eFramework for Technical Legacy System Assessments

This document outlines a simple, yet comprehensive, framework that can be used to assess a legacy software system in support of modernization and digital transformation efforts. This framework assesses legacy systems by following a People, Process, and Technology process (in that order).

A basic legacy assessment *can* be accomplished on a system in an isolated way, however, a more holistic legacy assessment—one that can anticipate areas of deeper focus and stand up better to entrenched industry scrutiny—is best accomplished when the organization understands its Peoples' needs, its Processes and the extent to which the legacy systems' limitations have shaped those processes, and clarity on how the current Technology is or is not meeting users' needs and enabling better business processes.

A strategy for modernization can then be conceived when that triad is well understood; assuming there exists a shared understanding and a strong buy-in of a vision for modernization and the organization knows "where the puck is going".

Phase 1.0 – People and Process

This first phase of the assessment does not spotlight the technology, instead, it focuses on the **People** and **Processes** behind the legacy system.

The **People** part of the equation focuses on understanding user needs and pain points. Rather than looking at a technical capability and asking if a system can or cannot do X, we begin by asking the users what it is that they need in a technology-agnostic way and what the current pain points of their current processes and tools are.

The **Process** part looks at key business processes, and we attempt to understand if the processes in place work well for the humans affected, or, as is often the case, whether they are not serving the organization well enough and merely exist as workarounds because of limitations of the legacy system. The true reasons why an organization has certain processes are often unknown and, in our experience, can often be traced back to a legacy technical reason. In imagining the vision for a new system it's important that we go back to the People

and Processes at play or we risk architecting a new system that brings along many of the shortcomings of the one it is meant to replace.

A study of user needs allows us to begin to formulate a needs assessment and begin to shape a vision for what a “modernized” system might look like, as well as expose critical human factors that a digital transformation strategy would need to address in order to succeed.

Phase 1.1 – Legacy System Assessment

Following People and Process is Technology. The system can be analyzed across the following dimensions to give us an understanding of the high-level components that make up the system. This analysis also surfaces some of the trade-offs and limitations of specific qualities of the system.

Architecture

Dimension	**Definition / Answers Sought (illustrative)**
Application Integrations	How well do applications integrate with themselves? With other systems?
End-User Consumption	How do the users interact with the system? Are UIs intuitive, or easy to iterate on? What volume of errors do users report, and does the system enable quick fixing and deploying fixes?
Extensibility	How extensible are systems and applications that make up the system? What is the average cost (in time) of extending the system to do new things?
Interoperability	How easy is it for the system to exchange information with other systems? What protocols and schemas does the system make use of? Are they standards-compliant (eg. RESTful JSON/XML, gRPC/Thrift) or proprietary/esoteric?
Data Sharing & Portability	Is data stored and accessible by the system in a way that enables easy portability of this data to other systems (eg. Other services, reporting systems, applications)?

Code (Languages & Frameworks)	What languages and frameworks are used? Is the market size for expertise in these languages and frameworks diminishing or expanding? Is community support and documentation for these languages and frameworks readily available and up to date? Does the programming language used “fit” the problem it’s solving or are potential solutions constrained and shaped around the language/framework?
Adequacy	How adequate is the architecture in supporting people’s needs and business processes? How readily can the system adapt to new users’ needs?

Maintenance and Support

Dimension	**Definition / Answers Sought (illustrative)**
Infrastructure:	
- Hardware Maintenance	Does the system run on commodity hardware, or does it require specialized/expensive/hard to obtain hardware?
- Network Maintenance	What are the network topology and overall network requirements of operating the system? Can network requirements be accomplished with commodity hardware?
- Software Maintenance	What are the software requirements for operating this system (from Operating System up to Application Code)? Are the software packages maintained and versions up-to-date? What is the relative cost of upgrading software and libraries that the system relies on? Is software written in a way that is easily and repeatedly testable?
- Database Maintenance	Is the database system a modern one, or a legacy system (eg. IBM Db2)? Is the database relational, document-based (nosql), hierarchical, combination and does its structure match the problem it’s solving?

