Features Proposal for Project Option #10

Team Number: 10
Team Members: Bao Pham - 63726483, Toby Nguyen - 53802864, Abdalla Abbas - 95708517, Adam Badry - 92262062, Brett Hardy 59854307, Samuel Alexander - 27322569

— § —

1 *Project Scope and Usage Scenario*

The basic usage scenario involves three main user groups:

- **End Users** who interact with the UI Service (through a graphical or command-line interface) to configure scans, track progress, and view dashboards, timelines, etc.;
- **Analysis Users,** who rely on the Analysis Service to process scanned data, generate metrics, and store results for comparisons across different scans;
- **Administrators or Data Managers,** who use the File Export Service to manage outputs, customize formats (CSV, JSON, PDF, Markdown), and maintain history or logs for re-use.
- The workflow allows users to scan files, analyze them, and export results in formats suitable for reporting, monitoring, or further processing.

1.1 Project Scope

The ***Mining Digital Work Artifacts*** application is designed to assist individuals in organizing, analyzing, and extracting insights from their accumulated digital work artifacts. These artifacts include source code repositories, documents, media files, archives, and related metadata. The system will enable users to have the following features:

- **Artifact Collection:**
  - Scan selected storage devices and directories
  - Customization and filtering of allowed files: types, size, extensions, hierarchy-depth, and grouping.
- **Processing and Analysis:**
  - Extract metadata (timestamps, authorship, size, format, etc.).
  - Analyze content to detect productivity patterns, activities, interests, and the field of discipline.
- **Insights and Visualization:**
  - Generate analytical summaries (e.g., project activity timelines, most-used file types).
  - Provide dashboards and visual reports for personal or organizational review.
- **Report Generation:**
  - Export findings into structured formats such as CSV, PDF, JSON, and Markdown.

1.2 Usage Scenario

Developed to aid and help individuals regarding SWE, below are the most common audiences and their possible usage of the application:

- **Developers:** Scan local Git repositories to track project contributions and export a portfolio-ready summary of commits, languages, and productivity trends.
- **Researchers:** Organize accumulated datasets and papers, analyzing output trends over a research timeline.
- **Students:** Aggregate coursework artifacts across semesters to showcase skills in resumes or project portfolios.
- **Freelancers:** Generate exportable client-ready reports summarizing delivered code, documents, and design work.
- **Organizations:** Measure collective productivity by scanning shared repositories and producing analytical overviews of team contributions.

