Bilkent University
Department of Computer Engineering

Senior Design Project
T2527
CollabHub

Analysis and Requirement Report

Tuna Göksal | 22203827
Yiğit Özhan | 22201973
İbrahim Çaycı | 22103515
Moin Khan | 22101287
Ömer Edip Aras | 22203238

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

19/12/2025

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Senior Design Project course CS491/2.

Contents

1 Introduction

In the 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 requires usage of proprietary tools unlike programming files. Thus, in the modern software environment there are no sufficiently capable 3D collaboration tools like programming collaboration tools, and this needs to be solved.

The objective of this project is to develop CollabHub, a specialized, high-performance version control platform engineered exclusively for the architectural domain. Unlike generic version control systems, CollabHub addresses the specific limitations imposed by large, proprietary RWT files, aiming to increase productivity by reducing the time architects spend managing file conflicts.

2 Current System

Architectural design collaboration is currently handled through a combination of file-based workflows and proprietary coordination platforms. In most practice environments, architects work on local copies of large RWT files and exchange updated versions via shared network drives, cloud storage services, or email. Version tracking in this process is informal and typically relies on manual file naming conventions or external documentation. As the number of collaborators and project complexity increase, this approach often leads to overwritten changes, loss of intermediate design states, and uncertainty about who made a particular modification and when.

Several commercial tools attempt to improve collaboration by offering centralized, cloud-based environments. Autodesk BIM Collaborate enables teams to publish and review models using Autodesk's cloud infrastructure and supports workflows such as design coordination and clash detection [2]. While these features improve accessibility and coordination across teams, they operate primarily at the level of complete model versions and require commercial licensing. During informal discussions conducted with architecture students, it was noted that BIM Collaborate is not included in standard student licensing plans, which limits its accessibility in academic settings.

Similarly, platforms such as 3D Repo provide cloud-based model visualization, issue tracking, and comparison between uploaded revisions[3]. These tools are effective for design review and coordination but treat architectural models as review artifacts rather than actively versioned design assets. They do not manage the internal structure of proprietary RWT files, nor do they support controlled merging of concurrent changes directly within the authoring environment. Feedback gathered from architecture students also indicated that subscription costs for tools such as 3D Repo and similar BIM coordination platforms are considered prohibitively expensive for regular student use.

Generic version control systems such as Git are not suitable for architectural workflows. RWT files are large proprietary binary files, making textual differencing impractical and storage inefficient. These systems lack awareness of architectural semantics and cannot provide visual feedback or safe conflict resolution for geometric and parametric changes.

As a result, existing solutions focus on file sharing, cloud-based coordination, or model review, often with significant cost barriers and without providing structured version control tailored to architectural design files. In contrast, CollabHub is designed as a free system developed within an academic context, prioritizing accessibility alongside technical capability. By avoiding licensing barriers and focusing on RWT-specific version tracking, branching, and controlled merging, CollabHub aims to make advanced collaboration mechanisms available to a broader range of users, particularly students, small teams, and research-oriented environments.

3 Proposed System

3.1 Overview

CollabHub is designed as a specialized version control platform engineered exclusively for the architectural domain, with strict support for the .rwt file format. The system allows multiple architects to collaborate within a shared project space, providing features similar to GitHub or Bitbucket but tailored for the visual nature of architectural design. The main objective is to prevent the limitations imposed by generic frameworks by utilizing a core engine optimized to track changes specifically within .rwt files.

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

- **Client Side:** The client application serves as the interface for the architect. It is built as a Graphical User Interface (GUI) provided as a plugin to the RWT development environment, combining development and version control environments. It includes a Desktop GUI for history management, an RWTFileWatcher background service, and a Local RWTViewer.
- **Server Side:** The backend is designed for scalability and efficiency. It includes an API Gateway for request routing, an RWT Version Control Engine to track file lineage, and an Object Storage layer to handle heavy RWT data efficiently.

3.2 Functional Requirements

The following requirements define the specific behaviors and functions the system must support:

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.

3.3 Non-functional Requirements

3.3.1 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.

3.3.2 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.

3.3.3 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.

3.3.4 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.

3.3.5 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.

3.4 Pseudo Requirements

This section details the implementation constraints, economic and ethical limitations, and engineering standards that mandate how the system must be constructed.

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.

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.

Ethical and Privacy 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.

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].

