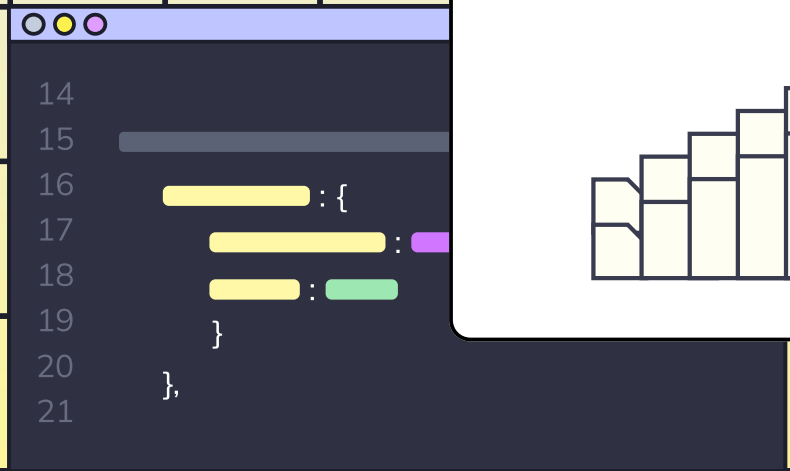
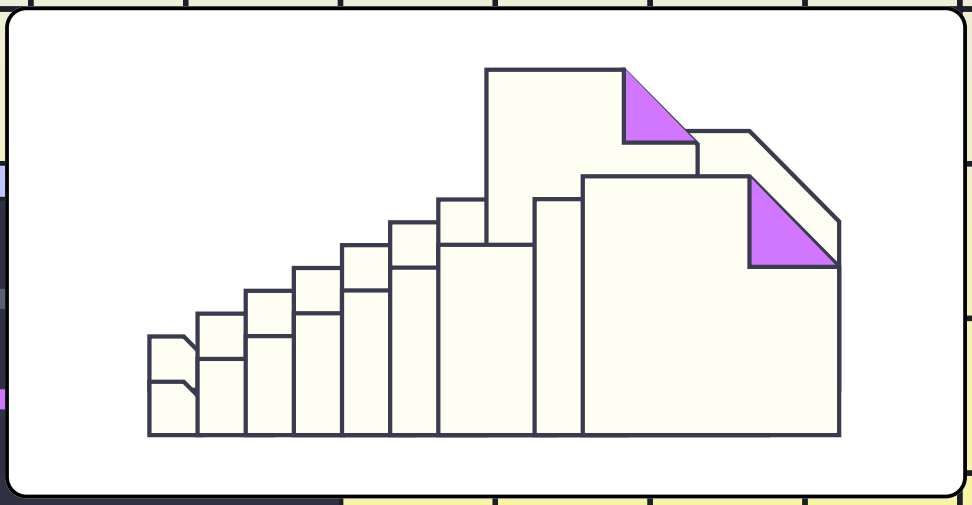
swimm

