Bilkent University
Department of Computer Engineering

Senior Design Project

CS 491

CollabHub

Project Specifications Report

Team Members:
Moin Khan – 22101287
Yiğit Özhan – 22201973
İbrahim Çaycı – 22103515
Tuna Göksal – 22203827
Ömer Edip Aras – 22203238

Supervisor: Ayşegül Dündar Boral
Instructors: Mert Bıçakçı, İlker Burak Kurt

November 28, 2025

Contents

1 Introduction

1.1 Description

In last two decades the architectural industry has obtained some software tools thanks to advancement in computer technology. There are some software tools for 2D drawings and 3D modeling. Since these files are complex and can be modified a few times, these files require version control systems for storage and version control [1]. However, there are no sufficient 3D version control tools because they do not provide ability to collaborate with other project contributors. The reason for this incapability is that 3D files are large and viewing them require usage of proprietary tools unlike programming files. Thus, in modern software environment there is no sufficiently capable 3D collaboration tools like programming collaboration tools, and this needs to be solved.

Our project for the given problem, CollabHub, is a specialized, high-performance version control platform engineered exclusively for the architectural domain, with strict support for the .rwt file format. Unlike generic version control systems that treat all files as simple text, CollabHub will be built to understand the specific properties of RWT files. The main objective of CollabHub is to increase productivity by reducing the time architects spend managing file conflicts and restoring lost data. The system allows multiple architects to collaborate within a shared project space, providing features similar to GitHub/Bitbucket but tailored for the visual nature of architectural design. By solving the problem of managing large RWT files, CollabHub delivers value by preventing the limitations imposed by generic frameworks.

Main functionalities are:

- **RWT-Specific Versioning:** The core engine will be optimized to track changes specifically within .rwt files, ensuring that metadata and visual updates are recorded accurately.
- **Visual Change Tracking:** Instead of textual differences (lines of code changed), CollabHub will provide a visual comparison tool that renders two versions of an RWT file and highlights visual differences.
- **Collaboration-Based Workflow:** The platform supports branching and merging, enabling architects to test new design concepts in a safe branch without affecting the main project file until they are ready to merge.

1.2 High Level System Architecture & Components

The CollabHub system utilizes a client-server architecture designed so as to handle the high bandwidth and processing requirements of large 3D architectural files.

1.2.1 Client Side Architecture

The client application is the interface for the architect. Since architects are not software technology savvy as much as software engineers, architects may not be comfortable with Command Line Interfaces (CLI). Therefore, the client side will be built as a Graphical User Interface (GUI). This GUI will be provided as a plugin to RWT development environment since RWT is a proprietary software and visualization can be done by using the dev tools provided by RWT creators. In addition, this will give architects an advantage by combining development and version control environments.

- **Desktop GUI:** A user-friendly GUI that denotes commit history, and branch management.
- **RWT File Watcher:** A background service that monitors the changes to .rwt files. When a save is detected, it prompts the user to commit changes.
- **Local RWT Viewer:** This allows users to preview changes and visual differences.

1.2.2 Server Side Architecture

The server backend is designed for scalability, efficiency and data security.

- **API Gateway:** Serves as the entry point for all client requests, handling authentication, and routing to appropriate microservices.
- **RWT Version Control Engine:** The core logic component that will track the lineage of every RWT file version, managing branches and merge requests.
- **Data Encryption & Security:** AES-256 encryption [2] for all stored RWT files will be implemented, so architectural intellectual property will remain secure against unauthorized access.
- **Storage Layer:** Object storage model will be used to store heavy RWT data efficiently.

1.3 Constraints

1.3.1 Implementation Constraints

- **Proprietary RWT Format:** The .rwt file format is proprietary, so parsing such files often requires using specific provided developer toolkits, and libraries.
- **File Size and Bandwidth:** RWT files can exceed several gigabytes. The implementation must utilize efficient compression algorithms to ensure that syncing a project does not consume excessive bandwidth or take an unreasonable amount of time.

1.3.2 Economic Constraints

- **High Storage Costs:** Due to the large size of RWT files and the requirement to store historical versions, the project will create high cloud storage costs compared to standard text-based version control applications.
- **Development Costs:** Licensing the necessary SDKs or toolkits to legally parse and display proprietary RWT files may require financial support.

1.3.3 Ethical Constraints

- **Intellectual Property Protection:** Architects will trust our platform by uploading their designs. Any security breach can result in the theft of their intellectual property, will cause severe reputational damage.
- **Privacy of Location Data:** If the system tracks user activity or login locations for security, this data must be handled according to strict privacy standards, ensuring it is not used for unauthorized surveillance.

