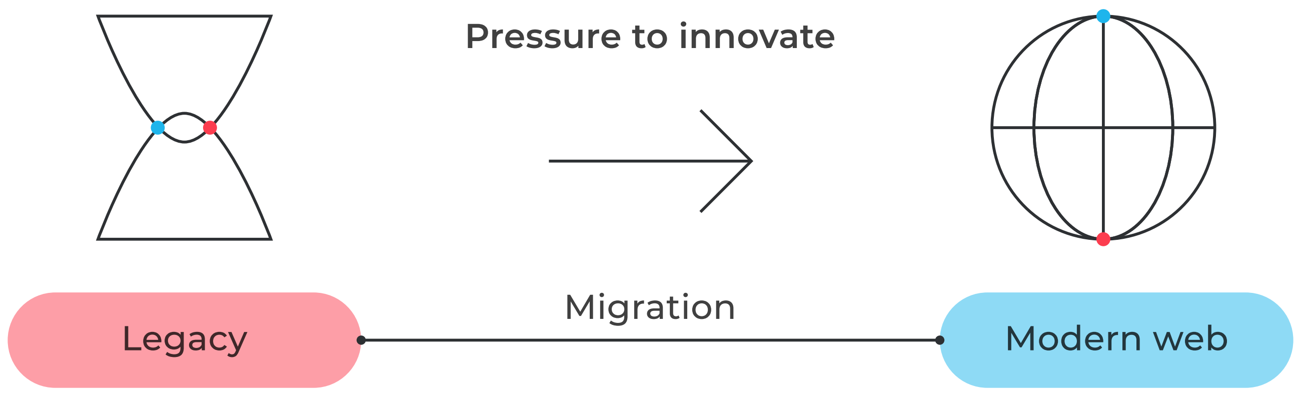
WHITEPAPER

Application migration

Your guide to starting right, coordinating with your stakeholders, and completing your migration successfully

vaadin}>



Legacy symptoms

Whether the economy is slow or booming, the **pressure to innovate** has always been there for organizations, regardless if they aim to scale, grow, or even just survive. This challenge to innovate becomes more difficult for these organizations when they use **legacy applications** that do great jobs at keeping them firmly **on track with old ways of doing business**.

Today’s trends in digital transformation have pushed this pressure to adapt to extreme levels, however. The ability of an organization not just to enact change that is centrally and carefully planned, but to **enable multiple concurrent changes that are driven from agile, decentralized initiatives** is now the goal. Engineering processes like continuous delivery have emerged to meet this new need for speed in enabling new business requirements at this pace.

Legacy applications hurt the most by slowing the organizations that run them from implementing changes that enable innovation. Brittle application architectures, expensive or outdated technology that is hard to integrate with other systems, and the overall look and feel not reflecting the organization's branding are all contributing factors.