# Automating legacy code documentation

A unique approach



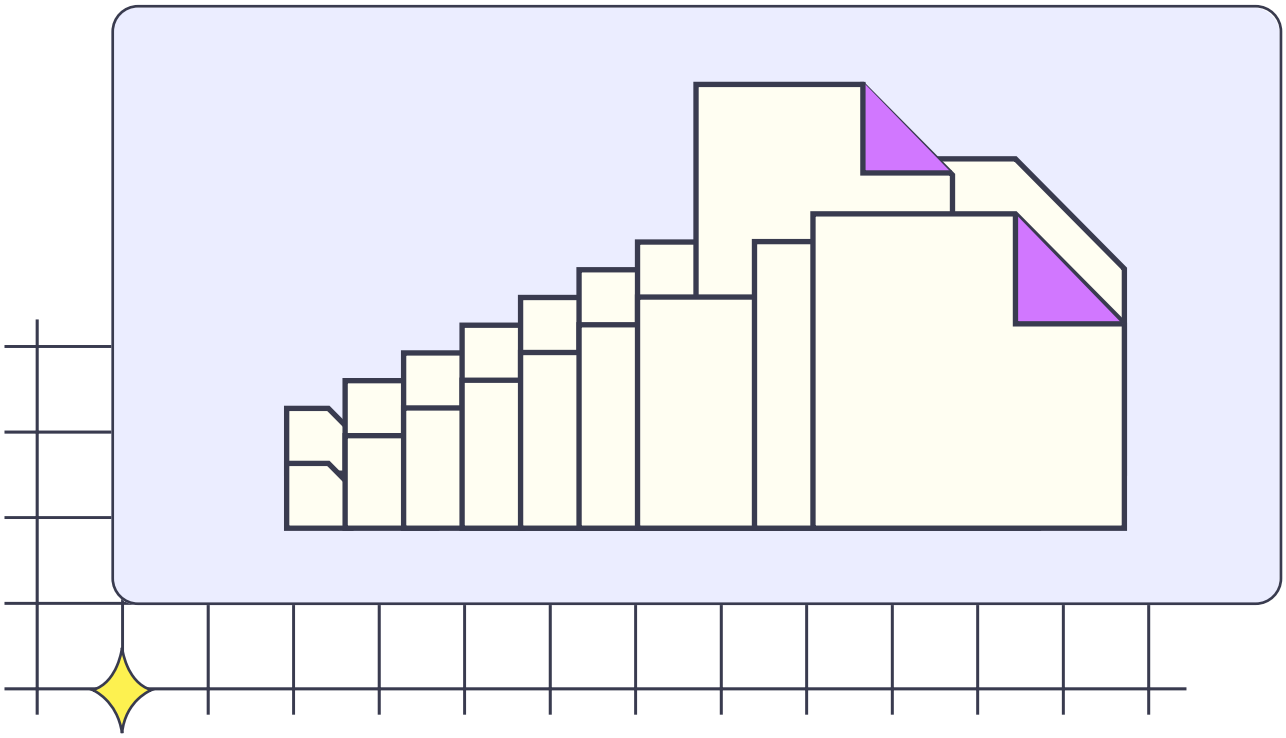
# Table of Contents

Chapter 1

# Executive summary

## Introduction:

Legacy code poses a significant challenge for many organizations, impacting productivity, innovation, and risk management. This whitepaper explores Swimm's solution for automatically documenting legacy codebases, addressing the critical need for efficient knowledge transfer and code understanding in modern software development environments.



In 2022, legacy code cost the United States $520 billion, underscoring why addressing it is an executive priority for enterprises.

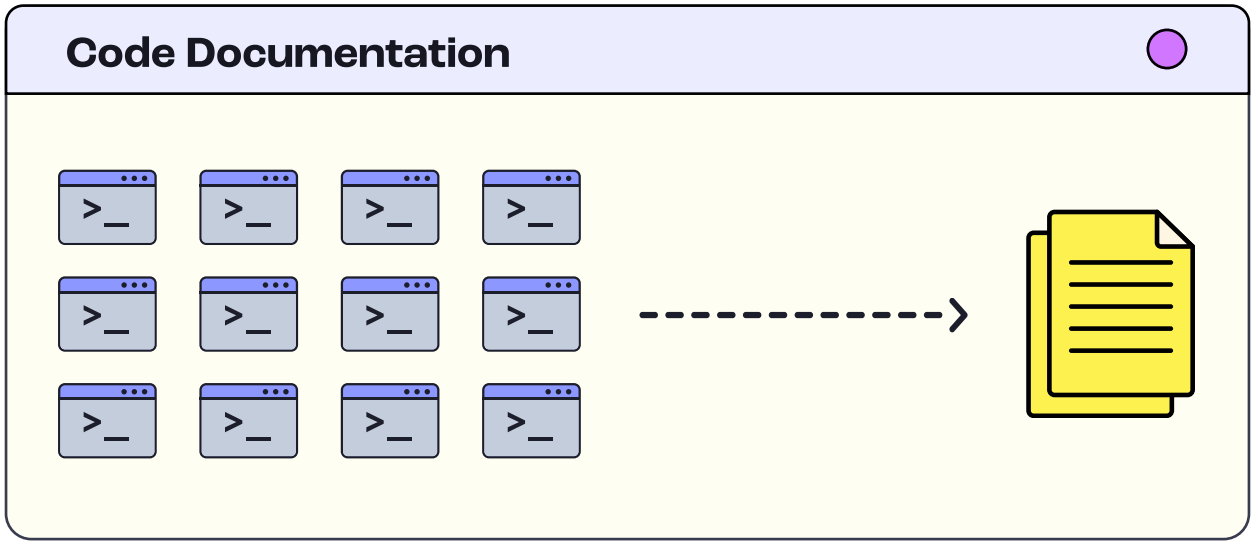
While not all legacy code requires modernization, effectively maintaining or modernizing it begins with a clear understanding of your existing codebase.

