Li'nage

Line-Level Version Control System

Technical Architecture Specification

Development Team

December 2025

Document Information

Version:	1.0
Status:	Technical Specification
Classification:	Internal

Abstract

Li'nage is a version control system that tracks code changes at the line level rather than the file level. The system implements three diff algorithms (Myers, Patient, Minimal), supports multi-protocol authentication (HTTP, SSH, OAuth), and provides commit graph visualization. This document specifies the system architecture, data models, and implementation plan.

Contents

1 Introduction

1.1 System Overview

Li'nage extends traditional version control by maintaining complete histories of individual line modifications. Where conventional systems treat files as atomic units, Li'nage records every line change with associated metadata.

1.2 Core Capabilities

- Line-level change tracking across all commits
- Three diff algorithm strategies with runtime selection
- Multi-protocol authentication for remote repositories
- Directed acyclic graph (DAG) visualization of commit history
- Snapshot-based recovery and rollback
- AI-assisted commit detection and tracking

1.3 Target Platform

- Operating System: Windows 10/11 (x64)
- Framework: .NET 7.0+
- Language: C# 11.0+
- UI: Windows Forms

2 System Architecture

2.1 Architectural Pattern

The system follows a four-layer clean architecture:

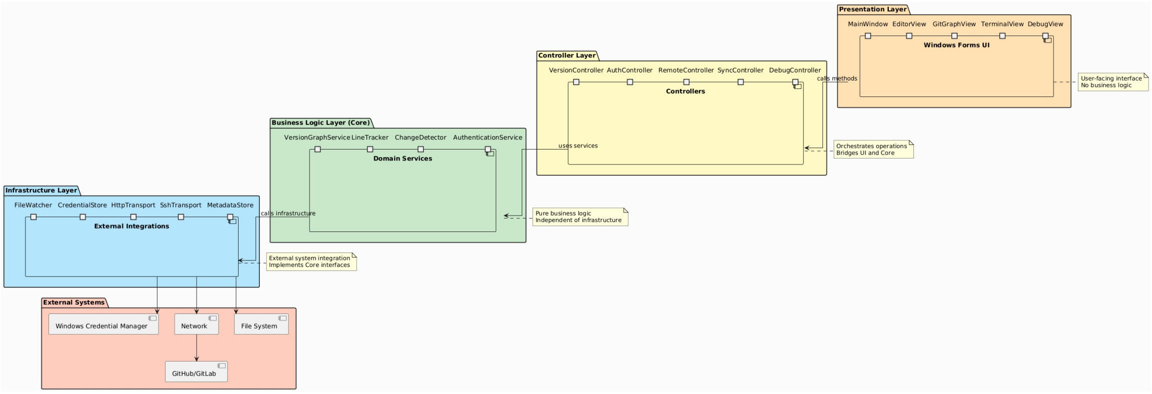


Figure 1: Four-layer architecture with clear separation of concerns

2.2 Layer Responsibilities

2.2.1 Presentation Layer

Provides user interface without business logic.

- MainWindow: Repository management and commit interface
- EditorView: Code editing with line tracking visualization
- GitGraphView: Commit DAG visualization
- TerminalView: Command-line operations
- DebugView: System diagnostics

2.2.2 Controller Layer

Orchestrates operations between UI and services.

- VersionController: Version control operations (commit, branch, merge, rebase)
- AuthController: Authentication and credential management
- RemoteController: Remote repository communication
- SyncController: Local and remote synchronization

2.2.3 Business Logic Layer

Contains domain logic independent of infrastructure.

- VersionGraphService: DAG management and branch operations
- LineTracker: Line-level change detection using diff strategies
- ChangeDetector: File change monitoring and conflict identification
- AuthenticationService: Credential lifecycle management