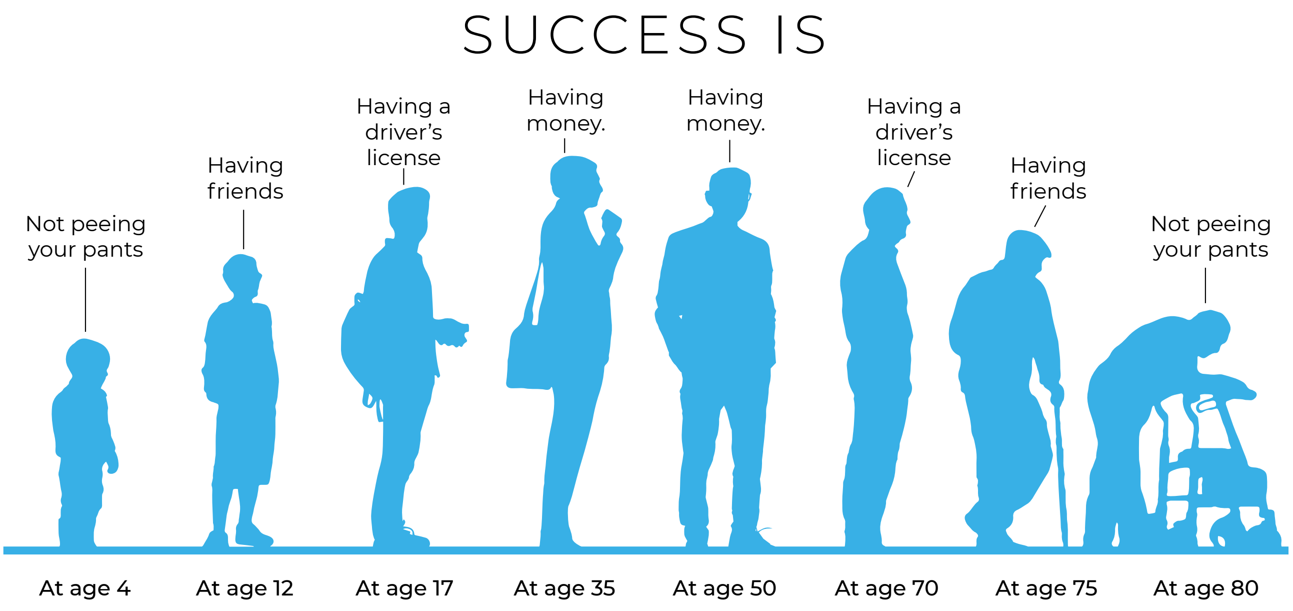
From VB6 to Flash, hundreds of billions of lines of legacy code are in production today that are not HTML5-compliant and don't run on the latest browsers and devices. **This code is simultaneously at the heart of keeping businesses running, while ultimately keeping technological advances out of reach that will enable key innovations.**

In this series on application migration we will take a closer look at legacy applications and explore how organizations can be **faster to recognize legacy symptoms** and **distinguish what key business drivers are behind finding a cure**. We'll continue with practical guidelines on deciding on an appropriate solution from the most common approaches and how to move from deciding on an outcome to a plan of action that will lead to a successful completion. We wrap up this article with a few tips on the most common pitfalls organizations encounter as they attempt to leverage their legacy applications and processes, and move forward at the same time.

Recognize the challenges

Bespoke enterprise applications of any significant complexity in production today did not appear overnight. Typically they were first launched in a simple form and **gradually grew and evolved with the organization** to provide increasing value. In most common scenarios, this evolution slows as the **value eventually plateaus**.

A useful analogy to the plateau of legacy applications is the internet humor describing the bell curve of life:



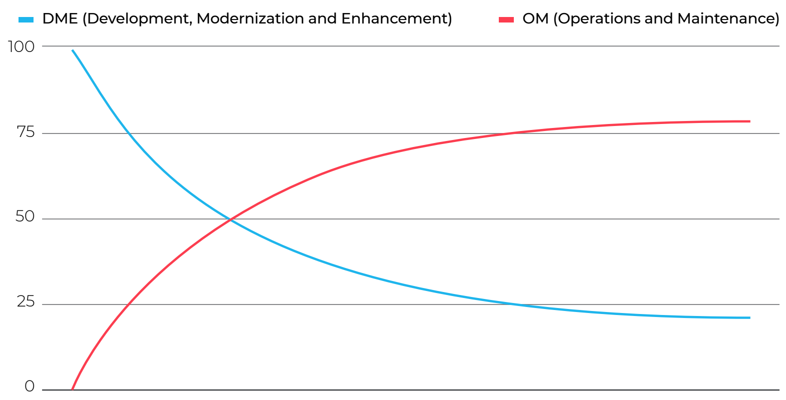
How we observe the plateau effect in enterprise applications helps us define the term "legacy". Both **gradual and sudden events** in the application's context can lead to the plateau.

