Large Codebase SDK: Extending OpenHands for Legacy Migration and Code Refactors at Scale

Calvin Smith
calvin@openhands.dev

1 The Problem

Enterprise organizations run on legacy code. Across industries—financial services, insurance, government, healthcare—critical business systems are written in languages like COBOL, and the expertise to maintain them is disappearing. COBOL alone poses a significant risk:

- **200B+** lines of COBOL still running [6]
- **95%** of US ATM swipes touch COBOL [11]
- **92%** of COBOL developers retiring by 2030 [8]

The workforce that understands these systems is aging out, and replacement hires are increasingly difficult to find—even as salaries for COBOL engineers continue to rise year over year.

There are simply fewer and fewer people who can maintain, debug, or extend these business-critical systems.

Manual Migration Mostly Fails. The natural response is to migrate away from legacy languages. But legacy modernization has a poor track record: 74% of organizations that start a modernization project fail to complete it [1], and 70% of digital transformation initiatives fail to meet their stated goals [7], finishing over budget or with significant scaling and performance losses.

The Missing Piece. An effective migration requires deep expertise in *both* the legacy system and the target language. Such a combination is not common, and without both you risk a migration leaving you with a modern implementation still rooted in the legacy architecture. You may have succeeded in moving away from COBOL, but your code base is no more maintainable than before and may have lost business logic in the translation.

2 How AI Agents Can Help

AI coding agents are uniquely suited to the legacy migration problem. Modern large language models have broad familiarity with both legacy and modern programming languages, which let them bridge the dual-expertise gap that makes human migration so difficult.

Proof-of-Concept. In a previously-published blog post, we demonstrate how OpenHands can successfully migrate COBOL to Java [9]. It includes a reference implementation comprising about 470 lines of Python built on the OpenHands Agent SDK [12] and relies on techniques like *iterative refinement* to steer solutions toward defined quality bars, even for code bases too large for an agent to handle in a single pass.

The reference implementation provides an example COBOL program, making it a low-barrier way to demonstrate the efficacy of AI agents in modernizing legacy code.

3 Difficulties in Scaling Up

The iterative refinement workflow works well for small-to-medium code bases. But legacy enterprise systems are orders of magnitude larger and understanding the business logic requires a holistic view.

This presents unique challenges for coding agents, even when equipped with modern language models that process hundreds of thousands of tokens at once. Performance starts declining well before models hit their context limit: agents get stuck in loops, lose track of where they are, and quit before the job is done [5].

The Parallelism Problem. The natural fix is to rely on multiple agents. Modern agentic tools support this paradigm via *subagent delegation* [2, 3], where an agent orchestrates other agents, each with their own task and context, before bringing the results together. Some workloads allow these sub-agents to run in parallel, which dramatically improve the speed with which an agent can resolve a task.

But parallelism introduces a serious coordination problem in software development tasks: if two agents touch interdependent code, they can make incompatible decisions and invalidate each other's work. For example, if an agent attempts to fix a class and generates two sub-agents, one might restructure the interface while the other is writing code that depends on the old version, resulting in a frustrating cycle of agents undoing each other's progress.

4 Scaling with the Large Codebase SDK

The Large Codebase SDK solves the scaling problem by combining AI agents with static analysis. It extends the OpenHands Agent SDK with dependency-aware tooling, so agents work with the structure of the code rather than against it.

In a recent evaluation, the Large Codebase SDK was used to successfully migrate a 40+ program, 60+ file COBOL project to Java. In doing so it:

- Removed mainframe dependencies so the source could be run locally
- Established behavior tests to ensure the migration preserved business logic
- Produced a Java implementation that met partner requirements for both functionality *and* quality

The same workflow without the Large Codebase SDK demonstrated all the issues typical of single-agent solutions, and only managed to migrate a third of the necessary business logic.

Dependency Graph. The SDK uses `tree-sitter` [4] to parse source code and build a *dependency graph* that maps how functions, classes, imports, inheritance, and API calls interact throughout the entire code base. The result is easy to visualize and clearly demonstrates the complexity of even small software projects. Fig. 1 contains an example dependency graph of a portion of OpenHands itself.

Figure 1. A dependency graph of the OpenHands SDK [10]. Nodes correspond to files in the `openhands.sdk` module, and edges to imports (oriented from importers to importees). Node size is scaled with the number if incoming edges. The red edges indicate cycles in the import structure.

Dependency analysis is language-aware, with full support for Java, Python, Scala, and COBOL. Additional languages can be supported on request.

Understanding the Dependencies. The dependency graph provides a granular view of the code before an agent ever gets involved. Using a combination of graph-theoretic algorithms and a semantic understanding of the underlying code, this granular view is transformed into batches of files that can be processed independently. Batches contain related files, so that a human can easily review the generated changes, and are ordered so that agents will resolve a batch only when all of its dependencies have been handled. A batch graph generated from the dependencies in Fig. 1 is shown in Fig. 2.

