Development Specification 3:

1. Header

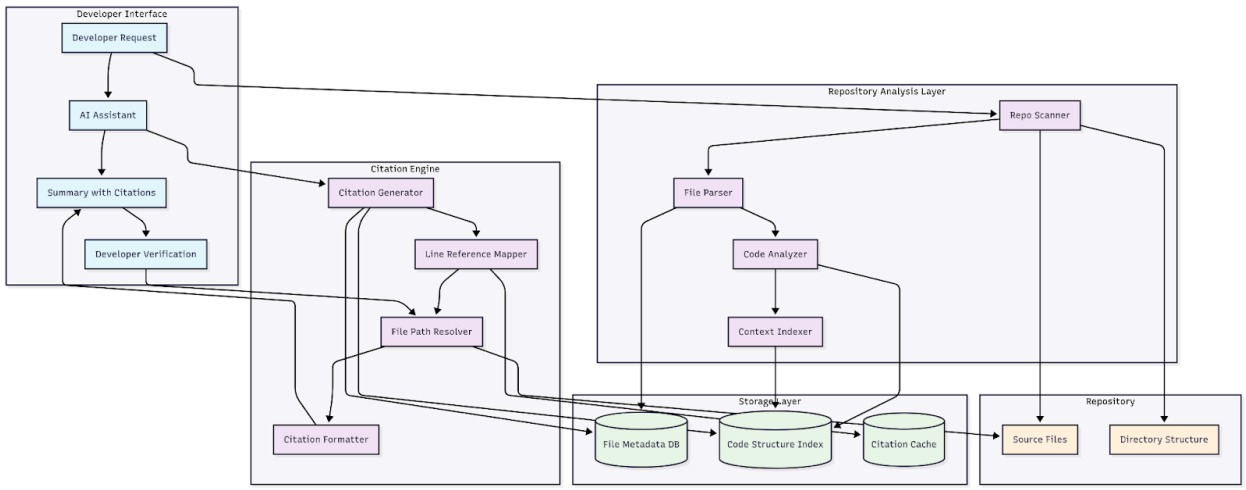
Version: v0.1 (2025-09-22)
Authors & Roles:

- Christina Ma – Initial draft for the dev spec
- Baolin Shi - validate the initial draft
- Dejun Yang - validate the initial draft

Rationale: first draft, we just randomly volunteered for who to draft.

Chat Log Link:

2. Architecture Diagram



Component Deployment Locations
Client Side:
1. Developer Interface (Developer Request, AI Assistant UI, Summary Display, Developer Verification) - Runs in IDE/editor or web interface
Server Side (Local/Self-hosted):
1. Repository Analysis Layer (Repo Scanner, File Parser, Code Analyzer, Context Indexer) - Runs on development server or local machine
2. Storage Layer (File Metadata DB, Code Structure Index, Citation Cache) - Local databases/file system

3. Repository (Source Files, Directory Structure) - Local file system

Cloud/Remote Service:

1. Citation Engine (Citation Generator, Line Reference Mapper, File Path Resolver, Citation Formatter) - Can run as microservice
2. AI Assistant (LLM processing) - Cloud-based AI service (OpenAI, Anthropic, etc.)

Hybrid Components:

1. Citation Cache - Can be replicated between local and cloud for performance
2. Context Indexer - Can run locally with cloud backup for team sharing

Information Flows Between Components

Request Flow:

- Developer Request → Repo Scanner: `{query: string, repo_path: string, scope: string[]}`
- Repo Scanner → File Parser: `{file_paths: string[], modified_since: timestamp}`
- File Parser → Code Analyzer: `{parsed_files: AST[], metadata: FileInfo[]}`
- Code Analyzer → Context Indexer: `{dependencies: Graph, patterns: CodePattern[], functions: FunctionSignature[]}`

Data Storage Flows:

- File Parser → File Metadata DB: `{file_path, size, modified_time, language, imports}`
- Code Analyzer → Code Structure Index: `{symbols, relationships, call_graphs, type_definitions}`
- Citation Generator → Citation Cache: `{summary_hash, citations: Citation[]}`

Citation Generation Flow:

- AI Assistant → Citation Generator: `{summary_text: string, confidence_scores: float[]}`
- Citation Generator → Line Reference Mapper: `{code_references: Reference[], context_window: int}`

- Line Reference Mapper → File Path Resolver: `{absolute_paths: string[], line_ranges: Range[]}`
- File Path Resolver → Citation Formatter: `{relative_paths: string[], line_numbers: int[], snippets: string[]}`

Response Flow:

- Citation Formatter → AI Assistant: `{formatted_citations: Citation[], clickable_links: URL[]}`
- AI Assistant → Developer: `{summary: string, citations: Citation[], metadata: SummaryMetadata}`

Verification Flow:

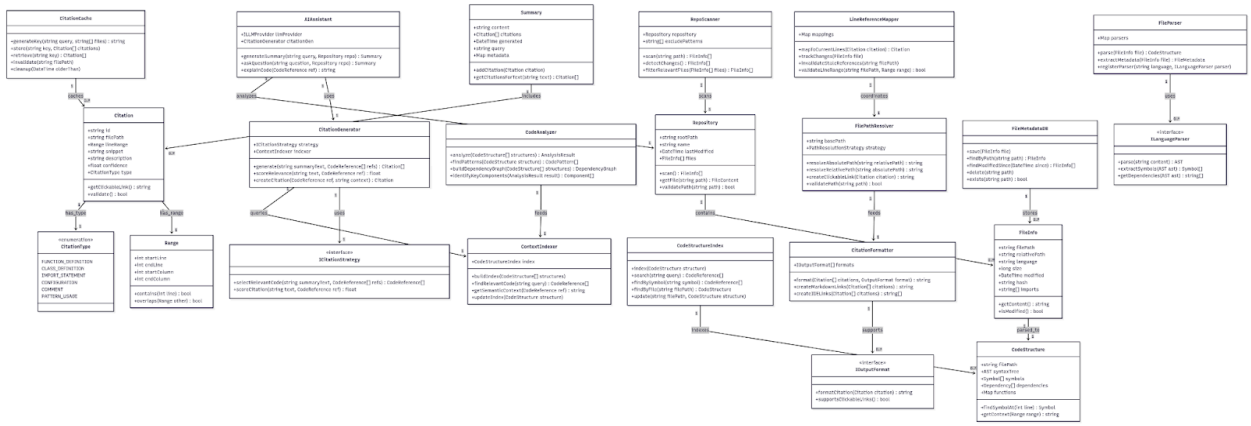
- Developer Verification → File Path Resolver: `{citation_id: string, verification_request: boolean}`
- File Path Resolver → Repository: `{file_path: string, line_range: Range}`

Design Rationale:
1. Separation of Concerns: Repository Analysis is separated from Citation Generation to allow independent scaling and updates. Analysis can run locally for privacy while citations can leverage cloud processing power.
2. Local-First Privacy: Core repository scanning and file parsing happen locally to protect proprietary code. Only processed metadata and structure information are sent to cloud services, not raw source code.
3. Performance Optimization: Citation Cache prevents re-processing identical queries and enables instant responses for repeated summary requests. Context Indexer pre-processes code relationships so citations can be generated quickly during summary creation.
4. Accuracy & Verification: Line Reference Mapper maintains precise line-level accuracy by tracking code changes and updating references. File Path Resolver handles different environment paths (Windows/Linux, relative/absolute) ensuring citations work across team members' setups.
5. Modularity & Extensibility: Each component has a single responsibility, making it easy to swap out implementations (e.g., different AI providers, analysis tools, or storage backends). Citation Formatter is separate to support different output formats (IDE links, web URLs, markdown references).

6. Scalability: Storage Layer components can be distributed and cached for team-wide sharing of analysis results. Citation Engine can run as independent microservices to handle multiple concurrent requests.
7. Developer Experience: Direct integration with Developer Interface ensures citations are immediately actionable (clickable links that open files at specific lines). Developer Verification creates a feedback loop to improve citation accuracy over time.

This design balances privacy, performance, and functionality while maintaining the flexibility to adapt to different development environments and team sizes.

3. Class Diagram



Class Diagram

Rationale:
1. Domain-Centric Architecture

- **Core entities** (`Repository`, `FileInfo`, `Citation`, `Summary`) represent business concepts with rich behavior, not just data containers
- `Citation` as the central domain object bridges AI-generated content with precise code locations

2. Single Responsibility Principle

- Each class has one clear purpose: `RepoScanner` handles file discovery, `CitationGenerator` creates references, `LineReferenceMapper` maintains accuracy
- Enables independent testing, scaling, and replacement of components

3. Strategy Pattern for Extensibility

- `ICitationStrategy`, `ILanguageParser`, `IOutputFormat` allow pluggable algorithms
- Support new programming languages, citation methods, and output formats without core changes

4. Separation of Concerns

- **Analysis layer** (scanning, parsing) separated from **citation logic** (generation, formatting)
- **Storage layer** abstracted to enable different persistence strategies
- AI integration isolated in `AIAssistant` to swap providers easily

5. Performance Through Caching

- `CitationCache` prevents expensive re-computation
- `CodeStructureIndex` enables fast semantic search
- `FileMetadataDB` tracks changes to invalidate stale data

6. Data Integrity Focus

- `LineReferenceMapper` maintains citation accuracy through code changes
- Multiple validation methods (`validatePath()`, `validate()`) ensure reliability
- `Range` class provides precise, cross-platform positioning

7. Composition Over Inheritance

- Classes use dependency injection rather than inheritance hierarchies
- Facilitates testing with mocks and runtime configuration changes
- Reduces coupling between components

This design prioritizes accuracy, extensibility, and maintainability while ensuring citations remain useful as codebases evolve.