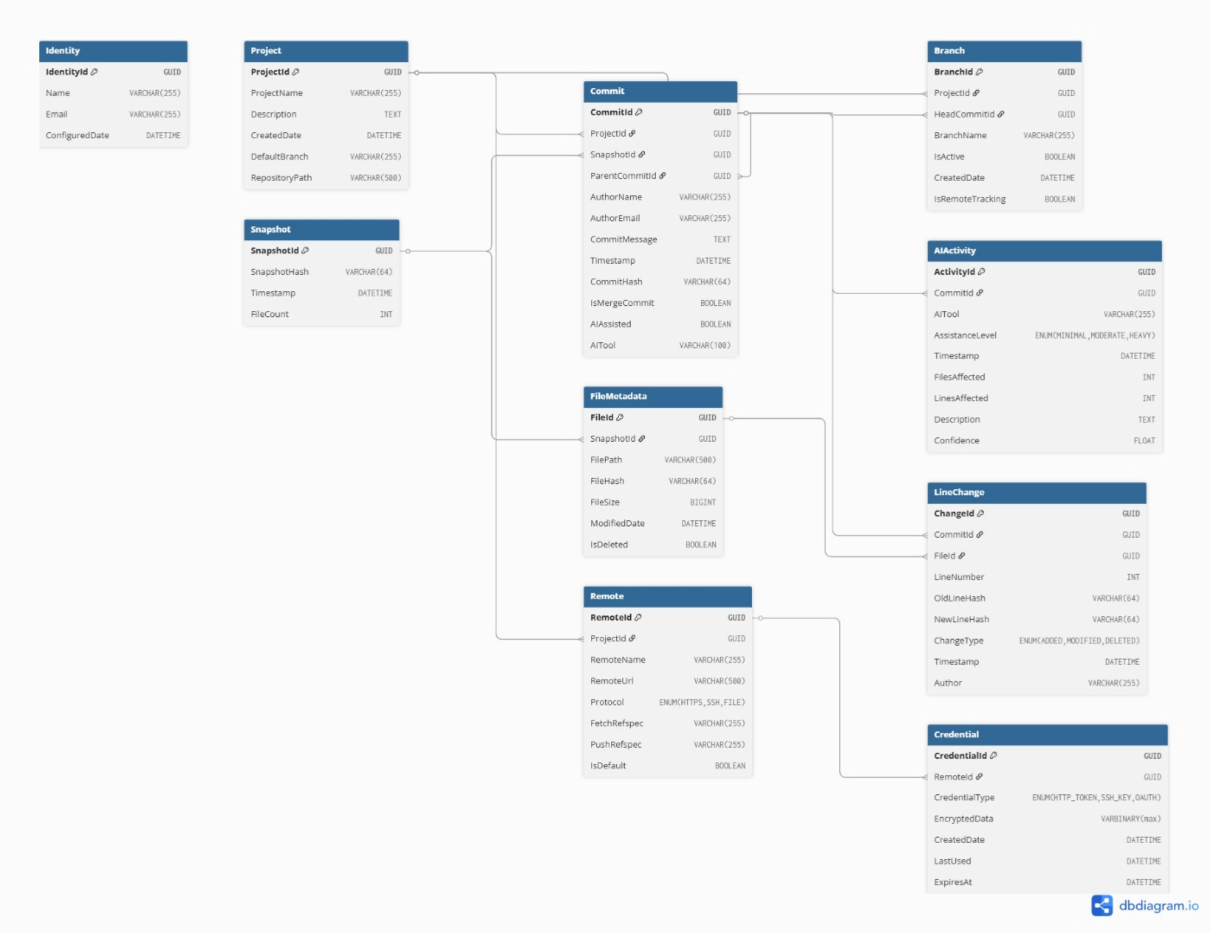
2.2.4 Infrastructure Layer

Implements external system integrations.

- FileWatcher: File system change monitoring
- CredentialStore: Windows Credential Manager integration
- HttpTransport: HTTP/HTTPS protocol implementation
- SshTransport: SSH protocol with key authentication
- MetadataStore: Persistent storage for commits and snapshots

3 Data Model

3.1 Entity-Relationship Model



Identity	
IdentityId	GUID
Name	VARCHAR(255)
Email	VARCHAR(255)
ConfiguredDate	DATETIME

Project	
ProjectId	GUID
ProjectName	VARCHAR(255)
Description	TEXT
CreatedDate	DATETIME
DefaultBranch	VARCHAR(255)
RepositoryPath	VARCHAR(500)

Snapshot	
SnapshotId	GUID
SnapshotHash	VARCHAR(64)
Timestamp	DATETIME
FileCount	INT

Commit	
CommitId	GUID
ProjectId	GUID
SnapshotId	GUID
ParentCommitId	GUID
AuthorName	VARCHAR(255)
AuthorEmail	VARCHAR(255)
CommitMessage	TEXT
Timestamp	DATETIME
CommitHash	VARCHAR(64)
IsMergeCommit	BOOLEAN
AIAssisted	BOOLEAN
AITool	VARCHAR(100)

FileMetadata	
FileId	GUID
SnapshotId	GUID
FilePath	VARCHAR(500)
FileHash	VARCHAR(64)
FileSize	BIGINT
ModifiedDate	DATETIME
IsDeleted	BOOLEAN

Remote	
RemoteId	GUID
ProjectId	GUID
RemoteName	VARCHAR(255)
RemoteUrl	VARCHAR(500)
Protocol	ENUM(HTTPS,SSH,FILE)
FetchRefspec	VARCHAR(255)
PushRefspec	VARCHAR(255)
IsDefault	BOOLEAN

Branch	
BranchId	GUID
ProjectId	GUID
HeadCommitId	GUID
BranchName	VARCHAR(255)
IsActive	BOOLEAN
CreatedDate	DATETIME
IsRemoteTracking	BOOLEAN

AIActivity	
ActivityId	GUID
CommitId	GUID
AITool	VARCHAR(255)
AssistanceLevel	ENUM(MINIMAL,MODERATE,HEAVY)
Timestamp	DATETIME
FilesAffected	INT
LinesAffected	INT
Description	TEXT
Confidence	FLOAT

LineChange	
ChangeId	GUID
CommitId	GUID
FileId	GUID
LineNumber	INT
OldLineHash	VARCHAR(64)
NewLineHash	VARCHAR(64)
ChangeType	ENUM(ADDED,MODIFIED,DELETED)
Timestamp	DATETIME
Author	VARCHAR(255)

Credential	
CredentialId	GUID
RemoteId	GUID
CredentialType	ENUM(HTTP_TOKEN,SSH_KEY,OAUTH)
EncryptedData	VARBINARY(max)
CreatedDate	DATETIME
LastUsed	DATETIME
ExpiresAt	DATETIME

Figure 2: Complete entity-relationship diagram

3.2 Core Entities

3.2.1 Project

Root container for version control.

- ProjectId: GUID (primary key)
- ProjectName: String
- Description: Text
- DefaultBranch: String
- RepositoryPath: String (file system path)
- CreatedDate: DateTime

3.2.2 Commit

Immutable version snapshot.

- CommitId: GUID (primary key)
- ProjectId: GUID (foreign key to Project)
- SnapshotId: GUID (foreign key to Snapshot)
- ParentCommitId: GUID (self-referencing foreign key, nullable)
- AuthorName: String
- AuthorEmail: String
- CommitMessage: Text
- Timestamp: DateTime
- CommitHash: String (SHA-256, unique)
- IsMergeCommit: Boolean
- AIAssisted: Boolean

3.2.3 Snapshot

Filesystem state at commit time.

- SnapshotId: GUID (primary key)
- SnapshotHash: String (SHA-256, unique)
- Timestamp: DateTime
- FileCount: Integer

3.2.4 FileMetadata

Individual file properties within a snapshot.

- FileId: GUID (primary key)
- SnapshotId: GUID (foreign key to Snapshot)
- FilePath: String (500 characters max)
- FileHash: String (SHA-256)
- FileSize: BigInt (bytes)
- ModifiedDate: DateTime
- IsDeleted: Boolean

3.2.5 LineChange

Individual line modification record.

- ChangeId: GUID (primary key)
- CommitId: GUID (foreign key to Commit)
- FileId: GUID (foreign key to FileMetadata)
- LineNumber: Integer
- OldLineHash: String (SHA-256)
- NewLineHash: String (SHA-256)
- ChangeType: Enumeration (ADDED, MODIFIED, DELETED)
- Timestamp: DateTime
- Author: String

3.2.6 Branch

Development line reference.