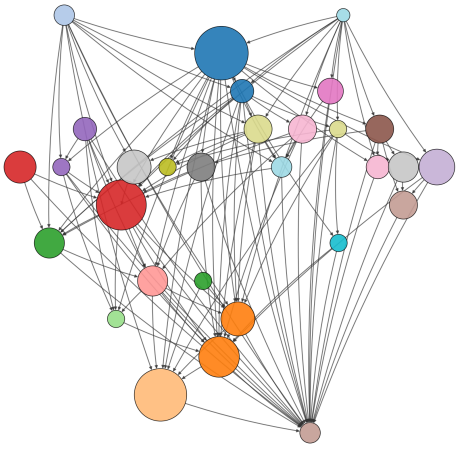

Figure 2. A batch graph generated from the `openhands.sdk` dependencies shown in Fig. 1. Nodes represent groups of files (and scaled by the number of files), while edges capture inter-batch file dependencies. Note the simpler structure, lack of cycles, and clear ordering: starting from the bottom and working upwards, we explore the batches only when all of their dependencies have already been explored.

Concretely, that means the Large Codebase SDK orchestrates agents to work on files *only when changes cannot conflict with earlier work*. This ensures progress is durable and the resulting code base is coherent. As an added benefit, the Large Codebase SDK understands when two batches of work are truly independent, enabling safe parallelism at enterprise scale.

Extending the OpenHands SDK. The Large Codebase SDK provides a number of abstractions to orchestrate the full refactoring pipeline: tracking per-batch progress, handling failures and retries, and coordinating parallel execution. It uses *fixers* (instances of OpenHands) to apply changes and *verifiers* (OpenHands, CLI tools for type-checking or security scanning, LLM review) to validate the changes.

Because it is built on OpenHands, the Large Codebase SDK can spin up work locally or in the cloud and is model-agnostic. It works with Anthropic, OpenAI, Azure, or locally-hosted models. This is critical for organizations with data sensitivity requirements or regulatory

constraints that prevent sending proprietary code to external APIs.

5 Get Started

OpenHands is currently accepting applications for the Q1 2026 Proof of Concept program. Participants work directly with our forward-deployed engineering team to validate real workflows and outcomes, with a path toward broader enterprise rollout.

Who It's For. The PC program is designed for multi-user teams with large codebases—COBOL or otherwise—that want to validate AI-assisted refactoring and migration workflows before committing to enterprise-wide adoption. Participants get direct engineering support, a voice in shaping the product, and success measured by proof and readiness.

References

[1] Advanced. 2020 mainframe modernization business barometer report. Business Wire, May 2020. 74% of organizations have started a legacy system modernization project but failed to complete it. https://www.businesswire.com/news/home/20200528005186/en/74-Of-Organizations-Fail-to-Complete-Legacy-System-Modernization-Projects-New-Report-From-Advanced-Reveals.

[2] Anthropic. Create custom subagents. Claude Code Documentation, 2025. Accessed: 2026-03-09. https://docs.anthropic.com/en/docs/claude-code/sub-agents.

[3] Anthropic. How we built our multi-agent research system. Anthropic Engineering Blog, 2025. Accessed: 2026-03-09. https://www.anthropic.com/engineering/multi-agent-research-system.

[4] Max Brunsfeld, Patrick Thomson, Josh Vera, Andrew Hlynskyi, Phil Turnbull, and Timothy Clem. Tree-sitter: An incremental parsing system for programming tools. GitHub, 2018. https://github.com/tree-sitter/tree-sitter.

[5] Andy Chung, Yichi Zhang, Kaixiang Lin, Aditya Rawal, Qiaozi Gao, and Joyce Chai. Evaluating long-context reasoning in llm-based webagents, 2025.

[6] Gartner Group. COBOL usage statistics, 1997. Reported 200 billion lines of COBOL running 80% of the world's business. Cited in Nextgov (2009) and other industry publications. https://www.nextgov.com/modernization/2009/07/cobol-remains-old-standby-at-agencies-despite-showing-its-age/203338/.

[7] McKinsey & Company. Digital transformation failure rates, 2023. 70% of digital transformation initiatives fail to meet their objectives. Accessed: 2026-03-09. https://www.ft.com/partnercontent/teamviewer/70-per-cent-of-transformation-projects-fail-and-everyones-ignoring-the-same-fix.html.

[8] Modernization Intel. Mainframe modernization: Research, cost data & migration strategies, 2026. Industry analysis: 220,000 COBOL developers remaining (average age 58.3), 92% retiring by 2027-2030, 47% of organizations cannot fill COBOL roles. Accessed: 2026-03-09. https://softwaremodernizationservices.com/mainframe-modernization.

[9] Graham Neubig. Refactoring COBOL to Java with AI agents. OpenHands Blog, December 2025. Accessed: 2026-03-09. https://openhands.dev/blog/20251218-cobol-to-java-refactoring.

[10] OpenHands. software-agent-sdk: A clean, modular sdk for building ai agents with openhands v1. https://github.com/OpenHands/software-agent-sdk, 2025. Accessed: 2026-03-11.

[11] Reuters. COBOL blues, April 2017. Reuters graphic showing 43% of banking systems built on COBOL, 95% of ATM swipes rely on COBOL code, 80% of in-person transactions use COBOL. https://thenewstack.io/cobol-everywhere-will-maintain/.

[12] Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, Valerie Chen, Robert Brennan, and Graham Neubig. The OpenHands software agent SDK: A composable and extensible foundation for production agents, 2025.