PharmaSUG 2026 - Paper ET-390

A Practical Roadmap for Modernizing Legacy Clinical Applications

David Ward, Triam Ltd

ABSTRACT

Many pharmaceutical organizations depend on long-standing analytical applications that are increasingly difficult to maintain, validate, and extend. While these systems often remain mission-critical, accumulated technical debt can hinder productivity, increase operational risk, and limit the adoption of modern analytical techniques. This paper presents a practical, incremental roadmap for modernizing legacy clinical applications while maintaining continuity with existing SAS-based and regulated workflows.

The discussion begins with guidance on project planning and assessing modernization readiness, then moves to strategies for tool and language selection. Attendees will gain a clearer understanding of how alternative products and open-source languages can be evaluated and introduced alongside established platforms, minimizing disruption and regulatory risk. The paper also explores software tools that support the migration process itself, including capabilities for documentation generation, code discovery and cataloging, dependency management, and integration with version control, release, and deployment practices.

This paper is intended for data engineers, statistical programmers, and technical leads responsible for maintaining or modernizing legacy analytical systems. Participants will leave with a clear, actionable framework for planning modernization efforts that balance innovation, operational stability, and regulatory confidence.

INTRODUCTION: THE LEGACY REALITY IN PHARMA

Within pharmaceutical and clinical research organizations, analytical applications often persist far beyond their originally intended lifecycle. Systems developed years or even decades ago continue to support critical workflows in data preparation, statistical analysis, reporting, and regulatory submission. These applications, frequently built on SAS and tightly coupled to validated processes, remain deeply embedded in operational and compliance frameworks.

Their longevity is not accidental. Legacy clinical applications endure because they encode institutional knowledge, have undergone extensive validation, and are trusted to produce reproducible results in highly regulated environments. Replacing such systems outright introduces nontrivial risk, particularly when regulatory submissions, audit trails, and historical comparability must be preserved.

At the same time, these systems accumulate technical debt. Over time, incremental modifications, evolving requirements, and ad hoc extensions produce codebases that are increasingly difficult to maintain and extend. Common symptoms include tightly coupled logic, undocumented dependencies, brittle execution paths, and workflows that rely heavily on implicit knowledge held by a small number of experienced contributors. As a result, organizations face growing challenges in onboarding new team members, adapting to changing analytical demands, and integrating modern tools or methodologies.

This tension—between the stability required by regulated workflows and the flexibility demanded by modern analytics—defines the legacy reality in clinical environments. Modernization is therefore not optional; however, it must be approached with precision. Aggressive rewrites or wholesale platform replacements may introduce unacceptable operational and regulatory risk, while inaction allows complexity to compound.

A more viable approach is incremental modernization: a deliberate, staged process that improves maintainability, extensibility, and interoperability without disrupting validated workflows. This paper presents a practical roadmap for such efforts, focusing on how organizations can assess readiness, define appropriate modernization strategies, and introduce new tools and practices alongside existing systems in a controlled and auditable manner.

DEFINING "MODERNIZATION"

Modernization is often conflated with wholesale system replacement or large-scale rewrites; however, in regulated clinical environments, these approaches are frequently impractical or impossible. A more useful definition considers modernization as the **progressive improvement of an existing system's structure, maintainability, and interoperability while preserving its validated behavior**.

This distinction is critical. Rewriting an application from scratch may offer architectural clarity, but it also resets validation status, introduces uncertainty in reproducibility, and requires substantial re-verification effort. Similarly, full platform replacement can disrupt established workflows and necessitate retraining, process redefinition, and regulatory realignment.

Modernization, by contrast, emphasizes **continuity with controlled evolution**. Existing systems are not discarded but incrementally improved through targeted interventions. These may include modularizing monolithic codebases, externalizing configuration, introducing version control, standardizing execution patterns, or integrating complementary tools that extend existing capabilities.

A key dimension of modernization is the distinction between **incremental improvement and "big bang" migration**.