1.4 Professional and Ethical Issues

The development of CollabHub requires strict commitment to professional engineering ethics.

- **Data Integrity and Safety:** Architecture is a discipline where precision is vital. A corruption in the version control process can lead to structural flaws. Therefore, the system must guarantee 100% data integrity during the transfer and storage of RWT files.
- **Confidentiality:** We must treat all user data as confidential. Debugging processes should never involve accessing a client's proprietary RWT files without explicit permission.
- **Transparency on Limitations:** The team must be honest about the limitations of the application. The limitations must be clearly stated to the architects to prevent false confidence.

1.5 Standards

The project will follow stated engineering standards to ensure reliability, security, and maintainability.

- **IEEE 830 (Software Requirements Specifications):** This standard will be used to document all functional and non-functional requirements, ensuring the scope is clearly defined and agreed upon [3].
- **UML 2.5.1 (Unified Modeling Language):** All diagrams, such as Class, Sequence, and Component diagrams, will be created using standard UML notation to ensure clarity.
- **AES-256 (Advanced Encryption Standard):** All data storage will be encrypted using AES-256 to prevent unauthorized access [2].

2 Design Requirements

2.1 Functional Requirements

1. The system will extract all relevant architectural data from the project file and convert it into a structured format suitable for backend processing.
2. The system will support uploading the extracted base version of the file to the backend, establishing the initial reference state for future comparisons.
3. The client will compare the current file state with the previously stored version to identify additions, modifications, and deletions.
4. Only the identified differences will be transmitted to the backend instead of resending the entire project file.
5. Each new version will be stored on the backend together with basic metadata such as timestamp and user information.
6. The client will allow users to pull the latest available version from the backend and apply the corresponding changes to their local file.
7. When a change cannot be applied (e.g., due to missing or altered referenced components), the system will notify the user of the conflict.

8. Where supported, the client will utilize built-in visual comparison tools to highlight differences between versions.
9. A custom visual comparison tool may be developed in later stages, but it is not a requirement for the current phase of the project.
10. The client interface will provide simple controls that allow the user to push changes, pull updates, and view available versions within the application.

2.2 Non-Functional Requirements

2.2.1 Usability

1. The user interface will be integrated directly into the existing workflow to ensure that architects can easily perform version control tasks without additional training.
2. Core actions such as pushing and pulling updates will be clearly presented and simple to execute.
3. The system will provide clear and understandable feedback messages indicating the success or failure of an operation.
4. Formal usability studies or advanced UX evaluations are not required at this stage of the project.

2.2.2 Reliability

1. A version will only be stored if all associated data has been fully and correctly received by the backend.
2. The system will report common errors such as invalid data or network failures to the user.
3. Incomplete or corrupted differences will not be stored or applied.
4. Advanced reliability guarantees such as redundancy or system failover are not applicable at this stage.

2.2.3 Performance

1. The system will conceptually improve efficiency by transmitting only differences between versions rather than entire files.
2. The client is expected to behave with responsiveness typical of modern desktop applications, without strict numerical performance targets.
3. Formal performance benchmarking, stress testing, and optimization requirements are not applicable at this stage of development.

2.2.4 Supportability

1. The system will maintain a modular internal structure to make future enhancements and maintenance easier.
2. Basic logging of push, pull, and error events will be implemented to support debugging.
3. Developer documentation appropriate for a university-level project will be provided.

4. Long-term maintenance planning and external API exposure are not required at this stage.

2.2.5 Scalability

1. The backend will be structured to logically support multiple projects, even if testing is limited to a smaller scope.
2. The version storage structure will be designed so that it can be extended in future phases without major redesign.
3. Large-scale deployment requirements such as heavy concurrency, load balancing, or distributed infrastructure are not applicable at this stage.

3 Feasibility Discussions

3.1 Market & Competitive Analysis

Multiple platforms exist in the architectural collaboration ecosystem, but none directly target file-format-aware version control for RWT files.

3D Repo [4] is a cloud-based viewer and coordination tool. It supports issue tracking and model federation but does not provide branching, merging, or internal diffing of proprietary files. During earlier discussions, it was determined that while 3D Repo allows online review, it is not a version-control system in the Git sense and does not manage the architectural file itself.

Autodesk BIM Collaborate [5] provides cloud-based project coordination and dashboards. It offers clash detection and design review features but does not support Git-style branching or user-controlled version lineage. You previously asked whether BIM Collaborate is “the same as our project,” and the conclusion was that it is similar conceptually but does not attempt deep version tracking of a specific binary format.

