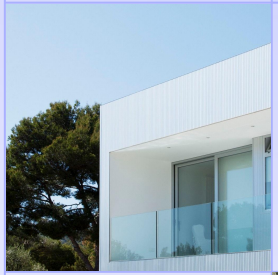
Team 9 Runtime Terror
F25

Repository Architecture Diagramming



Statement of Purpose:

This project aims to turn code into clarity by automating architecture visualization, making complex systems understandable and collaboration effortless.

USER PERSONAS

Lily Chen
Beginning Software Developer

21, First-year CS student, San Diego

Needs and motivations

- Understand real codebases visually instead of reading dense source files
- Explore open-source projects for learning and inspiration

Pain Points

- READMEs are text-heavy and hard to follow
- Hard to see how modules and folders are connected

How We Can Help

Our tool automatically visualizes a repo's internal logic and structure, turning messy code into an intuitive diagram that helps Lily *learn by seeing*.

Eric Wang
Newly Onboarded Software Engineer

25, Junior software engineer, Seattle

Needs and motivations

- Quickly grasp core project logic and architecture
- Locate where assigned features on in the codebase

Pain Points

- Sparse onboarding documents
- Hard to link assigned tasks to actual code components

How We Can Help

By generating architecture diagrams and supporting task-focused prompts, our tool gives Eric *a map of the codebase* from day one.

Cassie Hang
Computer Science Researcher

31, University researcher, Minneapolis

Needs and motivations

- Analyze how structure and visualization affect codebase comprehension
- Study architecture complexity across branches and commits

Pain Points

- Manual diagram creation is slow and inconsistent
- Hard to align research metric across codebase evolution

How We Can Help

Our automated visualization engine provides consistent, analyzable mappings, helping Cassie transform *code evolution into measurable insights*.

MORE USER PERSONAS

Tyler Roberts
Product Manager

25, Product Manager, Los Angeles

Needs and motivations

- Visualize the architecture of a feature directly from the source code
- Ensure the user experience aligns with system capabilities and constraints

Pain Points

- Hard to explain different modules interact without relying solely on engineers' explanations
- Hard to visualize how backend components connect to the UI or API endpoints

How We Can Help

Our instant generated diagrams provides analyzable diagrams, helping help Tyler plan and run his meetings and his sprint reviews.

Blake Li
Software Team Lead

30, Senior software engineer, Irvine

Needs and motivations

- Visualize team codebase system architecture
- Track system architecture diagrams that evolve across sprints

Pain Points

- Hard to detect emergent technical debt or architectural drift early
- Difficult to confirm whether implementation matches the original design

How We Can Help

Our generated architecture diagrams provide Blake *a* mapping of the entire codebase to visualize and track the system architecture.

Freddie Reynolds
QA and Test engineer

27, QA engineer, Minneapolis

Needs and motivations

- Analyze development flow and identify potential risks to build better tests
- Visualize the branch structure and commits of a repo

Pain Points

- Difficult to track branch structure and commits of a repo
- Time-consuming to understand breakdown of each component

How We Can Help

Our automated visualization engine provides consistent, analyzable mappings, helping Freddie design better test cases to ensure the codebase implements the system architecture and design .

TOPICS OF RISK AND POTENTIAL CONCERNS

01

Essential Complexity

1. Cross-Language Parsing (Syntactic and Semantic nuances)
2. Repository Scale → Memory and Performance bottlenecks

02

Cognitive Complexity

1. Role Diversity → Varied Abstraction Levels
2. Large Diagrams → Reduced Visual Comprehension

03

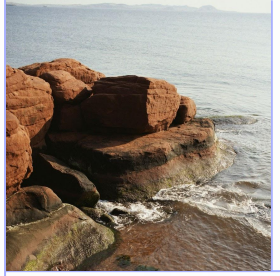
Temporal Complexity

1. Version evolution → Repositories evolve continuously

04

Structural Complexity

1. Scale of Interconnections → Dependencies multiply rapidly with repository size
2. Large Diagrams → Reduced Visual Comprehension

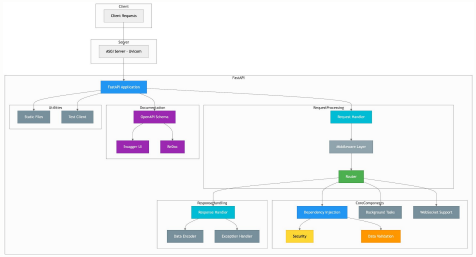


PREVIOUS WORK

Before starting on this project, we found some existing software related to this topic. Although they offer impressive functionality, they still have some inherent limitations.

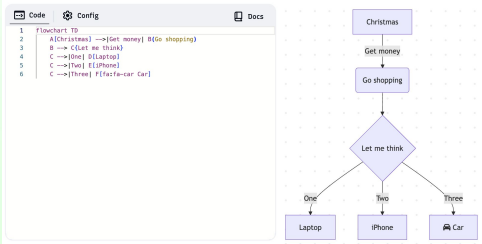
GitDiagram

Instant repo-structure diagrams — closest competitor; validates demand but lacks deeper semantic + version insights.