- *Big bang approaches* attempt to replace an entire system in a single transition, often requiring parallel validation and extensive coordination. While theoretically clean, these efforts are high-risk and frequently stall or fail due to scope, resource constraints, or shifting requirements.
- *Incremental approaches* break modernization down into manageable units—individual programs, modules, or workflow stages—allowing organizations to deliver value progressively while limiting risk exposure.

In practice, incremental modernization often results in **hybrid environments** in which legacy SAS-based workflows coexist with newer tools and languages. Rather than viewing this coexistence as transitional, it should be treated as a stable and intentional architecture. SAS remains essential for validated reporting and regulatory deliverables, while open-source tools such as Python or R may be introduced for tasks such as data exploration, feature engineering, or advanced modeling.

The objective, therefore, is not to eliminate legacy systems, but to **reposition them within a broader, more flexible analytical ecosystem**. Modernization succeeds when systems become easier to understand, safer to modify, and more capable of integrating with emerging tools—without compromising the reliability and traceability that clinical environments require.

ASSESSING MODERNIZATION READINESS

Modernization efforts frequently fail not because of flawed implementation, but because they begin at the wrong time, target the wrong systems, or proceed without sufficient organizational alignment. Before selecting tools or defining a roadmap, teams must first determine whether they are positioned to modernize effectively—and, equally important, where to begin.

Modernization readiness can be evaluated across three dimensions: **application suitability, technical characteristics, and organizational context**. Taken together, these factors provide a structured basis for identifying candidate systems and defining an appropriate scope of effort.

IDENTIFYING CANDIDATE APPLICATIONS OR MODULES

Not all legacy applications are equally suitable for modernization. Some systems are deeply stable, rarely modified, and tightly bound to validated processes; others are actively evolving, operationally burdensome, or increasingly misaligned with current analytical needs. Effective modernization begins by targeting the latter.

Candidate applications typically exhibit one or more of the following characteristics:

- **High change frequency**
  Applications that require frequent updates—whether due to evolving study designs, data standards, or reporting requirements—tend to accumulate complexity quickly and benefit most from improved structure and tooling.
- **Operational fragility**
  Systems that fail unpredictably or depend on manual intervention are strong candidates for refactoring and workflow standardization.
- **Limited transparency**
  Applications with unclear data flow, implicit dependencies, or minimal documentation create risk in both maintenance and validation contexts.
- **Onboarding difficulty**
  If new team members require extensive guidance to understand or modify a system, this is a strong signal that structural improvements are needed.
- **Partial reuse potential**
  Modules that are replicated across studies or projects—often with minor variations—are particularly valuable targets, as modernization can yield immediate efficiency gains through standardization.

Rather than attempting to modernize entire systems at once, organizations should prioritize **bounded components**—individual programs, pipelines, or functional modules—where improvements can be delivered incrementally and validated independently.

TECHNICAL INDICATORS

Once candidate applications are identified, their technical characteristics should be evaluated to determine both **complexity** and **modernization effort**. The goal is not to produce an exhaustive audit, but to identify structural properties that will influence feasibility and timing.

Key technical indicators include:

- **Code complexity and structure**
  Large, monolithic programs with deeply nested logic, extensive macro usage, or limited modularization are more difficult to modify safely. However, they may also offer high returns if decomposed into smaller, well-defined components.
- **Dependency profile**
  Applications that rely on external data sources, shared libraries, or tightly coupled upstream/downstream processes require careful coordination. Hidden or undocumented dependencies are particularly problematic.
- **Data flow clarity**
  Systems in which inputs, intermediate datasets, and outputs are not clearly defined—or are distributed across multiple locations—are harder to reason about and validate.
- **Execution model**
  Workflows that depend on manual steps, implicit ordering, or environment-specific behavior (e.g., hard-coded paths, session state) introduce fragility and complicate automation.
- **Performance constraints**
  In some cases, legacy systems persist because they are performant and well-optimized for existing workloads. Modernization efforts should account for whether performance must be preserved or can be improved through alternative approaches.

These indicators should be used to estimate both risk and effort, allowing teams to plan modernization work in a way that delivers early value without destabilizing critical systems.

