# Upstream jank-lang specific patches back to CppInterOp

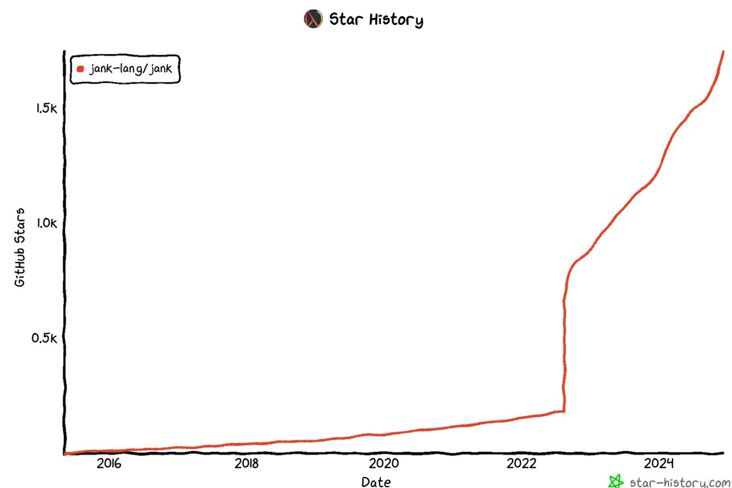
Iva Fezova

# Why i chose this project

Maintaining a large set of downstream patches is a significant burden for any language project, and jank-lang is no exception.

I chose this project because it offers a unique opportunity to work at the intersection of LLVM/Clang and language interoperability while solving a real-world engineering problem.

My goal is to transform these isolated modifications into high-quality, upstream-standard contributions that benefit the entire CppInterOp ecosystem, ensuring long-term stability for both project



# The  jank Programing Language

The jank programming language relies on a specialized set of downstream patches to CppInterOp to meet its complex interoperability requirements.

Currently, these modifications are maintained separately from the upstream project, which significantly increases long-term maintenance effort and technical debt.

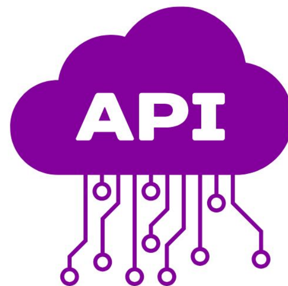
By integrating these patches into the main CppInterOp repository, the project aims to reduce this overhead and ensure that jank remains compatible with future compiler updates.

# CppInterOp Framework

CppInterOp is a C++ interoperability library built on top of LLVM and Clang that enables various programming languages to interact with C++ code using compiler-level information. The implementation work for this project primarily interacts with LLVM and Clang APIs, requiring strict adherence to upstream coding styles and design rules. This framework serves as the foundation for the project, where jank-specific patches will be refactored into maintainable and generally applicable improvements.

# Project Objectives

- Convert downstream-only modifications into well-designed, generally applicable improvements.
- Align jank-specific code with upstream project standards and API stability.
- Establish a repeatable process for classifying patches based on size and complexity.
- Prioritize self-contained changes to maximize the number of accepted contributions.

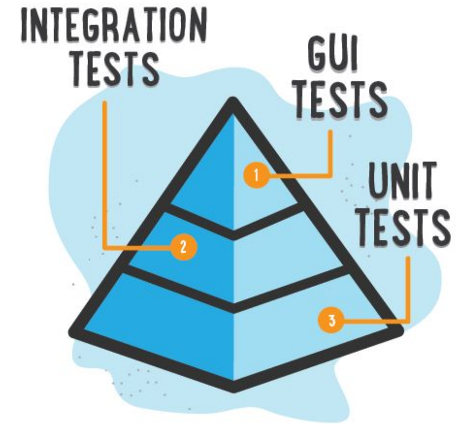


# Refactoring strategy

- Simplify patches by removing jank-specific assumptions to make functionality universal.
- Keep language-specific behavior separate via simple interfaces or configuration options.
- Focus on modularity to ensure that new features do not disrupt existing CppInterOp logic
- Maintain strict version compatibility with underlying Clang and LLVM components

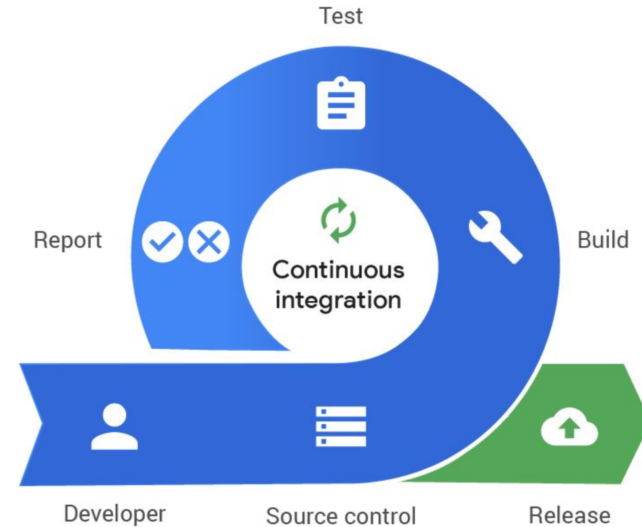
# Technical workflow

- Review approximately 85 existing patches to determine their functional intent.
- Categorize patches into "easy", "medium", and "hard" for prioritized upstreaming.
- Add regression and unit tests to ensure stability and prevent breaking changes.
- Prepare clear documentation and technical descriptions for every submission



# Execution approach

- Run Continuous Integration (CI) to ensure all refactored patches pass successfully across different environments.
- Provide clear PR descriptions and documentation for all submissions.
- Submit patches in batches to allow for thorough review and adaptation.
- Aim to reduce the downstream stack by achieving ~50% patch acceptance.



# Expected outcomes

- **Stack Reduction:** Significantly minimize the downstream patch set for the jank-lang project.
- **Maintenance Framework:** Provide a useful system for future patch evaluation and maintenance.
- **Community Impact:** Contribute robust, tested features to the broader CppInterOp user base.

# Thank you!