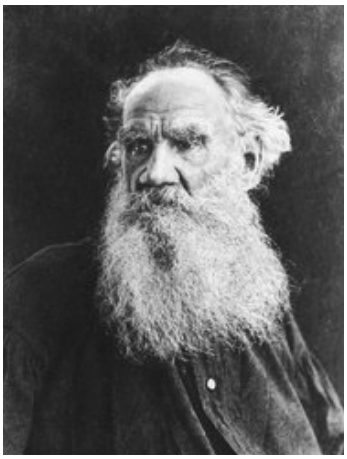
Luxoft
A DXC Technology Company

The perils of legacy modernization

An ebook for navigating complex modernization choices



Ever heard of the Anna Karenina principle? Leo Tolstoy's Russian classic begins by stating that, "All happy families are alike; each unhappy family is unhappy in its own way."

If you have legacy IT and have looked into issues and solutions that come along with it, you may find it interesting that the world of legacy modernization initiatives inversely reflects this principle and states that all modernization failures are alike, while success is a combination of unique ingredients.

In this article we look at the generality of legacy IT modernization failures, provide an overview of common modernization approaches and share the experience from one of our clients who attempted a mainframe modernization five times before it was successfully completed. We also outline some considerations on what can help organizations succeed where others have failed.

Legacy is expensive to run and expensive to modernize

IT operations and maintenance are expensive. They account for around 75% of the entire IT spend worldwide. Normally, the most expensive systems to support are the oldest ones, and mainframe systems created 20–50 years ago would very likely be at the top of the list. However, they continue running despite growing run and change costs — at least until they fail and don't recover themselves.

These reasons make legacy systems in general (and old mainframe monoliths in particular) a dangerous and expensive part of an organization's technology landscape with constantly growing operations cost and the chance of failure increasing every day, forcing IT managers to conduct modernization initiatives.

If you could replace or fix your legacy system without much effort and risk, you've probably done it already. However, it's often a very challenging and complex endeavor, resulting in modernization projects accounting for almost a third of the entire development and enhancement spend worldwide. Twenty-nine percent of this spending is wasted on failed efforts. The 2021 Mainframe Modernization Business Barometer Report found that 77% of organizations have started a legacy system modernization project but failed to complete it. If you don't want to end up in this category, you need to honestly assess your modernization strategy for general failure attributes, and proceed to a thorough analysis of modernization scope and solution space.

What are the similarities of all modernization failures:

Unclear understanding of modernization scope

Underestimation of effort

High expectations of chosen approach and reliance on a 'magic solution'

C-suite and IT leadership disconnect

In the next sections we'll try to describe how modernization projects typically end up with these attributes. We'll start with an explanation of our inside-view on general modernization approaches.

Modernization approaches and pitfalls

Modernization methodology fundamentals state that there are several generalized approaches which are usually considered legacy modernization treatments — also commonly known as the R-treatments. Each of them works perfectly for a particular subset of cases, but they don't necessarily integrate easily with the others, and of course, each comes at its own price. We have laid out our thoughts on these approaches, where they are best applicable and where often overlooked caveats are.

- **Retire** — an approach that simply lets you shut your application down. Best case scenario, this comes as the result of an application portfolio analysis that outlines duplication of functionality or its obsoleteness. Do not forget to retain the data you might need and think about how you can access it.

- **Retain** — your mainframe still does the job, so keep the application and instead focus on modernizing the peripherals: Modernize your legacy landscape with CI/CD, web-enable your UI, introduce REST API integration points. This usually comes with extra vendor lock-in that adds to your current spend.

  It's often not an option for non-IBM legacy platforms as the toolset to support modernized Software Development Lifecycle there simply couldn't exist, and even with mainstream legacy platforms some of your components would be doomed to collect further dust.

- **Replace** — getting a COTS product from a trusted vendor with support and tailoring services is an excellent option. If this was possible for your business, it's probably already been done. The likes of what is left out usually includes custom-made solutions with company IPs. Even if there's a good candidate for replacement, business requirements recovery and aggregation is often a multiyear saga with business process adjustment battles, pulling other applications in, compromises between business and product tailoring capabilities and eventually turning legacy modernization into a global business initiative. Worth considering, but IT leadership should remember this will be business driven with an unpredictable timeline and outcome.