- BranchId: GUID (primary key)
- ProjectId: GUID (foreign key to Project)
- HeadCommitId: GUID (foreign key to Commit)
- BranchName: String
- IsActive: Boolean
- CreatedDate: DateTime
- IsRemoteTracking: Boolean

3.2.7 Remote

External repository reference.

- RemoteId: GUID (primary key)
- ProjectId: GUID (foreign key to Project)
- RemoteName: String
- RemoteUrl: String
- Protocol: Enumeration (HTTPS, SSH, FILE)
- FetchRefspec: String
- PushRefspec: String
- IsDefault: Boolean

3.2.8 Credential

Authentication material for remotes.

- CredentialId: GUID (primary key)
- RemoteId: GUID (foreign key to Remote)
- CredentialType: Enumeration (HTTP_TOKEN, SSH_KEY, OAUTH)
- EncryptedData: Binary (encrypted with AES-256)
- CreatedDate: DateTime
- LastUsed: DateTime (nullable)
- ExpiresAt: DateTime (nullable)

3.2.9 AIActivity

AI-assisted development tracking.

- ActivityId: GUID (primary key)
- CommitId: GUID (foreign key to Commit)
- AITool: String
- AssistanceLevel: Enumeration (MINIMAL, MODERATE, HEAVY)
- Timestamp: DateTime
- FilesAffected: Integer
- LinesAffected: Integer
- Description: Text
- Confidence: Float

4 Domain Model

4.1 Class Structure

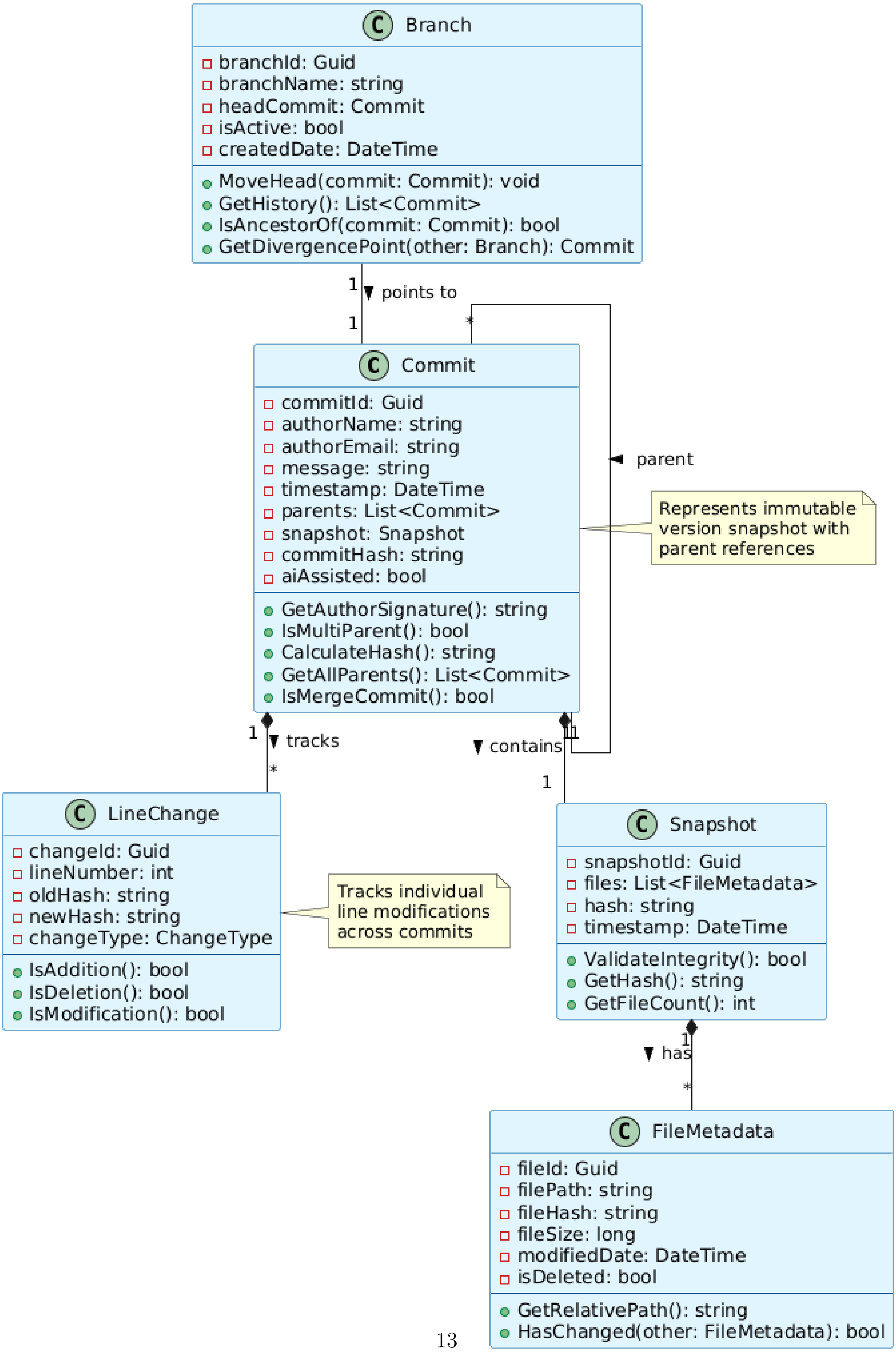


Figure 3: Core domain class diagram

4.2 Class Specifications

4.2.1 Commit

Represents an immutable version snapshot.

Properties:

- commitId: GUID
- authorName, authorEmail: String
- message: String
- timestamp: DateTime
- parents: List<Commit>
- snapshot: Snapshot
- commitHash: String (SHA-256)
- aiAssisted: Boolean

Methods:

- GetAuthorSignature(): String
- IsMultiParent(): Boolean
- CalculateHash(): String
- GetAllParents(): List<Commit>
- IsMergeCommit(): Boolean

4.2.2 Snapshot

Captures filesystem state.

Properties:

- snapshotId: GUID
- files: List<FileMetadata>
- hash: String
- timestamp: DateTime

Methods:

- ValidateIntegrity(): Boolean
- GetHash(): String
- GetFileCount(): Integer

4.2.3 FileMetadata

Tracks individual file properties.