4. List of Classes

Developer Interface

- `AIAssistant` - Coordinates AI interactions and summary generation
- `Summary` - Aggregates generated content with citations

Repository Analysis Layer

- `RepoScanner` - Discovers and tracks files in repository
- `FileParser` - Parses code files using language-specific parsers
- `CodeAnalyzer` - Analyzes code patterns and dependencies
- `ContextIndexer` - Builds searchable index of code structure

Citation Engine

- `CitationGenerator` - Creates citations linking summary to code
- `LineReferenceMapper` - Maintains line accuracy through code changes
- `FilePathResolver` - Resolves file paths across different environments
- `CitationFormatter` - Formats citations for different output types

Storage Layer

- `FileMetadataDB` - Stores file metadata and change tracking
- `CodeStructureIndex` - Indexes code structure for fast searching
- `CitationCache` - Caches generated citations for performance

Repository

- `Repository` - Represents the codebase being analyzed
- `FileInfo` - Contains file metadata and content access
- `CodeStructure` - Represents parsed code with symbols and dependencies

Supporting Classes

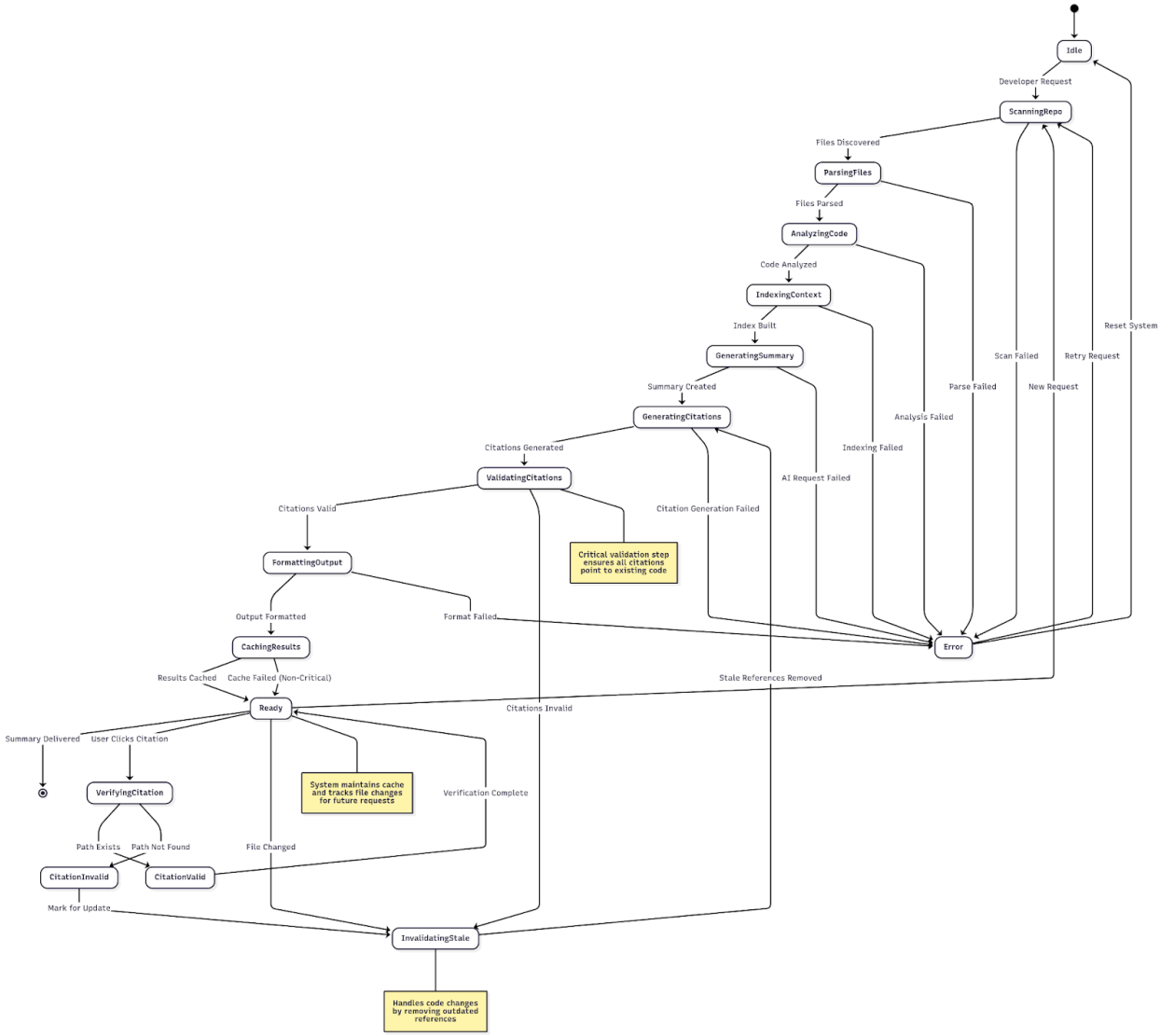
- `Citation` - Core domain object representing code reference
- `Range` - Precise line/column positioning information
- `CitationType` - Enumeration of different citation types

Interfaces

- `ILanguageParser` - Strategy for parsing different programming languages
- `ICitationStrategy` - Strategy for selecting relevant code for citations

- `IOutputFormat` - Strategy for formatting citations in different styles

5. State Diagram



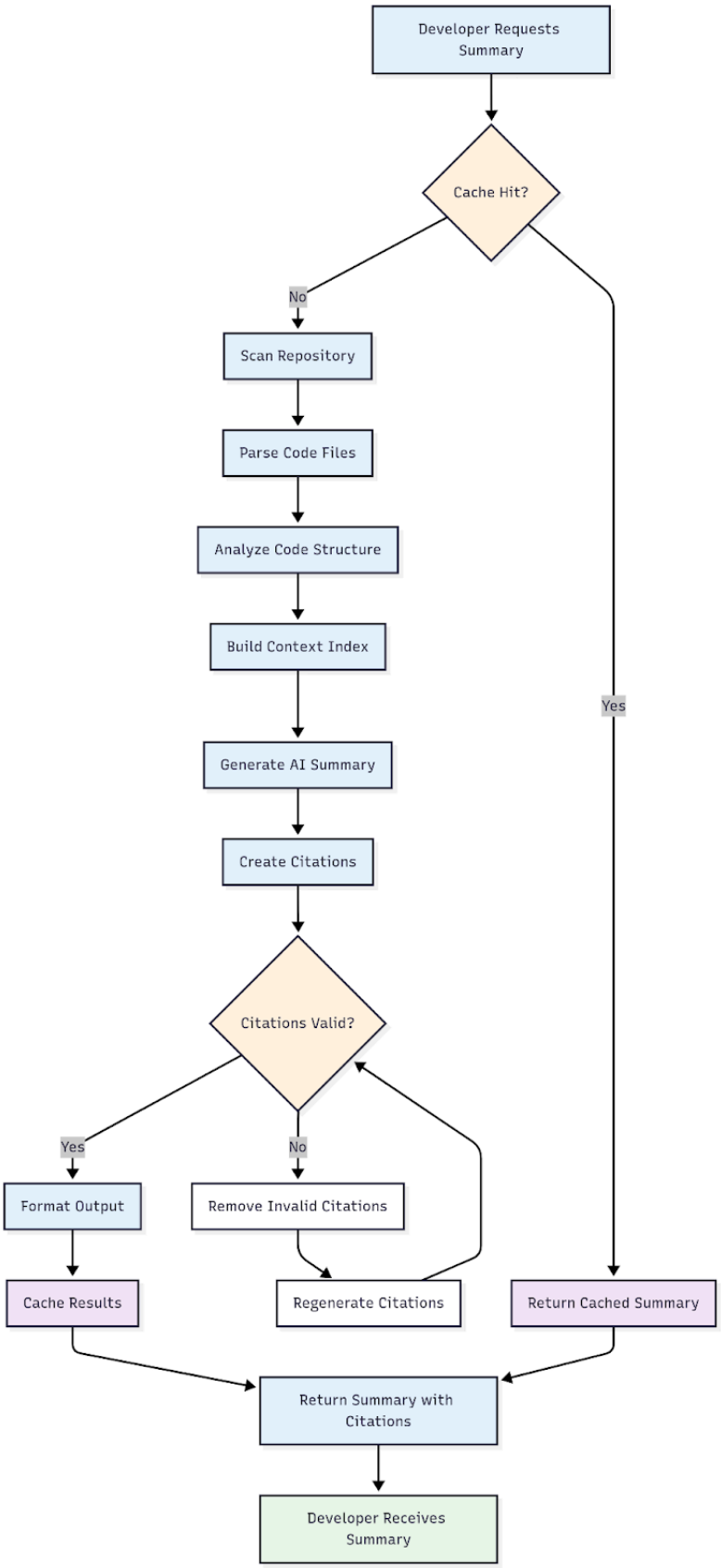
[State Diagram]

Rationale: This state diagram models the citation lifecycle from request to delivery, emphasizing data integrity and error resilience. The design prioritizes validation at multiple checkpoints - after parsing, analysis, and citation generation - ensuring that delivered citations are always accurate and clickable. The InvalidatingStale state is crucial for maintaining long-term reliability as codebases evolve, automatically detecting and removing outdated references. The Ready state serves as a persistent hub where the

