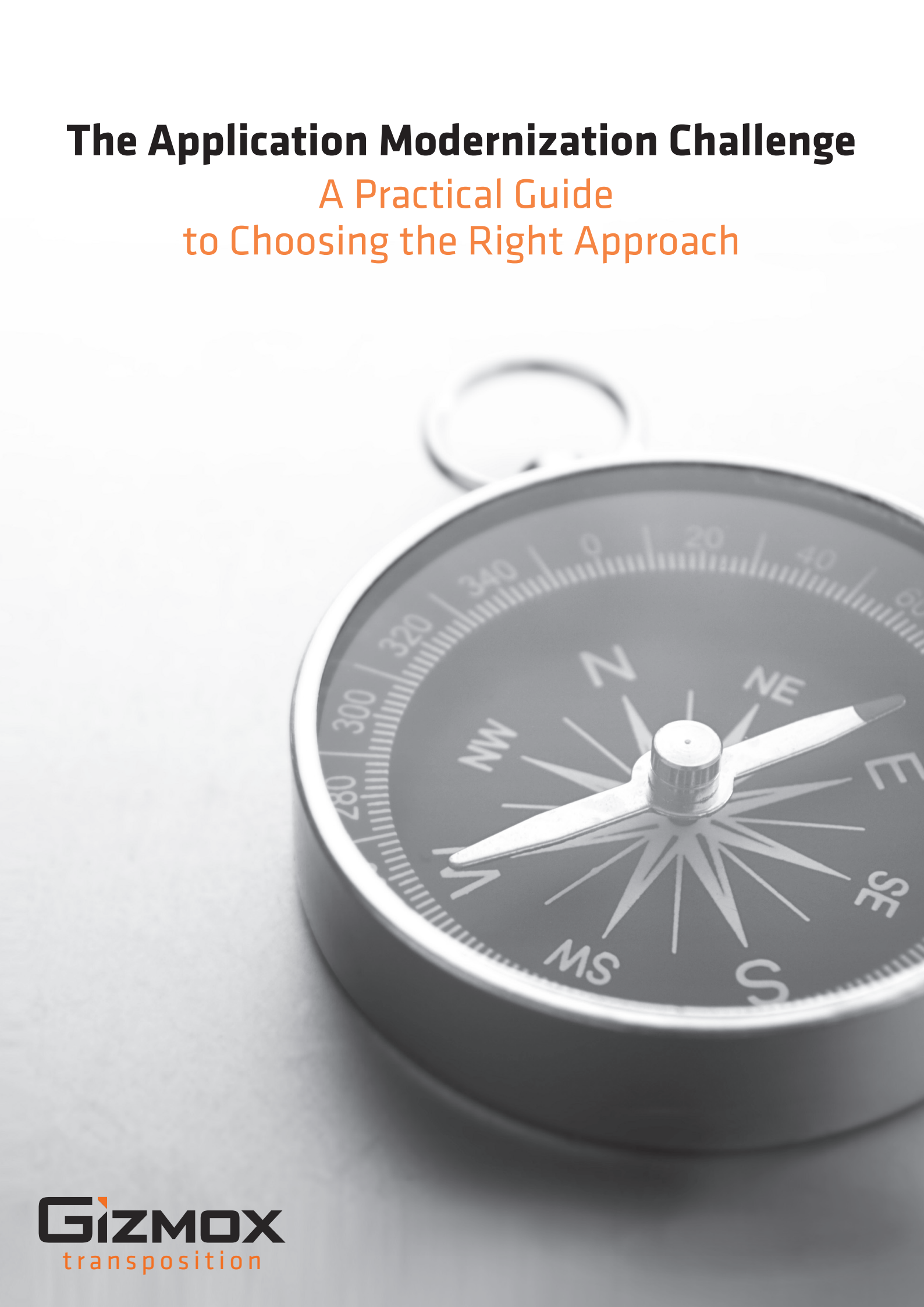
The Application Modernization Challenge

A Practical Guide to Choosing the Right Approach



Gizmox transposition

Table of Contents

The Application Modernization Challenge:
A Practical Guide to Choosing the Right Approach

The Zombie Appocalypse

As an IT executive in a medium-sized or large enterprise, you must be feeling the increasing pressure to modernize your organization's core business applications.
Across industries and business sectors, competitive advantage and business growth are increasingly derived from adopting new technologies. Enterprise technology has to evolve in order to keep pace with competition and ever changing business needs. Whether it is a modern environment, a mobile app, a cloud-based service or a new web app, business leaders in your organization are all in agreement that it should be launched ASAP. However, your legacy applications are not architected to support integration with newer applications, components and devices.