Sudden events typically occur outside the organization and are related to technology and its suppliers. Examples are if a technology **supplier announces termination of support, goes bankrupt or ceases operations, or unilaterally changes licensing arrangements which make the cost of ownership infeasible**. In these cases there are sudden, clear events that disqualify the underlying technology as strategic for the future, sharply increasing the resistance to invest further in application maintenance. Depending on the risk to business, these changes may have an urgent impact on foreseeable future operations that eclipses the business need to innovate.

Gradual events on the other hand are at the heart of the "creeping normal", the barely perceptible gradual plateau and decline of enterprise applications. They are a good example of the boiling frog scenario that pushes companies to petrification of their applications, systems and business models.

The gradual plateau is experienced by developers and the business alike although developers will typically be the first to have their work affected. **For *developers*, there is a decrease in productivity** as code changes during adaptive maintenance affect a growing number of use cases and dependencies with external services. Hours spent doing maintenance will be less effective as more time is spent on testing, and less time is spent on purposefully writing new correct code.

The slowdown in productivity is to a large degree responsible for the often-cited 80/20 effect in legacy applications, where we observe 80% of the budget is spent on the maintenance of the old systems and only 20% on new products. This cascades to business in slower time to market of new technology-enabled products, and slower rate of change in general.



When new strategic applications are built the expenditure will count as development and modernization, but once the application is put into production, operations and maintenance costs quickly rise so companies reach the 80% maintenance plateau.

source: https://www.whitehouse.gov/wp-content/uploads/2018/02/ap_16_it-fy2019.pdf

Business experiences the plateau through gradual changes in the state of the art in IT that makes the application costlier to support and **harder to integrate with other systems**. Recent articles from Gartner offer a useful cost-oriented definition of legacy applications, namely applications for which the **cost of addressing their technical debt** exceeds the cost of replacing them. Gartner defines technical debt as the deviation of an application from non-functional requirements of compatibility, maintainability, reliability, usability, efficiency, portability, and security.

Understand where the pain is greatest

Different groups of application stakeholders are affected differently by the legacy effect, so while stakeholders might be unified in not liking the old, they rarely see eye to eye on the solution. Accordingly, organizations often have a hard time collectively arriving at a decision on how best to move forward.

If we divide the stakeholders into software maintainers, the product owners, the users, and senior management, we get different perspectives on the legacy problem:

The ***software maintainers*** typically will be concerned with the developer view on technical debt: how programming activities of the past have led to **decreased maintainability** of the application today. To be faster, cheaper and more effective at meeting user's evolving requirements, developers often propose redeveloping the applications from scratch. For small applications or applications with excellent modularity built in, this can be viable. This tendency can be magnified by the pressure of developers to have up to date skills.

Product owners will be mostly concerned with time to market of new products (digital or otherwise); and will require **extending and easily**

integrating with existing core business systems of record. This requirement to integrate will put pressure on non-functional areas of the applications, so low-invasive solutions like presentation extension become attractive.

The ***users***, to the continuous surprise of many developers, are not tech-driven and are concerned in the first place with **doing their jobs efficiently**. For business processes of any moderate complexity, experienced users have often gained a considerable amount of expertise with the existing application and in their business area. Perception of value of the existing application will be a tradeoff of functional correctness of the application, business value, and expertise on the one hand versus non-functional inadequacies of the application on the other.