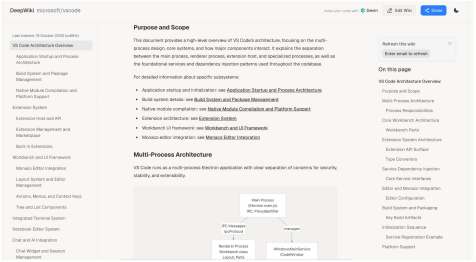
MermaidJS

Text-to-diagram rendering — useful formatting backend, but requires manual authoring, not automatic extraction.



DeepWiki

AI-generated repo documentation + structure views — inspires automated understanding but limited architectural depth.



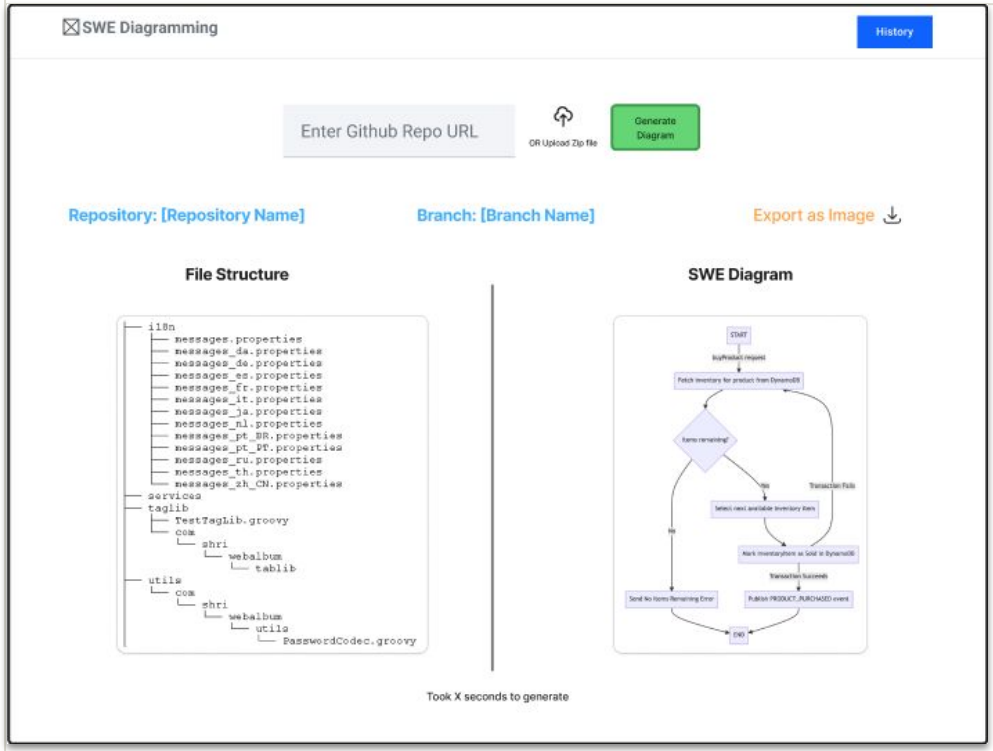
awesome-diagramming

A curated index of visualization tools — shows the ecosystem but doesn't analyze or auto-diagram code.

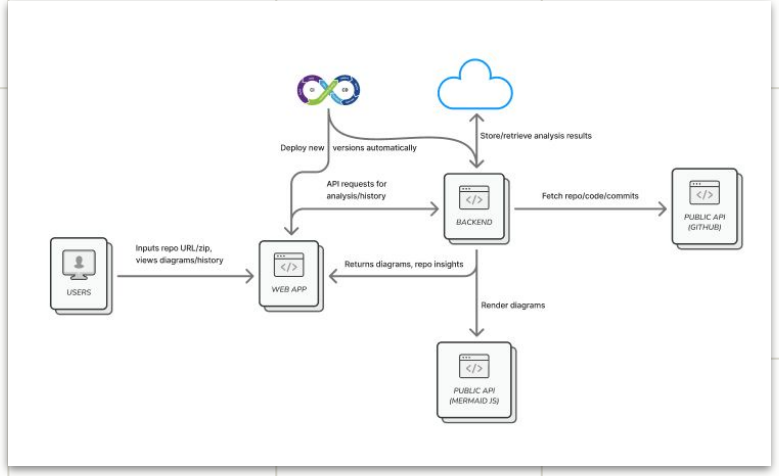
FEATURES AND PRIORITIES

Feature	Core	Medium Priority	Exploratory
Input & Access	▪ Import via GitHub links ▪ Zip file upload: Allow users to upload source ZIPs as an alternative to GitHub links	▪ Share to tools like slack, jira, github ▪ Private Access Tokens: Allow users to analyze private repositories.	
Repository Insights	▪ List all components and latest commit author/date	▪ Repo metrics (file count, language, folder depth ▪ Provide a brief summary of what each component contains	▪ Coding language analysis for the repo
Visualization	▪ Generate an architecture diagram for the repo, able to select specific commit/branch	▪ Dropdown to select branches from repo to display ▪ Navtools to move to parts of the codebase in the diagram quickly	▪ Diagram change history (diff view) ▪ Export Mermaid JS code ▪ Built-in option to copy/edit Mermaid JS code
Usability & Personalization	▪ Export diagram as PNG/SVG or embedded links ▪ Cached repo history: Save previously analyzed repositories.	▪ Construct in-depth documentation based on the diagram	▪ Annotate diagrams with notes and highlights ▪ User-type based specialized analysis

WIREFRAMES



SYSTEM DESIGN



THANK YOU