Maintenance costs for legacy systems increase over time. Developers with needed expertise in out-of-date technologies become hard to find and expensive to contract. Security updates for legacy systems become obsolete. Enterprises that try to postpone the inevitable, find that the cost of inaction is often the loss of significant business opportunities.
Legacy modernization is a moving target. The pace of change in the IT world is such that the use of RAD tools such as VB6 and WinForms - that were considered state of the art just a few years ago - has created 'zombie' legacy systems – difficult to integrate, impossible to secure, expensive to maintain, and inaccessible remotely with standard devices. However, those legacy applications still hold significant business value and therefore, must be modernized.

Black-box or White-box?

While the meaning of 'Legacy Systems' may change over time, the challenges of migrating critical business applications remain the same. Code modernization projects aim to salvage the investment made in infrastructure, business logic and functionality, and at the same time develop an application that would be maintainable, customizable and compatible with new user interface (UI) standards such as HTML5.

Application modernization approaches broadly fit into two categories: Black-box and White box modernization. This white paper will compare the main code modernization techniques; Transposition, Automatic Migration and Manual Migration, in the context of common modernization use-cases, to help you determine the right approach for your project.

BLACK-BOX

A Black-box approach to migration means that inner workings of the migration process are obstructed from view, as well as any user input into the migration process.

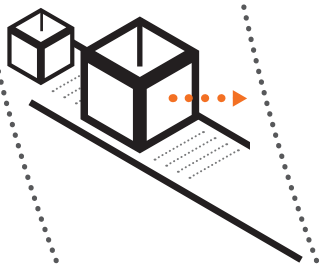
The Black-box approach is commonly used for version or language upgrades, but falls short when the application requires architectural changes or refactoring onto a new platform and user exprience.

Automatic Migration
is a method used to upgrade or migrate applications, that fit a predetermined configuration and do not require customized reengineering.

WHITE-BOX

The White-box approach allows the flexibility to intervene in the migration process and impact application structure, code patterns and user exprience.

A White-box approach is commonly applied to architectural changes or the refactoring of applications onto a new platform and user exprience.



Manual Rewrite
A complete manual rewrite of an application is advised when existing business logic becomes obsolete, or when the application requires re-purposing.

Transposition
Transposition is a patent pending modernization technology that employs semantic code understanding, for context-aware application migration. Use it to preserve an applications' business logic and functionality, or when your application needs reengineering, or refactoring onto a new platform and user experience

Selecting the Target Platform

The target platform decision is, of course, a compromise between the **a**. new need (be it competitive advantage, security or maintenance, to name a few), **b.** the desired result (Mobile, Web or Cloud operability) and **c.** the challenges of modernization.

Legacy data architectures are especially at odds with Distributed and Service Oriented architectures (e.g. Client/Server, Web, Cloud). When the gap to bridge between legacy and target is a core architectural one, we recommend avoiding the black-box approach of an automated solution, which leaves very little room for refactoring and re-architecting.

Budgeting & Outsourcing

Application modernization is so specialized, that it is hardly ever done in-house. When the need to modernize presses, most large enterprises turn to their Systems' Integrator (SI) of choice, for a manual rewrite project scoping and price quote. While turning to a trusted partner is never a bad idea, SI project pricing structure usually includes a 'Time and Materials' (T&M) bracket that can easily spiral out of control, in terms of time-to-market, internal resources and budget.

Manual rewrite projects are typically between four to ten times longer and more expensive than automated projects or Transposition[1]. Moreover, manual rewrite projects involve a high level of risk, as their failure rate can reach 65%[2].