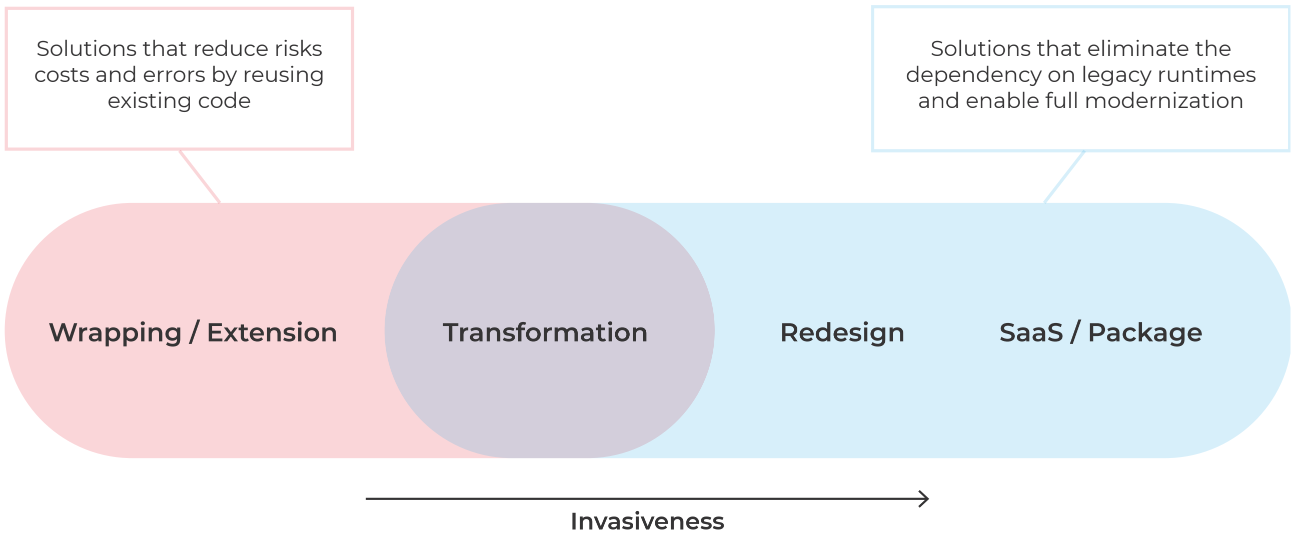
Senior management will be aware of the **investment** by the organization to build or implement the application currently in use. The cost not just to build the software but to test, document, and provide training for it will be taken into account and used as a guide to predict the cost to implement a new application. This group will also be weighing the **build vs buy** question as they consider mergers, acquisitions, and consolidated reporting over multisite locations. For applications that perform generic utility like "payroll" "cost accounting" there will be a strong case to drop bespoke solutions in favor of commercial packages for ease of integration.

Choosing the right path forward

There is **more than one way to replace an existing legacy system** with an alternative that has fewer legacy characteristics. The available options can be listed in order of their invasiveness. Invasiveness is a key variable to track since **higher invasiveness is tied to greater change but also higher expense, risk and project duration**.

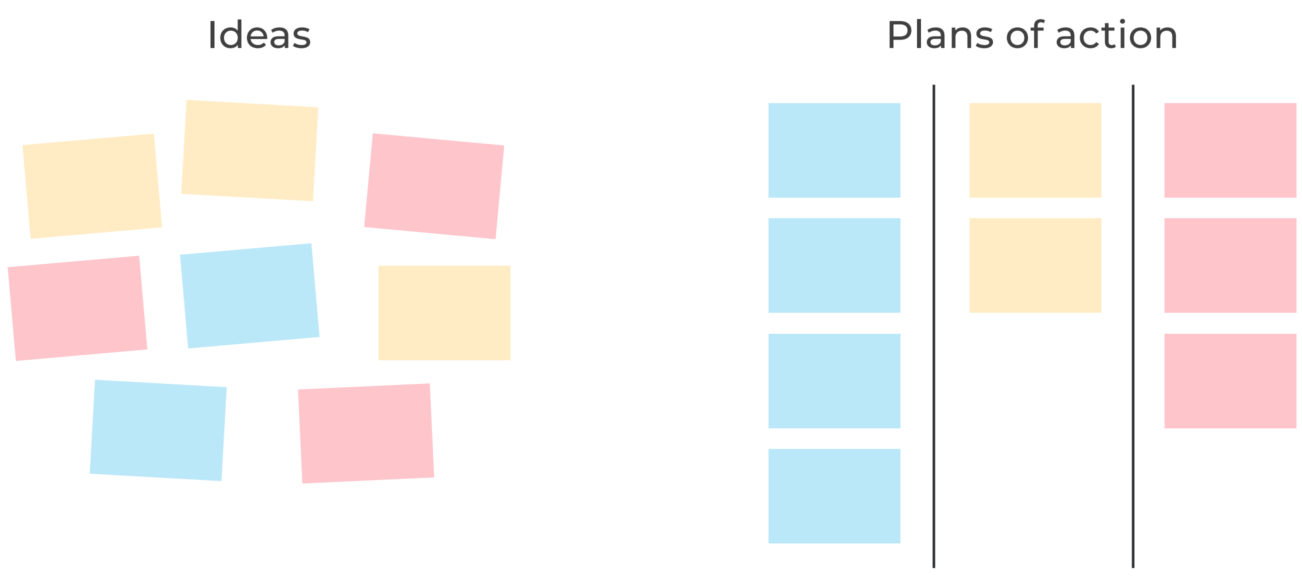
The least invasive approach is to ***not change the code*** at all, but to tap from a plethora of creative technical tools that bridge the most problematic non-functional issues through **wrapping**. Examples are **emulators, cross-compilers, transpilers, presentation extenders, multi-platform runtimes, or even screen scrapers**. These let you either build like the old and run in a new environment, or even build and run in the old and extend the experience to a new platform. This approach has been viable for organizations seeking short-term relief for platform and interoperability problems. They have the benefit of being easy to implement, but at the same time help lock the legacy application in place, making it harder to remove.

