COSC 499 Capstone Software Engineering Project

Mining Digital Work Artifacts – Team17

2025/26

Week 4 Project Proposal

Parsa Aminian
Raunak Khanna
Yuxuan Sun
Shuyu Yan
Michelle Zhou

1. Project Scope and Usage Scenario

This project delivers a local-first desktop tool designed to give users a clear account of their work activity. After installation, users select the folders or GitHub repositories they wish to analyze. The system then systematically iterates through a wide range of artifact types which includes source code, documents, downloads, design files, images, or even videos. This information processes the results into a unified dashboard. The dashboard provides clear metrics on what you worked on, where you put in the effort the most, and how much progress you truly made, expressed in percentage progress indicators familiar from platforms such as Workday showing the user how close they are to finishing their degree. Multiple user groups benefit in distinct ways: **students** can export structured reports (PDF/CSV) to strengthen their CVs and portfolios, **early career software engineers** can present verified evidence of productivity for appraisals, and **project managers** or **human resources professionals** can review artifact-based progress rather than relying on self-reported claims, which are often biased or prone to error.

2. Proposed Solution

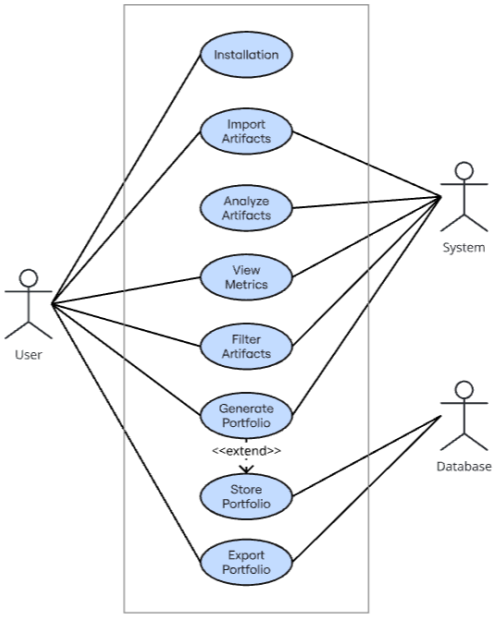
We propose a local-first desktop application that turns scattered digital artifacts into structured, verifiable insights while ensuring all processing remains secure on the user's own device. Once folders or GitHub repositories are selected, the system scans deeply across code, documents, images, videos, downloads, and design files. It extracts key details such as names, types, sizes, and dates, removes duplicates through hashing, and organises results into meaningful categories. The output is presented in a single

dashboard that shows timelines of activity, balances of artifact types, and percentage-based progress indicators inspired by Workday's degree tracker. Every metric is fully **traceable**: users can drill down from any chart to the exact files or commits that generated it, much like ChatGPT's deep research mode which not only provides an answer but also reveals its sources. Providing metrics helps us in avoiding the "black box" problem common in existing tools.

Our approach is distinctive because it combines privacy, breadth, and engagement. Unlike teams that may analyse only code or rely on cloud services (for example: Github), our solution unifies diverse artifact types, runs entirely on-device, and includes safeguards such as aggregation and light noise to allow safe sharing of trends without exposing sensitive details. It also supports **reflection and growth** by highlighting major **patterns** and **milestones** for the user, enabling comparisons across projects or time periods, and providing **goal-tracking** features with **percentage-completion** feedback. To motivate users, **lightweight badges** mark key milestones (similar to Khanacademy), adding a layer of **gamification** without overwhelming the professional focus of the tool. Robust engineering ensures reliability through resumable scans, integrity checks, CPU throttling, and cross-platform support, while security is reinforced with an encrypted database, a token-protected local API, and export audit logs. Looking ahead, the modular design allows seamless extensions for résumé or LinkedIn exports, integration with cloud storage or Jupyter notebooks, and collaborative team dashboards. In short, this solution delivers a practical, privacy-first, and verifiable analytics platform that not only

showcases digital work but also encourages reflection, motivation, and achievement, enabling users to present their progress with clarity and confidence in their work.