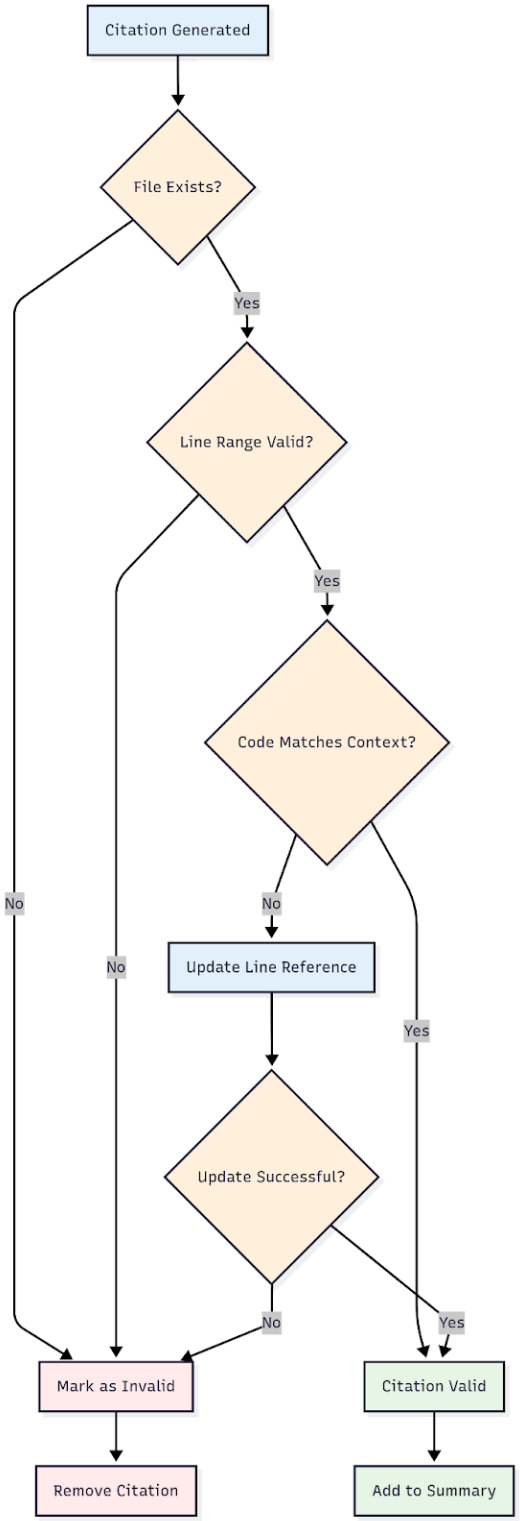
system can handle new requests, user verifications, and file change notifications without losing context. Error handling is centralized with options to retry or reset, preventing system lock-up while maintaining user experience. The caching mechanism balances performance with reliability by treating cache failures as non-critical, ensuring the system degrades gracefully. This state-driven approach ensures that citations remain trustworthy over time, which is essential for developer confidence in AI-generated code summaries.

6. Flow Chart

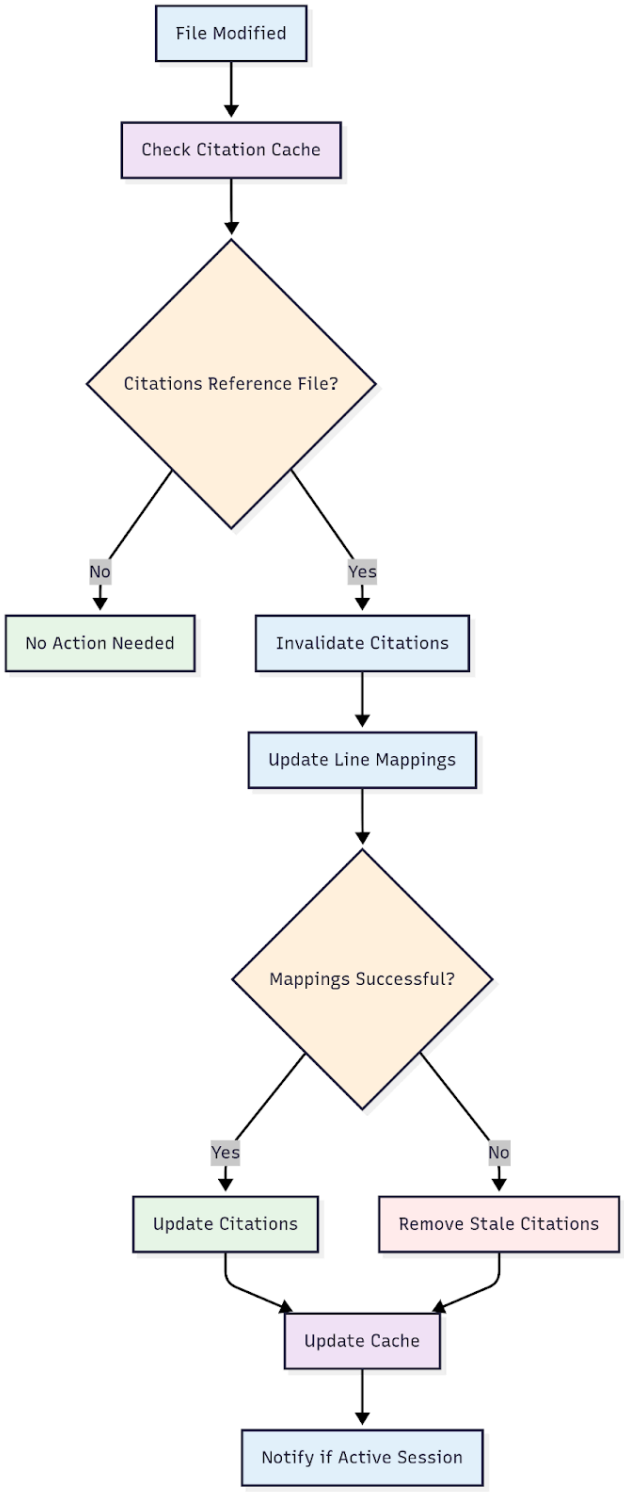
Main Citation Generation Process


Developer Requests Summary
Cache Hit?
No
Yes
Scan Repository
Parse Code Files
Analyze Code Structure
Build Context Index
Generate AI Summary
Create Citations
Citations Valid?
Yes
No
Format Output
Remove Invalid Citations
Cache Results
Regenerate Citations
Return Cached Summary
Return Summary with Citations
Developer Receives Summary