Alternatively, some enterprises approach an automated solution vendor for a quote. When budgeting for an automated migration project, always remember that architectural changes and UI will need to be refactored outside the automation tool, which will incur additional costs.

Other considerations include the length of the required code freeze for your selected modernization path. A manual rewrite project usually lasts many months and sometimes years. Any ongoing programming and maintenance performed during the project period would need to be integrated into the application, adding complexity to the process. Automatic migration is typically a shorter process. However, it will require a code freeze for the entire period. Transposition will require the shortest freeze, because the project duration is as short as in automatic migration (or shorter), but ongoing changes are formulated as rules. The fresh code can be re-integrated by applying existing rules.

1 Based on Gartner Group study "Forecasting the Worldwide IT Services Industry: 1999,1" and Gizmox internal Data

2 The Standish Group Report, 2014

Engagement Scoping

Imagine a contractor coming to install a new fireplace in your house. Now assume you no longer have the water and power infrastructure schemes. Without figuring those out first, the contractor is likely to hit a vital pipe or cable, which will double the duration and cost of the project.

A manual rewrite project scoping is guided by many - well tested, yet generalized - rules of thumb. Your enterprise application has evolved through accretion. Even if your original code documentation is available, which is rarely the case, it is probably not very useful at this stage. Much of your legacy code had been rewritten over the years with layers and patches. Some of those are likely to be redundant, often providing extended capabilities in a completely undocumented way. Moreover, at the time your legacy application was written, best practices did not include current standards like object oriented models or n-Layers architecture. This condition is commonly referred to as 'spaghetti code'. Without proper reverse engineering, no rewrite project scoping can truly provide an accurate pre-engagement assessment. Indeed, Research by Standish Group found that all manual rewrite projects are disposed to an average of 189% budget and time overrun.

The pre-engagement assessment capabilities of Transposition on the other hand, includes an algorithmic process, equivalent to a detailed reverse engineering that would make sense of the most tangled legacy 'spaghetti'. Transposition assessment can identify all the hidden 'pipes and cables' up front, to provide a very accurate scoping of the project.

Internal Resource Allocation

A manual rewrite project requires considerable internal resource allocation. In particular, the reverse engineering, or business logic re-capture stage of the specification is human resource intensive. Moreover, non-technological enterprises (but say, finance, or health) often do not employ technical staff with the required skill-set to manage a complex application modernization project. In most cases, a rewrite would involve expert reverse engineering and would require expertize in the source and target environments – a rare combination of skills.

Automatic migration requires relatively little internal resources during the migration, because it does not involve reengineering and relies on a preconfigured, generic process. However, internal resources are required in the post-migration phase, to customize the delivered code.

Transposition is a software-guided and human controlled process that algorithmically 'understands' your legacy application. It can provide optimized analysis, assessment and gap bridging suggestions. Transposition delivers a fully re-engineered and customizable application – all with minimal internal resource allocation. Alternatively, you may pick and choose standalone parts of the project, to be managed in-house.

Migration

Comparing an automated migration project to Transposition is like comparing machine translation to the product of a human translator (equipped with the best machine translation tools). The former is textual (literal), out-of-context. The latter is semantic and context-aware. Automated migration is a 'Black-box' technique that can be compared to textual translation. This means the migrated application is a textual translation of the source application, sub-optimal to the target platform and new user experience.

Manual rewrite projects are highly dependent on the volatile human factor.

Transposition is a software-guided, semantically aware and human controlled process that minimizes common human errors like integration collisions or non-repetitive code patterns. In addition, it is a 'White-box' technique, meaning that customization is possible at any stage and optimized for the target platform and desired user experience.

Quality Assurance (QA) and Maintenance

Software testing can be a painful, costly and time-consuming process. In manual rewrite projects, the average 'bug' ratio is around 20-30 bugs per 1000 lines of code[3]. In contrast, Transposition bug ratio is dramatically reduced to around 1 bug per each 1000 line of code.
In automatic migration, the migrated code is generically configured and machine-generated. It has to cover all common use cases with pre-configured rules and therefore likely to include excessive, generic code lines. The result is difficult to test, maintain and extend. In contrast, the code resulting from Transposition is native and optimized to provide specific application functionality. This makes transpositioned code easier to test, maintain and extend, by means of target platform standards and developers.

