# ScopeDocs: Living Documentation for High-Velocity Engineering Teams

## TL;DR

Fast-growing engineering teams constantly struggle with fragmented documentation, forgotten context, and knowledge loss across systems. ScopeDocs solves this by providing automated documentation with memory and traceability, deeply integrated into GitHub, Linear, and Slack at the MVP. The solution is tailored for engineering teams needing persistent architecture/RFC history, onboarding context, change logging, and reliable integration mapping—without becoming yet another docs portal.

---

## Goals

### Business Goals

- Achieve 85% adoption rate among engineering teams at target high-velocity startups within 6 months of launch.
- Reduce onboarding time for new engineers by 30% compared to baseline.
- Drive at least one meaningful code- or process-related insight per week per team through documentation surfacing.
- Demonstrate measurable reduction (20%) in context-seeking interruptions in target user interviews post-launch.

### User Goals

- Access trustworthy documentation that automatically stays up-to-date with current code, tickets, and decisions.
- Instantly answer "why is this here?" and "who changed this?" for any module or system.
- Integrate seamlessly into daily workflows (PRs, incidents, onboarding) with minimal context switching.
- Accelerate ramp-up for new hires with slide-based, ADHD-friendly documentation spanning live code, architectural diagrams, and inner source annotations.

### Non-Goals

- Not aiming to replace external product/marketing or compliance documentation.
- Will not support non-engineering/technical teams (finance, HR) in the MVP.
- No support for legacy, on-prem-only VCS or ticketing integrations at launch (e.g., Perforce, Jira Data Center).
- ScopeDocs is not for general business or user-facing documentation (e.g., product manuals, help centers).
- Does NOT generate or manage API reference docs.
- Not designed as a full-featured documentation portal or as a replacement for broader knowledge bases like Confluence or Notion.

---

## User Stories

- **Core - Individual Contributor - Engineers (Engineer, Senior Engineer, On-call Developer)**
    - Goal: Understand, modify, and debug the system quickly with reliable, up-to-date context.
    - As an engineer,
        - I want documentation and examples to automatically reflect recent code changes so I never rely on stale information.
        - I want to explore a codebase with guided overviews and contextual explanations so I can orient myself quickly in unfamiliar areas.
        - I want to annotate code and have that knowledge visible to teammates so important context isn't lost in chat threads.
        - I want to trace a feature from RFC through PRs and deployments so I understand intent and can debug effectively.
        - I want onboarding summaries that highlight key files, history, and ownership so joining a repo is seamless.
        - *As an engineer on call*, I want to jump directly from an incident to relevant code, docs, and past fixes so I can resolve issues quickly.
        - I want to see previous incidents and their resolutions when working in related code so recurring issues are avoided.
    - ***New Hire / Onboarding Engineer***
        - Goal: Ramp up quickly without overloading teammates.
            - As a new hire,
                - I want guided walkthroughs of the system so I can begin contributing safely in my first weeks.
                - I want onboarding summaries showing code structure and ownership so I know who to approach when needed.
                - I want to search for a system or concept and see architecture, decisions, and ownership so I avoid repeatedly asking for tribal knowledge.
                - I want to review historical architecture and related code changes so I understand both how and why the system evolved.
- **Technical Leadership - Tech Lead, Engineering Manager**
    - Goal: Maintain system clarity, reduce repeated discussions, and enable team autonomy.
    - As a technical leader,
        - I want a canonical log of architectural decisions so teams don't repeat past discussions.
        - I want system integration points and dependencies surfaced so risks are visible during planning and reviews.
        - I want code changes to link back to discussions and decisions so historical intent is preserved.
        - I want new team members to self-onboard using accurate system context so leadership time isn't spent answering historical questions.
- **Product & Delivery - Product Managers**
    - Goal: Understand technical changes affecting product delivery.
    - As a PM,
        - I want to see which parts of the codebase changed for a given ticket so I understand why a feature was implemented.

        - I want updated system diagrams and change summaries so I can understand technical decisions affecting releases.
- **Data & Platform Lead**
    - Goal: Maintain reliable data pipelines and system evolution.
    - As a data/platform lead,
        - I want updated diagrams and dependency mappings so pipeline upgrades don't cause regressions.
        - I want audit trails for data model changes linked to tickets and discussions so system evolution remains traceable.

---

## Functional Requirements

- **Core Code Ingestion & Sync (Priority: Critical)**
    - Real-time GitHub ingestion: Codebase, commits, diffs.
        - GitHub: Pull RFCs, PR discussions, commit history, and code change context.
    - Map code, PRs, and linked Linear tickets bi-directionally.
        - Linear: Ingest issue and change decisions, link tasks to code changes.
    - Slack: Surface architecture/thread discussions, tie to change logs.
    - Detect and highlight undocumented/changed code regions.
    - Surfacing commit history and author context per file/line.
- **Workflow Integrations (Priority: High)**
    - Linear ticket linking and status syncing.
    - Slack integration to surface knowledge base in context (e.g., when a code snippet is posted).
    - PR comment bot that links docs to reviews.
- **UI Paradigms (Priority: Critical)**
    - Slide-based navigation: "Story mode" for code exploration and onboarding.
    - ADHD-friendly layouts: split focus panes, quick-jump anchor links, collapsible summaries.
    - Inline annotation and comment threads tied to code lines or logical modules.
- **Bidirectional Documentation Sync (Priority: High)**
    - Updates in docs automatically suggest PRs to code comments (and optionally vice versa).
    - Support for living architectural diagrams: synced to detected code structure and routes.
- **AI/Automation Enhancements (Priority: Stretch, not MVP)**
    - Summarize code changes and auto-suggest doc updates using AI.
    - Automatically generate "what changed this week" briefs.