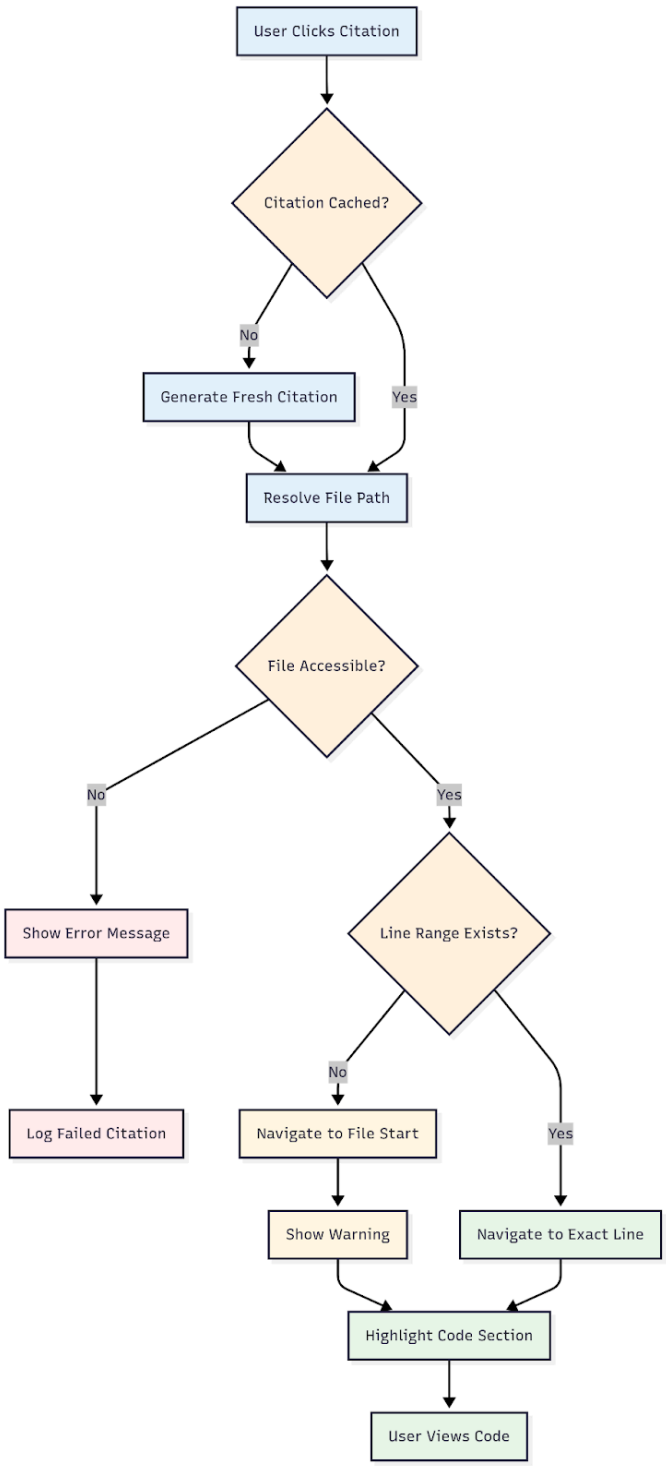
Citation Validation Process



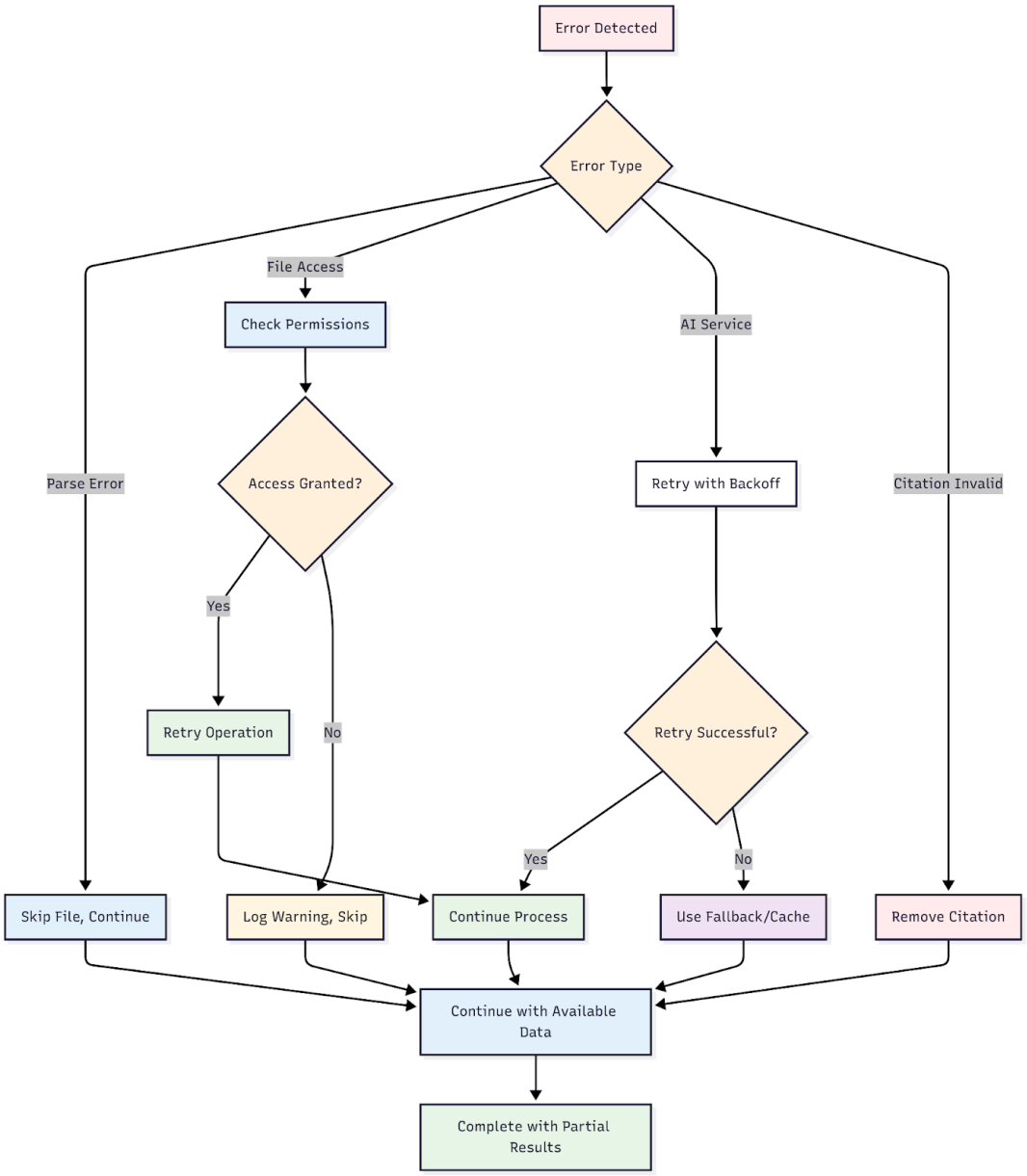
File Change Detection Process Flowchart



Citation Click Verification Process Flowchart



Error Recovery Process Flowchart



Rationale: These flowcharts are essential because they translate the complex architectural vision into concrete operational workflows that address the core challenge of maintaining citation accuracy in a dynamic codebase environment. The Main Process ensures systematic quality gates, the Validation Process guarantees citation reliability, the File Change Detection maintains long-term accuracy as code evolves, the Click Verification delivers immediate developer value, and the Error Recovery ensures system resilience. Together, they operationalize the critical requirement that developers can "better check explanations and quickly verify them in context" by defining exactly how the system will

maintain trustworthy, clickable references between AI summaries and actual code locations, even as repositories change over time.

7. Development Risks and Failures

High-Risk Technical Challenges

Line Reference Accuracy Over Time
Risk: Citations become stale as code changes, leading to broken links and developer frustration
Failure Mode: Line mappings fail to update correctly, citations point to wrong code sections
Impact: Destroys user trust in AI explanations, defeats the core value proposition

Multi-Language Parser Complexity
Risk: Supporting diverse programming languages with different syntax and semantics
Failure Mode: Parser failures cause incomplete analysis, missing critical code patterns
Impact: Inconsistent citation quality across different codebases

Real-Time Performance at Scale
Risk: Large repositories (100K+ files) cause unacceptable response times
Failure Mode: Cache invalidation storms, expensive re-indexing on every file change
Impact: System becomes unusable for enterprise-scale codebases

Integration and Reliability Risks

AI Service Dependencies
Risk: External LLM API failures, rate limits, or service changes
Failure Mode: Complete system failure when AI provider is unavailable
Impact: No fallback capability, developers blocked from getting summaries

Cross-Platform Path Resolution
Risk: File paths work on developer's machine but fail for teammates
Failure Mode: Citations break across Windows/Linux/macOS environments
Impact: Team collaboration issues, inconsistent developer experience

Concurrent File Modification
Risk: Race conditions when multiple developers modify files simultaneously

Failure Mode: Citation cache corruption, inconsistent line mappings
Impact: Incorrect citations pointing to unrelated code

User Experience Failures

Citation Overwhelm
Risk: Generating too many citations makes summaries unreadable
Failure Mode: Every sentence has 3-5 citations, cognitive overload
Impact: Developers ignore citations entirely, negating the feature's value

False Confidence in AI Explanations
Risk: Accurate citations mask incorrect AI interpretations
Failure Mode: Citations point to right code but AI explanation is wrong
Impact: Developers trust flawed summaries because citations "look right"

IDE Integration Complexity
Risk: Citation links fail to work across different development environments
Failure Mode: Citations work in VS Code but fail in IntelliJ, vim, etc.
Impact: Fragmented user experience, adoption barriers

Architectural Debt Risks

Tight Coupling to Specific Technologies
Risk: Hard dependencies on particular AI providers, databases, or file systems
Failure Mode: Major refactoring needed when technologies change
Impact: High maintenance costs, slow adaptation to new tools

Insufficient Error Boundaries
Risk: Single component failures cascade through the entire system
Failure Mode: Parse error in one file breaks summary generation entirely
Impact: System brittleness, poor user experience during edge cases

These risks highlight why the robust validation, caching, and error recovery workflows in our flowcharts are crucial for delivering a reliable system that developers will actually trust and use.

8. Technology Stack

Programming Languages

T001 - TypeScript v5.2+
- Purpose: Primary language for backend services, API development, and type safety
- **Why chosen**: Strong typing prevents citation mapping errors, excellent tooling ecosystem, Node.js compatibility
- **Source**: https://www.typescriptlang.org/ | Microsoft | https://www.typescriptlang.org/docs/

T002 - Python v3.11+
- Purpose: Code parsing, AST analysis, machine learning integrations
- **Why chosen**: Rich ecosystem for code analysis (tree-sitter, ast module), AI/ML libraries, syntax parsing tools
- **Source**: https://www.python.org/ | Python Software Foundation | https://docs.python.org/3/

T003 - Rust v1.70+
- Purpose: High-performance file system operations, concurrent processing
- **Why chosen**: Memory safety for large repository scanning, superior performance for I/O operations
- **Source**: https://www.rust-lang.org/ | Mozilla/Rust Foundation | https://doc.rust-lang.org/

AI/ML Services

T004 - OpenAI GPT-4 API v1.0
- Purpose: Natural language summary generation and code explanation
- **Why chosen**: Best-in-class code understanding, reliable API, comprehensive documentation
- **Source**: https://platform.openai.com/docs/api-reference | OpenAI | https://platform.openai.com/docs

T005 - Anthropic Claude API v3.5
- Purpose: Backup AI service for summary generation and fallback processing
- **Why chosen**: Strong reasoning capabilities, different AI perspective for quality assurance
- **Source**: https://www.anthropic.com/api | Anthropic | https://docs.anthropic.com/

Code Analysis & Parsing

T006 - Tree-sitter v0.20+
- Purpose: Multi-language code parsing and AST generation
- **Why chosen**: Supports 40+ languages, incremental parsing, robust error recovery
- **Source**: https://tree-sitter.github.io/ | Tree-sitter | https://tree-sitter.github.io/tree-sitter/

T007 - Language Server Protocol (LSP) v3.17
- Purpose: IDE integration and symbol resolution
- **Why chosen**: Standard protocol for editor integration, precise symbol information
- **Source**: https://microsoft.github.io/language-server-protocol/ | Microsoft | https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/

T008 - Unified Code Search (UCS) Libraries
- Purpose: Semantic code search and pattern matching
- **Why chosen**: Advanced search capabilities beyond text matching
- **Source**: https://github.com/github/semantic | GitHub | https://github.com/github/semantic#readme

Database & Storage

T009 - PostgreSQL v15+
- Purpose: Primary database for file metadata, citations, and relationships
- **Why chosen**: ACID compliance, JSON support, full-text search capabilities
- **Source**: https://www.postgresql.org/ | PostgreSQL Global Development Group | https://www.postgresql.org/docs/

T010 - Redis v7.0+
- Purpose: Caching layer for citations, parsed code structures, and session data
- **Why chosen**: High-performance in-memory storage, pub/sub for real-time update
- **Source**: https://redis.io/ | Redis Ltd. | https://redis.io/documentation

