



Systematic IT Modernization

Strangling the Beast

Transforming legacy systems & optimizing for
fast flow business enablement

Shawn Davison
Managing Partner

The Challenge To Modernize

Businesses today are often under extreme pressure to transform their information technology. There is pressure to become more nimble, more entrepreneurial, and to incorporate the rising wave of digital capabilities. Sales and Marketing consistently raise the issue that their ability to engage and service customers is being impaired by the aging capabilities of the company's technology infrastructure or foster the perception that IT cannot move fast enough. Moreover, Management faced with revenue objectives increasingly demand hyper-responsiveness to rapidly changing market conditions.

Standing still and resting on past brand success and legacy systems is no longer acceptable. Often, the cost to maintain aging infrastructure is escalating yearly and becoming an inordinate portion of their budget. Emerging technologies such as Mobile, Cloud, IoT, and Analytics lower barriers to entry and enable new market entrants to leapfrog incumbents at alarming speed. Navigating the path to modernization

Driven by this exploding pace of competition and innovation, IT and development leaders struggle with a rising critical consideration —how to modernize underperforming capabilities.

can be difficult and complex. Within the IT group, leaders confront numerous obstacles to transformation.

Another challenge is that the composition, culture, and capabilities of the development team may be out of sync with current industry practices, restricting their ability to respond to the evolving needs of the business and the customer. In light of these conditions, most executives grasp the imperative of modernization. The challenge is tackling the problem with the most efficient approach.

The Path To IT Modernization

IT Modernization is fundamentally the process of systematically retooling infrastructure + pragmatic software refactoring of legacy systems, to realign them with business needs.

There are multiple ways to transform a legacy system, including a wholesale replacement or a software rewrite. The challenge with wholesale replacement is that valuable business rules and operational knowledge may be lost in the transition. Further, IT leaders may have experienced a painful wholesale replacement due to significant customization, because in retrospect, the software did not meet key business requirements.

The alternative perspective is that custom software development or a rewrite of a proprietary legacy system is the best way forward. However, not all software refactoring methods are created equal.

Frequently, IT teams embark on a major software rewrite by employing a monolithic approach that relies on extensive upfront planning, minimal early stage feedback on assumptions, and extended timeframes between product deliverables. This approach typically results in miscommunications, cost overruns, and failed expectations. In our experience, a more effective approach to software modernization is "systematic refactoring".

The Components Of Systematic IT Modernization

Systematic IT modernization starts with the premise that the most performant IT groups function best in a highly adaptive learning environment. This means minimizing upfront planning and focusing on defining and building small, incremental yet viable pieces of functionality. In this way, features can be quickly and inexpensively tested with the customer, thereby validating assumptions, reducing overall project risk and allowing the team to rapidly learn and adapt. One of the best known strategies for incremental, systematic refactoring of legacy software is often referred to as strangulation or "strangling the beast" based on work done by Martin Fowler.