[1] Krasner, Herb. “The Cost of Poor Quality Software in the US: A 2022 Report.” Consortium for Information & Software Quality (CISQ), 15 December 2022, https://www.it-cisq.org/wp-content/uploads/sites/6/2022/11/CPSQ-Report-Nov-22-2.pdf. Accessed 26 August 2024.

Chapter 2

# The challenge: Legacy code and the documentation deficit

In the lifecycle of software development, today's innovative project inevitably becomes tomorrow's legacy system. As codebases grow and evolve, they accumulate layers of complexity, often outpacing documentation efforts. When you start working on a codebase, there is a small team where every engineer knows their part of the code, and probably knows enough about the other parts of the code. Gradually, what began as a well-understood project, held together by the shared knowledge of a core team, gradually transforms into a labyrinth of code that few fully comprehend.




This evolution creates a knowledge deficit that poses significant challenges for organizations. Team members come and go, taking valuable context with them, while new developers struggle to navigate an unfamiliar landscape. The once-clear mental map of the system becomes fragmented, with knowledge scattered across the organization or lost entirely.

When effective documentation about the codebase exists, it serves as a written form of the knowledge that was once held in the engineers' minds. The problem in legacy code is that documentation is outdated (as the code changed since it had been written), ineffective, or doesn't exist at all.

## The consequences

### 01. Onboarding inefficiency

New team members face a steep learning curve, requiring weeks or months to become productive as they piece together understanding of undocumented structures and flows.

### 02. Maintenance risks

Without clear documentation, even minor changes can have unforeseen consequences, leading to increased bugs and system instability.

### 03. Modernization hurdles

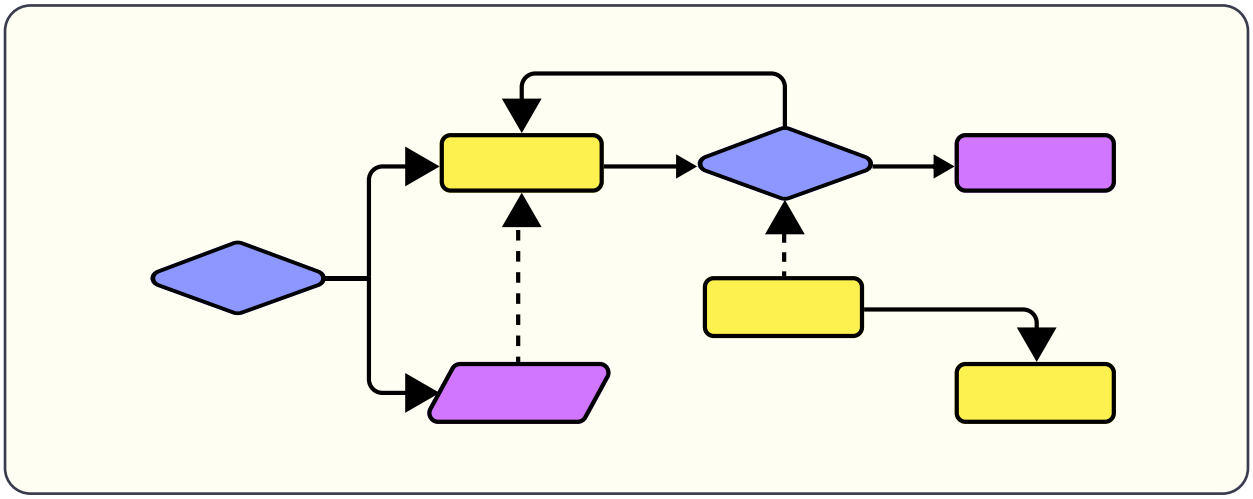
Lack of comprehensive system understanding impedes modernization efforts, making it difficult to refactor or replace legacy components quickly and safely.

### 04. Debugging delays

Developers spend excessive time tracing issues through poorly documented code, significantly slowing down problem resolution and increasing downtime.

### 05. Innovation bottlenecks

Adding new features or integrating modern technologies becomes increasingly challenging without a clear map of the existing codebase, stifling innovation and agility.