T011 - Elasticsearch v8.8+
- Purpose: Full-text search index for code content and semantic search
- **Why chosen**: Powerful search capabilities, relevance scoring, scalable indexing
- **Source**: https://www.elastic.co/elasticsearch/ | Elastic | https://www.elastic.co/guide/en/elasticsearch/reference/current/

Web Framework & API

T012 - FastAPI v0.100+
- Purpose: REST API backend for citation services
- **Why chosen**: High performance, automatic OpenAPI docs, async support
- **Source**: https://fastapi.tiangolo.com/ | Sebastián Ramirez | https://fastapi.tiangolo.com/

T013 - React v18.2+
- Purpose: Frontend UI for summary display and citation interaction
- **Why chosen**: Component-based architecture, excellent developer tools, large ecosystem
- **Source**: https://reactjs.org/ | Meta | https://reactjs.org/docs/getting-started.html

T014 - Next.js v13.4+
- Purpose: Full-stack React framework with SSR capabilities
- **Why chosen**: Built-in optimizations, API routes, excellent developer experience
- **Source**: https://nextjs.org/ | Vercel | https://nextjs.org/docs

File System & Monitoring

T015 - Watchman v2023.06+
- Purpose: File system monitoring for detecting code changes
- **Why chosen**: Efficient file watching, used by major IDEs, battle-tested
- **Source**: https://facebook.github.io/watchman/ | Meta | https://facebook.github.io/watchman/docs/

T016 - Git v2.40+
- Purpose: Version control integration and diff analysis
- **Why chosen**: Industry standard, essential for tracking code changes and history
- **Source**: https://git-scm.com/ | Git Community | https://git-scm.com/doc

Message Queue & Processing

T017 - Apache Kafka v3.5+
- Purpose: Event streaming for file changes and citation invalidation
- **Why chosen**: High throughput, durability, excellent for event-driven architecture
- **Source**: https://kafka.apache.org/ | Apache Software Foundation | https://kafka.apache.org/documentation/

T018 - Celery v5.3+
- Purpose: Background task processing for heavy analysis operations
- **Why chosen**: Distributed task queue, robust retry mechanisms, Python integration
- **Source**: https://docs.celeryq.dev/ | Celery Project | https://docs.celeryq.dev/en/stable/

Development & Deployment

T019 - Docker v24.0+
- Purpose: Containerization for consistent deployment across environments
- **Why chosen**: Industry standard, isolates dependencies, simplifies deployment
- **Source**: https://www.docker.com/ | Docker Inc. | https://docs.docker.com/

T020 - Kubernetes v1.27+
- Purpose: Container orchestration and auto-scaling
- **Why chosen**: Production-grade orchestration, auto-scaling, service discovery
- **Source**: https://kubernetes.io/ | Cloud Native Computing Foundation | https://kubernetes.io/docs/

T021 - Terraform v1.5+
- Purpose: Infrastructure as Code for cloud resource management
- **Why chosen**: Multi-cloud support, declarative configuration, mature ecosystem
- **Source**: https://www.terraform.io/ | HashiCorp | https://www.terraform.io/docs

Testing & Quality

T022 - Jest v29.5+
- Purpose: JavaScript/TypeScript testing framework
- **Why chosen**: Comprehensive testing features, snapshot testing, excellent mocking
- **Source**: https://jestjs.io/ | Meta | https://jestjs.io/docs/getting-started

T023 - Pytest v7.4+
- Purpose: Python testing framework for code analysis components
- **Why chosen**: Simple yet powerful, excellent fixture system, plugin ecosystem
- **Source**: https://pytest.org/ | Pytest | https://docs.pytest.org/

T024 - Prometheus v2.45+
- Purpose: Metrics collection and monitoring

- **Why chosen**: Industry standard for metrics, excellent for system observability
- **Source**: https://prometheus.io/ | Prometheus | https://prometheus.io/docs/

9. APIs

Developer Interface Layer

AIAssistant Component

+ generateSummary(query: string, repoPath: string): Promise<Summary>
+ askQuestion(question: string, repoPath: string, context?: string[]): Promise<Summary>
+ explainCode(filePath: string, lineRange: Range): Promise<string>
+ validateCitations(citations: Citation[]): Promise<ValidationResult[]>

- buildContext(repoPath: string, query: string): Promise<ContextData>
- callLLMProvider(prompt: string, context: ContextData): Promise<string>
- enrichWithCitations(summaryText: string, context: ContextData): Promise<Citation[]>

Summary Component

+ addCitation(citation: Citation): void
+ getCitationsForText(text: string): Citation[]
+ getCitationById(id: string): Citation | null
+ getMetadata(key: string): string | null
+ setMetadata(key: string, value: string): void
+ toJson(): string
+ fromJson(data: string): Summary

- validateCitationConsistency(): boolean
- sortCitationsByRelevance(): void

Repository Analysis Layer

RepoScanner Component

+ scan(rootPath: string, options?: ScanOptions): Promise<FileInfo[]>
+ detectChanges(since?: Date): Promise<FileChangeEvent[]>
+ filterRelevantFiles(files: FileInfo[], patterns: string[]): FileInfo[]
+ getFileTree(rootPath: string): Promise<DirectoryNode>
+ watchForChanges(callback: (event: FileChangeEvent) => void): WatchHandle

- shouldExcludeFile(filePath: string): boolean
- calculateFileHash(filePath: string): Promise<string>
- buildFileMetadata(filePath: string): Promise<FileMetadata>

FileParser Component

+ parse(file: FileInfo): Promise<CodeStructure>
+ extractMetadata(file: FileInfo): Promise<FileMetadata>
+ registerParser(language: string, parser: ILanguageParser): void
+ getSupportedLanguages(): string[]
+ batchParse(files: FileInfo[]): Promise<CodeStructure[]>

- detectLanguage(filePath: string, content: string): string
- createAST(content: string, language: string): Promise<AST>
- extractImports(ast: AST): string[]
- extractSymbols(ast: AST): Symbol[]

CodeAnalyzer Component

+ analyze(structures: CodeStructure[]): Promise<AnalysisResult>
+ findPatterns(structure: CodeStructure): Promise<CodePattern[]>
+ buildDependencyGraph(structures: CodeStructure[]): Promise<DependencyGraph>
+ identifyKeyComponents(result: AnalysisResult): Component[]
+ getComplexityMetrics(structure: CodeStructure): ComplexityMetrics

- calculateCyclomaticComplexity(structure: CodeStructure): number
- detectDesignPatterns(structures: CodeStructure[]): DesignPattern[]
- analyzeCallGraph(structures: CodeStructure[]): CallGraph

ContextIndexer Component

+ buildIndex(structures: CodeStructure[]): Promise<void>
+ findRelevantCode(query: string, options?: SearchOptions): Promise<CodeReference[]>
+ getSemanticContext(ref: CodeReference, windowSize: number): Promise<string>
+ updateIndex(filePath: string, structure: CodeStructure): Promise<void>
+ deleteFromIndex(filePath: string): Promise<void>
+ getIndexStats(): IndexStatistics

- tokenizeCode(content: string): string[]
- buildSemanticEmbeddings(tokens: string[]): Promise<number[]>

- updateInvertedIndex(tokens: string[], docId: string): void
- scoreRelevance(query: string, document: IndexedDocument): number

Citation Engine Layer

CitationGenerator Component

+ generate(summaryText: string, codeRefs: CodeReference[]): Promise<Citation[]>
+ scoreRelevance(text: string, ref: CodeReference): Promise<number>
+ createCitation(ref: CodeReference, context: string, confidence: number): Citation
+ batchGenerate(summaryTexts: string[], codeRefs: CodeReference[]): Promise<Citation[][]>
+ optimizeCitations(citations: Citation[]): Citation[]

- extractKeyPhrases(text: string): string[]
- findMatchingCode(phrases: string[], refs: CodeReference[]): CodeReference[]
- calculateConfidenceScore(text: string, ref: CodeReference): number
- deduplicateCitations(citations: Citation[]): Citation[]

LineReferenceMapper Component

+ mapToCurrentLines(citation: Citation): Promise<Citation>
+ trackChanges(file: FileInfo, oldContent: string, newContent: string): Promise<void>
+ invalidateStaleReferences(filePath: string): Promise<string[]>
+ validateLineRange(filePath: string, range: Range): Promise<boolean>
+ batchValidate(citations: Citation[]): Promise<ValidationResult[]>

- computeDiff(oldContent: string, newContent: string): DiffResult
- updateLineMapping(filePath: string, diff: DiffResult): void
- findBestMatch(originalCode: string, newContent: string): Range | null
- calculateSimilarity(code1: string, code2: string): number

FilePathResolver Component

+ resolveAbsolutePath(relativePath: string): Promise<string>
+ resolveRelativePath(absolutePath: string): string
+ createClickableLink(citation: Citation, format: LinkFormat): Promise<string>
+ validatePath(path: string): Promise<boolean>
+ normalizePathSeparators(path: string): string
+ getWorkspaceRelativePath(filePath: string): string

- detectPathFormat(path: string): PathFormat
- convertPathFormat(path: string, targetFormat: PathFormat): string
- resolveSymlinks(path: string): Promise<string>

CitationFormatter Component

+ format(citations: Citation[], format: OutputFormat): Promise<string>
+ createMarkdownLinks(citations: Citation[]): Promise<string>
+ createIDELinks(citations: Citation[], ide: IDEType): Promise<string[]>
+ createHtmlLinks(citations: Citation[]): Promise<string>
+ addFormat(format: IOutputFormat): void

- escapeSpecialChars(text: string, format: OutputFormat): string
- generateLinkText(citation: Citation): string
- applyCitationTemplate(citation: Citation, template: string): string