ORGANIZATIONAL INDICATORS

Even when a system is technically suitable for modernization, success depends heavily on organizational readiness. Modernization introduces new tools, practices, and responsibilities, all of which require alignment across teams.

Key organizational indicators include:

- **Skill availability**
  Teams must have, or be willing to develop, proficiency in any new languages, tools, or development practices introduced during modernization.
- **Stakeholder alignment**
  Business, statistical, and regulatory stakeholders must share a common understanding of modernization goals, particularly with respect to risk tolerance and expected outcomes.
- **Support for process change**
  Modernization often entails adopting version control, structured testing, automated workflows, or new documentation practices. Organizations must be prepared to support these changes operationally.
- **Time and resource allocation**
  Incremental modernization still requires dedicated effort. Without explicit allocation of time and resources, efforts tend to stall or be deprioritized in favor of immediate deliverables.
- **Validation and compliance readiness**
  In regulated environments, teams must be prepared to document changes, justify decisions, and support validation activities associated with modified workflows.

A lack of readiness in any of these areas does not preclude modernization, but it does suggest that initial efforts should be scoped more narrowly and focused on building capability alongside technical improvement.

READINESS CHECKLIST

To support practical decision-making, the following checklist can be used to assess whether a given application—or organization—is prepared for modernization:

Application Readiness

- The system requires frequent updates or exhibits operational friction
- Core workflows can be broken down into discrete, testable components
- Data inputs, outputs, and dependencies can be identified with reasonable effort

Technical Readiness

- Code structure, while complex, is accessible and modifiable
- Dependencies are known or can be discovered
- Execution can be observed, replicated, and incrementally refactored

Organizational Readiness

- Team members have, or can develop, relevant technical skills
- Stakeholders support incremental change over wholesale replacement
- Time and resources are allocated for modernization activities
- Validation and documentation processes can accommodate change

Applications that meet most of these criteria are strong candidates for modernization. Those that do not may still be addressed, but should be approached more cautiously, often beginning with documentation, dependency discovery, or limited refactoring efforts.

FROM ASSESSMENT TO ACTION

The outcome of a readiness assessment is not simply a binary decision, but a **prioritized modernization backlog**. By evaluating systems across application, technical, and organizational dimensions, teams can:

- Identify high-value, low-risk starting points
- Order work to deliver incremental improvements
- Align modernization efforts with broader organizational capabilities

This structured approach ensures that modernization begins with achievable, well-understood targets, establishing momentum and reducing the likelihood of stalled or abandoned initiatives.

BUILDING A TECHNOLOGY ROADMAP

Once candidate systems have been identified and readiness assessed, the next step is to define a structured roadmap for modernization. In regulated clinical environments, this roadmap must balance competing priorities: improving flexibility and maintainability while preserving validated behavior, ensuring traceability, and minimizing disruption to ongoing studies.

A practical roadmap is not defined by a single end state, but by an **ordered progression of capabilities**. Rather than attempting to modernize all aspects of a system simultaneously, organizations should define short-, medium-, and long-term objectives that align with both technical feasibility and organizational readiness.

DEFINING TIME HORIZONS

A useful way to structure modernization efforts is through three overlapping time horizons, each representing a different level of change and associated risk.

Short-Term: Stabilization and Visibility

Short-term efforts focus on improving understanding and control of existing systems without altering core behavior. The objective is to reduce operational risk while laying the groundwork for future changes.

Typical activities include:

- Introducing **version control** for legacy codebases
- Establishing **consistent project structures** and naming conventions
- Documenting **data flows, dependencies, and execution steps**
- Capturing and centralizing **logs and outputs**
- Eliminating environment-specific assumptions (e.g., hard-coded paths)

These changes are generally low-risk from a validation perspective, as they do not alter analytical logic but instead improve transparency and reproducibility.

Medium-Term: Structural Improvement and Integration

Medium-term efforts focus on improving the internal structure of applications and enabling interoperability with additional tools and languages.