— § —

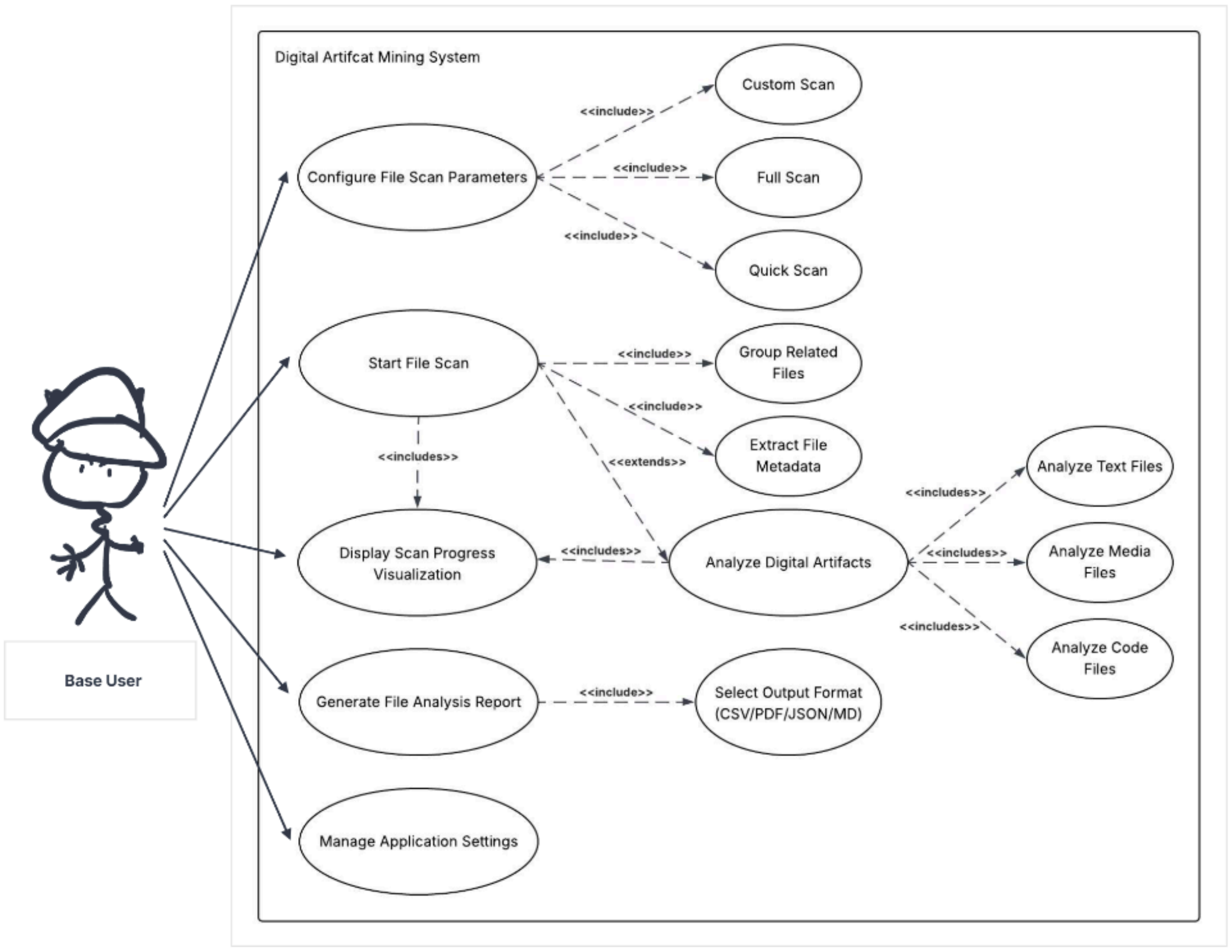
2 *Proposed Solution*

Our solution to the Mining Digital Work Artifacts consists of a locally run application that will scan, compile, and output project, skill, and statistical insights to the user. We intend to use the programming language Rust, which affords us a high-performance, memory-safe, and efficient language to build our application upon. With Rust, we plan to use the egui library to form our front-end GUI for easy user access, as well as implement the Cargo testing framework to ensure functionality. Our system will also run locally and be available with no internet connection, lowering the requirements to accommodate and offer the service to all users in almost all circumstances.

One of the major advantages our system will hold over other groups is specifically the security of the data and files on the user's computer that are being scanned. As this program is locally run, there is no chance that the data will be leaked or spread online. The program does not directly interface with LLMs or even store the data in its entirety; the risk of a data breach is minimal, if not zero. This allows us to confidently provide analytics to the user in a way that is both ethical and secure, as the only data exported will be contained within the dashboard on the GUI and a user-specified output file (CSV, PDF, JSON, Markdown, etc.).

— § —

3 *Use Cases*



Use Case 1: Configure File Scan Parameters

- Primary Actor: Base User
- Description: The configuration parameters for the method of search, and which files are considered
- Precondition: User has launched and loaded the application
- Postcondition: File search is prepared with a specific set of search conditions
- Main Scenario:
    1. User clicks on file scan configuration
    2. User configures parameters for the desired method of file search
    3. User saves configuration parameters
    4. The system sets the current parameters to what has been saved
- Extensions:

        1. If configuration parameters have been saved, update file search settings or request configuration parameters on search; otherwise, an error is displayed.