Strangler Architecture Evolution

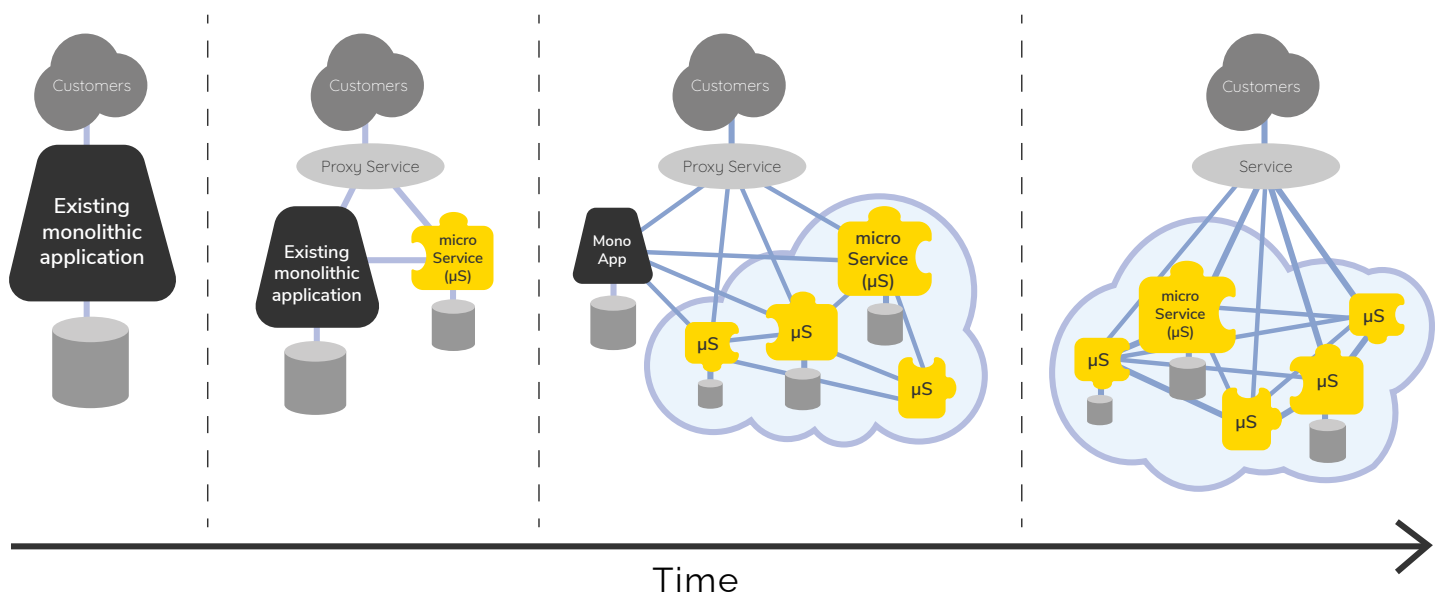


Fig. 1: Martin Fowler first introduced the concept of a Strangler Application aka Strangler Pattern in 2004. Since then, it has been used widely as a pragmatic and cost effective method for legacy software transformation!. Modern proxy services enable the abstraction and management of legacy and modern microServices. (<https://martinfowler.com/bliki/StranglerFigApplication.html>)

How To “Strangle The Beast”

Systematically refactoring a legacy environment optimizes project success by allowing the IT team to confidently lead transformation and deliver business value in a timely manner. Our approach to legacy strangulation leverages four key components: a transformation strategy, the right team topology, a continuous delivery pipeline, and modern development tools that enable Automation.

Transformation Strategy (Strangler Architecture Pattern)

Major software rewrites fail most often due to project cancellation risk, which can usually be attributed to underestimating effort and the resulting cost overruns. Agile + LEAN + DevOps principles and practices go a long way to alleviate this risk; however, it's not enough to ensure success when replacing or rewriting a legacy system. A transformation strategy is necessary to enable an organization to go from Point A (project start) to Point B (project completion) with predictability. At DevIQ, our primary legacy transformation strategy is an adoption of a mature concept and pattern called the Strangler Architecture.

The premise of the Strangler Architecture is that the legacy system is complex enough

that it needs to be split up into parts and migrated over a prescribed period of time--enabling the business to continue to operate as needed. The bigger challenge in this approach is knowing where to “cut” a monolith to create “seams” or appropriate scope and interfaces. Modern proxy service technologies, such as Envoy Proxy (<https://www.envoyproxy.io/>), enable abstraction and management of the parts.

Glossary

DevOps (Development + Operations)

A culture and practice that integrates software development and IT operations activities while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably

CI (Continuous Integration)

A software development method where team members integrate their work on a continual basis. This accelerates deployment and ensures that all developers are working with the most up-to-date iteration of the code. Builds are verified using automated testing to detect integration errors as quickly as possible.

CD (Continuous Delivery)

A modern method of deploying software into production or into the hands of users safely and quickly in a sustainable way.

The Right Team Topology

How you organize your teams to execute on software development greatly impacts the outcome and ability to create “fast flow”. Team Topologies principles, based on Conway’s Law have become the de facto standard for creating high-performance software delivery organizations.