Properties:

- fileId: GUID
- filePath: String
- fileHash: String (SHA-256)
- fileSize: Long
- modifiedDate: DateTime
- isDeleted: Boolean

4.2.4 LineChange

Records line-level modifications.

Properties:

- changeId: GUID
- lineNumber: Integer
- oldHash: String
- newHash: String
- changeType: ChangeType (enumeration)

Methods:

- IsAddition(): Boolean
- IsDeletion(): Boolean
- IsModification(): Boolean

4.2.5 Branch

Represents development line.

Properties:

- branchId: GUID
- branchName: String
- headCommit: Commit
- isActive: Boolean
- createdDate: DateTime

Methods:

- MoveHead(commit: Commit): Void
- GetHistory(): List<Commit>
- IsAncestorOf(commit: Commit): Boolean
- GetDivergencePoint(other: Branch): Commit

5 Service Layer

5.1 Service Architecture

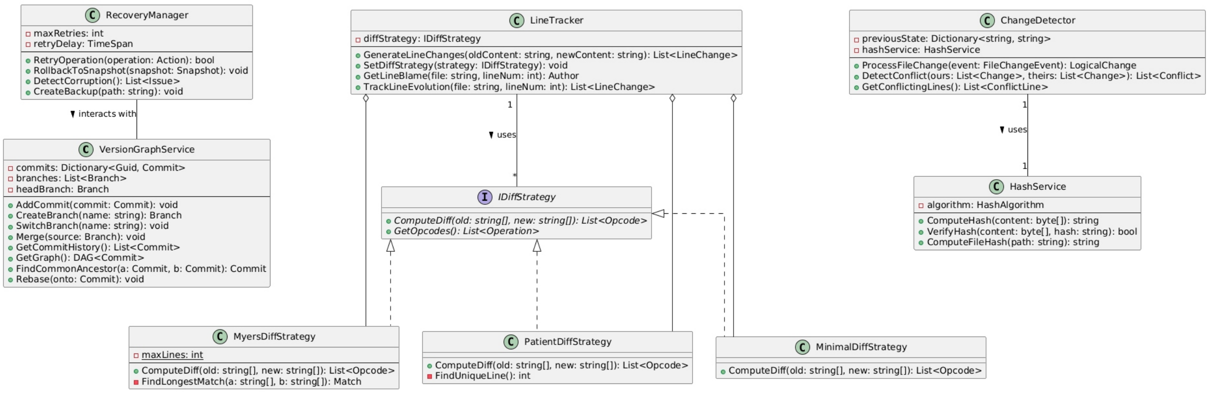

RecoveryManager
maxRetries: int
retryDelay: TimeSpan
RetryOperation(operation: Action): bool
RollbackToSnapshot(snapshot: Snapshot): void
DetectCorruption(): List<Issue>
CreateBackup(path: string): void
interacts with
VersionGraphService
commits: Dictionary<Guid, Commit>
branches: List<Branch>
headBranch: Branch
AddCommit(commit: Commit): void
CreateBranch(name: string): Branch
SwitchBranch(name: string): void
Merge(source: Branch): void
GetCommitHistory(): List<Commit>
GetGraph(): DAG<Commit>
FindCommonAncestor(a: Commit, b: Commit): Commit
Rebase(onto: Commit): void
LineTracker
diffStrategy: IDiffStrategy
GenerateLineChanges(oldContent: string, newContent: string): List<LineChange>
SetDiffStrategy(strategy: IDiffStrategy): void
GetLineBlame(file: string, lineNum: int): Author
TrackLineEvolution(file: string, lineNum: int): List<LineChange>
1
uses
*
IDiffStrategy
ComputeDiff(old: string[], new: string[]): List<Opcode>
GetOpcodes(): List<Operation>
MyersDiffStrategy
maxLines: int
ComputeDiff(old: string[], new: string[]): List<Opcode>
FindLongestMatch(a: string[], b: string[]): Match
PatientDiffStrategy
ComputeDiff(old: string[], new: string[]): List<Opcode>
FindUniqueLine(): int
MinimalDiffStrategy
ComputeDiff(old: string[], new: string[]): List<Opcode>
ChangeDetector
previousState: Dictionary<string, string>
hashService: HashService
ProcessFileChange(event: FileChangeEvent): LogicalChange
DetectConflict(ours: List<Change>, theirs: List<Change>): List<Conflict>
GetConflictingLines(): List<ConflictLine>
1
uses
1
HashService
algorithm: HashAlgorithm
ComputeHash(content: byte[]): string
VerifyHash(content: byte[], hash: string): bool
ComputeFileHash(path: string): string

Figure 4: Service layer with diff strategy pattern

5.2 VersionGraphService

Manages commit DAG and branch operations.

Responsibilities:

- Maintain commit graph structure
- Execute merge and rebase operations
- Compute common ancestors
- Provide history traversal
- Ensure DAG integrity

Key Methods:

- AddCommit(commit: Commit): Void
- CreateBranch(name: String): Branch
- SwitchBranch(name: String): Void
- Merge(source: Branch): Void
- GetCommitHistory(): List<Commit>
- GetGraph(): DAG<Commit>
- FindCommonAncestor(a: Commit, b: Commit): Commit
- Rebase(onto: Commit): Void

5.3 Diff Strategy Pattern

Three pluggable diff algorithms implement the IDiffStrategy interface.

Strategy	**Algorithm**	**Use Case**	**Complexity**
Myers	Myers O(ND)	General purpose	$O(N + D^2)$
Patient	Patient diff	Long files, unique lines	$O(N^2)$ worst
Minimal	Greedy minimal	Fast approximation	$O(N \log N)$

Table 1: Diff algorithm comparison (N = lines, D = edit distance)

5.4 LineTracker

Generates and tracks line-level changes.

Responsibilities:

- Process file changes using configured diff strategy
- Generate LineChange objects with metadata
- Track line evolution across commits
- Provide blame information per line
- Support runtime strategy switching

5.5 ChangeDetector

Monitors changes and detects conflicts.