3. Use Cases



Use Case 1: Installation and Application Set Up

- **Primary Actor:** User
- **Description:** The process of installing the desktop application and configuring it.
- **Precondition:** User has downloaded the installer.
- **Postcondition:** Application is installed locally and ready for use.
- **Main Scenario:**
    1. User runs installer
    2. System installs necessary components
    3. User launches application
- **Extensions:**

1. Installation fails due to insufficient permissions granted. System notifies the user and prompts for reinstallation.

Use Case 2: Import Artifacts

- **Primary Actor:** User
- **Description:** The process of selecting artifacts to be scanned/mined.
- **Precondition:** Application is installed and running.
- **Postcondition:** Selected resources are registered and able to be processed.
- **Main Scenario:**
    1. User selects “Add Resource”.
    2. User browse local folders for files and repositories to be considered.
    3. System confirms selection and adds resources to a list of sources.
- **Extensions:**
    1. If a repository cannot be accessed, system notifies user with a pop up.

Use Case 3: Analyze Artifacts

- **Primary Actor:** System
- **Description:** The process of analyzing and extracting metadata.
- **Precondition:** User has added at least one source.
- **Postcondition:** Artifacts are scanned and organized into categories.
- **Main Scenario:**
    1. User selects “Scan Sources”.
    2. System identifies files (documents, programming code, various media).

3. System extracts metadata.
4. System removes duplicates.
5. Data are organized and stored locally

- **Extensions:**
    1. If a file is corrupted, system skips ahead and logs an error.

Use Case 4: View Metrics

- **Primary Actor:** User
- **Description:** The process of viewing productivity insights/completion progress for projects.
- **Precondition:** Artifacts are imported and analyzed.
- **Postcondition:** Metrics are displayed.
- **Main Scenario:**
    1. User navigates to metric section.
    2. User selects specific project.
    3. System displays progress charts/graphs.
- **Extensions:**
    1. If no projects are available, system prompts user to import artifacts.

Use Case 5: Generate Portfolio

- **Primary Actor:** User
- **Description:** The process of creating a report highlighting source artifacts.
- **Precondition:** Resources are analyzed and organized.

- **Postcondition:** Viewable portfolio can be saved or exported/downloaded.
- **Main Scenario:**
    1. User clicks “Generate”.
    2. User reviews selected data.
    3. System compiles data.
- **Extensions:**
    1. If insufficient artifacts are found, system prompts user to upload additional sources.

Use Case 6: Store Portfolio

- **Primary Actor:** User
- **Description:** The process of storing exportable portfolios in the system database.
- **Precondition:** Portfolio was created without error.
- **Postcondition:** User can export previous portfolios any time.
- **Main Scenario:**
    1. User selects “Save”.
    2. System stores portfolio within database
    3. Portfolios are organized and categorized.
- **Extensions:**
    1. If the user attempts to save prior to

Use Case 7: Search and Filter Artifacts

- **Primary Actor:** User

- **Description:** The process of finding specific artifacts.
- **Precondition:** Artifacts are imported and organized.
- **Postcondition:** Relevant artifacts are displayed.
- **Main Scenario:**
    1. User enters key words or applies filters.
    2. System searches indexed data.
    3. System returns results.
- **Extensions:**
    1. If no results are returned, system notifies user.

Use Case 8: Export/Share Portfolio

- **Primary Actor:** User
- **Description:** The process of downloading or sharing portfolios.
- **Precondition:** Portfolio was created without
- **Postcondition:** Portfolio is available in desired format.
- **Main Scenario:**
    1. User selects “Export”.
    2. User chooses output format.
    3. System generates exports.
- **Extensions:**
    1. If file size is too large, system splits the export.