Fig. 2: Team Topologies (<https://teampologies.com/key-concepts>)

The separation between product-oriented Platform teams and Stream-aligned teams has become particularly important in reducing dependencies and increasing feature velocity.

DevOps is not a project or technology; it is a cultural movement for enabling continuous improvement.

Integrating DevOps practices into a legacy environment can be a significant challenge. Changing the culture is as important as the process and technologies. Selecting and cultivating the right team structure is critical to success (<https://web.devopstopologies.com/>).

Continuous Delivery Pipeline (Agile + DevOps Culture + Tools)

Continuous Delivery embraces the integration of multiple software creation trends, including LEAN Agile methodologies and DevOps. Agile software development methods are not new but they have evolved as practitioners applied LEAN approaches to operations work.

Agile has become the prescribed method for software development for most modern organizations. However, in many cases these organizations are implementing a hybrid approach, commonly known as “Water SCRUM Fall”. Gated upfront requirements and estimation processes, in combination with siloed production and release operations, have effectively stifled the vision of what Agile promised, resulting in similar costs to pre-Agile methods and marginal benefits in some cases.

LEAN extends the Agile philosophy of short development cycles to the practice of experimenting with small, viable product deliveries. This is especially effective when applied to legacy refactoring projects.

DevOps Culture builds on the value of collaboration between development and operations staff throughout all stages of the software development life cycle.

Continuous Delivery Pipeline

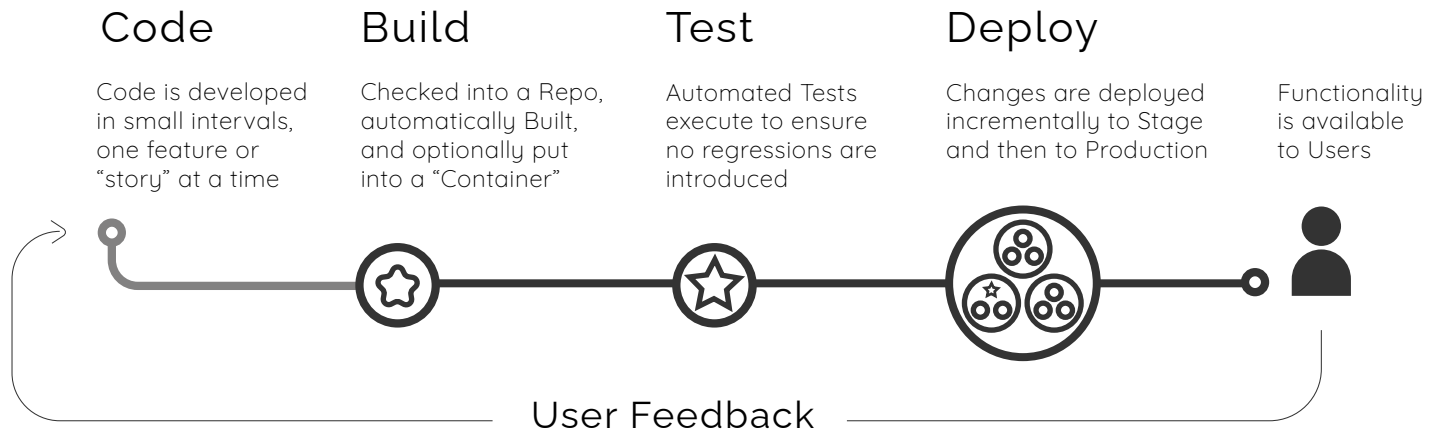


Fig. 3: DevIQ Continuous Delivery Pipeline (simplified).

One of the primary tenets of DevOps culture is "sharing information proactively". Collaboration tools such as Slack and Teams have accelerated this cultural movement.

