered a comprehensive data and software solution to control risk and compliance. The product adhered to Blue Sky Law (state laws in the U.S. that regulate the offering and sale of securities ostensibly to protect the public from fraud). The solution reduced compliance costs by:

- Monitoring and management of sales activities
- Electronic filing of products
- Filing renewals for existing products
- Expense management and reductions in costs

Devbridge released the initial version within a year and public release in another six months. Our iterative, agile process uncovered recommendations for further enhancements in future sprints.

MODERNIZE: THE ACTION PLAN

With a solid understanding of the risks involved and varied approaches, set out to modernize. While rebuilding takes time, making this move can significantly alter the future for organizations. Below is an action plan for modernization initiatives leveraging Devbridge's proven-best practices working with Global 2000 enterprises across multiple industries.

DETERMINING THE EFFORT

Decide what type of effort offers the most benefits for the business. Focus on outcomes over feature parity. Rather than trying to identify and build everything based on guesswork (i.e., what you think your customers may need), prioritize what's relevant to customers noted in user research or the current business model. Evaluate the entire system and replicate select elements with clarity on all the features known or unknown.



Outcome

Timeline Funding

Challenges Risks (business) Alternatives

Integrations

Systems

Dependencies

GETTING STARTED WITH A LEAN REQUIREMENTS WORKSHOP

The workshop is designed to challenge the team to think critically about the best digital product for the enterprise. Gather stakeholders together in a room for 1-2 days. The Devbridge team facilitates activities.

- Users, roles, and goals
- Story map or service blueprint
- Risks and impediments
- Technical feasibility

At the end of the workshop, the team is armed with critical assets informing the work.

SIX KEY OUTCOMES

1 Defined business goal

A clearly defined business goal with success metrics

2 MVP requirements

The minimum amount of requirements necessary to kickoff the design process

3 Hidden requirements

Bringing people from different functions together to uncover what impacts goals, scope, and priorities

4 Shared understanding

A shared understanding of the business process, end users, and their pain points

5 Scope & priorities

An agreement on scope and priorities to meet the business goals

6 Product release strategy

A phased approach to releasing your product to market

WHEN REPLACING AN APPLICATION: CREATE A STORY MAP OF AN ENTIRELY NEW SYSTEM.

Stakeholders jot ideas down on post its and present ideas, discussing each concept with the group. The dev team arranges the epics, features, and stories into the story map. Epics and features flow from left to right in a logical user path (e.g., searching, checkout, and then booking). Arrange the stories under each feature in a logical flow to visualize the entire scope of work.

As a reminder, don't rely too heavily on feature parity. Yes, there is some value in flagging the options no longer needed and old functionality to avoid potential failure. However, now is a prime opportunity to redefine the software synced to the company's current and future needs. Focus on delivering outcomes. Be deliberate and work with stakeholders to only identify features that benefit customers or business needs.





WHEN REBUILDING AN APPLICATION: CAPTURE THE AREAS REQUIRING REWORK IN A SERVICE BLUEPRINT.

A service blueprint provides the entire team insight into the business systems, third-party integrations, and manual steps featured in the legacy systems relevant to customers and back-office personnel. Break the monolithic legacy system into domains, prioritizing each to inform the initial iterative component build and release of the new system. Highlight specific pain points as well as low-risk opportunities to set aside for future workshops, estimate, and build. Capture risks, impediments, technical feasibility, and priority items.

DEFINING SCOPE

After the workshop, estimate the effort based on business goals, story points, and complexity factors. Determine the drivers influencing the work ahead (e.g., the cost and time).



Increase the team size or run parallel teams to hit the product release date or expedite or larger system efforts.



Keep the team size smaller and spread the development across several, incremental releases to pace funding better aligned with the business or access revenue from new features support further investment.

Sometimes a workshop is not enough to inform the initial build. If there are still unknowns, difficulty evaluating risks, or in need of additional information, add a discovery phase to better define the work, including:

- Heuristics
- User research
- Competitive analysis
- Ideation
- Service design
- Technical spikes
- Prototypes

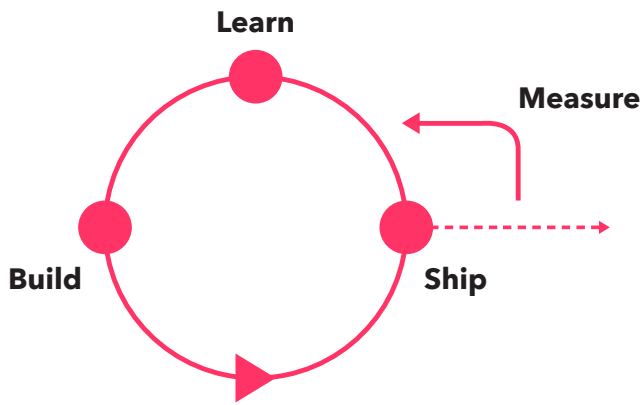
ESTABLISHING SUCCESS METRICS

Craft metrics to measure and monitor success. Set specific, measurable, achievable goals. During each sprint and release, validate the approach against the established metrics. If running off-course, reset and set new success metrics based on the company's current state, future state, or market conditions. Partner with consultants to support teams needs to scale up or down based on demand, provide specialty services, or fill technology knowledge gaps.

