PharmaSUG 2026 - Paper ET-212

A Novel Approach to Inter-Collaboration using IDEs and GitHub

Sydney Hyde and Tamara Martin, Bristol-Myers Squibb

ABSTRACT

In the pharmaceutical industry, specific programming languages have long been the dominant standard. However, open-source technologies have seen increasing adoption, driven by active user communities, and rapid innovation. As a result, new graduates entering the industry bring increasingly diverse programming backgrounds. In a fast-paced environment, it is critical to leverage both existing expertise and the skills of the next generation of programmers.

This goal is often hindered by limited overlap in programming language experience across teams, which can constrain projects to a subset of available talent and lead to duplicated effort. This paper presents a collaborative system, built on existing platforms, that enables cross-language collaboration. Using practical examples, we demonstrate how combining GitHub repositories with modern interactive development environments (IDEs) enables seamless collaborative workflows. This approach allows organizations to fully utilize the strengths of all team members, promote cross-language code reuse, and accelerate development without requiring programmers to abandon their preferred tools or workflows.

INTRODUCTION

Traditionally, the pharmaceutical industry has relied on the use of a single programming language within individual projects. The choice of language has typically been driven by what was deemed most appropriate for the project's goals, or more commonly, by the language with which the majority of the team was most familiar.

In recent decades, however, both the development of new programming languages and the continued refinement of established industry tools have significantly expanded the technical landscape. This growing diversity is also reflected in the range of programming backgrounds among professionals entering the workforce. Depending on their academic training or specific coursework, new talent often specializes in one primary programming language, with limited exposure to others.

While this diversity of specialization offers clear benefits, it has also introduced new challenges within the industry. When teams are required to standardize on a single programming language for a project, they effectively limit participation to those who are proficient in that language. As a result, project assignments are often constrained by tool familiarity rather than overall capability.

This limitation leads to two key challenges. First, it reduces the effective size of the available internal talent pool, resulting in smaller teams with limited ability to support one another. Second, solutions developed in one programming language are often not directly compatible with others, leading to duplicated effort when similar functionality must be recreated across different environments.

To address these challenges, this paper proposes a platform-neutral system that facilitates cross-programming-language collaboration. By leveraging version-controlled repositories and language-agnostic design principles supported by modern programming tools, teams can collaborate more effectively across technical boundaries. This approach enables organizations to better utilize their full talent pool while also promoting the reuse of existing solutions without requiring reimplementation in different programming languages.

MOTIVATION AND PROBLEM LANDSCAPE

Modern analytics and software organizations are increasingly composed of professionals with diverse technical roles and skill sets, including statisticians, analysts, programmers, data scientists, and software engineers. While these varied backgrounds foster innovation and enable specialized contributions, they

also introduce complexity in how teams collaborate, particularly when individuals rely on different tools, programming languages, and development environments.

In many cases, limitations in contribution are not due to a lack of capability, but rather the misalignment of tools and programming languages across the organization. Individuals may be unable to fully apply their expertise when required technologies do not integrate smoothly with those they are familiar with, creating barriers to effective collaboration.

As organizations grow in size and scope, these challenges become more pronounced. Teams often operate within fragmented workflows, where development environments are isolated and integration points are loosely defined. Contributors frequently develop solutions within their own contexts, making assumptions about structure, inputs, and outputs that are not easily understood or reused by others. This lack of shared context makes it difficult to review, integrate, or extend existing work.

In the absence of shared infrastructure or a standardized integration framework, collaboration often becomes ad hoc and heavily reliant on manual coordination rather than systematic processes. Teams may depend on informal methods to exchange code and results, leading to duplicated effort, inconsistent standards, and delays in delivery, particularly when multiple contributors are working across different technologies.

Additionally, individuals may be limited in their ability to contribute effectively, not due to a lack of expertise, but because the tools or programming languages required for a project do not align with their experience. As a result, valuable skill sets remain underutilized, and collaboration opportunities are restricted by technical barriers rather than functional needs.

These challenges reflect common day-to-day realities and highlight the need for a unified yet flexible collaboration model. Rather than enforcing a single programming language or technical stack, the focus should shift toward establishing shared structures, clear interfaces, and consistent workflows that support interoperability across diverse technical environments.