3.5 System Models

3.5.1 Scenarios

Scenario Name: User Sign-Up **Participating Actors:** Architect (Collaborator) **Entry Condition:** The user has installed the plugin, opened Revit, and clicked the "Sign Up" button in the CollabHub ribbon tab. **Exit Condition:** The user is successfully registered and authenticated within the plugin. **The Flow of Events:**

1. The plugin opens a modal window requesting details: name, email, and password.
2. The system sends confirmation email
3. If credentials are confirmed, the system creates the user record.

Scenario Name: User Login **Participating Actors:** Architect (Collaborator), Project Owner **Entry Condition:** The user clicks the "Login" button on the CollabHub panel.**Exit Condition:** The user is authenticated, and the plugin connects to the server. **The Flow of Events:**

1. The user enters credentials into the plugin's login panel.
2. The system verifies credentials against the server.
3. If correct, the plugin retrieves the user's project list.

Scenario Name: Create New Project **Participating Actors:** Project Owner **Entry Condition:** The user has a model open in Revit that is not yet version-controlled. **Exit Condition:** The currently open model is initialized as a CollabHub repository. **The Flow of Events:**

1. The user navigates to the CollabHub tab and clicks "Create Project."
2. The user enters a project name and description in the plugin popup.
3. The system links the currently open file to a new remote repository.
4. The plugin performs the initial extraction of the open model.
5. The scenario ends when the plugin indicates the project is "Tracked."

Scenario Name: Invite Team Member **Participating Actors:** Project Owner **Entry Condition:** The user is logged into the plugin and is the owner of the active project. **Exit Condition:** An invitation is sent. **The Flow of Events:**

1. The user clicks the "Settings" icon in the CollabHub panel.
2. The user selects the "Team" tab.
3. The user enters the email of the collaborator.
4. The system validates the email.
5. The system sends the invitation.
6. The scenario ends when the UI shows "Invitation Pending."

Scenario Name: Push Changes to Server **Participating Actors:** Collaborator **Entry Condition:** The user has made changes to the open Revit model and saved them locally. **Exit Condition:** The changes are synced to the server. **The Flow of Events:**

1. The user clicks the "Push" button in the CollabHub panel.
2. The plugin locks the UI to prevent further edits during the process.
3. The system extracts data directly from the active Revit API context.
4. The system compares this data against the last known server state.
5. The system transmits only the differences (diffs).

6. The user receives a "Success" toast notification in the Revit window.

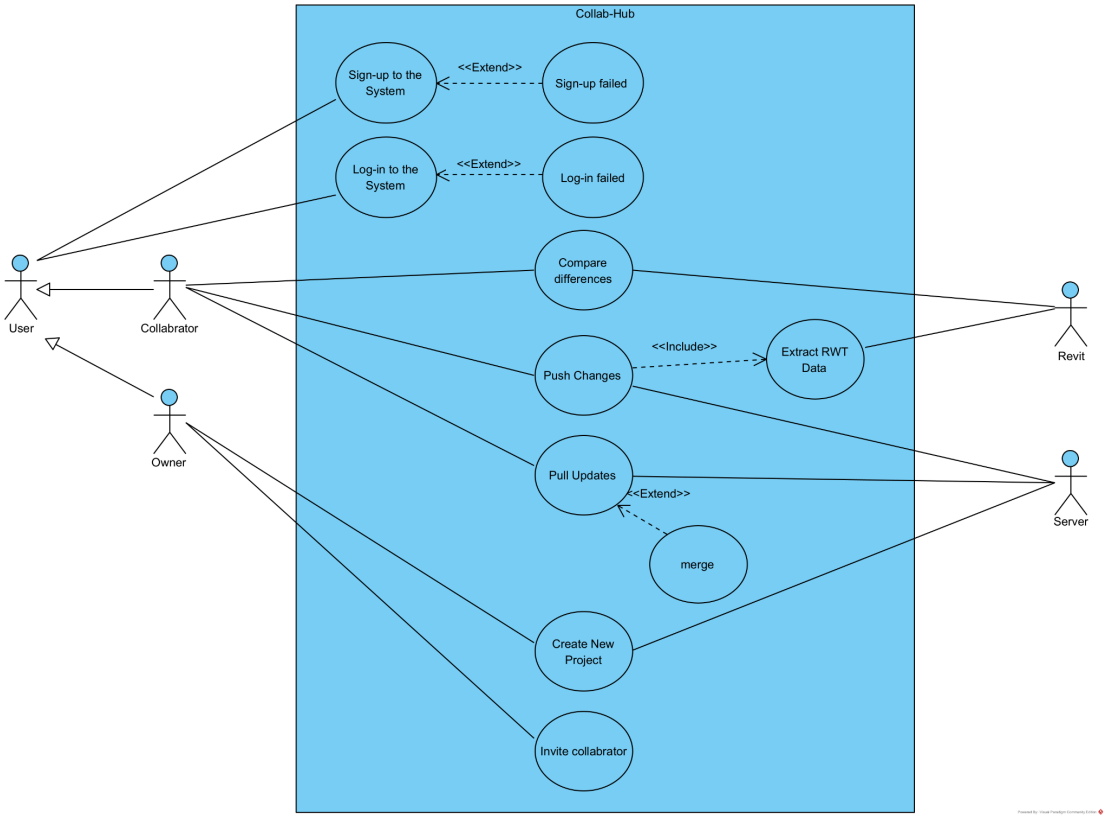
Scenario Name: Pull Updates from Server **Participating Actors:** Collaborator **Entry Condition:** The plugin notification icon shows that new commits are available from the team. **Exit Condition:** The open Revit model is updated with team changes. **The Flow of Events:**