BUILDING AND DELIVERING

Rather than trying to modernize the entire application with a big bang, use an agile approach—breaking legacy monoliths into components and services. Reference the workshop assets, like a service blueprint or story map, to inform the product roadmap. Focus on building high-priority elements.

Demine which aspects to work on and in what order. Run daily standups to ensure teams remain aligned during execution delivery cycles. Slice the workload into manageable increments with two-week sprints released every two to three months—working toward a twelve to eighteen-month final delivery. Leverage an iterative build-measure-learn approach to evaluate success after each release and reincorporate learnings from recent releases back into the product. Once all components are assembled and interconnected, remove and replace the legacy system.



MODULAR MODERNIZATION

As a foundation, a modular approach enables organizations to manage applications at an individual component level, relying on the system's data as a building block. To implement, build, test, and run applications in a virtualized environment. Modular builds include a set of connected micro-applications or microservices along with corresponding compartmentalized domain data. Start with the least dependent components of the legacy application identified in the story map or service blueprint. Then, stitch the components together, creating a way to scale pieces of the work banking on successes or quickly responding to failure.

ANALYZING THE OUTCOME

User research, lean requirements, and dual-track scrum enable teams to deliver fast. Pivots throughout delivery are possible due to the nimble process. Agile, cross-functional teams own the product backlog, user research, design, testing, development, and release strategy.

Unlike building a widget, digital product requirements often shift throughout delivery. Markets change. Stakeholders view working demos and raise questions. Supplement initial bias with user research.

Work with stakeholders to remedy concerns and review analytics to identify features to retire or add. Maintain the underlying software and libraries and continue to foster continuous release cycles using DevSecOps. Dual-track scrum allows the team to react to changing requirements with minimal rework and ship a product faster.



INVEST IN PRODUCTS, NOT PROJECTS

Don't risk reverting to an outdated, risk-riddled, rotted state. Instead, provide ongoing support and maintenance to keep systems viable. The premise is rather simple.

- 1 Research
- 2 Design
- 3 Build
- 4 Maintain
- 5 Support

Keep applications evergreen through a steady stream of features releases and fixes. Agile alone is not enough. To maintain and support the product, fund the change at the product level instead of a single project. Adjust the investment model to accommodate ongoing oversight—continually investing in products over time.

Adopting a product over project mentality at all levels of the organization is key to successful legacy modernization efforts. A product mindset challenges the team responsible for the application to think about the long-term goals in small iterative pieces. Evaluate the product against

established metrics within the build-measure-learn loop, empowering the team to learn from and celebrate successes—as well as quickly react to failure. Build a culture around products should not end after the first release. Keeping systems modernized with continued support and maintenance enables enterprises to respond to market conditions and business needs quickly.



AUTHOR

Ed Price, Director of Technology

Ed has 25 years of experience working in IT and development for organizations such as JPMorgan Chase, BMO Harris Bank, U.S. Department of Homeland Security, and Fonts.com by Monotype. He is a frequent contributor to Devbridge.com.

OUR EXPERIENCED TEAMS DELIVER SOFTWARE **4X FASTER** THAN THE INDUSTRY AVERAGE.

We are a full stack research, design, and software delivery company. Our teams facilitate a creative, iterative process, powered by some of the most passionate individuals in the industry. We build extraordinary custom applications that solve complex problems and deliver measurable results.

Our scalable cross-functional product teams and tailored services transform businesses in agriculture, aviation, financial services, fraud & forensics, healthcare, hospitality, logistics, and manufacturing.



Mobile applications for B2B and B2C

We create elegant, intuitive mobile applications to enhance the customer experience.



Product research and design

We build custom applications that solve real business problems to drive measurable value.



Scaling product delivery

We go beyond team aug and dedicate full cross-functional teams to scale feature delivery.



Data strategy and microservices

From dashboards to master data strategy, we can help you leverage your data to grow your business.



Legacy software modernization

Replatform aging systems and legacy architecture to meet the needs of your business now.



Workflow optimization

Streamline your business processes to reduce time, costs, and improve productivity.



**FASTER THAN
YOU'RE USED TO.**

**getstarted@devbridge.com
312.242.1642**