By prioritizing interoperability, modular design, and standardized conventions, organizations can create an environment where collaboration is both scalable and inclusive. This approach allows teams to maintain individual flexibility while ensuring that contributions can be effectively integrated into cohesive, end-to-end workflows, ultimately improving efficiency, reducing redundancy, and strengthening alignment across teams.

PROPOSED COLLABORATIVE SYSTEM

To address these challenges, a structured yet flexible collaboration model is required. The following section introduces a repository-centered system designed to enable this approach.

A. SYSTEM OVERVIEW

The proposed system, hereafter referred to as “The System,” is a repository-centered approach designed to securely interface with clinical data. The repository, such as GitHub, serves as a programming language–agnostic environment where individual program files written in different languages can be organized and linked together into unified workflows using standardized file structures.

This approach facilitates not only the reuse of previously developed code but also enables broader participation by allowing contributors with different programming backgrounds to work within the same project. In doing so, it removes barriers that would otherwise limit participation based on language proficiency.

To support these objectives, The System emphasizes compatibility across multiple programming environments. This is achieved through the use of standardized structures, naming conventions, and configuration formats that are independent of any specific programming language. These conventions enable teams to integrate tools and components developed in different languages without requiring

changes to the underlying technologies. As a result, the system is flexible, scalable across projects, and adaptable to the evolving technical demands of the pharmaceutical industry.

Specifically, standardized formats such as YAML- or JSON-based pipeline definitions, along with environment-agnostic conventions, allow a consistent pipeline framework to be applied across applications built in different languages or frameworks. This abstraction enables integration with various build tools, testing frameworks, container runtimes, and deployment targets without requiring pipeline redesign. Consequently, the system supports uniform pipeline behavior, predictable automation, and reduced operational complexity across projects.

B. CORE PRINCIPLES

The System is guided by four core principles:

Tool Neutrality:

Workflows are not dependent on any specific programming language. Individual components can be developed using the most appropriate language for the task.

Interoperability:

Program outputs are stored in widely readable, language-neutral formats (e.g., YAML, JSON), enabling seamless data exchange between components.

Flexibility:

Contributors are able to use their preferred programming languages and development tools, allowing them to work within their areas of expertise.

Reproducibility:

The repository structure, supported by version control, ensures traceability and validation of workflows, maintaining a single source of truth across multiple contributors.

C. INTEGRATION WITH MODERN DEVELOPMENT INTERFACES

These principles are further enabled through integration with modern development environments. Features such as multi-file editing, debugging tools, and repository synchronization are now standard across many integrated development environments (IDEs).

One of the most impactful advancements has been the widespread adoption of extensions and plugins. These tools act as bridges between different programming languages and IDEs, allowing contributors to work within their preferred environments while maintaining connectivity to shared repositories. This capability supports collaboration without requiring standardization on a single language or development interface.

D. ENABLING CROSS-ENVIRONMENT COLLABORATION

Effective collaboration across diverse technical environments requires the establishment of shared conventions that span tools, programming languages, and platforms. Rather than enforcing a single technology choice, The System focuses on defining common structures and interfaces that enable independent work while maintaining interoperability.

Standardization is achieved through consistent directory layouts, naming conventions, and data handoff

formats that are shared across projects and teams. These conventions ensure that code, configurations, inputs, and outputs follow predictable patterns, allowing contributors to quickly understand and navigate unfamiliar repositories. This structure also supports automated integration processes and reliable exchange of artifacts, improving both efficiency and onboarding.

In addition, modular design enables parallel development without language dependencies. By clearly defining module boundaries, inputs, outputs, and configurations, contributors can develop components independently without disrupting integration. This approach reduces coordination overhead, minimizes merge conflicts, and supports incremental updates as technologies evolve.

For example, a statistical programmer may generate ADaM datasets using SAS, while a data scientist performs exploratory analysis in R, and another contributor develops validation scripts in Python. By defining standardized input and output formats, these components can be integrated into a single workflow without requiring reimplementation in a common language.

Clear, tool-agnostic documentation templates further enhance collaboration by providing consistent guidance regardless of the tools used. These templates help preserve critical knowledge, ensure consistent understanding of system usage and configuration, and reduce reliance on informal or undocumented processes. As a result, onboarding is improved, cross-functional code reviews are facilitated, and long-term maintainability is strengthened.

Collectively, these practices enable collaboration to scale across heterogeneous environments by emphasizing structure, clear interfaces, and shared standards. The result is a flexible model that supports individual productivity while enabling cohesive, end-to-end workflows across the organization.