4. Requirements, Testing, Requirement Verification

Tech Stack

Application Form

- Electron desktop app, local-first, offline by default.
- Process model: Renderer (UI) + Main Process (app)

Renderer

- React + TypeScript (or Vue)
- Charts: Recharts or ECharts
- UI kit: Tailwind

Main

- Node.js + TypeScript

Database

- SQLite

CI/CD Deployment

- GitHub Actions

Test Framework

- Unit: Vitest
- Component: React Testing Library

Requirement	**Description**	**Test Cases**	**Who**	**H/M/E**
Artifact Scanning	System should scan a selected directory and identify files of supported types (code, docs, images).	-Positive:Select a directory and confirm files detected. - Negative: point to empty folder, expect no results. - Error handling: invalid path.	Parsa	Medium
Metadata Extraction	Extract metadata (file path, size, type, timestamp) for each artifact.	- Positive: Verify metadata correctness for known files.	Parsa	Medium

		- Negative: very large file. - Negative: corrupted file.		
Database Insertion	Store extracted artifact metadata in database schema	-Positive: Insert known files, query DB, verify records. - Negative: duplicate insertion avoided. - DB schema validation.	Yuxuan	Hard
Search & Filter	User can search artifacts by type, date, or keyword.	- Positive: Search code files only. - Filter by time range. - Edge: no	Shuyu	Medium

		matches returned.		
Privacy / Opt-Out	Allow users to exclude directories or delete scanned artifacts.	- Positive: Mark folder as ignored → confirm no scan. - Delete record → verify DB removal. - Edge: delete non-existent record.	Yuxuan	Hard
Performance & Scalability	Handle large datasets efficiently	-Scan 10,000 files within 10s - Stress test with 1GB+ media files - Negative: Scan a folder without read	Raunak	Hard

		permission → error shown		
Data Integrity	Ensure scanned metadata is not lost during crashes	- Simulate crash mid-scan, restart → data persists - Positive: Normal shutdown and restart → scanned data still available - Export DB → verify contents	Michelle	Hard
Export & Reporting	Allow exporting artifacts/metrics as CSV/JSON	- Positive: Export valid dataset → fields complete - Export empty dataset → valid empty file	Raunak	Medium

Cross-Platform Compatibility	System runs on Windows/Mac/ Linux	- Positive: Test path parsing on Win/Linux - Verify UI launches on Mac - Negative: Unsupported OS → show error message	Parsa	Medium
Error Logging	Record errors for debugging & user support	- Positive: Normal operation → no error logs created -Invalid path generates error log - DB failure logged	Michelle	Medium

		correctly		
User Authentication	Secure login to system before accessing artifacts	- Positive: Valid login → access granted - Invalid password → denied - Session timeout → require re-login	Yuxuan	Medium
Generate Productivity Metrics	Compute metrics (e.g., #artifacts, time trends, complexity indicators)	- Positive: Generate metrics for 10 artifacts - Negative: empty dataset → return “no metrics” - Performance: large dataset within 5s	Michelle	Hard

Store Portfolio	Save portfolio with artifacts + metrics to DB or file	- Positive: Save portfolio and reload → data consistent - Negative: attempt to save empty portfolio → system prevents save / warning shown - Error: DB disconnected → error message	Shuyu	Medium
Generate Highlights	Auto-generate key project highlights for portfolio	-Positive: Select top 3 largest/most recent artifacts - Negative: no artifacts in portfolio →	Shuyu	Hard

		system returns "no highlights available" - Verify highlight text generated correctly		
Export & Share Portfolio	Allow exporting portfolio as CSV/JSON or sharing	- Positive: Export CSV → fields correct - Empty portfolio → valid empty file - Share link opens correctly	Raunak	Medium

5. Proposed Workload Distribution

Ownership areas Workload

1. **Ingestion and preprocessing pipeline (core coding)**

   Design and implement file discovery, type detection, hashing, metadata extraction, and queuing.

   Success metric: end to end ingest speed at least two hundred files per minute on a typical laptop with zero data loss and reproducible hashes.

2. **Metadata schema and storage (coding + indexes)**

   Define the canonical schema for artifacts and implement persistence with indexes that support fast queries.

   Success metric: typical lookups under one hundred milliseconds for common filters such as type, owner, date, and project tag.