Automation is the systematic process of refactoring software development, test, and deployment systems to reduce delivery time. Also known as Continuous Integration / Continuous Deployment (CI/CD), automation dramatically improves software quality, in combination with Test Driven Development (TDD), by replacing repetitive tasks that no longer have to be done manually, thus reducing churn and increasing speed-to-market. By incorporating automated tests in the CI process, regression bugs—defects that are introduced in previously working functionality—can be reduced significantly if not eliminated.

The resulting Continuous Delivery Pipeline enables faster Mean Time to Repair (MTTR), which is more relevant today than traditional Mean Time Between Failure (MTBF) metrics.

The combination of LEAN Agile + DevOps Culture + Automation Tools creates a partnership between all relevant business owners including Product Management, Developers, QA, and most importantly both internal and external customers. The result is a culture and practice of operations and development working closely together throughout the entire refactoring lifecycle. This includes iterative development cycles and interactive communication during design, development, Continuous Delivery, and production support. Figure 3 illustrates key components of a Continuous Delivery Pipeline.

Modern Development Tools

What makes a systematic IT modernization or refactoring approach even more attractive in today's business climate is the powerful combination of a Continuous Delivery Pipeline with the latest generation of development technologies. State-of-the-market tools and Cloud platforms can deliver a whopping 10x or more improvement in terms of transforming legacy applications into market leading capabilities, productivity and performance. Utilizing proven leading technologies ensures the longest possible product life, as well as providing a platform that

can be responsively enhanced as market conditions change.

Continuous Delivery Framework (Structured Process + Proven Tools)

How do you pick the right tools? There are so many tools available now, that it's difficult to keep up and know what is best for solving the problem at hand. Open source software has become ubiquitous, and the pendulum has moved from concern of use to critical requirement. The challenge is staying current and having a strategy for retooling in this ever-accelerating world. Ideally, what is needed is a transformation framework that transcends the current specific technology — enabling a way to systematically retool the development