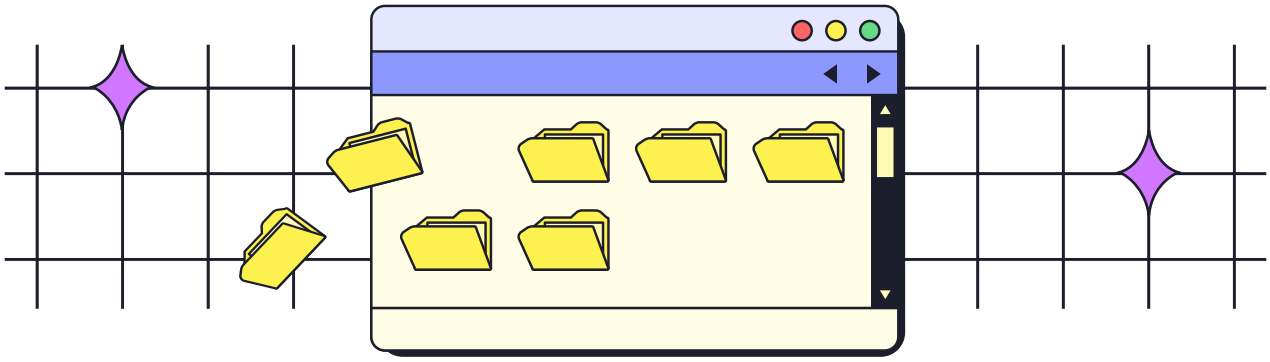
These challenges underscore the critical need for a solution that can bridge the gap between legacy code and modern development practices. By providing comprehensive, up-to-date documentation, organizations can unlock the value hidden in their legacy codebases, enabling efficient maintenance, faster modernization efforts, and renewed innovation. The key lies in finding a way to rapidly and accurately document existing systems, transforming impenetrable legacy code into a well-mapped, understandable asset that can drive business value for years to come.

Chapter 3

# Limitations of LLMs for code documentation

Effective documentation of a codebase would include both a high level overview of the system, as well as in-depth walkthroughs of main flows in the code. The documentation should help the reader separate the wheat from the chaff, focus on the most important details while obscuring the rest. The documents should describe things that are not easily deduced from looking at the code - for example, flows that span across multiple files or repositories.

While LLMs have shown promise in various natural language tasks, they face significant challenges when it comes to comprehensive code documentation.



## The challenges

### 01. Lack of deep code understanding

LLMs struggle to trace complex flows and understand the overall structure of large codebases.

### 02. Difficulty in identifying important components

LLMs cannot reliably determine which parts of a codebase are most critical or interesting without additional context.

### 03. Non-deterministic outputs

LLMs produce different results for the same input, making it challenging to ensure consistent documentation.

### 04. Inability to perform deep Code analysis

LLMs cannot perform the deep, language-specific static analysis required to accurately trace code flows and relationships.

### 05. Potential for hallucinations

LLMs may generate convincing but incorrect information, which is particularly dangerous in a code documentation context.

Chapter 4

# Swimm's approach

### Deep static code analysis: Tackling language complexity

Swimm's approach to static code analysis is built on a foundation of language-specific parsing, crucial for accurately understanding diverse codebases.

#### Language-specific understanding

Programming languages differ in many ways. To perform an accurate analysis of a codebase, Swimm uses our specific plugins when necessary in legacy code languages.  The rest of the process is language agnostic.

#### Cross-Language Analysis

- **Identifying logical components:** Based on deep static analysis, our cross-language algorithms identify various logical components in any codebase. These components represent the hierarchy of the knowledge, and as a result - the proposed hierarchy of the documents.
- **Tracking and identifying important flows:** We find all flows within the codebase, and then deterministically rank them. We create both in-depth and high-level overview docs for the important flows.

#### Organization-specific analysis

When we work with enterprise clients, we also generate plugins that match structures and files that are specific to their organizations.

Chapter 5

# Swimm Knowledge Layer: Foundation for code understanding

### 01. Comprehensive code representation

Our knowledge base captures a complete representation of the codebase, including both source code and any existing documentation.

### 02. Language-agnostic storage

After parsing language-specific features during the initial analysis, we store information in a unified format. This allows for consistent analysis and documentation generation across different programming languages.

### 03. Efficient information retrieval

We've implemented specialized retrieval mechanisms optimized for code-related information. It is optimized for finding relevant information from big codebases.