PRACTICAL EXAMPLES (AGNOSTIC USE CASES)

A. MULTI-CONTRIBUTOR DEVELOPMENT SCENARIO

In typical multi-contributor workflows, team members operate within different local environments based on their roles and expertise while collaborating through a shared repository. For example, one contributor may develop data preparation logic in a statistical programming environment, another may implement validation checks using a scripting language, and a third may focus on orchestration or reporting.

Each contributor commits their work to a clearly defined module within the repository, following shared directory structures and interface standards. Clear module interfaces allow components to be developed independently and integrated without modification. For instance, a data transformation module can feed directly into analytics or reporting modules developed by other contributors, regardless of the tools used. This enables parallel development while maintaining a unified workflow.

B. REPOSITORY-CENTERED REVIEW AND COLLABORATION

Collaboration activities such as branching, peer review, and change tracking are managed centrally within the shared repository. Contributors submit changes through isolated branches, where updates are reviewed based on standardized structures, configuration files, and documented interfaces rather than local execution environments.

Reviewers do not need to replicate the author’s development setup. Instead, they focus on code organization, adherence to conventions, interface definitions, and documentation completeness. This allows contributors from different technical backgrounds to participate in the review process, reduces reliance on specialized environments, and keeps collaboration focused on integration readiness and maintainability.

C. ENVIRONMENT-AGNOSTIC EXECUTION PIPELINES

Execution workflows are defined using environment-agnostic scripts or workflow descriptors that outline the logical sequence of steps rather than language-specific commands. A pipeline may include stages such as data ingestion, transformation, validation, and output generation, with each stage invoking modules through standardized interfaces.

Dependencies are managed at the module or execution level, ensuring that each component runs within its required environment without impacting others. This enables components written in different programming languages to operate within the same pipeline while maintaining consistent execution and reproducibility across environments.

D. REUSABLE, CROSS-CONTEXT COMPONENTS

Reusable components such as templates, configuration files, and utility functions can be shared across projects and toolchains. Examples include standardized configuration schemas, logging conventions, validation rules, and data quality checks that can be applied regardless of implementation language.

By centralizing these assets within the repository, teams reduce duplication of effort and promote consistency. Components developed once can be reused across multiple workflows, improving efficiency and reducing long-term maintenance overhead.

ORGANIZATIONAL BENEFITS

The modular, tool-agnostic approach provides several key organizational benefits by enabling more effective collaboration across diverse technical environments.

A. MAXIMIZING STAFF EXPERTISE

Contributors are able to work within their areas of expertise without being constrained by a single mandated technology. This increases participation, improves quality, and reduces friction associated with tool standardization.

B. REDUCING REDUNDANCY AND INCREASING EFFICIENCY

A centralized repository of reusable components minimizes duplicated effort. Teams can build upon existing solutions rather than recreating similar functionality, leading to faster delivery and more consistent outputs.

C. ACCELERATING ONBOARDING

Standardized structures, interfaces, and documentation reduce the learning curve for new contributors. Individuals can quickly understand workflows without needing to adopt unfamiliar tools.

D. STRENGTHENING COMPLIANCE AND AUDITABILITY

Version-controlled workflows enhance traceability, governance, and accountability. Changes are documented and auditable, supporting regulatory compliance and improving transparency.

IMPLEMENTATION CONSIDERATIONS

To successfully adopt this collaboration model, organizations should begin by establishing foundational standards and supporting infrastructure. Key initial steps include:

- Defining a standardized repository structure, including directory layouts and naming conventions
- Establishing common data exchange formats (e.g., CSV, JSON, YAML) to support interoperability
- Creating clear module interface definitions, including expected inputs and outputs
- Implementing version control practices, including branching strategies and code review processes

- Developing documentation templates to ensure consistency across contributors

Adopting these practices incrementally allows teams to transition toward a tool-agnostic model without disrupting existing workflows. Initial implementation within pilot projects can help refine standards, validate the approach, and demonstrate value prior to broader adoption.

DISCUSSION

Implementing a modular, tool-agnostic collaboration model introduces both technical and organizational challenges. One practical consideration is the establishment of standardized naming conventions and directory structures. While these may seem minor, they are critical for enabling consistency, supporting onboarding, and ensuring long-term maintainability across contributors and projects.