3. **Privacy and security guardrails (coding)**

   Local only processing mode, selective redaction, permission checks, and secure deletion workflow.

   Success metric: redaction applied to all designated fields verified by automated tests and a manual spot check script.

4. **Analytics and insights (coding)**

   Implement artifact level features such as recency, velocity, uniqueness score, and basic clustering of similar items.

Success metric: top five insights generated correctly on a seeded demo workspace and validated against ground truth.

5. **CI, testing, and developer tooling (coding and git)**

   Set up unit and integration tests, data fixtures, and a smoke test runner.

   Success metric: ninety percent pipeline code coverage, green build on main, and a one command local setup.

6. **Architecture documentation and demos (presentation and doc)**

   Own the system architecture diagram, data flow diagrams, and a short demo script.

   Success metric: teammates can explain the pipeline in two minutes using Parsa's diagram and script.

Team roles

- **Parsa Aminian:** Responsible for ingestation and preprocessing pipeline, documentation and demos, artifact scanning, meta data extraction, making sure the software is cross platform ready.

- **Michelle Zhou:** Responsible for delivering reliable and insightful analysis, ensuring data integrity, implementing error logging, and generating accurate productivity metrics.

- **Shuyu Yan:** Responsible for portfolio persistence, artifact search/filtering mechanisms, and automated highlight generation.

- **Raunak Khanna:** Responsible for data processing, front end development and back end development, helping write tests for various methods.

- **Yuxuan Sun:** Responsible for database schema design, handling runtime data read/write and storage, ensuring performance optimization, participating in front-end and back-end design, and involved in system testing.

Sprint plan and deliverables

Sprint 1 discovery and scaffolding

Deliverables: ingestion spikes for two file families code and pdf, draft metadata schema, repo structure, minimal run script.

Collab: align with team on tech stack and success metrics.

Sprint 2 ingestion MVP

Deliverables: directory crawl, file type sniffing, hashing, basic metadata extraction, local storage, ten unit tests, ingest CLI.

Collab: API contract with search or UI teammate.

Sprint 3 indexing and query

Deliverables: indexed storage, filter and sort, pagination, query benchmarks, profile report,

fifteen tests.

Collab: hand off a simple service endpoint for UI integration.

Sprint 4 privacy pass

Deliverables: redaction rules, local only mode switch, secure delete workflow, audit log, threat model checklist, tests.

Collab: quick UX for redaction review with the UI owner.

Sprint 5 analytics v1

Deliverables: recency and velocity features, similarity fingerprints, top five insights per user, validation notebook.

Collab: agree on insight copy and cards with the team.

Sprint 6 performance and scale

Deliverables: parallel ingestion, batched writes, backpressure, memory caps, large catalog benchmarks, tuning notes.

Collab: coordinate with teammate running end to end scenarios.

Sprint 7 polish and docs

Deliverables: refined architecture diagram, runbook, contributor guide, troubleshooting, demo script and sample dataset.

Collab: dry run the final presentation with the team.

Sprint 8 release and handoff

Deliverables: tagged release, reproducible demo environment, metrics snapshot, postmortem and backlog for future work.

Interfaces and collaboration points

1. **Storage interface**

   Provide a simple service with create read update delete for artifacts and a search endpoint with filters.

   Consumers: search module and UI.

2. **Event hooks**

   Emit events on artifact discovered, artifact indexed, artifact redacted, artifact deleted.

   Consumers: analytics module and notification bar in UI.

3. **Redaction policy file**

   YAML or JSON file checked into the repo that declares fields to mask or drop.

   Consumers: ingestion and UI redaction review.

Risks and mitigations

1. **Heterogeneous file types**

   Mitigation: plug in extractor pattern with graceful fallback to raw metadata.

2. **Large catalogs**

   Mitigation: streaming processing, batching, and clear memory ceilings with backpressure.

3. **Privacy gaps**

   Mitigation: default to redaction first and require explicit allow for sensitive fields, plus unit tests on real looking fixtures.