Finally in Transposition, the migrated application is functionally equivalent to the legacy one, allowing cost-efficient reuse of your legacy test scripts.

3 Gizmox internal data

Your Application Modernization Project: Key Decision Points

Legacy App
VBv, Win Forms, ASP, PowerBuilder

Selecting
Target Platform

Budgeting & Outsourcing

Engagement Scoping

Internal Resource allocation

Migration

New App
Saas, Mobile, Multitenant

QA & Maintenance

Manual Rewrite

- Supports monolithic-to-distributed re-architecture
- High budget
- 'Time & Materials' bracket prone to up to 189% overrun[4] of allocated budget
- High risk: over 65% failure5
- Project scoping does not include pre-engagement reverse engineering and therefore inaptly predictis resources
- HR-intensive
- Qualified staff with source and target environments skill set is scares and expensive
- Prone to human error (e.g. Integration collision, non-repetitive patterns)
- 65% fail rate[6]
- Testing & iterations only possible post-compilation

Transposition

- Software guided monolithic-to-distributed re-architecture
- Low budget
- Assessment utilizing semantic code understanding and reverse engineering approach, provides precise scoping and fixed price quote upfront
- Shortest code freeze. Ongoing changes are formulated as rules that can be re-applied to modified code
- Minimal and Flexible. Modular levels of outsourcing engagement available
- Software-guided and controlled process
- Semantic, context-aware code refactoring
- Customizable Business logic and UI
- Reduced percentage of programing 'bugs'
- Native code quality. Highly maintainable and extendable
- Cost-efficient reuse of legacy test scripts

Automatic Migration

- Rearchitecting is manual and only possible post-migration
- Low budget
- Pricing does not include rearchitecting, which is manual and potentially risky
- Code freeze required
- Minimal for the migration phase, HR-intensive for post migration phase
- Text-based, contextless
- Business logic and UI are only customizable post migration
- Preconfigured, machine-generated code, prone to excessive 'code-noise', which is hard to maintain

Project time
Short
Long

Approach
Black box
White box

4-5-6 The Standish Group Report, 2014

Conclusions & Recommendations

Approaches to application modernizations vary and are designed to meet different needs. It is therefore a futile exercise to try and discredit or validate any single approach or method. From start to finish of your enterprise application modernization project, different considerations should impact your approach.

While a manual rewrite might be the most flexible process, it also incorporates the highest risks. Much like a rewrite, Transposition offers high flexibility in reengineering the application, as well as the protection of a software controlled environment, that minimizes the human error liability. Automatic migration - although relatively risk-free, in terms of project execution - is limited, and doesn't deliver much value, when the target platform necessitates refactoring or reengineering of the application, and a new user experience.

Our detailed examination shows that Transposition is advantageous when refactoring applications from one platform onto a different one and in almost every aspect of the project execution, while a Manual Rewrite has clear advantages when application logic and functionality have to be rewritten. Finally, an Automatic Migration is recommended for version upgrades, with no refactoring and reengineering needs.

Here are some specific recommendations:

- Engage in a manual rewrite project if - and only if - your application logic is obsolete. No lesser change merits the time, risks and cost involved
- To temporarily extend the life of an application, for a version, code or language upgrade, use automatic migration. This approach delivers quick fixes and short-term gains, in terms of budget and project length.
- To repurpose your application when its business logic is still valuable, but your code, platform, language or user experience are out-of-date, use Transposition. This will allow a complete reengineering and user experience refactoring, foror the cost and durationth of an automated migration (or less) and the code quality of a rewrite (or better).

Gizmox
transposition

GET A FREE ASSESSMENT

Transposition Assessment Wizard

Download and run on your application code

Get detailed statistics analyzing your code

Work with Gizmox to get a cost estimate on transposing your application

DOWNLOAD

Gizmox Transposition provides an enterprise level solution for bringing business applications to the latest platforms and operating systems such as, latest desktop Win8, 10, or web, cloud and mobile. Gizmox Transposition is fast and risk-free, using innovative, patented solution and service.
To learn more, please contact us at **consult@gizmoxts.com**