- **Rewrite** — same as Replace, but with your own solution, no compromises required. Rebuild your application following modern architectural guidelines and see it become cloud native. It requires a complete recovery of what your application functionally is, and reintegration with the ecosystem once ready. The cleanest approach of all. However, complexity and costs grow exponentially with the size of the application. It might be very expensive if your application is medium to large, but it's the best option for smaller ones.

- **Rehost** — also known as lift and shift. Take all your legacy technology and port it to a new platform using some COTS environment emulation solution. The most popular solution with a few heavy-hitter product vendors out there. Kicks the can down the road, you get a new vendor lock-in, but it's the fastest option with the least amount of risk, effort, change and learning for the current operations team. Allows you to decommission your legacy hardware and keep something running until you Replace or Rewrite it, unless you decide to keep it Rehosted, because COBOL is not that bad, and applications written in COBOL are often rather ergonomic. Most of the solutions allow some level of integration with modern technologies. Where's the catch?

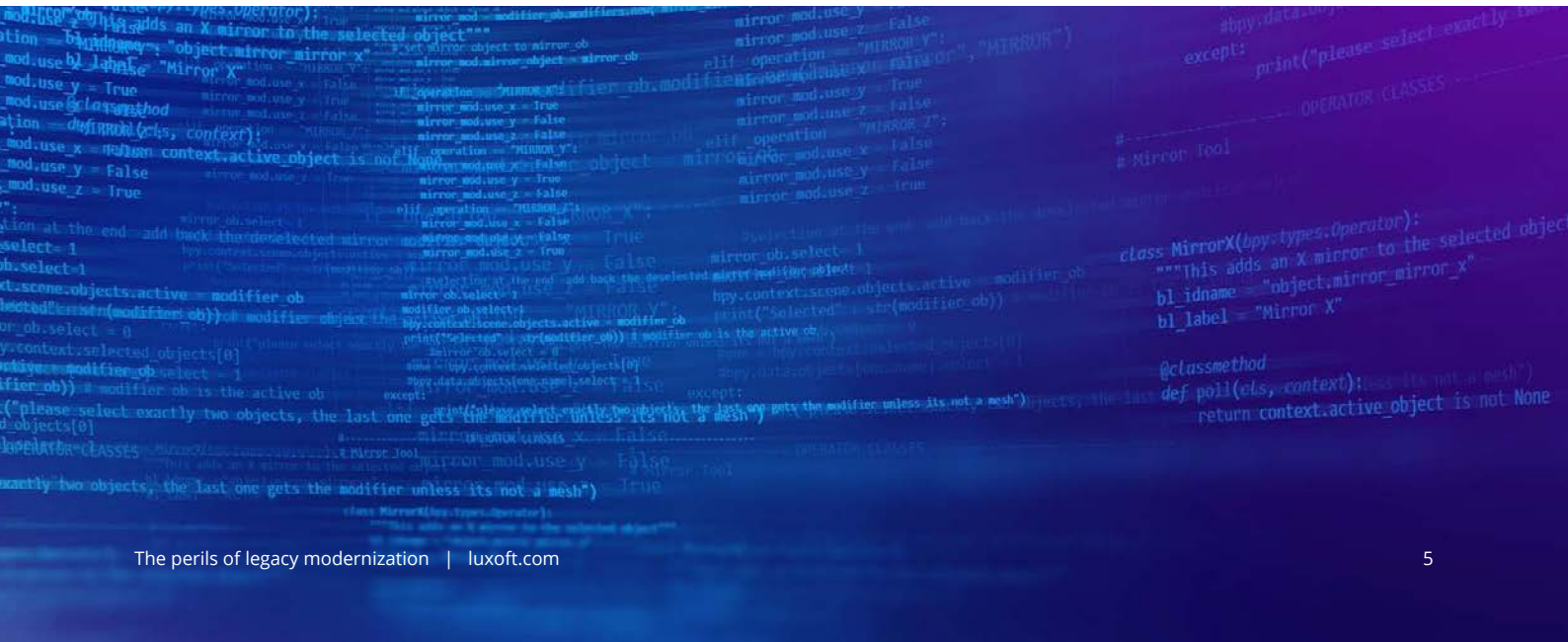
Well, some legacy components are not in the scope of emulation products: Reporting systems, assembler or other corner-case languages, CASE tools (Computer Aided Software Engineering code generators, popular in the early 1990s), for example. A lot of Rehost vendors specialize in a particular technology, provide different support for cloud enablement models and use different runtime licensing. If your technologies are mainstream (such as commonly used IBM products) you can proceed with few risks and a high chance of one vendor covering all. But, as soon as you need to combine solutions your risk grows exponentially.