Responsibilities:

- Monitor file system via FileWatcher
- Detect concurrent modification conflicts
- Map logical to physical changes
- Generate conflict resolution suggestions

5.6 HashService

Provides cryptographic hashing.

Responsibilities:

- Compute SHA-256 hashes
- Verify content integrity
- Support file and content-level hashing

5.7 RecoveryManager

Implements recovery and rollback.

Responsibilities:

- Retry logic with exponential backoff
- Rollback to previous snapshots
- Detect and repair corruption
- Create backups before destructive operations

6 Authentication

6.1 Authentication Architecture

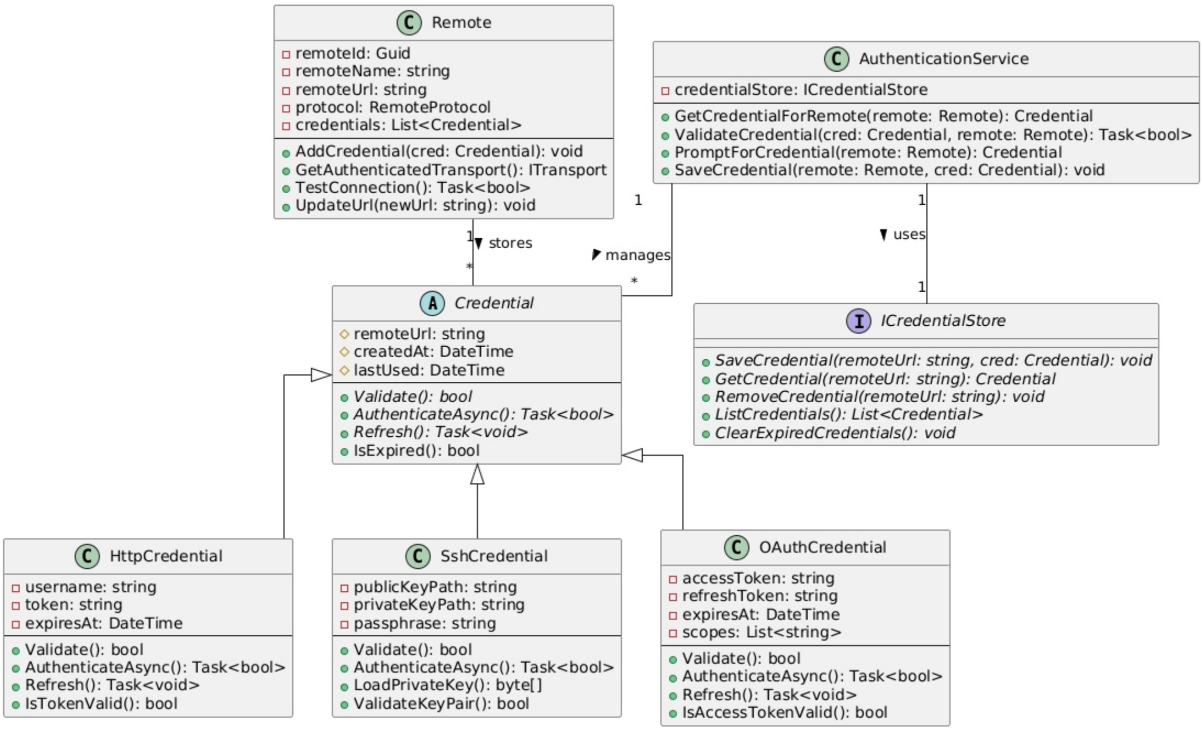

C Remote
remoteId: Guid
remoteName: string
remoteUrl: string
protocol: RemoteProtocol
credentials: List<Credential>
AddCredential(cred: Credential): void
GetAuthenticatedTransport(): ITransport
TestConnection(): Task<bool>
UpdateUrl(newUrl: string): void
C AuthenticationService
credentialStore: ICredentialStore
GetCredentialForRemote(remote: Remote): Credential
ValidateCredential(cred: Credential, remote: Remote): Task<bool>
PromptForCredential(remote: Remote): Credential
SaveCredential(remote: Remote, cred: Credential): void
1
stores
*
1
manages
*
1
uses
1
A Credential
remoteUrl: string
createdAt: DateTime
lastUsed: DateTime
Validate(): bool
AuthenticateAsync(): Task<bool>
Refresh(): Task<void>
IsExpired(): bool
I ICredentialStore
SaveCredential(remoteUrl: string, cred: Credential): void
GetCredential(remoteUrl: string): Credential
RemoveCredential(remoteUrl: string): void
ListCredentials(): List<Credential>
ClearExpiredCredentials(): void
C HttpCredential
username: string
token: string
expiresAt: DateTime
Validate(): bool
AuthenticateAsync(): Task<bool>
Refresh(): Task<void>
IsTokenValid(): bool
C SshCredential
publicKeyPath: string
privateKeyPath: string
passphrase: string
Validate(): bool
AuthenticateAsync(): Task<bool>
LoadPrivateKey(): byte[]
ValidateKeyPair(): bool
C OAuthCredential
accessToken: string
refreshToken: string
expiresAt: DateTime
scopes: List<string>
Validate(): bool
AuthenticateAsync(): Task<bool>
Refresh(): Task<void>
IsAccessTokenValid(): bool

Figure 5: Credential hierarchy and authentication flow

6.2 Credential Hierarchy

6.2.1 Credential (Abstract Base)

Properties:

- remoteUrl: String
- createdAt: DateTime
- lastUsed: DateTime

Methods:

- Validate(): Boolean
- AuthenticateAsync(): Task<Boolean>
- Refresh(): Task<Void>
- IsExpired(): Boolean

6.2.2 HttpCredential

HTTP Basic or token authentication.

Properties:

- username: String

- token: String
- expiresAt: DateTime

Methods:

- IsTokenValid(): Boolean

6.2.3 SshCredential

SSH key-based authentication.

Properties:

- publicKeyPath: String
- privateKeyPath: String
- passphrase: String

Methods:

- LoadPrivateKey(): Byte[]
- ValidateKeyPair(): Boolean

6.2.4 OAuthCredential

OAuth 2.0 flow implementation.