Reuse-driven ***transformation*** is a linear **process that aims to unlock value from the existing application code** by removing dependencies on unstrategic framework runtimes and replacing them with equivalents on newer ones. This allows a high degree of reuse but the changes can lead to side effects that need to be discovered through testing and then resolved. This approach works well for larger, more complex core business applications, especially when heavily customized for a small number of organizations. When taking small steps, like migrating from Vaadin 8 to Vaadin 10, even very large applications can be migrated in a single rollout iteration. There are many ways to apply automation to increase the speed, quality and consistency of the way the new application is generated. The approach typically aims at **feature parity** however can also be extended to include the **redesigns of a select number of interfaces** that are used the most.


Solutions that reduce risks costs and errors by reusing existing code
Solutions that eliminate the dependency on legacy runtimes and enable full modernization
Wrapping / Extension
Transformation
Redesign
SaaS / Package
Invasiveness

Application ***redesign*** is an exercise that leverages the opportunity of **starting fresh** to arrive at both an **improved user experience** and a release from the **complexity and inflexibility of the current application**. Redesign is a creative, non-linear process in which designers empathize with users while they imagine their work free from the kinematics of their current user interfaces. The result is an application that will be more intuitive for new users and will ultimately enable improved customer experience. For existing applications that are being commercialized, **redesign may be key for the product to remain competitive**. Even though it is design-driven, reuse and recovery of information from the existing application can be secondary objectives to improve quality and efficiency in creating the final product.

The most impactful approach is to turn away from bespoke solutions entirely and switch to commercially available ***packaged applications***, either hosted on or off the premises. Salesforce, Trello, Sage, Snow, G Suite and SAP are examples. The main argument for moving to packages is **standardization**, which can provide benefits that extend **throughout the organization** and beyond **into the supply chain**. Senior management stands to gain from this approach as they make the organization better ready for merger and acquisition deals, and also users gain by acquiring more reusable skills.



Turning ideas into plans of action

An important source of risk in application migration is ensuring the **project ends successfully** through setting clear, achievable goals from the beginning. For the two migration approaches on either extreme of the invasiveness spectrum, CIOs will be ultimately responsible for outcomes but not for evaluating what would be “best practice” to make the vision reality:

- Wrapping techniques are by their very nature non-invasive so their implementation will require negligible planning and coordination;
- Package implementations will typically follow vendor-specific rollout processes leaving us only with non-technical problems to be creatively solved by the business - the impact on our working practices and on existing integration points.