Another challenge involves integrating version control systems, such as GitHub, with modern development environments like Visual Studio Code. While these tools provide robust functionality through extensions and plugins, technical limitations or inconsistencies can occasionally require additional coordination and troubleshooting.

More broadly, adopting this approach requires a cultural shift within organizations. Rather than focusing on specific tools or programming languages, teams must align around shared workflows, interfaces, and outcomes. This shift moves the focus away from how work is performed and toward what is produced and how it integrates into the broader system.

Clear communication, well-defined interfaces, and consistent documentation become essential components of collaboration. Success depends not on a single tool, but on shared standards and a common understanding of how components interact within the system. This alignment reduces silos, strengthens collaboration, and supports long-term sustainability as technologies and team structures evolve.

While enforcing a single tool or tightly controlled environment may provide short-term consistency, it often limits scalability and participation over time. In contrast, a flexible, tool-agnostic approach supports more inclusive and adaptable workflows. Contributors can use tools aligned with their expertise while still integrating seamlessly into shared processes. This reduces dependency on specialized knowledge, minimizes bottlenecks, and enables organizations to evolve alongside changing technologies and team compositions.

FUTURE DIRECTOIONS

As the pharmaceutical and broader analytics industries continue to evolve, the need for flexible, scalable, and interoperable collaboration models will only increase. While the system proposed in this paper provides a strong foundation for enabling cross-programming-language collaboration, several opportunities exist to further enhance and expand this approach.

One area of future development is the integration of automated workflow orchestration and validation processes. Incorporating continuous integration and continuous delivery practices can strengthen consistency, improve reproducibility, and reduce manual oversight by automatically executing workflows and validating outputs.

Another important direction is the increased use of metadata-driven design. Defining workflows, module interfaces, and data exchange requirements using structured formats such as YAML or JSON allows systems to become more dynamic, adaptable, and reusable across projects. This approach also enhances transparency and traceability, which are particularly important in regulated environments.

Containerization technologies, such as Docker, present an additional opportunity to standardize execution environments while preserving flexibility. By encapsulating dependencies at the module level, teams can

ensure consistent behavior across development and production environments, reducing integration challenges.

From an organizational perspective, continued development of governance frameworks and best practices will support broader adoption. Establishing standardized templates, onboarding materials, and usage guidelines will help ensure consistency while maintaining flexibility across teams.

Finally, emerging technologies such as artificial intelligence and machine-assisted development tools offer opportunities to further enhance collaboration. Capabilities such as automated code review, documentation generation, and cross-language translation can reduce barriers between technologies and improve overall productivity.

Ongoing evaluation and iteration of this model in real-world applications will be essential. As teams adopt and refine the system, feedback can be used to improve usability, strengthen standards, and ensure long-term effectiveness.

CONCLUSION

The pharmaceutical industry is experiencing a shift toward increasingly diverse technical ecosystems, where professionals bring a wide range of programming skills, tools, and perspectives. While this diversity has the potential to drive innovation, traditional approaches that rely on a single programming language or tightly controlled environments can limit collaboration, reduce efficiency, and create unnecessary duplication of work.

This paper introduced a repository-centered, tool-agnostic collaboration model designed to address these challenges. By leveraging version control systems such as GitHub, along with standardized structures, modular design principles, and language-neutral data formats, the proposed system enables contributors to work within their preferred tools while participating in a cohesive, integrated workflow.

Through this approach, teams are no longer constrained by programming language boundaries. Instead, they can focus on defining clear interfaces, reusable components, and consistent documentation practices that support interoperability across environments. This maximizes the use of available talent, promotes reuse of existing solutions, and improves overall efficiency.

Beyond its technical advantages, this model supports a broader shift in how collaboration is approached within organizations. By emphasizing shared standards, transparency, and clear communication, teams can move away from fragmented, ad hoc processes toward more structured and scalable workflows. This shift is critical for enabling long-term sustainability in complex and evolving environments.

As the industry continues to advance, adopting flexible and interoperable collaboration frameworks will be essential for improving efficiency, ensuring reproducibility, and supporting innovation. The approach outlined in this paper provides a practical and scalable path forward.

Contact Information

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

Sydney Hyde
Bristol-Myers Squibb
860.395.7110
Sydney.Hyde@bms.com
https://www.linkedin.com/in/sydney-hyde/

Tamara Martin
Bristol-Myers Squibb
423.742.1449
Tamara.Martin@bms.com
https://www.linkedin.com/in/tamaratestermartin/