Security	What does the security landscape look like for the system? Are 3rd party software package vulnerabilities quickly discovered and patches made readily available? Are systems proprietary or do they have a large open-source community that works to find and patch vulnerabilities? Are the needs of the system such that the network is easy to monitor and secure? Does all inter-system communication happen over TLS?
Availability and Costs of Skills/Expertise	Applicable to all areas above: Is the pool of personnel that has the experience and skills to continue building on and maintaining all of the subsystems increasing or decreasing? Will the languages, frameworks, and databases currently used still be supported and have a large active community 5, 10, 15 years from now?
Licensing Costs	Applicable to all areas above: Are there licensing costs for any systems? If so, are there alternative open-source solutions or options with lower licensing costs? Are licensing decisions approached methodically or are they a result of vendor lock-in?

Organization

Dimension	**Definition / Answers Sought (illustrative)**
Technical Maturity	How mature is the organization at building and delivering technical solutions? Are software, systems, and network engineering internal or outsourced?
Executive Sponsorship	Does the legacy system have entrenched support for maintaining the status quo or support for modernizing?
Readiness for Change	How ready are leaders in the organization for this change? Are users willing to endure change for a better system, or are they happy and effective with their current systems and workarounds?

Regulatory Compliance	Does the current system conform to all necessary regulatory requirements? Does your current system take advantage of recent developments in regulatory requirements? How readily can the system adapt to a changing regulatory landscape?

Phase 2-3 – Modernization Approaches & Implementation

Once we have sufficient information from the People, Process, and Technology phases, we can begin to create a strategy for an approach to modernizing a system.

The first question that should be answered is “Why?” What are the key drivers of digital modernization? Are there specific user needs, regulatory compliance concerns, security vulnerabilities, or other drivers of digital transformation? Clarity around this will enable our strategy when picking which parts of the component to begin to work on and help guide buy vs build decisions.

In general, modernizing a legacy system can be accomplished in the following ways:

- **Encapsulate:** Wrapping with a more modern framework that exposes modern means of communicating with a legacy system via new APIs. Useful when the inner workings of a system are a “black box” that we don’t have the expertise to change directly.
- **Replatform/Refactor/Rebuild:** Rebuild or rewrite the application component from scratch while preserving its scope and external interfaces.
- **Replace:** Eliminate the component altogether and replace it, building in new requirements and user needs, and altering dependent components to work with this new one (often via interim layers of abstractions).

In practice, an Agile modernization of a large enough system will include all of these methods happening simultaneously and combined in creative ways to make incremental, forward progress.

The *actual* implementation of these strategies is beyond the scope of this framework...

This approach is informed by our hands-on experience working on modernization efforts in the federal government and the private sector, and draws inspiration from resources like the following:

"IEEE, Towards a Framework to Assess Legacy Systems", https://ieeexplore.ieee.org/document/6721915

"Legacy System Modernization: How to Transform the Enterprise for Digital Future", https://www.altexsoft.com/whitepapers/legacy-system-modernization-how-to-transform-the-enterprise-for-digital-future/

"An assessment strategy for identifying legacy system evolution requirements in eBusiness context", http://csis.pace.edu/~marchese/CS775/Proj1/aversano_legacy.pdf

"Dropbox: How we rolled out one of the largest Python 3 migrations ever", https://blogs.dropbox.com/tech/2018/09/how-we-rolled-out-one-of-the-largest-python-3-migrations-ever/

"Managing legacy system costs: A case study of a meta-assessment model to identify solutions in a large financial services company", https://www.sciencedirect.com/science/article/pii/S2210832716301260

"Systems @Scale 2018 – Rebuilding the Inbox: The Corporate and Cultural Adventures of Modernizing Yahoo Mail", https://atscaleconference.com/videos/systems-scale-2018-rebuilding-the-inbox-the-corporate-and-cultural-adventures-of-modernizing-yahoo-mail/

Victor's musings and existential questions related to RPMS:
Can RPMS be modernized? The answer, I feel, is simultaneous Yes and No; depending on the definition of "modernized".

Although mostly a semantics question, a shared understanding and some agreement on the answers to these existential questions can help:
At what point is RPMS no longer considered to be "RPMS"?
Would we still consider the system to be "RPMS":

- If we replace the database layer but keep the user interfaces the same?
- If we replace individual middle-tier applications with new services written in languages other than MUMPS, but leave the database and the UI elements untouched?
- If we systematically refresh user interfaces to fit user needs, leaving other layers untouched?