Typical activities include:

- **Modularizing monolithic programs** into discrete, testable components
- Introducing **parameterization and configuration management**
- Standardizing **execution workflows** (e.g., pipelines, orchestration scripts)
- Integrating **complementary tools** (e.g., Python or R for specific tasks)
- Establishing **automated validation checks** where appropriate

At this stage, systems begin to evolve in meaningful ways, and coordination with validation and quality teams becomes increasingly important.

Long-Term: Architectural Evolution

Long-term efforts address broader architectural concerns, enabling systems to support new analytical paradigms and scale more effectively.

Typical activities include:

- Decoupling data processing, analysis, and reporting layers
- Introducing service-based or pipeline-oriented architectures
- Enabling scalable execution environments (e.g., distributed or cloud-based systems)
- Standardizing interfaces between tools and components
- Aligning with enterprise-level data and platform strategies

These changes may require more substantial validation effort and organizational alignment, but they position systems for sustained adaptability.

LANGUAGE AND TOOL SELECTION CRITERIA

Selecting appropriate tools and languages is a critical component of modernization, particularly in environments where regulatory expectations constrain both implementation and change management. Rather than selecting tools based solely on popularity or perceived capability, organizations should evaluate them against a consistent set of criteria:

- **Fitness for purpose**
  The tool should address a clearly defined need—such as data manipulation, statistical modeling, or workflow orchestration—more effectively than existing approaches.
- **Interoperability**
  The tool must integrate with existing systems, including SAS-based workflows, data sets, and reporting pipelines.
- **Reproducibility and traceability**
  It must be possible to document, version, and reproduce results generated by the tool in a manner consistent with regulatory expectations.
- **Operational supportability**
  The organization must be able to install, manage, and maintain the tool, including handling dependencies, updates, and security considerations.
- **Team capability**
  The learning curve associated with the tool should be aligned with team skill levels or supported by training plans.
- **Validation impact**
  The introduction of the tool should be evaluated in terms of its effect on validation scope, including documentation and testing requirements.

This criteria-driven approach helps prevent tool sprawl and ensures that each addition to the ecosystem contributes measurable value.

WHEN OPEN SOURCE MAKES SENSE—AND WHEN IT DOES NOT

Open-source tools such as Python and R have become central to modern analytics, offering extensive libraries and active ecosystems. However, their adoption in regulated environments requires careful consideration.

Open source is often well-suited for:

- Exploratory data analysis and prototyping
- Advanced modeling or statistical methods not readily available in SAS
- Data transformation tasks that benefit from flexible libraries

In these contexts, open-source tools can complement existing workflows without immediately impacting validated outputs.

However, open source may be less appropriate when:

- Strict validation requirements apply to final outputs
- Long-term support and stability are critical
- Dependency management introduces unacceptable operational risk
- Auditability and traceability cannot be easily established

In practice, successful modernization strategies do not position open source as a replacement for SAS, but as a **complementary capability**. SAS remains central to validated reporting and submission workflows, while open-source tools extend analytical flexibility in upstream or non-regulated stages.

AVOIDING TOOL PROLIFERATION

One of the common failure modes in modernization efforts is uncontrolled expansion of tools and technologies. While introducing new capabilities is often necessary, each additional tool increases cognitive load, operational complexity, and support requirements.

To mitigate this risk, organizations should:

- Establish a **curated set of approved tools and languages**
- Define **clear use cases** for each tool
- Standardize **integration patterns** between systems
- Periodically review and rationalize the technology stack

The objective is not to minimize tools at all costs, but to ensure that each tool has a defined role within a coherent architecture.

FROM ROADMAP TO EXECUTION

A well-defined roadmap provides more than a list of initiatives—it establishes a **series of achievable steps** aligned with both technical and organizational readiness.

Effective roadmaps:

- Prioritize **high-impact, low-risk improvements** early
- Deliver **incremental value** rather than deferring benefits
- Align with **validation and compliance processes**
- Provide **clear decision points** for advancing to more complex changes

By structuring modernization as a progression rather than a single transformation event, organizations can improve legacy systems in a controlled, sustainable manner while maintaining the stability required in clinical environments.