While all product vendors support only their solutions, an integrator needs to step up and take responsibility for the end-to-end functional readiness; including integration, adequate test coverage with extensive automation, project management and deployment models. Also, some vendors don't have the capacity to support their product tailoring for you at a scale the rest of the project needs. All of the above is why, when you engage to Rehost, you need a delivery/integration partner with nerves of steel to get all the parts together and delivered on time. Most emulation solutions come with proprietary runtime license fees.

Lastly, remember that the emulation layer (even if supported by a trusted vendor) is another layer of complication for your application. This should be taken into account if you have performance considerations. Some things just won't work in an emulated environment at the scale you may need.

- **Rearchitect** — leverage automated conversion tools to reuse all legacy components of your application by transforming them to a modern technology stack. The key here is the transformation of your business logic from a legacy language to Java, green screens to web pages, batch scripts to modern scripting. These transformation tools allow for some level of refactoring on the fly, and with serious revision of business logic during transformation you can even bring meaningful names to your entities and attributes. This takes a long time, yet you're still going to end up with COBOL written in Java (so-called "JOBOL").

On the other hand, the Rearchitect approach is invaluable when converting metadata such as green screen definition, data dictionary and properties to the target technology format. Also, automatic conversion, or transformation, engines have a valuable subproduct that is used in application assessments — they provide a full blueprint of your application with component breakdown, dependency analysis reports and the like. Sometimes these can even guide your Rewrite with some tools providing business rules mining capabilities. The drawbacks are similar to Rehost: Vendors are rarely able to support all required technologies, though some are ready to try and stretch their capabilities beyond operating their existing offering (be ready to become their guinea pig). Plus there are performance concerns (native compilation vs. managed code, memory management paradigm adaptation), and more risk (not exactly a lift and shift). Test automation is vital with any automatic acceleration, as every transformation engine tweak results in the potential for fundamental or local regression with a need for a round of complete retest.



And again, while the resultant product will technically be in Java, a regular Java developer might refuse to accept it as Java, as it is still COBOL in Java syntax. So, SME COBOL developers will not understand the code anymore, and new Java developers might not understand it either — quite a conundrum. A lot of clients pull the plug on the Rearchitect approach down the road because the timeline, cost and complexity don't meet initial expectations.

- Reengineer — a combination of all. An art of compromise. If no other treatment worked for you on its own, this will because all technical challenges are solvable with the right approach.

  A typical scenario when modernizing one legacy component means needing to upgrade other interconnecting components, which may also be legacy, but of a different flavor, with a different nature of missing parts, and with a different solution space. Reengineering brings the greater flexibility of a Rewrite together with the acceleration of Rehost and Rearchitect.

  The caveat here is that typically, other approaches are single tool centric, whereas this approach is never the silver bullet that everyone is after. Unfortunately, Reengineering can be more expensive than lift and shift approaches. It is easier to believe in alternatives, and Reengineering is usually considered only when you're backed against the wall with several previous unsuccessful attempts. But everything comes at a price, and an early decision to Reengineer usually means saving on failures.

  Finally, finding a tool-agnostic partner that is able to identify all the ingredients of your success with this approach is a challenge.

The next section tells the story of a representative modernization case where the client tried a variety of options and spent 20 years failing, before finding the approach that worked. Luxoft saw the situation evolving and took an active role in the successful outcome. We believe there are good lessons to be learned for future mainframe migrations.