Use Case 2: File Scan

- Primary Actor: Base User
- Description: The process of scanning files
- Pre-condition: The scan parameters have been configured
- Post-condition: Files are scanned and sorted to be processed
- Main Scenario:
    1. User clicks the “start scan” button to begin the scanning
    2. Files are separated or excluded based on the configuration parameters
    3. Metadata is extracted from the files
- Extensions:
    1. If the scan is cancelled or an error occurs, the display visualization will update with a visualization indicating the scan has halted or is no longer in progress

Use Case 3: Display Scan Progress Visualization

- Primary Actor: Base User
- Description: The visual representation of the progress of the current file scan
- Pre-condition: A file scan has been started and is in progress
- Post-condition: File scan gets cancelled or reaches completion
- Main Scenario:
    1. User starts a file scan, initiating the scan progress visualization
    2. The visualization parameters are set according to the parameters of the file scan
    3. As files are scanned, the visualization updates to display the progress of the scan
    4. The scan finishes, and the visualization updates to represent a completed file scan
- Extensions:
    1. If the scan is cancelled or an error occurs, the display visualization will update with a visualization indicating the scan has halted or is no longer in progress
    2. After the scan completes, the file analysis starts with the metadata from the files

Use Case 4: Generate File Analysis Report

- Primary Actor: Base User
- Description: The process of exporting the analysis of files to a single summarized file
- Pre-condition: A file scan has been completed
- Post-condition: A (CSV/PDF/JSON/MD) file is created and exported to the user’s file system
- Main Scenario:
    1. User clicks the “Export Report” button
    2. User selects their desired file type for the export
    3. Using the file type as input, a file with data from the analysis is generated
    4. The file is saved on the local machine

- Extensions: Can be specialized or tailored for different disciplines, for example:
  - Myriad types of graphs, necessary datamines (t-values, distributions, frequency, etc.) for Data Scientists
  - Frequency, sizing, contribution size, and interest length for Employers
  - Difficulty, connections, variety of projects taken in, and colloquialness of contributions for Software Engineer Team Recruiter

Use Case 4: Manage Application Settings

- Primary Actor: Base User
- Description: The general application settings
- Pre-condition: User has launched the application
- Post-condition: Application settings are adjusted to match the user's selection
- Main Scenario:
  1. The user clicks on the "Application Settings" button
  2. The user changes or modifies some settings
  3. The user saves the changes and exits the settings menu
  4. The application loads the new settings
- Extensions:
  1. Customizable themes for users - aesthetic options for the users
  2. Selective displaying for users - being able to choose what features, windows, and insights to display on the GUI

— § —

4 Requirements, Testing, Requirement Verification

Our Technology stack will be Rust with a library known as egui. Rust has a built-in testing framework with Cargo. This allows us to have deeply integrated tests to guarantee that all functionality is atomic.

Requirement	Descriptions	Test Cases	Who	H/D/E
The program shall allow the user to choose what to scan (folders, disks, file types) for privacy reasons.	The program shall allow the user to configure parameters for search, in both the UI and the command-line interface. The difficulty of this will be finding an efficient way to filter the tree search. There are lots of open-source algorithms that may be used for the search.	Unit testing shall be used to verify the following: Test case: *Input:* A known folder and validated scan results *Operation:* Scan the known folder *Output:* Compare the operation result to the manually validated file for each. If the files match, the test passes.	Toby	Medium
The program shall support multiple scan modes: - All files (thorough search) - Quick scan (faster, lightweight) - Custom scan (user specifies paths or file types) - Incremental scan (only scan files changed since last scan)	The program shall have preconfigured search settings for the user to consent to with differing levels of search depth. This is easy, given that searches are going to be parameterized anyway, so we just have to pre-make parameterized queries for the search.	Unit testing shall be used to verify the following: Test case: *Input:* A known folder and validated scan results *Operation:* Scan the known folder *Output:* Compare the operation result to the manually validated file for each. If the files match, the test passes.	Toby	Easy
The program shall be compiled for	The program shall run on all platforms	Testing shall be completed on each operating system	Adam	Medium