REGULATORY AND COMPLIANCE CONSIDERATIONS

In clinical and pharmaceutical environments, modernization efforts must be evaluated not only in terms of technical merit, but also in terms of their impact on regulatory compliance. Analytical systems are often embedded within validated workflows that support regulatory submissions, internal quality processes, and external audits. Any change to these systems—whether structural or functional—must therefore preserve traceability, reproducibility, and documented control.

Modernization does not eliminate these requirements; rather, it introduces new considerations, particularly in hybrid environments where legacy and modern tools coexist. A successful approach treats compliance not as a constraint to be worked around, but as a design requirement that informs how modernization is planned and executed.

TRACEABILITY AND REPRODUCIBILITY

Traceability is foundational to regulated analytics. It must be possible to demonstrate how data moves from source to output, how transformations are applied, and how results are generated. In legacy systems, this traceability is often implicit—encoded in established workflows and understood by experienced users.

Modernization efforts should make this traceability **explicit and systematized**.

Key practices include:

- **End-to-end data lineage**
  Clearly defining inputs, intermediate datasets, and outputs, along with the transformations that connect them.
- **Version-controlled code and artifacts**
  Ensuring that all scripts, configuration files, and dependencies are versioned and recoverable.
- **Deterministic execution**
  Designing workflows such that repeated execution with the same inputs produces the same outputs, regardless of environment.
- **Centralized logging and metadata capture**
  Recording execution details, including parameters, timestamps, and runtime context.

In hybrid environments, where SAS workflows coexist with Python or R components, traceability must extend across language boundaries. This requires consistent conventions for naming, data exchange, and execution sequencing, as well as clear documentation of how components interact.

VALIDATION AND DOCUMENTATION EXPECTATIONS

Validation requirements vary by organization and use case, but generally focus on demonstrating that systems perform as intended and produce reliable, reproducible results. Modernization introduces two primary validation considerations:

- **Change classification**
  Not all changes carry the same validation burden. Refactoring code without altering logic may require minimal validation, while introducing new tools or altering data transformations may require more extensive verification.
- **Scope control**
  Incremental modernization allows validation to be scoped to specific components rather than entire systems. This reduces effort and risk, provided that boundaries between validated and modified components are clearly defined.

Documentation plays a central role in both areas. Modernized systems should improve—not degrade—the quality and accessibility of documentation.

Effective practices include:

- Maintaining **clear specifications** of inputs, outputs, and processing steps
- Documenting **assumptions and transformation logic**
- Recording **changes and their rationale** in a structured manner
- Aligning documentation with **validation artifacts** such as test plans and results

Where possible, documentation should be generated or maintained alongside code, reducing the risk of divergence between implementation and description.

SUPPORTING AUDITS IN HYBRID ENVIRONMENTS

Audits—whether internal or external—require organizations to demonstrate control over their analytical processes. In hybrid environments, this can be more complex, as workflows may span multiple languages, tools, and execution environments.

To support auditability, modernization efforts should ensure that:

- **Workflow boundaries are clearly defined**
  Each stage of processing—regardless of language—has a well-understood role and interface.
- **Execution can be reconstructed**
  It must be possible to reproduce a given result by identifying the exact versions of code, data, and configuration used.
- **Tool usage is governed**
  The introduction of new tools is documented, justified, and incorporated into standard operating procedures where appropriate.
- **Access and control mechanisms are consistent**
  Permissions, audit trails, and change controls apply uniformly across legacy and modern components.

A common concern is that introducing open-source tools will reduce auditability. In practice, the opposite can be true—provided that these tools are integrated within a controlled framework that enforces versioning, logging, and reproducibility.

DESIGNING FOR COMPLIANCE

Rather than treating compliance as a downstream validation activity, modernization efforts should incorporate compliance considerations into system design.

This includes:

- Structuring workflows to make **data lineage explicit**
- Standardizing **interfaces between components**
- Embedding **logging and metadata capture** into execution processes
- Using **version control and release practices** that align with audit expectations

By designing systems with these principles in mind, organizations can reduce the incremental effort required for validation and audit support, while improving overall system transparency.