1. The user clicks the "Pull" button in the CollabHub panel.
2. The plugin downloads the differential data.
3. The plugin utilizes the Revit API to modify the current open document (e.g., moving walls, adding families).
4. If no conflicts exist, the Revit view refreshes to show the new geometry.
5. The scenario ends when the "Up to Date" status is displayed.

Scenario Name: Visual Comparison (Diff) **Participating Actors:** Collaborator **Entry Condition:** The user wants to compare their current open view with a past version. **Exit Condition:** The Revit viewport displays a color-coded analysis. **The Flow of Events:**

1. The user opens the "History" panel in the plugin.
2. The user selects a previous commit and clicks "Visual Diff."
3. The plugin generates a temporary "Analysis View" inside Revit.
4. The plugin overrides graphics: Green for new elements, Red for deleted, Yellow for moved.
5. The scenario ends when the user switches back to their standard view.

3.5.2 Use-Case Model



3.5.3 Object and Class Model

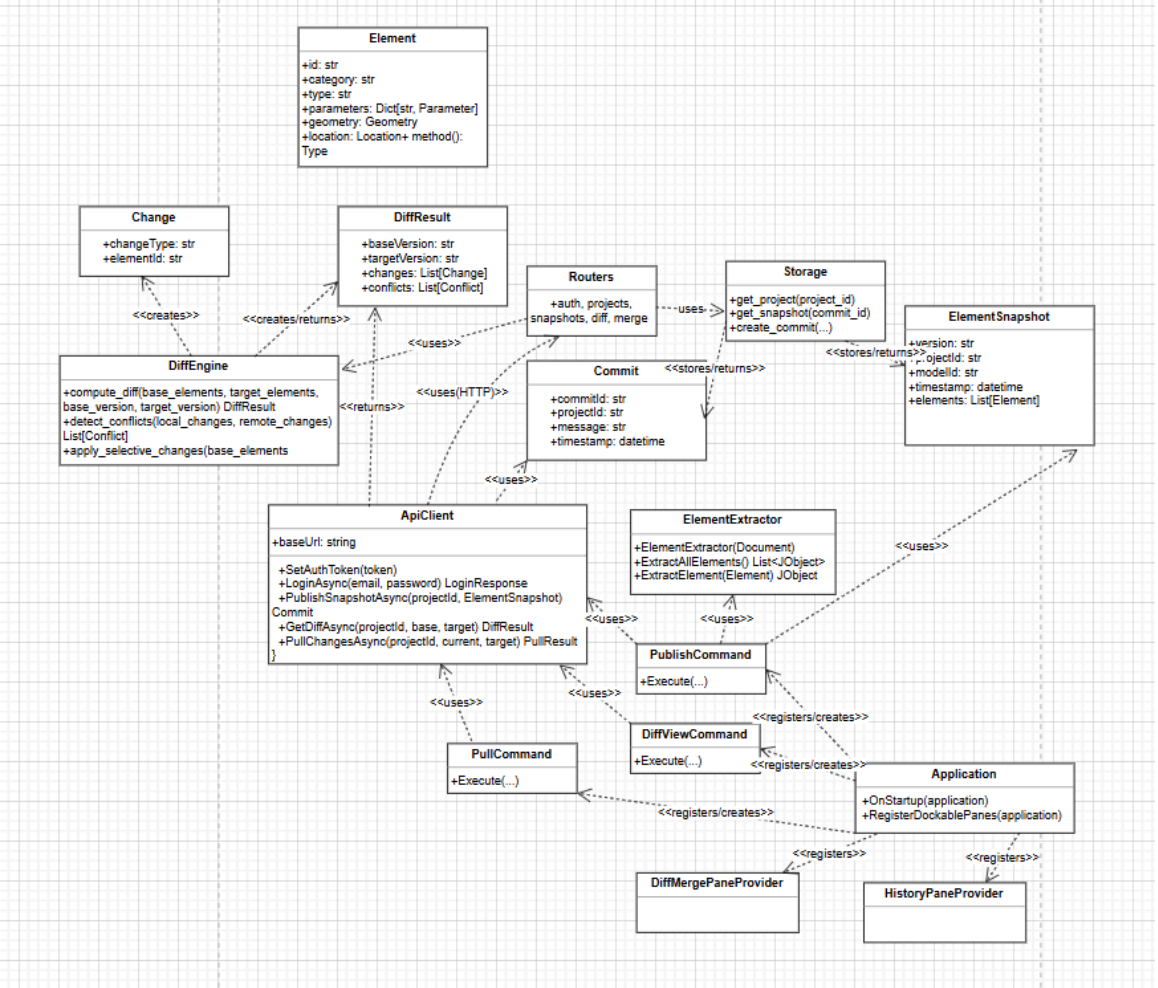
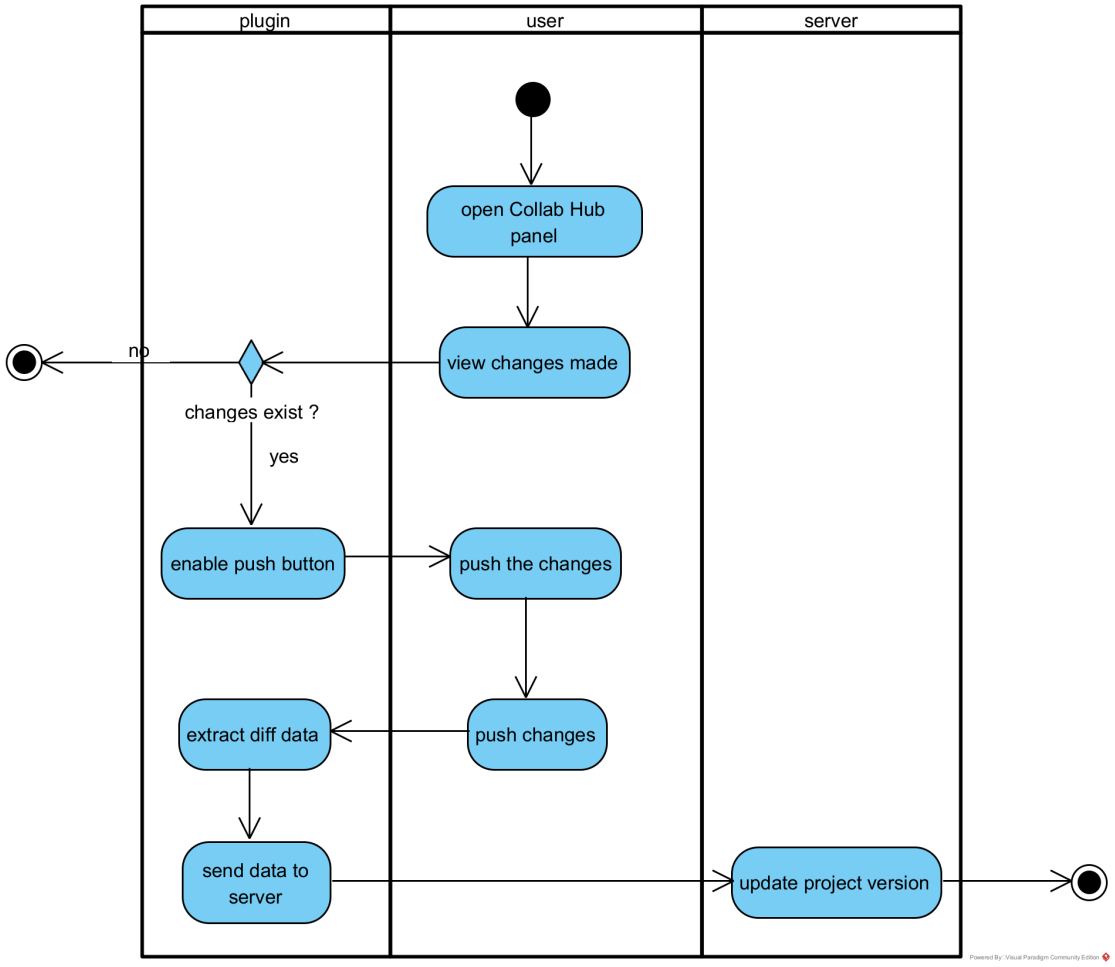