Properties:

- accessToken: String
- refreshToken: String
- expiresAt: DateTime
- scopes: List<String>

Methods:

- IsAccessTokenValid(): Boolean
- Refresh(): Task<Void>

6.3 AuthenticationService

Manages credential lifecycle.

Methods:

- GetCredentialForRemote(remote: Remote): Credential
- ValidateCredential(cred: Credential, remote: Remote): Task<Boolean>
- PromptForCredential(remote: Remote): Credential
- SaveCredential(remote: Remote, cred: Credential): Void

6.4 ICredentialStore Interface

Abstraction for credential persistence.

Methods:

- SaveCredential(remoteUrl: String, cred: Credential): Void
- GetCredential(remoteUrl: String): Credential
- RemoveCredential(remoteUrl: String): Void
- ListCredentials(): List<Credential>
- ClearExpiredCredentials(): Void

Implementation integrates with Windows Credential Manager using OS-level AES-256 encryption.

7 Workflows

7.1 Commit Creation

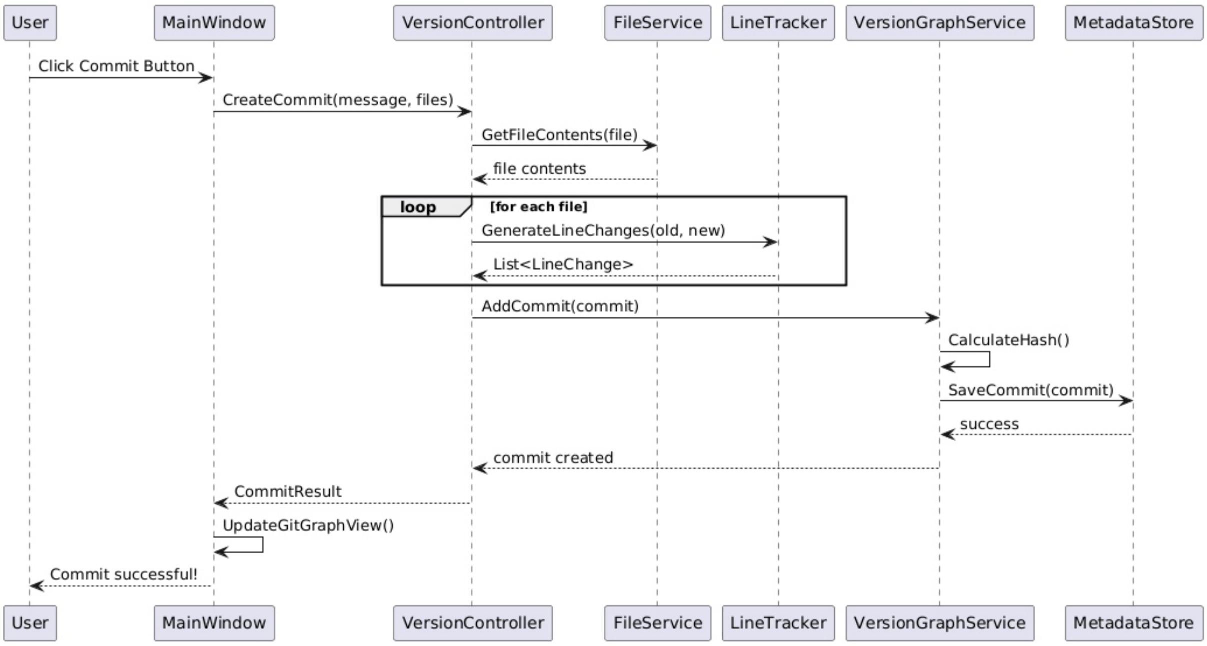


Figure 6: Commit creation sequence diagram

7.2 Commit Process

1. **Initiation:** User triggers commit via MainWindow
2. **Validation:** VersionController validates input (message, file selection)
3. **Content Retrieval:** FileService loads current file state
4. **Diff Computation:** LineTracker generates LineChange objects for each file
5. **Commit Assembly:** VersionGraphService creates commit with hash calculation
6. **Persistence:** MetadataStore serializes and writes commit data
7. **UI Update:** GitGraphView refreshes visualization
8. **Confirmation:** User receives success notification

7.3 Transaction Semantics

All commit operations follow ACID properties:

- **Atomicity:** Commit succeeds or fails as a single unit
- **Consistency:** DAG integrity is preserved
- **Isolation:** Concurrent commits are serialized
- **Durability:** Data persists to storage before confirmation

8 Component Architecture

8.1 Component Diagram

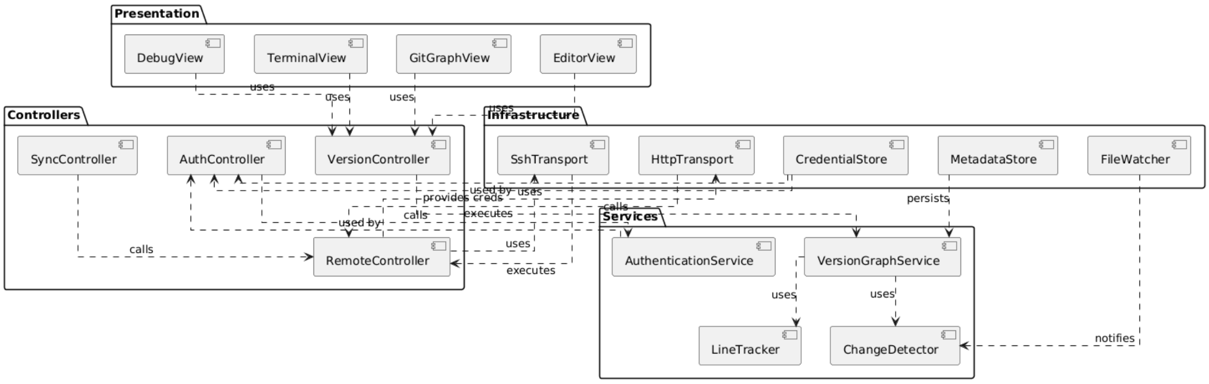


Figure 7: Component interactions and dependencies

8.2 Component Dependencies

Component	Depends On	Provides To
EditorView	VersionController	UI Layer
GitGraphView	VersionController	UI Layer
VersionController	VersionGraphService	Presentation
AuthController	AuthenticationService	Presentation, Remote
RemoteController	HttpTransport, SshTransport	SyncController
VersionGraphService	LineTracker, ChangeDetector	Controllers
LineTracker	IDiffStrategy	VersionGraphService
ChangeDetector	HashService, FileWatcher	VersionGraphService