---

## User Experience

**ScopeDocs – User Experience & Core Capabilities**

- **1. Entry Point & First-Time Experience**
    - Users typically discover ScopeDocs through onboarding invitations, Slack announcements, or repository access.
    - On first login, users see:
    - A concise welcome walkthrough explaining how ScopeDocs differs from static documentation tools.

    - An optional interactive tour showing:
    - Live documentation syncing with code
    - Story-mode documentation previews
    - Integration setup (e.g., ticketing and communication tools)
    - The goal is rapid comprehension with minimal cognitive load.
- **2. Core Product Experience**
    - **Step 1 — System Overview Dashboard**
        - Users land on a visual-first dashboard showing:
        - System dependency graph
        - Key system flows
        - Module ownership
        - Recently changed areas
        - Quick navigation to:
        - Open PRs
        - Incident history
        - Onboarding walkthroughs
        - The experience prioritizes clarity and visual navigation over dense text.
    - **Step 2 — Code & Documentation Exploration**
        - When a module or file is selected:
        - Code and documentation appear in a split view
        - **Users see:**
            - Current code
            - Commit and change history
            - Linked documentation
            - Ticket and discussion context
            - A slide-based module summary explains:
            - Why it exists
            - How it evolved
            - Upstream dependencies
            - Downstream consumers
            - Users understand both how and why a system evolved.
    - **Step 3 — Annotation & Contribution**
        - Users can add documentation or context directly:
        - Inline, markdown-style annotation interface
        - Notes can remain private or be published to the team
        - Contributions sync automatically with relevant code areas
        - Knowledge becomes persistent instead of living in chats.
    - **Step 4 — Continuous Sync & Discovery**
        - Documentation updates propagate automatically:
        - Changes appear during PR reviews
        - Updates surface via Slack notifications
        - Tickets link to relevant documentation
        - Users see indicators of changes since last visit
        - Documentation evolves alongside code.
    - **Step 5 — Onboarding Mode**

        - New hires can enter a guided onboarding flow featuring:
        - Prioritized system walkthroughs
        - Key system flows and ownership
        - Common pain points and troubleshooting guides
        - Progress tracking across onboarding material
        - This reduces onboarding dependency on senior engineers.
- **3. Memory & Traceability Engine (Core Capability)**
    - ScopeDocs maintains a persistent contextual graph linking:
    - RFCs
    - Tickets
    - Pull requests
    - Releases
    - Incidents
    - Code changes
    - Users can query:
        - Why a change occurred
        - When decisions were made
        - What systems were impacted
        - Who owns or contributed to components
        - This creates longitudinal system memory.
- **4. Integrations (High Priority)**
    - ScopeDocs connects documentation to live workflows:
    - Code Hosting
    - Pulls commit history, PR discussions, and code context.
    - Issue Tracking
    - Links tickets and decisions to code and documentation changes.
    - Communication Tools
    - Surfaces relevant discussion threads and associates them with system changes.
- **5. Flows for Key Engineering Pain Points (Medium Priority)**
    - Architecture & RFC Memory
    - Automatically records approvals, discussions, and implementation paths.
    - Codebase Onboarding
    - Generates contextual onboarding packets linking code, decisions, and ownership.
    - Change & Decision Logging
    - Maintains automatic documentation of major system changes and reversions.
    - Integration & Dependency Mapping
    - Continuously updates maps of service and system dependencies.
- **6. Advanced Features & Edge Cases**
    - To support real-world workflows:
    - Jump-to-author for recently modified files
    - Drift detection flags outdated documentation
    - Documentation merge conflict resolution tools
    - Toggle overlays to reduce UI clutter
    - Side-by-side documentation diffs
- **7. UX & Accessibility Principles**

    - The interface emphasizes accessibility and usability:
    - High-contrast visuals and adjustable motion controls
    - Slide-based presentation adaptable to screen sizes
    - Keyboard shortcuts and focus modes for power users
    - Persistent progress tracking
    - Screen-reader compatibility and semantic structure
- **8. Collaboration & Adaptation (Lower Priority)**
    - Future improvements include:
    - Inline suggestions or corrections
    - Ability to merge or split documentation artifacts as systems evolve

---

## Narrative

At Arcade, weeks fly by with back-to-back feature launches, emergency patches, and bug triage. With six engineers, three PMs, and a constant churn of new hires, tribal knowledge often goes missing in Slack threads or outdated Notion pages. When an on-call developer faces a 2 am incident, the only context is a cryptic comment in a months-old PR—or frantic DMs that break someone's sleep.