Case study

A client went through every option while trying to migrate from mainframe

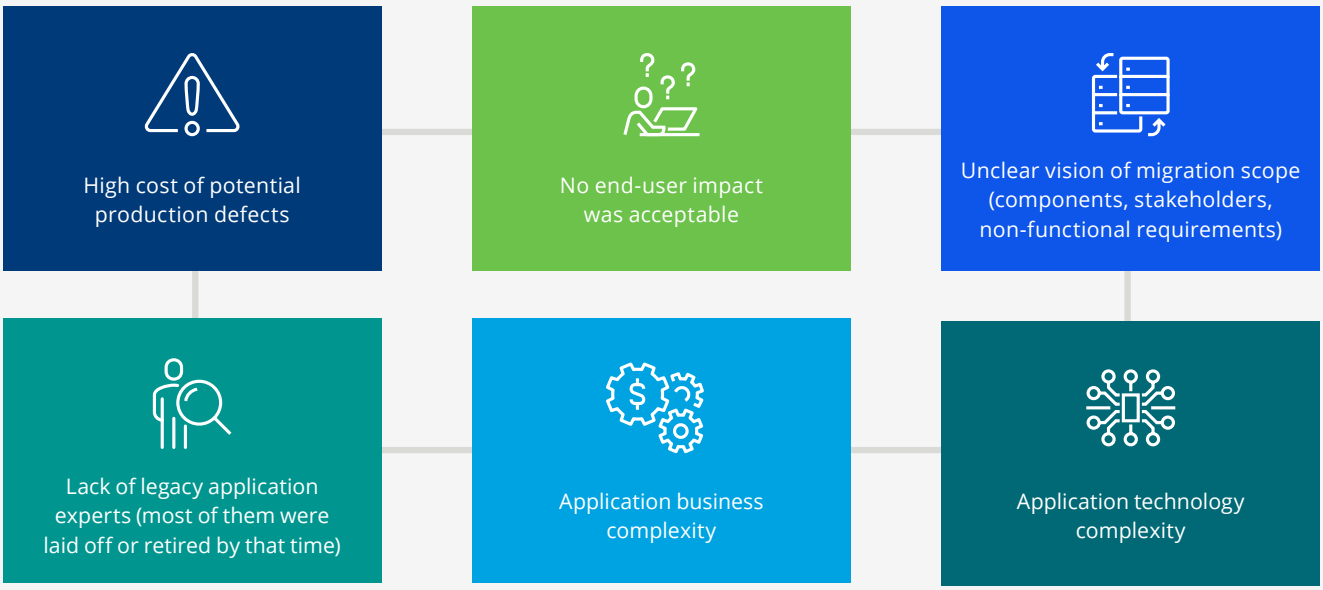
To provide a better understanding of how such failures happen and how to overcome them, we would like to tell you the story of one of our projects.

Our Fortune 500 client widely used mainframe in the past. Their early 21st century IT TCO reduction strategy included the decommissioning of their mainframe hosting data center and moving it to lower cost locations in the U.S., while also modernizing infrastructure technologies for lower cost operations.

By the time most applications were migrated away, one of the last systems residing on the mainframe was an in-house developed mission-critical COBOL/IMS application with very complex architecture and supported functionality. The inability to migrate it was holding the entire data center transition hostage. At the same time, for over a decade this system had been pending serious functionality extension for a new customer product line with legacy limitations making further application evolution nearly impossible.

Historically, there had been several attempts to give this system a new life, most of which were business-driven. It started with a partial **Replace** attempt in the late 1990s, a complete Replace was not possible due to the company's IP embedded into the code — which no other alternative products had — and ultimately it failed because of back-integration complexity. Then **Retain** was considered in the early 2000s, with web-enablement that failed because of an inability to technically drive this complicated code surgery in-house. A full **Rewrite** was then undertaken in the mid-2000s, that ended with the creation of a side application to support a few new business processes, which still depended on interconnection with the legacy application. Finally, a major take on a full Rewrite started in the early 2010s that unfortunately — being based on the need to extend existing functionality, generalize and transform it to support newer ways of doing business at the company — had not led to the application's migration to a new platform within 4 years. This is when the application had clearly become a top IT problem because the data center was no longer going to host it.

When IT took the lead of application modernization in 2014, they faced the following challenges:



Not all of the above challenges were clearly identified during the initial assessment. While assessing the solution space, the client's IT organization considered Rehost but wanted to modernize the technology stack, and chose to pursue a Rearchitect approach. A code transformation proposal by a chosen vendor was looking very promising with a commitment of a 12 month 'turnkey' project to be executed by the vendor. On paper, this looked rather straightforward from the vendor's perspective: COBOL is transformed to C++ (the language chosen by the client for performance considerations), green screens remain unchanged (to support extensive screen pumper-scraper automation used by the business), IMS DB migrates to MS SQL (preferred RDBMS vendor by the client).

One year into this project, the client invited Luxoft to consult on QA estimations. It appeared that an initial disconnect on the required migration effort had led to a chain of false assumptions and commitments, resulting in a severe underestimation of the overall scope. All delays (QA involvement in particular) were considered as final efforts to complete the project. Eventually, our involvement helped take the application over the line. However, this was only possible as a result of the gradual introduction of our capabilities and progress transparency, one step at a time:

1. We started with estimating 15,000 test cases of medium and high complexity as a full system test coverage, and proposed a few test strategy alternatives that allowed for optimization of test effort. Our client gave us lead on test design and test automation.