If the organization decides to go for a transformation or a redesign, this challenge of defining the best practice that fits the organization becomes more challenging. There are a few reasons:

- Migration projects are typically **unprecedented** for application developers and users alike - typically both groups are more used to forward engineering and adaptive maintenance of existing code. Because of this, alignment of expectations of a migration requires more effort from all stakeholders and misunderstandings occur more easily;

- While the existing **applications** might have gone into production with their initial release in one go, since their launch they will have **accumulated features** by rule of thumb at a rate of 8% per year or more. In terms of testing effort, deploying version 1.0 of the replacement of a 7 year old system will not be comparable to the launch of version 1.0 of the original;

- The core functionality of the existing applications will have been tested through intensive use. Users may no longer test the core features because they "just work" and user tests may only be focused on newly developed features. Still, to get the new system live, this **core functionality will have to be tested** in the new application and any knowledge forgotten will have to be regained;

- Development teams are typically highly skilled in the technologies and programming languages of the existing applications, and may **underestimate the learning curve** for the target development languages and architectural paradigms;

- Users but especially developer teams are right-sized for ongoing maintenance scenarios; expanding the capacity to accommodate **additional reengineering effort in parallel** will increase skills, communication, and organizational requirements that must be realistically planned for.

How to get the best result

Each migration is unique, but there are three basic things an organization can do to maximize the chance of success of any migration project. These are application portfolio management, reuse of existing artifacts, and following an incremental approach.

Application Portfolio Management (APM)

Why we need it

To apply migration energy and resources where they are needed most

What's it about

At the most basic level, APM helps organizations identify which applications they are running and where the source code is, so it is a useful starting point for planning a migration.

APM also goes further to help us analyze the value of the applications in our portfolios. With it we recognize that even if we select a framework, like Vaadin, as a strategic technical platform for the future, not all applications in our organization need to be rewritten to use this framework with equal urgency, if at all. APM comes in various flavors, and one of the most widely known is Gartner's TIME APM framework. APM frameworks typically categorize applications according to technical and business criteria, where an application can score either high or low on either. Applications that score high on business criteria (for instance, the application is an enabler of a key differentiator for the company) and low on technical criteria (for instance, high risk or cost of ownership) benefit the most from migration and should be prioritized.

Reuse existing artifacts

Why we need it

To facilitate testing and to keep newly developed features in synch between old and new

What's it about

An easy way to convince business that the new application is ready to be accepted is if the two applications, with identical data, can handle identical business processes to yield identical results. This is much easier when data structures and services of the production and modernized systems correspond or can at least be matched using a simple conversion.

Another reason why artifact reuse is worthwhile is that migration projects of large applications can easily take over a year to complete. In that time users will still expect new features to be implemented in the existing systems, and other updates may be required for regulatory compliance or to fix critical defects. In this dynamic setup, we want to make sure that things we change in the production system can be demonstrated to also work as intended in the new system. The more that production business logic, services, interfaces, and data structures can be leveraged in the new modernized version, the easier this synchronization becomes.

Find an incremental approach

Why we need it

An incremental approach will improve manageability and lower risk

What's it about

It's likely that business has a long list of wishes, and that IT has a far reaching plan to deploy a future-proof IT architecture. When considering larger business applications, it is unlikely that an attempt to migrate big-bang style all code while fulfilling all business wishes and all IT wishes will succeed. For IT and business to agree on an incremental modernization approach that will succeed, it is useful to understand which factors are actually relevant for driving the modernization and where the emphasis lies. Following this exercise, there are different approaches that could be considered:

- Modernize first the back-end, then the front-end
- Fully modernize and replace one module at a time
- First a technical migration and then a modernization
- Migrate view by view, direct or embed new views to web app, while some parts of the app still require the old desktop application

- Create an abstraction layer for the UI framework. This is often more expensive than doing a complete UI layer re-write, but allows running and developing the legacy version during the migration phase.

Want to learn more about our approach to migrating business applications based on deprecated technologies to the web?

CONTACT US