cross-platform execution (Windows, macOS, Linux)	Provided the low-level nature of our code, we must just recompile for each platform. The egui library is built around cross-platform development. There may be some code that needs localization			
The program shall run locally without an internet connection.	The program shall not require an internet connection, and shall not send user files off their computer for analysis. This is an easy requirement to meet; it just requires us to do all processing on the machine.	Manual testing shall verify this: *Operation:* Run a scan without the internet *Output:* The scan completes without error	Adam	Easy
The program shall scan and analyze the following artifact types: - Code: Git commits, repositories, languages used, version history, branch information - Docs: Word, PDFs, markdown, notes, Excel sheets, presentations - Media: images, design sketches, video files, audio files - Archives: ZIP, RAR, TAR files	The program shall fetch useful metadata from the following file types: .git folders, Word, PDF, markdown, notes, Excel sheets, JPG, Photoshop, video files, audio files, compressed archives, standard archives This will be more difficult to do as it will require learning about all of these file types and how to read their metadata.	Unit testing shall be used to verify the following: *Test case:* *Input:* A known folder and validated scan results *Operation:* Scan the known folder *Output:* Compare the operation result to the manually validated file for each. If the files match, the test passes.	Samuel	Hard
The program shall read metadata from files, including: - File name, type, size, timestamps (creation, modified,	The program shall parse file metadata for supported file types. This will be more difficult to do as it will require learning	Unit testing shall be used to verify the following: Test case: *Input:*	Toby	Hard

last accessed) - Contribution details (e.g., Git commit frequency, authorship) - Document properties (title, word count, page count, custom properties)	about all of these file types and how to read their metadata.	A known folder and validated scan results *Operation:* Scan the known folder *Output:* Compare the operation result to the manually validated file for each. If the files match, the test passes.		
The program shall allow the user to exclude sensitive file types	Users may exclude file extensions from being scanned This may be easy to implement after we create the necessary logic for parameterized searches.	Unit testing shall be used to verify the following: Test case: *Input:* A known folder and validated scan results *Operation:* Scan the known folder *Output:* Compare the operation result to the manually validated file for each. If the files match, the test passes.	Brett	Easy
The program shall avoid scanning common folders known to contain sensitive information, such as - Folders known to contain SSH keys - All files with a .env extension - All operating system files and program files	The program, by default, will avoid file types and folders known to commonly contain sensitive information. This task will be easy, as it just requires having built its parameters, ignoring sensitive files.	Unit testing shall be used to verify the following: Test case: *Input:* A known folder and validated scan results *Operation:* Scan the known folder *Output:* Compare the operation result to the manually validated file for each. If the files match, the	Abdalla	Easy

		test passes.		
The program shall analyze scanned results to compute: - Number of projects and project types - Timeline of work/project evolution - Lines of code, languages used, README.md files from project folders - Document word counts - Media/design file usage frequency - Insights into contributions for version-controlled projects	The program will take in files found in searches and run analysis on them. This task will be hard as it requires the implementation of the file analysis tool for Metadata. And turning the metadata into usable information	Unit testing shall be used to verify the following: Test case: *Input:* A known folder and validated scan results *Operation:* Scan the known folder *Output:* Compare the operation result to the manually validated file for each. If the files match, the test passes.	Brett	Hard
The program shall group files into potential projects and allow users to: - Mark files as belonging to a project - Mark ambiguous files for further analysis	The program shall keep track of which files are in which projects when it produces its results, then use those results in the future. This will be difficult to implement because it requires a side feature to allow users to give feedback and rewrite the results file during/after a scan. We can implement it by adding a lookup before asking the user for input, checking that the file wasn't marked as part of a project before	Manual Validation Required *Input*: A known ambiguous file *Operation*: Scan the file *Output*: the file can be marked, and is seen as non-ambiguous in follow-up scans	Adam	Medium
The program shall have a graphical user interface (GUI) for general users.	This requires that the users have a GUI option to show information more effectively. This will be a difficult task when combined with our	For our GUI testing, we would like to have students use the program and give us feedback on the UI and functionality.	Adam	Hard