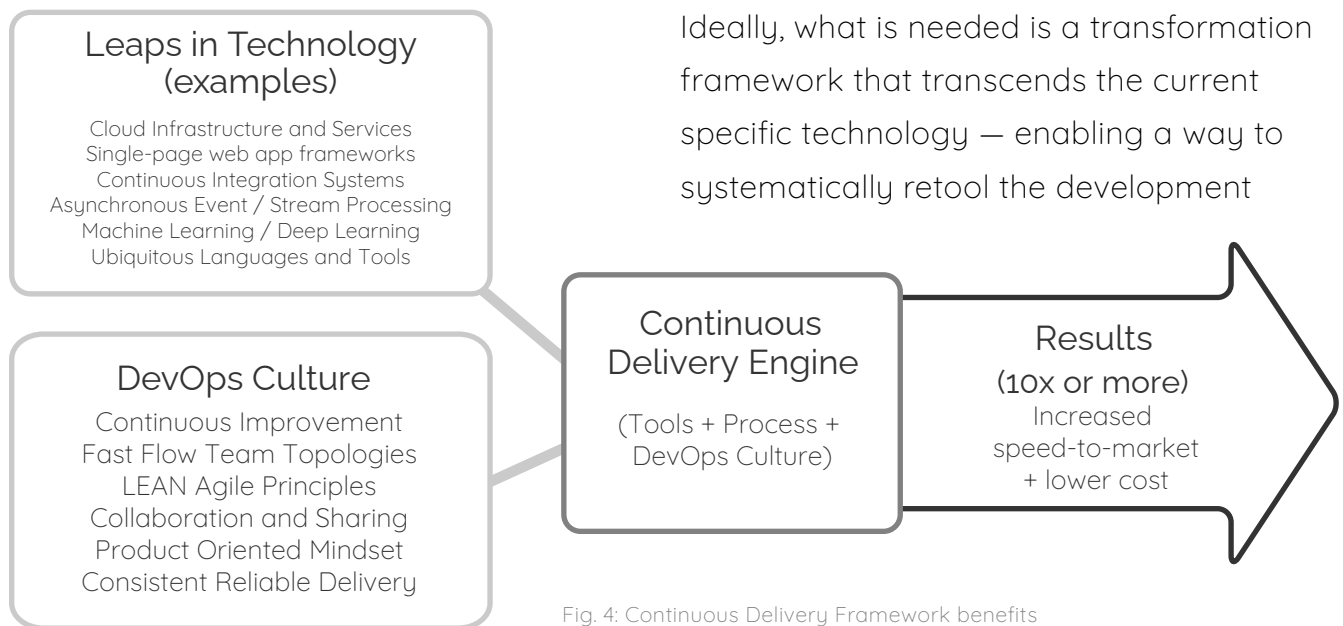


Fig. 4: Continuous Delivery Framework benefits

Side Benefits of Continuous Delivery:

- Supports Test Driven Development (TDD).
- Supports Serverless or Containerization deployment architectures.
- Enables migration from Monolithic → microServices (decomposition).
- Helps remove organizational silos, empowering teams to own their Service from Code to Production.
- IT Culture improvement — provides better accountability and ownership in delivery.

Glossary

MVP (Minimum Viable Product)

The scope that provides just enough business value to deploy to users/customers, yet does not add unnecessary features before getting the product out and receiving real user feedback.

TDD (Test Driven Development)

A process of developing software where requirements are used to build specific test plans and test cases so that the solution directly addresses all key requirements. Automated Unit Tests are built into the codebase from Day 1.

Containerization

A lightweight alternative to full machine virtualization that involves encapsulating an application in a container with its own operating environment.

microService

A software design technique that enables applications to be deployed as independent and discrete services. A microService generally follows the guidelines for a 12-factor App (<https://12factor.net/>).

Serverless Computing

A method of providing backend computing services on an as-used basis. The hardware is completely abstracted from the consumer and the services are charged based on usage during the execution of code vs. uptime.

Single Story Sprint

An Agile development practice enabling rapid deployment of a new feature, bug fix or unit of functionality all the way to Production, in minutes or hours vs. weeks or months.

process, just as business systems must transform to meet ongoing business needs.

A Continuous Delivery Framework is the glue that brings process and technology together in a cohesive way. It is designed to be deployed on highly scalable cloud infrastructure, and may be used for developing publicly accessible product or private/internal systems. It must support both new development and ongoing maintenance and feature development. The result is that it accelerates feature velocity, automates deployment and reduces overall delivery costs.

Transformation is everybody's job.
- W. Edwards Deming

Where To Draw The MVP Line

Many organizations have become familiar with the term Minimum Viable Product (MVP); however, few know where to draw the line. MVP is the minimum functionality that can be developed before “Single Story Sprints” can effectively be implemented. A Single Story Sprint is a small feature, bug fix or unit of functionality that can be deployed all the way to Production. A Continuous Delivery Pipeline that enables Single Story Sprints, effectively reduces delivery lead times from days, weeks or months, down to minutes.


Using LEAN principles early makes a significant impact. The product development cycle starts long before a developer starts coding. Decomposing products and features into small batches [or single story sprints] provides visibility into the flow of work from idea to production; improving IT performance and reducing deployment pain.

Conclusion

There is a significant need if not demand for a measured approach to modernizing aging IT capabilities. A systematic refactoring approach leveraging DevOps principles can be powerful; however, it is not enough without a proven transformation strategy.

A Strangler Architecture transformation pattern enables incremental change to critical IT systems, without the business risk of a rewrite or wholesale replacement.

Understanding and designing the right team topology is key to success, as well as building a Continuous Delivery Pipeline and DevOps culture that enables collaboration and a product-oriented delivery mindset.

It's not about whether the business needs to deliver 10x faster — it's about whether IT can step up to the business demands, when needed. It's about making small improvements and fixing failures more quickly, resulting in more progress in less time. Those that are increasing their IT agility are those that are likely to succeed. 



To learn more, contact us at:

info@DevIQ.io

+1 303-232-3840

or visit us on the web at

DevIQ.io



Driven to Improve Life.

Partners in smart product engineering, from concept to reality and beyond.