BALANCING INNOVATION AND CONTROL

A recurring tension in modernization efforts is the balance between adopting new tools and maintaining regulatory control. Overly restrictive approaches can limit innovation and perpetuate legacy constraints, while overly permissive approaches can introduce unacceptable risk.

A practical balance can be achieved by:

- Limiting new tools to **well-defined use cases**
- Introducing changes in **non-critical or upstream components first**
- Establishing **clear governance for tool adoption and usage**
- Expanding scope gradually as confidence and experience increase

This approach allows organizations to benefit from modern capabilities while maintaining the rigor required in clinical environments.

MIGRATION TOOLS

Modernizing legacy clinical applications is not solely a matter of rewriting or restructuring code; it is fundamentally an exercise in **understanding, controlling, and evolving complexity**. Migration tools play a critical role in this process by making existing systems more observable, enabling structured change, and supporting the transition to more maintainable architectures.

Rather than focusing on specific products, it is more useful to consider tooling in terms of the capabilities required at different stages of modernization: understanding what exists, defining what is needed, and managing complexity over time.

UNDERSTANDING WHAT EXISTS

A primary challenge in legacy environments is incomplete or implicit knowledge of how systems operate. Before meaningful modernization can occur, teams must establish a clear view of the current state.

Key tooling capabilities include:

- **Code discovery and cataloging**
  Tools that scan code repositories or file systems to identify programs, scripts, and associated assets, providing an inventory of what exists and where it resides.
- **Dependency analysis**
  Static or dynamic analysis tools that identify relationships between programs, datasets, libraries, and external systems. These are particularly valuable in SAS environments where dependencies may be embedded in macros or external includes.
- **Data lineage visualization**
  Tools that map how data flows through a system, from source inputs to final outputs, highlighting transformation steps and intermediate artifacts.
- **Documentation generation**
  Automated extraction of metadata, comments, and structural information to produce baseline documentation for legacy systems.

These capabilities transform opaque systems into observable ones, reducing reliance on tribal knowledge and enabling more informed decision-making.

UNDERSTANDING WHAT IS REQUIRED

Once the current state is understood, the next step is to define the target structure and the changes required to achieve it. Tooling at this stage supports planning, design, and controlled transformation.

Key capabilities include:

- **Refactoring support**
  Tools that assist in restructuring code—such as modularizing programs, extracting reusable components, or standardizing interfaces—while minimizing the risk of introducing defects.
- **Cross-language integration frameworks**
  Mechanisms for orchestrating workflows that span SAS, Python, R, and SQL, including the ability to pass data and control execution across components in a consistent manner.
- **Configuration and parameter management**
  Tools that externalize environment-specific settings, reducing reliance on hard-coded values and enabling more flexible execution.
- **Test harnesses and validation utilities**
  Frameworks that allow teams to compare outputs before and after changes, supporting incremental validation and regression testing.

At this stage, tooling enables teams to move from analysis to action, providing guardrails that make incremental change feasible in regulated environments.

MANAGING FUTURE COMPLEXITY

Modernization is not a one-time event; without appropriate controls, newly modernized systems can accumulate complexity just as legacy systems did. Tooling must therefore support ongoing maintainability and governance.

Key capabilities include:

- **Version control and change tracking**
  Systems that record changes to code, configuration, and documentation, providing traceability and supporting collaborative development.
- **Workflow orchestration**
  Tools that define and manage execution pipelines, ensuring consistent sequencing, error handling, and logging across multi-language workflows.
- **Environment and dependency management**
  Mechanisms for controlling library versions, runtime environments, and system configurations, reducing variability across executions.
- **Automated documentation and metadata capture**
  Continuous generation of documentation and execution metadata, ensuring that systems remain understandable over time.
- **Monitoring and observability**
  Tools that provide visibility into system behavior during execution, enabling rapid identification of failures or performance issues.

These capabilities ensure that modernization efforts produce systems that remain maintainable, auditable, and adaptable as requirements evolve.

TOOLING IN REGULATED CONTEXTS

In clinical environments, tooling must be evaluated not only for functionality, but also for its impact on compliance and validation.