	requirement to run on all platforms.			
The program shall also be executable from the command line for advanced users.	This requires that the program have a way to start without the graphical user interface. This means that the entire program must be written as service-based, so that we can have options to remove the GUI from the start sequence. This will be Hard because it requires that the rest of our code be very independent and multithreaded.	Unit testing shall be used to verify the following: Test case: *Input*: Manually validated the folder and Scan result output file *Operation*: Run the search from the command line *Output*: Compare the operation result to the manually validated file. If the files match, the test passes	Adam	Hard
The GUI shall include: - Scan progress visualization (with time estimate, discovered/ignored/excluded/minable files) - Post-scan dashboard showing: - Number of files/projects scanned - Timeline of projects and productivity - Skills worked on and improvements - Highlights (big projects, top repos, longest docs, most collaborators, etc.) - Ability to open folders containing chosen projects directly. - Error reporting	This requirement will require the team to collaborate. It will be a difficult task as the team is learning a new library for making desktop apps(egui library)	GUI features will require manual testing, consisting of - Verifying views - Verifying inputs matching outputs - Testing user experience - Testing UI Designs Each feature shall have a test case consisting of a given action and expected outcome (E.g, clicking the cancel button closes the error window)	Entire Team	Hard

with probable causes when scans fail.				
The program shall store results from a maximum of 2 previous scans.	This requires that the initial output file be saved in the program files. This is easy to do as the file is already made	Manual Verification shall be used here. Test case: We will run a scan, then check that it has been named appropriately and saved where expected.	Adam	Easy
The program shall allow export of insights in common formats: - CSV - JSON - PDF - Markdown	This requires that the program convert the initial report into different formats. This will be a Medium difficulty task, as outputting to different formats in easily readable ways is difficult. The people assigned to this will have to learn how each format works.	Unit testing shall be used to verify the following: Test case: *Input*: Scan result output file *Operation*: Convert the base file into each export type *Output*: Compare the operation result to a manually validated file for each. If the files match, the test passes.	Abdalla	Medium
The program shall allow customization and preview before exporting.	This requires that the program display opportunities for input while files are being processed. This may be difficult to implement as our scan is a background process, so we must process the feedback opportunities as they come from the analysis service.	Manual Verification Required: Run a scan, and check that user changes are reflected in the exported file.	Bao	Hard
The program shall maintain user settings	This requires that the program save a properties file	Manual Verification required:	Adam	Medium

between sessions.	with its settings, parse it on boot up, and write to it on setting changes. This will be a medium task as it requires deep integration with the user's settings	*Input*: New, non-default settings are applied *Operation*: Restart the program *Output*: The settings remain the same across restarts		
The program shall allow timespan filtering (scan only files created/modified within a set period).	The program shall allow the user to configure parameters for search, in both the UI and the command-line interface. The difficulty of this will be finding an efficient way to filter the tree search. There are lots of open-source algorithms that may be used for the search.	Unit testing shall be used to verify the following: Test case: *Input*: A folder with a variety of files to scan. *Operation*: Set a specific time range that excludes every file, except for the intended files. *Output*: Only the configured file is scanned.	Bao	Medium
The program shall present insights in both visual (charts, graphs) and statistical (numeric summaries) formats.	This requires that the users have a GUI option to show information more effectively. This will be a difficult task as it requires whoever is working on it to learn about different ways to analyze metadata, what metrics are valuable, and how to display the data. (May require the use of a library)	Manual Verification required: *Input*: A folder with validated scan results *Operation* Run a scan on the folder *Output*: Scan results displayed match the validated ones from the test case	Bao	Hard

— § —