2. After it was clear that we'd find more functional and performance defects than the transformation services provider could fix, we proposed our involvement in development activities and started fixing defects.

3. It was obvious that the initial, proposed approach from the transformation services provider was underestimated from a technical perspective: The proprietary runtime libraries required for the transformed code were experiencing serious performance hits, while fixing the IMS emulation on top of RDBMS to function properly would require a complete redesign. We proposed a whole new solution for migration and took over development leadership. Our solution was based on keeping business logic unchanged to eliminate functional risks with code transformation (Rehosting it), while picking up the DB emulation development at a scale that the project required (Rearchitecting it). The scope had additionally been extended to convert batch (Rearchitect), as this had been completely omitted from the initial body of work for the transformation services provider. Finally, a need to drive Rewrite/Replace/Rehost of other peripheral legacy application subcomponents (reporting, security, missing code, third-party utilities, etc.) had clearly transformed our approach to a Reengineer treatment.

4. Over time we started supporting the customer from integration and infrastructure perspectives.

5. Eventually we took control of the entire application migration (we were helping the customer up to this point) and we were asked to commit to a successful production delivery on a fixed price basis.

The mainframe was decommissioned after the migrated application had been implemented and in production for 2 weeks.

Apart from our delivery capabilities, what made the Luxoft Reengineer approach work? What is behind our solution?

Here is a quick outline of the legacy application itself and the approach from technology and methodology perspectives:

Legacy modernization complexity is best described by application statistics that picture it as a black box:

- **2M** COBOL SLOC
- **200** green screens
- **200** batch jobs with 16 hours of heavy batch every day
- **1300 IMS** segment types
- **60 TPS** (read-write) throughput
- **40** external interfaces

1. From the technology point of view, our solution involved:

- **OS**: Migrated from zOS to RHEL
- **COBOL**: Migrated all business logic as is, using GnuCOBOL compiler for Linux
- **JCL and JES2**: Auto-transformed JCL to Python (including IBM utilities) and created a Batch Framework with support of resilient output management capabilities
- **IMS DB**: Implemented IMSDB-to-SQL Adapter Layer and migrated data to RDBMS (MS SQL)
- **IMS DC**: Implemented transaction management layer, reused green screen GUI (TE3270, per customer request)
- **Performance**: Optimized performance by means of GnuCOBOL compiler optimization, application caching, DB performance optimization, business logic optimization
- **Reporting**: Migrated IBM CMOD from zOS to Win
- **Other languages**: Rewrote REXX to Python, Assembler to C/C++
- **Other COTS**: Migrated Syncsort, IBM MQ to new platform and back integrated to the application
- **Monitoring**: Established Splunk integration and created performance monitoring dashboards
- **DevOps**: Introduced robust CI/CD pipeline that involved Jenkins, Ansible, Kubernetes, Openshift, Terraform, Splunk, Grafana and for QA - HP QC, HP UFT, Python, Load Runner, Metabase

2. From the methodology point of view, our migration strategy included three steps:

1. We delivered **functional readiness** — migrated the application to the new platform and ensured it was functioning properly. This involved:

- Scope: Full migration scope definition, including inventory and engagement with all external stakeholders (~40 interfaces, ~20 user type groups/programs worldwide)
- Construction: POC, infrastructure, CI/CD and development activities related to migrated functionality enablement
- QA: Extensive test automation across several environment regions

2. Then we **focused on resilience** — ensured that the system was functional and performant under load. This involved:

- Scope: Non-functional requirements gathering and workload profile definition
- Construction: POC, infrastructure, CI/CD and development activities related to migrated functionality optimization
- QA: Test Automation Suite extension for load, concurrency and performance testing
- Support: Live Monitoring Solution for key application components

3. After successful production enablement, we provided **post-production hyper care** — ensuring the application was resilient in production in a 24/7 mode.

Finally, a major part of the success was the delivery excellence that had to be a primary ingredient for a complex journey like this. We tend to believe that modernization is full-scale engineering, not just integration of accelerating solutions. Therefore, best engineering practices are a must — including technology agnostic flexibility, robust delivery processes and reporting transparency. We conducted intermediate results report-outs and demos that brought reliability and faith in the positive outcome of our endeavor. And with DevOps practices introduced along the way, the application was finally ready to be integrated (with other applications) and be built upon. After 20 years of modernization attempts, the right, custom approach finally proved that every problem has a solution, and we're happy to be a part of it.