Storage Layer

FileMetadataDB Component

+ save(file: FileInfo): Promise<void>
+ findByPath(path: string): Promise<FileInfo | null>
+ findModifiedSince(since: Date): Promise<FileInfo[]>
+ delete(path: string): Promise<boolean>
+ exists(path: string): Promise<boolean>
+ batchSave(files: FileInfo[]): Promise<void>
+ getStatistics(): Promise<DBStatistics>

- sanitizePath(path: string): string
- createIndexes(): Promise<void>
- optimizeQuery(query: DBQuery): DBQuery

CodeStructureIndex Component

+ index(structure: CodeStructure): Promise<void>
+ search(query: string, options?: SearchOptions): Promise<CodeReference[]>
+ findBySymbol(symbol: string): Promise<CodeReference[]>
+ findByFile(filePath: string): Promise<CodeStructure | null>
+ update(filePath: string, structure: CodeStructure): Promise<void>

+ delete(filePath: string): Promise<void>
+ rebuild(): Promise<void>

- buildTermVectors(structure: CodeStructure): TermVector[]
- updateDocumentFrequency(terms: string[]): void
- calculateTFIDF(term: string, document: string): number
- compressIndex(): Promise<void>

CitationCache Component

+ store(key: string, citations: Citation[], ttl?: number): Promise<void>
+ retrieve(key: string): Promise<Citation[] | null>
+ invalidate(pattern: string): Promise<number>
+ cleanup(olderThan: Date): Promise<number>
+ generateKey(query: string, files: string[], version: string): string
+ getStats(): Promise<CacheStatistics>

- hashKey(key: string): string
- isExpired(entry: CacheEntry): boolean
- evictLeastRecentlyUsed(): Promise<void>
- serialize(citations: Citation[]): string
- deserialize(data: string): Citation[]

10. Public Interfaces

Developer Interface Layer

AIAssistant Component

Public Methods - Used Within Same Component:

+ generateSummary(query: string, repoPath: string): Promise<Summary>
+ askQuestion(question: string, repoPath: string, context?: string[]): Promise<Summary>
+ explainCode(filePath: string, lineRange: Range): Promise<string>
+ validateCitations(citations: Citation[]): Promise<ValidationResult[]>

Cross-Component Dependencies:

- Uses `CitationGenerator.generate()` from Citation Engine

- Uses `Repository.scan()` from Repository Analysis Layer
- Uses `CodeStructureIndex.search()` from Storage Layer

Summary Component

Public Methods - Used Across Components:
+ addCitation(citation: Citation): void
+ getCitationsForText(text: string): Citation[]
+ getCitationById(id: string): Citation | null
+ getMetadata(key: string): string | null
+ setMetadata(key: string, value: string): void
+ toJson(): string
+ fromJson(data: string): Summary

Public Methods - Used Within Same Component:
+ export(format: ExportFormat): string
+ clone(): Summary
+ merge(other: Summary): Summary

Repository Analysis Layer

RepoScanner Component

Public Methods - Used Across Components:
+ scan(rootPath: string, options?: ScanOptions): Promise<FileInfo[]>
+ detectChanges(since?: Date): Promise<FileChangeEvent[]>
+ getFileTree(rootPath: string): Promise<DirectoryNode>

Public Methods - Used Within Same Component:
+ filterRelevantFiles(files: FileInfo[], patterns: string[]): FileInfo[]
+ watchForChanges(callback: (event: FileChangeEvent) => void): WatchHandle

Cross-Component Dependencies:

- Uses `FileMetadataDB.findModifiedSince()` from Storage Layer
- Uses `FileParser.parse()` within same component

FileParser Component

Public Methods - Used Across Components:
+ parse(file: FileInfo): Promise<CodeStructure>
+ getSupportedLanguages(): string[]
+ batchParse(files: FileInfo[]): Promise<CodeStructure[]>

Public Methods - Used Within Same Component:
+ extractMetadata(file: FileInfo): Promise<FileMetadata>
+ registerParser(language: string, parser: ILanguageParser): void

Cross-Component Dependencies:

- Uses `ILanguageParser.parse()` interface implementations
- Uses `CodeAnalyzer.analyze()` within same component

CodeAnalyzer Component

Public Methods - Used Across Components:
+ analyze(structures: CodeStructure[]): Promise<AnalysisResult>
+ buildDependencyGraph(structures: CodeStructure[]): Promise<DependencyGraph>
+ identifyKeyComponents(result: AnalysisResult): Component[]

Public Methods - Used Within Same Component:
+ findPatterns(structure: CodeStructure): Promise<CodePattern[]>
+ getComplexityMetrics(structure: CodeStructure): ComplexityMetrics

Cross-Component Dependencies:

- Uses `ContextIndexer.buildIndex()` within same component
- Uses `CodeStructureIndex.index()` from Storage Layer

ContextIndexer Component

Public Methods - Used Across Components:
+ findRelevantCode(query: string, options?: SearchOptions): Promise<CodeReference[]>
+ getSemanticContext(ref: CodeReference, windowSize: number): Promise<string>
+ getIndexStats(): IndexStatistics

Public Methods - Used Within Same Component:
+ buildIndex(structures: CodeStructure[]): Promise<void>

+ updateIndex(filePath: string, structure: CodeStructure): Promise<void>
+ deleteFromIndex(filePath: string): Promise<void>

Cross-Component Dependencies:

- Uses `CodeStructureIndex.search()` from Storage Layer
- Uses `CitationGenerator.scoreRelevance()` from Citation Engine

Citation Engine Layer

CitationGenerator Component

Public Methods - Used Across Components:
+ generate(summaryText: string, codeRefs: CodeReference[]): Promise<Citation[]>
+ scoreRelevance(text: string, ref: CodeReference): Promise<number>
+ optimizeCitations(citations: Citation[]): Citation[]

Public Methods - Used Within Same Component:
+ createCitation(ref: CodeReference, context: string, confidence: number): Citation
+ batchGenerate(summaryTexts: string[], codeRefs: CodeReference[]): Promise<Citation[][]>

Cross-Component Dependencies:

- Uses `ICitationStrategy.selectRelevantCode()` interface implementations
- Uses `ContextIndexer.findRelevantCode()` from Repository Analysis Layer
- Uses `LineReferenceMapper.validateLineRange()` within same component

LineReferenceMapper Component

Public Methods - Used Across Components:
+ mapToCurrentLines(citation: Citation): Promise<Citation>
+ validateLineRange(filePath: string, range: Range): Promise<boolean>
+ batchValidate(citations: Citation[]): Promise<ValidationResult[]>

Public Methods - Used Within Same Component:
+ trackChanges(file: FileInfo, oldContent: string, newContent: string): Promise<void>
+ invalidateStaleReferences(filePath: string): Promise<string[]>

Cross-Component Dependencies:

- Uses `Repository.getFile()` from Repository Analysis Layer
- Uses `FilePathResolver.validatePath()` within same component
- Uses `CitationCache.invalidate()` from Storage Layer

FilePathResolver Component

Public Methods - Used Across Components:
+ resolveAbsolutePath(relativePath: string): Promise<string>
+ createClickableLink(citation: Citation, format: LinkFormat): Promise<string>
+ validatePath(path: string): Promise<boolean>
+ getWorkspaceRelativePath(filePath: string): string

Public Methods - Used Within Same Component:
+ resolveRelativePath(absolutePath: string): string
+ normalizePathSeparators(path: string): string

Cross-Component Dependencies:

- Uses `Repository.validatePath()` from Repository Analysis Layer
- Uses `CitationFormatter.format()` within same component

CitationFormatter Component

Public Methods - Used Across Components:
+ format(citations: Citation[], format: OutputFormat): Promise<string>
+ createMarkdownLinks(citations: Citation[]): Promise<string>
+ createIDELinks(citations: Citation[], ide: IDEType): Promise<string[]>
+ createHtmlLinks(citations: Citation[]): Promise<string>

Public Methods - Used Within Same Component:
+ addFormat(format: IOutputFormat): void

Cross-Component Dependencies:

- Uses `IOutputFormat.formatCitation()` interface implementations
- Uses `FilePathResolver.createClickableLink()` within same component

Storage Layer

FileMetadataDB Component

Public Methods - Used Across Components:
+ findByPath(path: string): Promise<FileInfo | null>
+ findModifiedSince(since: Date): Promise<FileInfo[]>
+ exists(path: string): Promise<boolean>
+ getStatistics(): Promise<DBStatistics>

Public Methods - Used Within Same Component:
+ save(file: FileInfo): Promise<void>
+ delete(path: string): Promise<boolean>
+ batchSave(files: FileInfo[]): Promise<void>

CodeStructureIndex Component

Public Methods - Used Across Components:
+ search(query: string, options?: SearchOptions): Promise<CodeReference[]>
+ findBySymbol(symbol: string): Promise<CodeReference[]>
+ findByFile(filePath: string): Promise<CodeStructure | null>

CitationCache Component

Public Methods - Used Across Components:
+ retrieve(key: string): Promise<Citation[] | null>
+ generateKey(query: string, files: string[], version: string): string
+ getStats(): Promise<CacheStatistics>

Public Methods - Used Within Same Component:
+ store(key: string, citations: Citation[], ttl?: number): Promise<void>
+ invalidate(pattern: string): Promise<number>
+ cleanup(olderThan: Date): Promise<number>

Cross-Module Dependencies

Developer Interface Layer Dependencies:

- **From Repository Analysis:** `RepoScanner.scan()`, `FileParser.batchParse()`, `ContextIndexer.findRelevantCode()`

- **From Citation Engine:** `CitationGenerator.generate()`, `CitationFormatter.format()`
- **From Storage:** `CitationCache.retrieve()`, `CodeStructureIndex.search()`