Table 2: Key component dependencies

8.3 Design Patterns

- **Strategy:** IDiffStrategy with Myers, Patient, Minimal implementations
- **Repository:** MetadataStore, CredentialStore
- **Observer:** FileWatcher notifying ChangeDetector
- **Factory:** Credential instantiation based on type
- **Command:** Version control operations
- **Template Method:** Credential.AuthenticateAsync()

9 Deployment

9.1 Deployment Topology

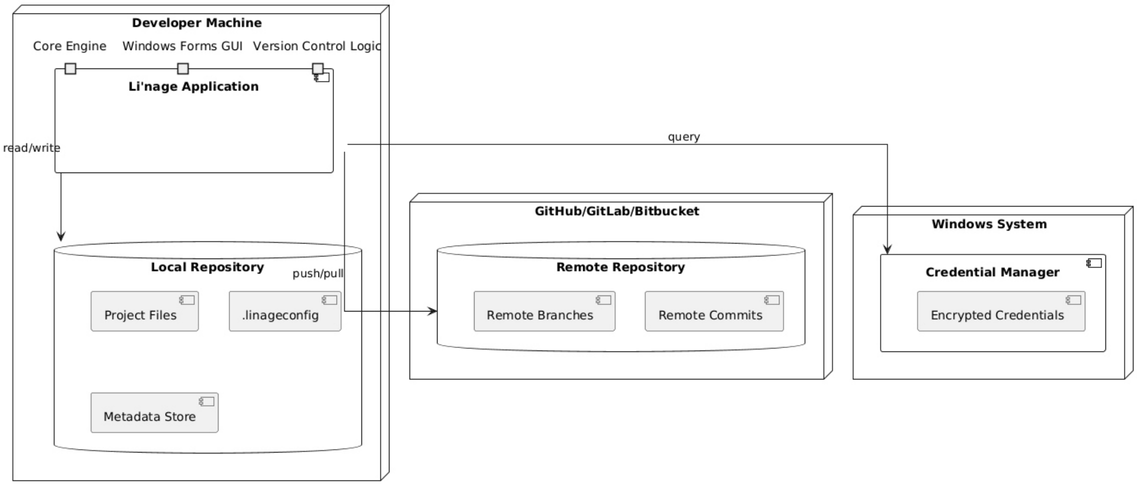


Figure 8: Deployment architecture

9.2 Components

9.2.1 Developer Machine

Local execution environment.

- **Li'nage Application:** Core engine and Windows Forms UI
- **Local Repository:** Project files, .linageconfig, metadata store
- **Metadata Store:** SQLite database with commit history, snapshots, line changes

9.2.2 Remote Repository

External Git hosting services (GitHub, GitLab, Bitbucket).

- Remote branches
- Remote commits
- Release tags

9.2.3 Windows System Integration

Operating system services.

- **Credential Manager:** Encrypted credential storage
- **File System:** NTFS with change notifications
- **Network Stack:** HTTP/HTTPS and SSH communication

9.3 Data Flow

- **read/write:** Application ↔ Local Repository
- **query:** Application → Windows Credential Manager
- **push/pull:** Application ↔ Remote Repository
- **monitor:** File System → FileWatcher → ChangeDetector

10 Performance and Security

10.1 Performance Characteristics

10.1.1 Diff Algorithm Performance

Algorithm	Time	Space	Best Case
Myers	$O(N + D^2)$	$O(N + D)$	$O(N)$
Patient	$O(N^2)$	$O(N)$	$O(N)$
Minimal	$O(N \log N)$	$O(N)$	$O(N)$

Table 3: Complexity analysis (N = lines, D = edit distance)

10.1.2 Performance Targets

Operation	Target
Commit creation (typical)	< 500 ms
Line tracking per file	< 10 ms
Diff (1000 lines)	< 100 ms
Branch switch	< 200 ms

Table 4: Performance targets

10.1.3 Caching Strategy

- Commit graph: In-memory DAG cache
- File hashes: Per-snapshot storage
- Credentials: Active session cache with expiration
- Line changes: Recent diff cache per file

10.2 Security

10.2.1 Credential Storage

- Windows Credential Manager integration
- AES-256 encryption for all credentials
- Private SSH keys encrypted with user-specific keys
- Memory cleared after authentication
- No plaintext storage in configuration files

10.2.2 Authentication Flow

1. Retrieve credential from secure storage
2. Validate and refresh if needed
3. Authenticate with remote
4. Log authentication event
5. Apply rate limiting on failures

10.2.3 Data Protection

- **File Integrity:** SHA-256 verification
- **Commit Immutability:** Cryptographic hashes prevent tampering
- **Snapshot Consistency:** Integrity checks on create/retrieve
- **Rollback Safety:** Automatic backups before destructive operations

11 Implementation Plan

11.1 Development Phases

11.1.1 Phase 1: Foundation (Weeks 1-5)

- Core domain classes (Commit, Snapshot, FileMetadata, LineChange, Branch)
- Database schema and migrations
- MetadataStore with SQLite backend
- DAG data structure
- Unit tests (>90% coverage)

11.1.2 Phase 2: Core Features (Weeks 6-10)

- IDiffStrategy interface and Myers implementation
- LineTracker service
- ChangeDetector service
- FileWatcher integration
- End-to-end workflow tests

11.1.3 Phase 3: Authentication (Weeks 11-13)

- Credential hierarchy (HTTP, SSH, OAuth)
- Windows Credential Manager integration
- AuthenticationService
- HttpTransport and SshTransport
- Remote push/pull operations

11.1.4 Phase 4: UI (Weeks 14-16)

- MainWindow and commit interface
- GitGraphView with DAG rendering
- EditorView with line tracking
- TerminalView and DebugView
- UI/UX testing