Considerations include:

- **Traceability support**
  The tool should enable or enhance the ability to track changes, execution history, and data lineage.
- **Reproducibility**
  It must be possible to recreate results using the tool in a controlled and documented manner.
- **Audit readiness**
  Outputs generated by the tool—logs, reports, metadata—should support audit and inspection processes.
- **Controlled adoption**
  Introduction of new tools should follow defined governance processes, including documentation, training, and validation where required.

In many cases, general-purpose tools can be used effectively in regulated environments, provided they are integrated within a framework that enforces these controls.

AVOIDING OVER-RELIANCE ON TOOLING

While tooling is essential, it is not a substitute for sound architecture or disciplined process. A common failure mode in modernization efforts is the assumption that adopting new tools will, by itself, resolve underlying structural issues.

To avoid this:

- Tools should be selected to support **clearly defined objectives**, not as exploratory additions
- Architectural decisions should precede tooling decisions wherever possible
- Teams should avoid introducing tools that duplicate existing capabilities without clear benefit
- Tool usage should be standardized to prevent fragmentation

The role of tooling is to **enable and reinforce modernization strategies**, not to define them.

LESSONS LEARNED AND COMMON PITFALLS

Modernization initiatives in clinical environments rarely fail due to lack of technical capability. More often, they stall or regress because of misaligned expectations, insufficient scoping, or an underestimation of organizational constraints. The following lessons reflect recurring patterns observed in real-world efforts and highlight where teams should exercise particular discipline.

TREATING MODERNIZATION AS A ONE-TIME EVENT

One of the most common pitfalls is framing modernization as a discrete project with a defined endpoint. This often leads to overly ambitious “big bang” initiatives that attempt to replace entire systems within a fixed timeframe. In practice, such efforts struggle under their own complexity. Requirements evolve, validation scope expands, and dependencies emerge that were not visible at the outset. As a result, timelines slip, confidence erodes, and teams may revert to maintaining legacy systems alongside partially completed replacements.

> Lesson**:**
> Modernization should be treated as a continuous process rather than a one-time transformation. Progress is best measured through incremental improvements—reduced friction, improved clarity, and increased adaptability—rather than complete system replacement.

UNDERESTIMATING LEGACY SYSTEM COMPLEXITY

Legacy clinical applications often appear simpler than they are. Years of incremental changes, implicit assumptions, and embedded domain knowledge create systems whose true complexity is not immediately visible. Teams that begin modernization without fully understanding existing dependencies, data flows, and edge cases frequently encounter unexpected behavior during refactoring or migration. This can lead to rework, validation challenges, or loss of trust in the modernization effort.

> Lesson**:**
> Invest early in discovery. Code cataloging, dependency analysis, and data lineage mapping are not optional overhead—they are prerequisites for safe and effective change.

OVER-RELIANCE ON REWRITES

The appeal of a clean rewrite is strong, particularly when legacy code is difficult to maintain. However, rewrites introduce substantial risk in regulated environments, where reproducing validated behavior is both necessary and nontrivial. Even when technically successful, rewrites often require extensive re-validation, delaying delivery of value and increasing organizational resistance.

> Lesson**:**
> Favor incremental refactoring over wholesale replacement. Where rewrites are necessary, constrain them to well-defined, low-risk components and validate them independently.

TOOL ADOPTION WITHOUT CLEAR USE CASES

Introducing new tools or languages without clearly defined roles can lead to fragmentation rather than simplification. Teams may experiment with multiple approaches, resulting in inconsistent patterns, duplicated functionality, and increased maintenance burden. This issue is particularly acute with open-source tools, where flexibility can lead to unstructured adoption.

> Lesson**:**
> Each tool should have a clearly defined purpose within the architecture. Adoption should be governed, documented, and aligned with specific use cases rather than individual preference.

IGNORING ORGANIZATIONAL READINESS

Technical modernization often outpaces organizational readiness. Teams may introduce new tools, workflows, or practices without sufficient training, documentation, or stakeholder alignment. This disconnect can manifest as resistance to change, inconsistent adoption, or reliance on a small group of individuals to maintain modernized components.