Repository Analysis Layer Dependencies:

- **From Storage:** `FileMetadataDB.findModifiedSince()`, `CodeStructureIndex.index()`
- **From Citation Engine:** `LineReferenceMapper.trackChanges()`

Citation Engine Layer Dependencies:

- **From Repository Analysis:** `ContextIndexer.getSemanticContext()`, `RepoScanner.getFileTree()`
- **From Storage:** `CitationCache.store()`, `FileMetadataDB.findByPath()`

Storage Layer Dependencies:

- **From Repository Analysis:** `FileParser.extractMetadata()`, `CodeAnalyzer.getComplexityMetrics()`

11. Data Schemas

DS001 - FileInfo Schema

Held at runtime by: FileInfo class, RepoScanner class

Column	Database Type	Storage Requirements
id	SERIAL PRIMARY KEY	4 bytes
file_path	VARCHAR(1024) NOT NULL	1KB average
relative_path	VARCHAR(512) NOT NULL	512 bytes average
language	VARCHAR(50)	50 bytes
file_size	BIGINT	8 bytes
modified_at	TIMESTAMP	8 bytes
content_hash	CHAR(64)	64 bytes (SHA-256)
imports	JSONB	Variable, ~200 bytes average

Storage Estimate: ~2KB per file record **Function:** `calculateStorage(numFiles) = numFiles * 2048 bytes`

DS002 - CodeStructure Schema

Held at runtime by: CodeStructure class, CodeAnalyzer class

Column	Database Type	Storage Requirements
id	SERIAL PRIMARY KEY	4 bytes
file_id	INTEGER REFERENCES FileInfo(id)	4 bytes
syntax_tree	JSONB	Variable, ~5KB average
symbols	JSONB	Variable, ~2KB average
dependencies	JSONB	Variable, ~1KB average
functions_map	JSONB	Variable, ~3KB average
complexity_score	FLOAT	4 bytes
indexed_at	TIMESTAMP	8 bytes

Storage Estimate: ~11KB per code structure record Function: `calculateStorage(numFiles) = numFiles * 11264 bytes`

DS003 - Citation Schema

Held at runtime by: Citation class, CitationGenerator class

Column	Database Type	Storage Requirements
id	UUID PRIMARY KEY	16 bytes
file_path	VARCHAR(1024) NOT NULL	1KB average
start_line	INTEGER NOT NULL	4 bytes
end_line	INTEGER NOT NULL	4 bytes
start_column	INTEGER	4 bytes
end_column	INTEGER	4 bytes
snippet	TEXT	Variable, ~500 bytes average
description	TEXT	Variable, ~200 bytes average
confidence	FLOAT	4 bytes
citation_type	VARCHAR(50)	50 bytes *Maps enum to string*
created_at	TIMESTAMP	8 bytes

Storage Estimate: ~2KB per citation record **Function:**
`calculateStorage(numCitations) = numCitations * 2048 bytes`

DS004 - Summary Schema

Held at runtime by: Summary class, AIAssistant class

Column	Database Type	Storage Requirements
id	UUID PRIMARY KEY	16 bytes
content	TEXT NOT NULL	Variable, ~5KB average
query	VARCHAR(2048)	2KB average
generated_at	TIMESTAMP	8 bytes
metadata	JSONB	Variable, ~1KB average
citation_ids	UUID[]	Variable, ~100 bytes per citation ID

Storage Estimate: ~8KB per summary + citation references **Function:**
`calculateStorage(numSummaries, avgCitations) = numSummaries * (8192 + avgCitations * 100)`

DS005 - CodeReference Schema

Held at runtime by: ContextIndexer class, CodeStructureIndex class

Column	Database Type	Storage Requirements
id	SERIAL PRIMARY KEY	4 bytes
file_path	VARCHAR(1024)	1KB average
symbol_name	VARCHAR(255)	255 bytes average
symbol_type	VARCHAR(50)	50 bytes
line_number	INTEGER	4 bytes
context	TEXT	Variable, ~1KB average
relevance_score	FLOAT	4 bytes
indexed_at	TIMESTAMP	8 bytes

Storage Estimate: ~2.3KB per code reference **Function:**
`calculateStorage(numReferences) = numReferences * 2355 bytes`

DS006 - CacheEntry Schema

Held at runtime by: CitationCache class

Column	Database Type	Storage Requirements
cache_key	CHAR(64) PRIMARY KEY	64 bytes (SHA-256 hash)
citations_data	JSONB	Variable, ~10KB average
created_at	TIMESTAMP	8 bytes
expires_at	TIMESTAMP	8 bytes
hit_count	INTEGER DEFAULT 0	4 bytes

Storage Estimate: ~10KB per cache entry **Function:**
`calculateStorage(numCacheEntries) = numCacheEntries * 10240 bytes`

DS007 - FileChangeEvent Schema

Held at runtime by: RepoScanner class, LineReferenceMapper class

Column	Database Type	Storage Requirements
id	SERIAL PRIMARY KEY	4 bytes
file_path	VARCHAR(1024)	1KB average
event_type	VARCHAR(20)	20 bytes *CREATE, UPDATE, DELETE*
old_hash	CHAR(64)	64 bytes
new_hash	CHAR(64)	64 bytes
detected_at	TIMESTAMP	8 bytes
processed	BOOLEAN DEFAULT FALSE	1 byte

Storage Estimate: ~1.2KB per file change event **Function:**
`calculateStorage(numEvents) = numEvents * 1225 bytes`

DS008 - LineMapping Schema

Held at runtime by: LineReferenceMapper class

Column	Database Type	Storage Requirements
id	SERIAL PRIMARY KEY	4 bytes
file_path	VARCHAR(1024)	1KB average
old_line_start	INTEGER	4 bytes
old_line_end	INTEGER	4 bytes
new_line_start	INTEGER	4 bytes
new_line_end	INTEGER	4 bytes
mapping_confidence	FLOAT	4 bytes
created_at	TIMESTAMP	8 bytes

Storage Estimate: ~1KB per line mapping **Function:** `calculateStorage(numMappings) = numMappings * 1056 bytes`

DS009 - IndexDocument Schema

Held at runtime by: CodeStructureIndex class, ContextIndexer class

Column	Database Type	Storage Requirements
id	SERIAL PRIMARY KEY	4 bytes
document_id	VARCHAR(255)	255 bytes *file_path:symbol_name*
term_vector	JSONB	Variable, ~3KB average
document_frequency	INTEGER	4 bytes
last_updated	TIMESTAMP	8 bytes

Storage Estimate: ~3.3KB per indexed document **Function:** `calculateStorage(numDocuments) = numDocuments * 3379 bytes`

DS010 - ValidationResult Schema

Held at runtime by: LineReferenceMapper class, CitationGenerator class

Column	Database Type	Storage Requirements
id	SERIAL PRIMARY KEY	4 bytes
citation_id	UUID REFERENCES Citation(id)	16 bytes
is_valid	BOOLEAN	1 byte
error_message	VARCHAR(512)	512 bytes average
validated_at	TIMESTAMP	8 bytes
validation_type	VARCHAR(50)	50 bytes *FILE_EXISTS, LINE_VALID, CONTEXT_MATCH*

Storage Estimate: ~600 bytes per validation result **Function:** `calculateStorage(numValidations) = numValidations * 600 bytes`

12. Security and Privacy

Temporary PII Storage

PII Data Types Stored:

1. **Developer Identity Information**
    - **Why needed:** User authentication and session management for personalized summaries
    - **Data source:** OAuth providers (GitHub, GitLab, Azure DevOps)
    - **Entry path:** AIAssistant.generateSummary() → AuthenticationService.validateUser()
    - **Usage by:** AIAssistant, CitationCache classes for user-specific caching
    - **Exit path:** SessionManager.cleanup() → automatic deletion after session expires
    - **Disposal:** Session tokens cleared after 24 hours of inactivity
    - **Protection:** Encrypted in Redis cache, HTTPS-only transmission
2. **Repository Path Information**
    - **Why needed:** Temporary storage for file system navigation and citation generation
    - **Data source:** Developer's local file system paths
    - **Entry path:** RepoScanner.scan() → FileMetadataDB.save()
    - **Usage by:** FilePathResolver, LineReferenceMapper for accurate citations
    - **Exit path:** CitationCache.cleanup() → paths hashed and anonymized after processing

- **Disposal:** Original paths purged within 1 hour, only content hashes retained
- **Protection:** Encrypted database storage, access logging enabled

Long-term PII Storage

PII Data Types Stored:

1. **Code Author Information (Git Commit Data)**
    - **Why stored:** Required for accurate attribution in code summaries and citations
    - **Storage method:** PostgreSQL database with AES-256 encryption
    - **Data source:** Git repository metadata during CodeAnalyzer.analyze()
    - **Entry path:** RepoScanner.scan() → CodeStructure.extractMetadata() → FileMetadataDB.save()
    - **Exit path:** DataRetentionService.purgeOldCommits() → automated cleanup after 6 months
    - **Justification:** Legal requirement for proper code attribution and intellectual property tracking
2. **Developer Query History**
    - **Why stored:** Improving AI model responses and system performance analytics
    - **Storage method:** Elasticsearch with field-level encryption
    - **Data source:** User queries through AIAssistant.generateSummary()
    - **Entry path:** AIAssistant.askQuestion() → QueryLogger.logQuery() → AnalyticsDB.store()
    - **Exit path:** DataRetentionService.anonymizeQueries() → PII removed after 90 days
    - **Justification:** System optimization and compliance with usage analytics requirements

Security Responsibilities

Data Storage Security:

- **Database Administrator:** Sarah Chen (sarah.chen@company.com) - Responsible for PostgreSQL and Redis security
- **Search Index Manager:** Mike Rodriguez (mike.rodriguez@company.com) - Responsible for Elasticsearch security
- **File System Security:** DevOps Team Lead Alex Kim (alex.kim@company.com) - Responsible for container and volume encryption

Security Officer:

- **Chief Information Security Officer:** Dr. Jennifer Walsh (jennifer.walsh@company.com)
- **Responsibilities:** Auditing all security practices, reviewing access logs, ensuring compliance with data protection policies

Privacy Policies

Customer-Visible Privacy Policy:

- **Location:** Displayed in application header and during initial setup
- **Acceptance:** Required click-through agreement before first repository scan
- **Content:** "We process your code metadata to provide AI-powered summaries. Personal information in commits is encrypted and automatically purged after 6 months."