What to consider
if you're modernizing

As mentioned earlier, if you haven't migrated from a legacy system yet, then you've either fixed the legacy part of it and live with a modern, reliable mainframe that is fit for purpose, or your case has been too complicated to finish or even try. The potential risks to the business may be intolerable, you may have limited budgets, extreme technical complexity, a tight timeline, a lack of understanding of what is under the hood of the system or the migration approach or all of the above.

If you're about to start your migration journey and have even selected the best option for your case, make sure you have thought about, and have acceptable answers for, the questions below. Not all of them are going to be relevant to your case, but these questions are crucial and often overlooked during the early stages of migration:

- Do you clearly see your application development/sunset strategy for the next decade?
- Do you have a clear definition of done for your modernization?
- Are you investing enough time and budget into upfront solutioning and a solid PoC before committing to one option?
- Does your application have performance considerations? Will the chosen approach meet your requirements?
- Do you completely understand the consequences of your technology fundamentals paradigm shift — such as procedural to object-oriented business logic, hierarchical to relational data storage and access, transactional to event-driven processing, batch to microservice operations, local to network storage, workload and traffic footprint going to cloud, security concepts revision?
- When will you start transformation of your DevOps and run processes? How fast will you need to be able to deliver post-migration? How much training is needed to run a modernized application?
- Do you have a fallback plan in case things go wrong once live?
- How expensive and time-consuming will it be to maintain your target solution in 5–10 years?
- What will you do if the selected option doesn't work? Is there a plan B? C?
- Do you have a contingency plan if the company IT strategy changes in a couple years?
- Who will take care of the migration of side activities — performance testing, NFR testing, integration with external systems, etc.?

- In cases of automatic migration — will the vendor be capable of reconciling the migrated logic and fix defects in the new system?

- Does your vendor understand your business and the consequences of potential outages?

- If you're introducing a vendor solution, how aware are you of their product development roadmap? How much will their proprietary licensing cost in a decade?

- Do you have an integrator partner, or are you relying on a turnkey service from the migration-solution product vendor?

- Do you have a change management strategy for your application to be adjusted in parallel with migration activities?

- Does going to the cloud for you mean moving your data center into the sky, or are you considering cloud-ready solutions that unlock elasticity and other TCO optimization instruments? Have these considerations been backed by a business case and is it worth the effort?

- How are you planning to assure quality of the migrated product? Do you know what is good enough? Do you have a well thought through acceptance strategy?

- Is the business community backing your modernization plans? Are there any compromises that need to be agreed to with the business community?

We hope these questions help you see a clearer picture, save budget and enjoy your modernization journey. These projects are always a one-off if a complete success.

We've got lots to discuss

Clients come to Luxoft when they need to migrate a complex system that is critical for their day-to-day business, while keeping risks to a minimum. With more than 20 years of mainframe sustaining and migration experience, and a core pool of top-class engineers fluent in both legacy and modern technologies including cloud, QA and DevOps, we've helped the world's largest enterprises design and successfully accomplish custom/hybrid migrations where standard ways haven't worked.

About **the author**

Ivan Aptekarev
Chief Architect, Cross-Industry Solutions
Linkedin profile

20+ years in IT, driving long-term customer engagements as well as quick solution deliveries with main focus on legacy application/infrastructure modernization landscape and other digital enablement initiatives. Hands-on software development attitude with ability to provide leadership to large distributed high velocity engineering teams.

Alexey Zagorodniy
Director, Engineering Solutions
Linkedin profile

11+ years in IT, 6.5 in Financial Services. Worked in Russia, Poland, US and the UK. Specializes in defining, setting up and leading global delivery engagements split between onshore and offshore locations. Proposes and implements effective software delivery outsourcing solutions to address the needs and goals of Financial Services clients. Has a background in hands-on management. PMP, Prince2 and PMI-ACP certified. Passed CFA level 1 exam.

About Luxoft

Luxoft is the design, data and development arm of DXC Technology, providing bespoke, end-to-end technology solutions for mission-critical systems, products and services. We help create data-fueled organizations, solving complex operational, technological and strategic challenges. Our passion is building resilient businesses, while generating new business channels and revenue streams, exceptional user experiences and modernized operations at scale.

luxoft.com