Enter ScopeDocs. Suddenly, every critical change, from the reason a table was renamed to the rationale behind a feisty API flag, is mapped into always-current, navigable documentation. A new backend hire hits "Story mode," scrolling through slide-visuals tracing architectural decisions, code owner handoffs, and last-week's incidents—all in sync with what's really running in prod. Engineers annotate critical functions; their insights propagate to Linear tickets and PR templates instantly. High-velocity teams unblock themselves, ramping new members in days, not weeks, and surfacing the "why" behind every change—without ever leaving Slack or their code review flow.

ScopeDocs turns documentation from a static drag into a living asset. No more duplicated effort, blind debugging, or knowledge hoarding. As the team grows, everyone stays in sync, builds context faster, and focuses on what matters: shipping, learning, and winning together.

---

## Success Metrics

- **User Adoption Rate:** % of engineers using ScopeDocs weekly.
- **Onboarding Time Reduction:** Days for new hires from first day to first PR merged.
- **Documentation Accuracy:** % of modules where documentation matches current code and tickets.
- **Context-Seeking Reduction:** # of internal pings/slack questions for "where/why/how" per team/week.
- **Engagement with Doc Changes:** # of meaningful doc updates/annotations per user/month.
- **System Uptime:** % availability, target 99.9%.

### User-Centric Metrics

- Weekly active users (WAU): Monitored via dashboard analytics.
- Onboarding completion rates (tracked per new hire account).
- User satisfaction/NPS on ease of finding knowledge and onboarding (quarterly survey).

### Business Metrics

- Reduction in average onboarding time (monitored by HR/IT).
- Lowered cost of handoffs and context loss (as measured by incident postmortems).
- Number of logos acquired in high-velocity startup segment.

### Technical Metrics

- Sync latency: Seconds from code/ticket change to doc update.
- Documentation accuracy audit: Scheduled reviews; % of outdated docs detected and resolved within SLA.
- Uptime and incident rates for real-time integration services.

### Tracking Plan

- User logins and session starts.
- Clicks on “Story mode” and onboarding slides.
- Creation, edit, and sync events for annotations and docs.
- Integration events: PR comment insertions, Slack queries served.
- AI auto-summary acceptance rate.

---

## Technical Considerations

### Technical Needs

- Real-time ingestion pipelines for code (GitHub), tickets (Linear), and Slack messages.
- Scalable backend for documentation storage, diffing, and querying.
- Front-end slide-based interface with flexible navigation and annotation APIs.
- Change detection service for code/doc drift and sync triggers.

### Integration Points

- GitHub (Public/Private Repos): Required for MVP for code/diff sync.
- Linear: Issue and project linkage.
- Slack: Message surfacing, onboarding rollouts, inline link previews.
- Future: Expand to other VCS or ticketing tools (not in MVP).

### Data Storage & Privacy

- Encrypted at rest and in transit.
- Docs and metadata mapped to team/org; logical multi-tenancy enforced.
- Minimal access principle: Only authorized repo or ticket data ingested.
- GDPR-compliance, with right-to-be-forgotten and audit tracking.

### Scalability & Performance

- MVP targets: 5–10 engineering teams, each with 5–30 users and codebases up to 500k LOC.
- Sub-second doc/code search; updates reflected in under 30 seconds.
- Async processing for expensive AI summarization (not critical-path in MVP).

### Potential Challenges

- Robust, low-latency code/ticket integration without over-polling APIs.
- Accurate change attribution when code is refactored or moved.
- Handling permissions and visibility (private repos, sensitive Linear tickets).
- Doc drift: Keeping documentation up-to-date with minimum manual effort.

---

## Milestones & Sequencing

### Project Estimate

- Small Team: 1–2 weeks for MVP launch.

### Team Size & Composition

- Small Team: 2 people (1 product-minded engineer, 1 design/front-end focused engineer).

### Suggested Phases

**Phase 1: MVP Build (1 week)**

- Key Deliverables:
    - Core GitHub integration and live code ingestion pipeline (Engineering).
    - Slide-based, ADHD-friendly UI with system overviews (Design/Front-end).
    - Inline annotation and PR bot for documentation sync (Engineering).
    - Live onboarding mode for new hires (Design/Front-end).
    - Secure multi-tenancy, basic Linear and Slack integrations (Engineering).
- Dependencies:
    - Access to target org GitHub, Linear, Slack APIs.
    - Test codebases and user sample from founding team.

**Phase 2: Pilot & Feedback Loop (1 week)**

- Key Deliverables:
    - Early rollout to internal team and 1–2 design partner startups.
    - Feedback collection: friction logs, UX pain points, critical bugs.
    - Weekly product iterations on doc drift detection, search, and notifications.
- Dependencies:
    - Willing design partner teams, real-world codebases.

**Phase 3: AI and Automation Enhancements (Stretch, post-MVP)**

- Key Deliverables:
    - Auto-summarization of changes.
    - Doc suggestion workflows driven by code/issue deltas.
- Dependencies:
    - Labeled data from MVP usage.
    - Opt-in from teams for AI features.

---