> Lesson**:**
> Modernization must include capability development. Training, documentation, and process alignment are integral to success, not ancillary concerns.

EXPANDING SCOPE TOO QUICKLY

Early successes in modernization can create momentum, but they can also encourage teams to expand scope prematurely. Attempting to modernize multiple systems simultaneously or introducing large-scale architectural changes too early can dilute focus and increase risk.

> Lesson**:**
> Scale deliberately. Use early efforts to establish patterns, validate approaches, and build confidence before expanding to broader systems.

NEGLECTING VALIDATION STRATEGY

In regulated environments, validation is sometimes treated as a downstream activity, addressed only after technical changes are implemented. This approach can result in unexpected rework, documentation gaps, or delays in deployment.

> Lesson**:**
> Incorporate validation planning into modernization efforts from the outset. Define how changes will be tested, documented, and approved before implementation begins.

FAILING TO DEFINE SUCCESS CRITERIA

Modernization efforts can lose direction if success is not clearly defined. Without measurable outcomes, teams may struggle to demonstrate value, secure continued support, or determine when a phase of work is complete.

> Lesson**:**
> Define success in practical terms. Examples include reduced onboarding time, improved execution reliability, clearer data lineage, or decreased time to implement changes.

FROM PITFALLS TO PRACTICE

Taken together, these lessons reinforce a central theme: successful modernization is less about selecting the right tools and more about **applying disciplined, incremental change within a constrained environment**.

Organizations that succeed tend to:

- Start with well-scoped, high-value targets
- Invest in understanding existing systems before modifying them
- Introduce new tools deliberately and with clear purpose
- Align technical changes with organizational capability and regulatory requirements
- Measure progress through incremental improvements rather than transformational milestones

By avoiding common pitfalls and maintaining a structured, incremental approach, teams can modernize legacy clinical applications in a way that improves flexibility and maintainability while preserving the stability and compliance that these environments demand.

CONCLUSION

Modernizing legacy clinical applications is often framed as a technical challenge, but in practice it is an exercise in balancing competing priorities: stability and change, compliance and flexibility, continuity and evolution. The systems in question are not merely outdated codebases; they are deeply embedded components of regulated workflows that have been shaped by years of validation, operational use, and institutional knowledge.

For this reason, modernization cannot be approached as wholesale replacement. Attempts to impose large-scale transformation without regard for existing constraints frequently introduce more risk than they resolve. At the same time, deferring modernization allows technical debt to accumulate, increasing fragility and limiting the ability to adopt new analytical methods.

The approach outlined in this paper positions modernization as a **structured, incremental process**. By assessing readiness across technical and organizational dimensions, defining a sequenced roadmap, and introducing tools and practices deliberately, organizations can improve legacy systems without disrupting validated workflows. This approach acknowledges that hybrid environments—where SAS-based systems coexist with newer tools and languages—are not transitional artifacts, but a stable and necessary aspect of modern clinical analytics.

A key principle throughout is that modernization should be **capability-driven rather than tool-driven**. New technologies provide value only when they address clearly defined problems and are integrated

within a controlled, reproducible framework. Similarly, improvements in structure, transparency, and workflow consistency often deliver greater long-term benefit than the introduction of entirely new platforms.

Equally important is the recognition that successful modernization depends as much on organizational alignment as on technical execution. Teams must develop the skills, processes, and governance structures required to support incremental change, while maintaining the documentation, traceability, and validation rigor expected in regulated environments.

Ultimately, the goal is not to eliminate legacy systems, but to **evolve them into more maintainable, understandable, and adaptable components of a broader analytical ecosystem**. Organizations that adopt this mindset—prioritizing disciplined progress over rapid transformation—are better positioned to reduce operational risk, improve productivity, and respond effectively to the increasing complexity of modern clinical analytics.

As analytical demands continue to grow and the technology landscape evolves, the ability to modernize without disruption will become an increasingly critical competency. A practical, incremental roadmap provides a sustainable path forward—one that enables innovation while preserving the reliability and trust upon which clinical decision-making depends.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

David Ward
Triam Ltd
david.ward@triamltd.com
https://triamltd.com/