### 04. Support for multiple use cases

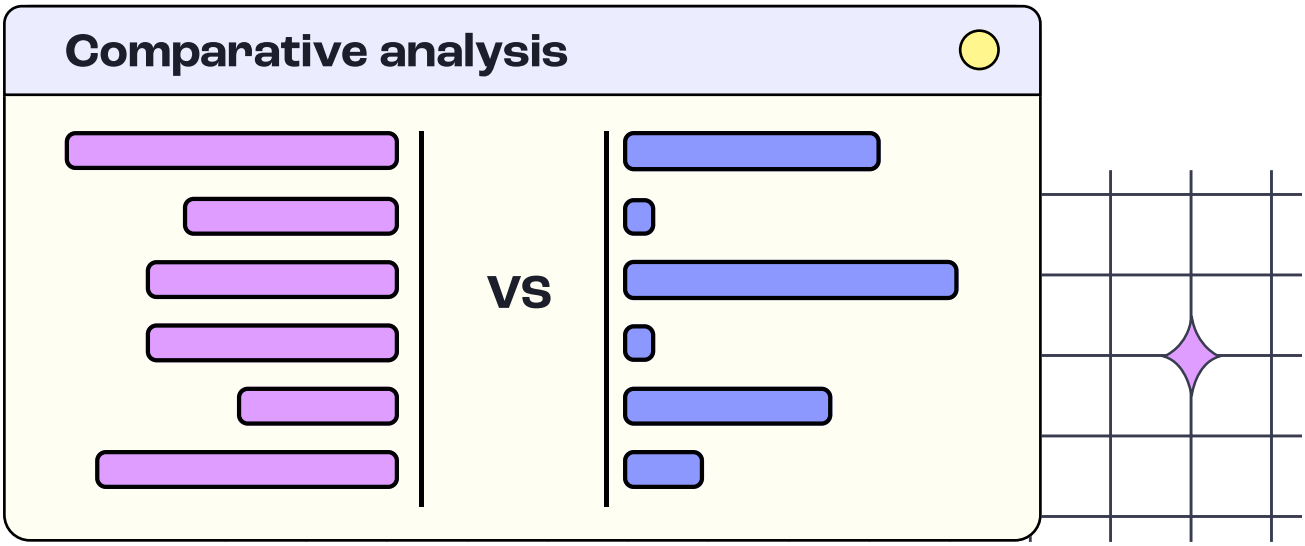
The knowledge base is designed to support various applications beyond just documentation generation. It provides the underlying data for our PR-to-doc, Snippet-to-doc and Ask Swimm features.

This internal knowledge base allows us to perform deep analysis and generate accurate documentation without sending any code to external servers, ensuring the security and privacy of your codebase.

Chapter 6

# Comparative analysis

Compared to traditional documentation tools and pure LLM-based solutions, Swimm's approach offers several advantages:




**Accuracy**

Our deterministic analysis provides more reliable results than pure AI-based solutions.

**Scalability**

Unlike manual or semi-automated tools, we can handle enterprise-scale codebases efficiently. This also means reduced costs that stem from many LLM calls with large context.

**Language coverage**

Our support for legacy languages sets us apart.

**Integration of AI and traditional techniques**

We leverage the strengths of both approaches, using AI for natural language generation while relying on proven static analysis techniques for code understanding.

Chapter 7

# Enterprise-ready solution

Swimm's Auto-docs feature is engineered to meet the demanding requirements of large-scale enterprises, offering a secure, scalable, and flexible documentation solution for legacy codebases.

### 01. Security and privacy

Our system processes all code locally, ensuring that sensitive intellectual property never leaves your infrastructure. For organizations with strict data governance policies, we offer on-premises deployment options, including the ability to use your own LLM instances.

### 02. Scalability and performance

Designed to handle enterprise-scale projects, Swimm can generate thousands of documents simultaneously, efficiently processing millions of lines of code.

### 03. Comprehensive language support

Our language-agnostic approach supports a wide range of technologies, including legacy languages like COBOL. This ensures comprehensive documentation coverage across diverse technology stacks. We can add support for additional languages used by our clients.

### 04. Customization

Recognizing that each organization has unique documentation standards, we enable customizable document templates to align with your specific requirements.

By addressing these key enterprise concerns, Swimm enables organizations to leverage advanced AI-assisted documentation while maintaining complete control over their codebase and infrastructure.

Chapter 8

# Conclusion

Swimm's Auto-docs feature represents a significant advancement in automated code documentation, particularly for legacy systems. Bx This approach not only preserves critical knowledge but also enables organizations to transform their legacy code from a liability into a valuable, understandable asset.

# swimm

Swimm helps enterprise developers quickly understand big, complex codebases—and automatically captures knowledge to fill in any documentation gaps.

**Learn more about Swimm**