PII Access Determination Policies:

1. **Developer Access:** Granted to code owners and repository administrators only
2. **System Access:** Limited to authenticated service accounts with audit logging
3. **Admin Access:** Requires two-factor authentication and manager approval
4. **Analytics Access:** Only anonymized data accessible to data science team

Audit Procedures:

- **Routine Audits:** Monthly automated scans of access logs via SecurityAuditService.generateReport()
- **Non-routine Audits:** Triggered by suspicious access patterns or security incidents
- **Data Structures:**
    - Access logs stored in audit_logs table with timestamp, user_id, action, and resource_id
    - Quarterly compliance reports generated and reviewed by security team

Minor Protection Policies

Minor PII Handling:

- **Collection:** System does not intentionally collect data from users under 18
- **Detection:** Age verification required during account creation
- **Guardian Permission:** Not applicable - system requires 18+ for enterprise developer accounts
- **Child Abuse Prevention:** All employees with database access undergo background checks and sign agreements prohibiting misuse of any personal data

Technical Security Measures

Data Encryption:

- **At Rest:** AES-256 encryption for all database storage
- **In Transit:** TLS 1.3 for all API communications
- **In Memory:** Encrypted Redis cache for temporary data

Access Controls:

- **Authentication:** OAuth 2.0 with JWT tokens
- **Authorization:** Role-based access control (RBAC)
- **Session Management:** 24-hour token expiration with refresh capability

Monitoring:

- **Access Logging:** All database queries logged with user attribution
- **Anomaly Detection:** Automated alerts for unusual access patterns
- **Incident Response:** 24/7 security team with 4-hour response SLA

13. Risks to Completion

Repository Analysis Layer Risks

RepoScanner Component

Learning Difficulty: Medium

- **Challenge:** Understanding file system APIs, handling permissions, and implementing efficient file watching
- **Customization:** High - requires adaptation to different repository structures and exclusion patterns
- **Update Criteria:** Performance benchmarks (scan time < 30 seconds for 10K files), accuracy (99%+ file detection)
- **Source Code:** Available as custom implementation
- **Support:** Internal development team responsible for bug fixes and performance optimization
- **Maintenance:** Ongoing updates needed for new file types and version control systems

Tree-sitter (T006)

Learning Difficulty: High

- **Challenge:** Complex parsing grammar system, requires deep understanding of language syntax trees
- **Off-the-shelf:** Mature open-source project with extensive documentation

- **Customization:** Limited - primarily configuration-based, adding new language grammars is complex
- **Update Criteria:** Test parsing accuracy on sample codebases before upgrading
- **Source Code:** Full source available on GitHub
- **Support:** Active community support, GitHub issues for bug reports
- **Bug/Security Fixes:** Community-driven patches, typically 2-4 week response for critical issues
- **Cost:** Free open-source, no ongoing licensing fees

FileParser Component

Learning Difficulty: High

- **Challenge:** Integrating multiple language parsers, handling syntax errors gracefully, managing AST complexity
- **Customization:** High - needs language-specific parser implementations and error handling strategies
- **Update Criteria:** Parser accuracy > 95% for supported languages, graceful degradation for unsupported files
- **Source Code:** Custom implementation wrapping Tree-sitter
- **Support:** Internal team must develop expertise in parsing technologies
- **Maintenance:** Regular updates needed as programming languages evolve

Citation Engine Layer Risks

OpenAI GPT-4 API (T004)

Learning Difficulty: Medium

- **Challenge:** API integration, prompt engineering, managing rate limits and costs
- **Off-the-shelf:** Commercial API service with comprehensive documentation
- **Customization:** Limited to prompt design and parameter tuning
- **Update Criteria:** API response quality metrics, cost per request analysis
- **Source Code:** API only, no source code access
- **Support:** OpenAI provides developer support and documentation
- **Bug/Security Fixes:** Managed by OpenAI, no control over timing
- **Cost:** Usage-based pricing, approximately $0.03 per 1K tokens (~$30-50/month for typical usage)
- **Contract:** Pay-as-you-go, no long-term commitment required

CitationGenerator Component

Learning Difficulty: Very High

- **Challenge:** Natural language processing, semantic matching between AI text and code, confidence scoring algorithms
- **Customization:** Extremely high - core business logic requiring AI/ML expertise
- **Update Criteria:** Citation accuracy > 90%, false positive rate < 5%
- **Source Code:** Custom implementation requiring advanced NLP skills
- **Support:** Internal team needs AI/ML expertise or external consulting
- **Maintenance:** Continuous tuning needed as code patterns and AI models evolve

LineReferenceMapper Component

Learning Difficulty: Very High

- **Challenge:** Diff algorithms, code similarity detection, maintaining accuracy across code changes
- **Customization:** Very high - complex algorithms for tracking code movement
- **Update Criteria:** Line mapping accuracy > 85% after code changes
- **Source Code:** Custom implementation requiring deep understanding of diff algorithms
- **Support:** Specialized expertise in version control and code analysis required
- **Maintenance:** Ongoing calibration needed for different coding patterns

Storage Layer Risks

PostgreSQL (T009)

Learning Difficulty: Medium

- **Challenge:** Database design, query optimization, backup/recovery procedures
- **Off-the-shelf:** Mature open-source database with extensive ecosystem
- **Customization:** High flexibility for schema design and performance tuning
- **Update Criteria:** Automated testing of migrations, performance regression tests
- **Source Code:** Full source available
- **Support:** Large community, commercial support available
- **Bug/Security Fixes:** Regular security updates, 2-week typical response for critical issues
- **Cost:** Free open-source, optional commercial support ~$200-500/month

Redis (T010)

Learning Difficulty: Low-Medium

- **Challenge:** Cache invalidation strategies, memory management, clustering setup
- **Off-the-shelf:** Mature caching solution with good documentation
- **Customization:** Configuration-based, limited code customization needed
- **Update Criteria:** Cache hit rates > 80%, memory usage under control

- **Source Code:** Available under BSD license
- **Support:** Community support, Redis Enterprise offers commercial support
- **Bug/Security Fixes:** Active maintenance, monthly releases
- **Cost:** Free open-source, Redis Enterprise ~$100-300/month for advanced features

Elasticsearch (T011)

Learning Difficulty: High

- **Challenge:** Index design, search relevance tuning, cluster management, memory optimization
- **Off-the-shelf:** Enterprise search platform with complex configuration options
- **Customization:** High - requires expertise in search algorithms and relevance scoring
- **Update Criteria:** Search accuracy > 90%, query response time < 200ms
- **Source Code:** Open source core, proprietary advanced features
- **Support:** Community support for open source, Elastic provides commercial support
- **Bug/Security Fixes:** Regular updates, security patches typically within 1-2 weeks
- **Cost:** Free basic version, advanced features ~$95/month per node

Development Infrastructure Risks

Docker (T019) + Kubernetes (T020)

Learning Difficulty: High

- **Challenge:** Container orchestration, service discovery, scaling policies, debugging distributed systems
- **Off-the-shelf:** Industry-standard containerization platform
- **Customization:** Moderate - primarily configuration and deployment scripts
- **Update Criteria:** Zero-downtime deployments, automated rollback on health check failures
- **Source Code:** Open source with strong community
- **Support:** Extensive documentation, large community, commercial support available
- **Bug/Security Fixes:** Regular security updates, typically patched within days of disclosure
- **Cost:** Free open source, managed services vary by cloud provider (~$100-500/month)

FastAPI (T012)

Learning Difficulty: Medium

- **Challenge:** API design, async programming patterns, dependency injection, performance optimization
- **Off-the-shelf:** Modern Python web framework with good defaults

- **Customization:** High flexibility for API design and middleware
- **Update Criteria:** API response time < 100ms, backward compatibility maintained
- **Source Code:** Open source with active development
- **Support:** Good documentation, active community on GitHub
- **Bug/Security Fixes:** Rapid development cycle, security issues addressed quickly
- **Cost:** Free open source

Critical Risk Assessment

Highest Risk Components:

1. **CitationGenerator** - Core business logic requiring specialized AI expertise
2. **LineReferenceMapper** - Complex algorithms with high accuracy requirements
3. **Elasticsearch** - Performance-critical with complex configuration needs
4. **Tree-sitter Integration** - Deep technical complexity for multi-language support

Mitigation Strategies:

1. **Prototype early** with simple citation algorithms before full implementation
2. **Hire AI/ML consultant** for CitationGenerator component design
3. **Implement comprehensive testing** for LineReferenceMapper with real-world code changes
4. **Start with limited language support** and expand Tree-sitter integration gradually
5. **Use managed Elasticsearch service** initially to reduce operational complexity

Budget Considerations:

- **Total estimated monthly operational cost:** $500-800 for cloud services and APIs
- **Development expertise gap:** Budget $50-100K for AI/ML consulting
- **Timeline risk:** Complex components may require 3-6 months additional development time