How CollabHub differs:

- It is not a cloud viewer or clash detection tool.
- It is not tied to a BIM ecosystem such as Autodesk.
- It focuses on RWT-specific versioning, not general-purpose upload-and-view workflows.
- It supports branching and merging of architectural design states.
- It allows vertex-based commenting, tightly integrated with version tracking.

Because existing tools focus on viewing and coordination rather than detailed version control, CollabHub fills a clear gap in the architectural software ecosystem.

Beyond field-specific tools, it is also useful to observe the broader software engineering landscape. Platforms such as GitHub have reshaped collaborative development by introducing structured version histories, distributed workflows, and peer review mechanisms [6]. These practices reduced integration conflicts and improved team coordination, eventually becoming an industry standard. The academic literature also highlights GitHub’s influence on large-scale collaborative development and community-driven software engineering [7]. Although architectural design differs from text-based programming, the underlying need for reliable change tracking is similar. This parallel illustrates the potential value of a domain-specific version control system for RWT files.

3.2 Academic Analysis

From an academic perspective, CollabHub builds upon several well-established areas of computer engineering research, particularly distributed version control, binary differencing, and secure data management. While these topics are covered conceptually across various courses, implementing them for large 3D architectural files introduces challenges not typically encountered in standard coursework.

A core requirement of CollabHub is identifying changes between two versions of a proprietary binary file. This relates directly to research on delta encoding and binary differencing. Classical algorithms such as the rsync rolling-checksum technique [8] and binary diff approaches like xdelta [9] demonstrate how differences between large binary objects can be computed and compressed efficiently. Although CollabHub does not require implementing these algorithms directly, the underlying principles influence how the system extracts and stores changes between RWT file versions.

CollabHub also aligns with academic material on distributed version control systems. Git's design philosophy—content-addressable storage, immutable history, and branch-based development—has been widely analyzed in the literature. Studies such as Bird et al.'s work on GitHub [7] demonstrate how structured version histories improve coordination in collaborative environments. CollabHub adapts similar ideas but applies them to a domain where version control is not normally available: architectural design files.

Secure data management is another area with strong academic foundations. The need to store and transmit large binary files safely reflects principles discussed in research on encrypted storage systems and secure cloud architectures. Techniques for ensuring data integrity and preventing tampering are supported by standard cryptographic practices such as AES-based encryption and integrity verification methods, both widely studied in the context of secure file storage [10].

Finally, integrating these technical components into a workflow usable by architects introduces human–computer interaction considerations. Academic work on domain-specific tool adoption highlights the importance of minimizing cognitive load and embedding new tools directly into existing environments [11]. This supports CollabHub's decision to integrate as an RWT plug-in, rather than requiring architects to adopt unfamiliar tools.

Overall, the project touches on multiple research-backed domains—binary differencing, distributed systems, secure storage, and human–computer interaction—making it both academically grounded and sufficiently challenging for a senior design project.

4 Glossary

RWT File: Proprietary architectural design file containing 3D models and metadata.
Version Control: Tracking file changes over time.
Diff Extraction: Identifying what changed between two file versions.
Branch: A separate line of development.
Merge: Combining changes from branches.
Backend: Server-side system managing data and logic.

Client Application: User-facing software for interacting with version control.
Metadata: Information such as timestamp and user.
Object Storage: Storage method optimized for large binary files.

5 References

[1] N. Leavitt, “Software Version Control Systems,” *Computer*, vol. 38, no. 6, 2005.

[2] National Institute of Standards and Technology, “Advanced Encryption Standard (AES),” FIPS PUB 197, 2001.

[3] IEEE, “IEEE Std 830-1998: Recommended Practice for Software Requirements Specifications,” 1998.

[4] 3D Repo Ltd., “3D Repo — BIM Collaboration Platform,” 2025. [Online]. Available: https://3drepo.com/

[5] Autodesk, “Autodesk BIM Collaborate,” 2025. [Online]. Available: https://www.autodesk.com/products/bim-collaborate/overview

[6] GitHub, “GitHub: Collaborative Software Development Platform,” 2025. [Online]. Available: https://github.com/

[7] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, and P. Devanbu, “Lessons from GitHub: Harnessing the Power of Open Source Communities,” *IEEE Software*, vol. 30, no. 4, pp. 26–31, 2013.

[8] A. Tridgell and P. Mackerras, “The rsync Algorithm,” *Technical Report TR-CS-96-05*, Australian National University, 1996.

[9] J. MacDonald, “File System Support for Delta Compression,” in *Proceedings of the USENIX Annual Technical Conference*, 2000.

[10] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Wiley, 1996.

[11] T. Green and M. Petre, “Usability Analysis of Visual Programming Environments: A ’Cognitive Dimensions’ Framework,” *Journal of Visual Languages & Computing*, 1996.