11.1.5 Phase 5: Advanced Features (Weeks 17-19)

- Patient and Minimal diff algorithms
- Conflict resolution UI
- AI activity tracking
- RecoveryManager
- Performance optimization

11.1.6 Phase 6: Finalization (Week 20)

- Documentation (API, user guide)
- Security audit
- Performance tuning
- Installer creation
- Beta testing

11.2 Milestones

Milestone	**Week**	**Deliverable**
M1: Data Model	5	Domain classes with tests
M2: Core VCS	10	Commit, branch, merge
M3: Authentication	13	Multi-credential support
M4: UI Complete	16	Full interface
M5: Feature Complete	19	All features
M6: Production	20	Release candidate

Table 5: Project milestones

12 Risk Analysis

12.1 Technical Risks

Risk	Impact	Mitigation
Large file performance	Slow diff computation	Streaming algorithms, file size limits, chunked processing
Merge conflicts	Data loss risk	3-way merge visualization, automatic conflict detection
Data corruption	Lost history	Checksums, automatic backups, corruption detection
Credential leaks	Security breach	OS-level encryption, audit logging
Network failures	Sync interruption	Retry with backoff, transaction rollback
Memory exhaustion	Application crash	Streaming operations, memory profiling
DAG corruption	Broken history	Integrity validation, repair utilities
Concurrent access	Data inconsistency	File locking, atomic operations

12.2 Project Risks

- **Schedule:** 20-week timeline requires strict milestone adherence
  - Mitigation: Bi-weekly reviews, scope adjustment
- **Resources:** Team availability and expertise requirements
  - Mitigation: Cross-training, documentation
- **Integration:** Windows platform dependencies
  - Mitigation: Early integration testing, abstraction layers
- **Adoption:** Learning curve for users
  - Mitigation: Documentation, tutorials, migration guides

13 Quality Attributes

13.1 Reliability

- Target uptime: 99.9% for local operations
- Recovery time: $<$ 1 minute
- Full backup on every commit
- Graceful error handling

13.2 Performance

- Commit creation: $<$ 500 ms (up to 10,000 lines)
- Line tracking: $<$ 10 ms per file
- Diff computation: $<$ 100 ms (1,000 lines)
- Branch operations: $<$ 200 ms
- Graph rendering: $<$ 2 seconds (10,000 commits)

13.3 Scalability

- 100,000+ commits per repository
- Unlimited files (performance degrades gracefully)
- Millions of line change records with indexing
- Hundreds of concurrent branches
- Multi-gigabyte repositories

13.4 Maintainability

- $>$ 85% unit test coverage
- 100% public API documentation
- Clean architecture with defined interfaces
- Static analysis compliance

13.5 Security

- OS-level credential encryption
- Cryptographic hashing for all content
- File system permission compliance
- Complete authentication audit trail

14 Technology Stack

14.1 Development Platform

- Language: C# 11.0+
- Framework: .NET 7.0 or .NET 8.0
- IDE: Visual Studio 2022 or JetBrains Rider
- UI: Windows Forms
- Target: Windows 10/11 (x64)

14.2 Data Storage

- Database: SQLite 3.x
- ORM: Entity Framework Core 7.0+
- Serialization: System.Text.Json
- File System: NTFS with change notifications

14.3 Security and Cryptography

- Hashing: SHA-256 (System.Security.Cryptography)
- Encryption: AES-256
- SSH: SSH.NET library
- HTTPS: System.Net.Http.HttpClient
- Credential Manager: Windows API

14.4 Testing

- Unit Testing: xUnit or NUnit
- Mocking: Moq
- Coverage: Coverlet + ReportGenerator
- Integration: xUnit with TestContainers
- Performance: BenchmarkDotNet

14.5 Build and Deployment

- Build: MSBuild (SDK-style projects)
- CI/CD: GitHub Actions or Azure DevOps
- Packages: NuGet
- Installer: WiX Toolset or Advanced Installer

14.6 Third-Party Libraries

Library	Purpose	License
SSH.NET	SSH protocol	MIT
LibGit2Sharp	Git reference	MIT
Serilog	Logging	Apache 2.0
Autofac	DI container	MIT

Table 7: Third-party dependencies

15 Appendix A: Glossary

Commit Immutable snapshot of code changes with metadata.

Snapshot Complete filesystem state at a specific point in time.

LineChange Record of an individual line modification.

Diff Computation of differences between file versions.

Branch Named reference to a commit sequence.

Merge Combining changes from multiple branches.

Rebase Re-applying commits on a new base.

Remote External repository reference.

Credential Authentication material for remotes.

DAG Directed Acyclic Graph of commits.

Conflict Incompatible concurrent changes.

Rollback Revert to previous state.

Hash Cryptographic fingerprint (SHA-256).

HEAD Current commit reference.

Ancestor Preceding commit in history.

Opcode Diff operation descriptor.

16 Appendix B: References

1. Myers, E. W. (1986). An O(ND) difference algorithm and its variations. *Algorithmica*, 1(1), 251-266.

2. Hunt, J. W., & Szymanski, T. G. (1977). A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5), 350-353.

3. Microsoft. (2024). Windows Credential Manager API Reference. Retrieved from https://docs.microsoft.co

4. Git Project. (2024). Git Internals. Pro Git Book, Chapter 10. Retrieved from https://git-scm.com/book

5. Hardt, D. (Ed.). (2012). The OAuth 2.0 Authorization Framework. RFC 6749.

6. Martin, R. C. (2017). *Clean Architecture*. Prentice Hall.

7. Gamma, E., et al. (1994). *Design Patterns*. Addison-Wesley.

17 Appendix C: Acronyms

Acronym	Definition
API	Application Programming Interface
DAG	Directed Acyclic Graph
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
SSH	Secure Shell
OAuth	Open Authorization
GUID	Globally Unique Identifier
SHA	Secure Hash Algorithm
AES	Advanced Encryption Standard
VCS	Version Control System
UI	User Interface
IDE	Integrated Development Environment
ORM	Object-Relational Mapping
JSON	JavaScript Object Notation
NTFS	New Technology File System
CI/CD	Continuous Integration/Deployment

Table 8: Acronyms and abbreviations