Figure : Class & Object Diagram

3.5.4 Dynamic Models

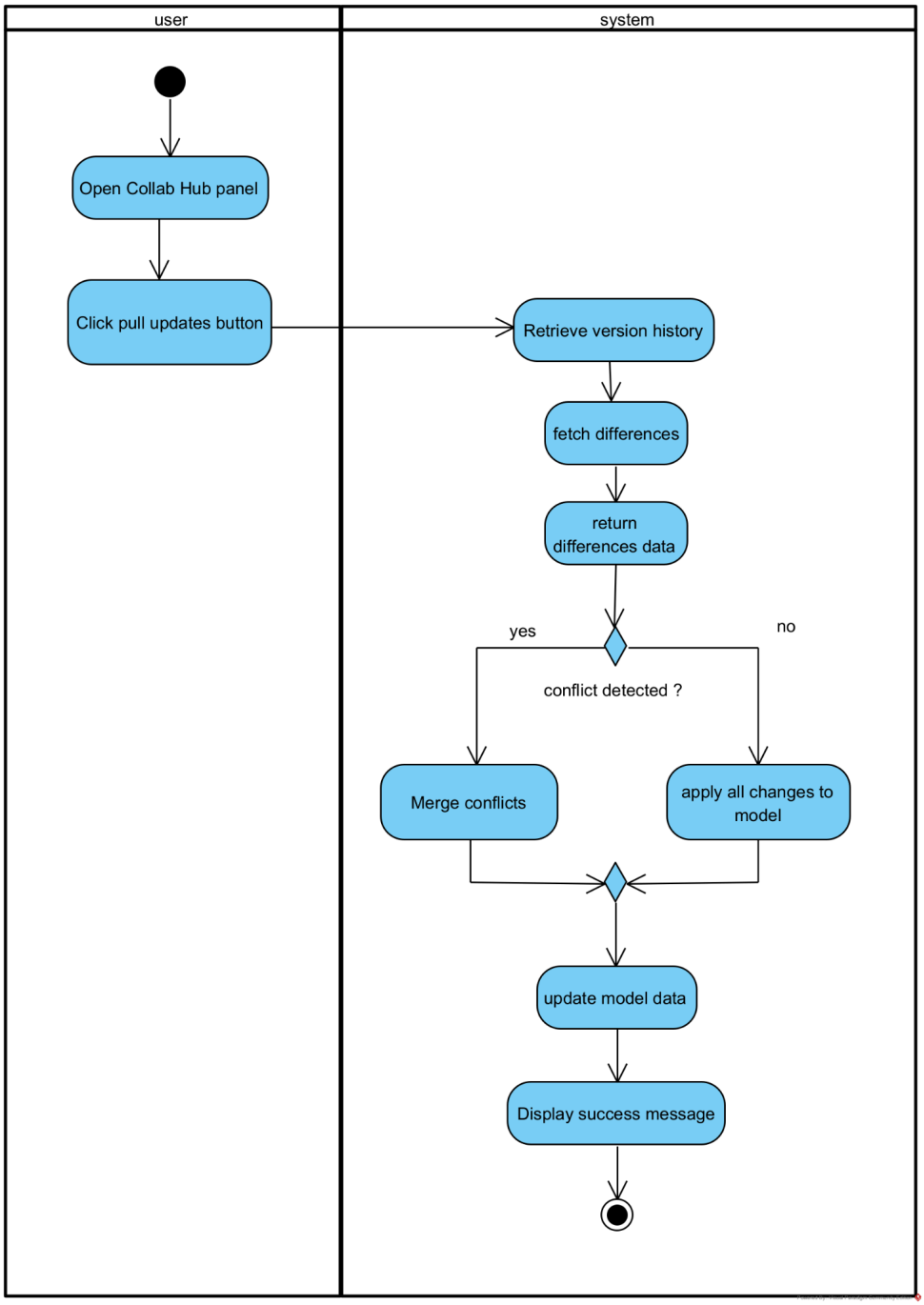
3.5.4.1 Activity Diagrams

3.5.4.1.1 Push Changes



Above is the activity diagram that shows the process for pushing the changes made to the Collab Hub.

3.5.4.1.2 Pull Changes



Above is the activity diagram that shows the process for pulling the changes from the Collab Hub.

3.5.4.2 Sequence Diagrams

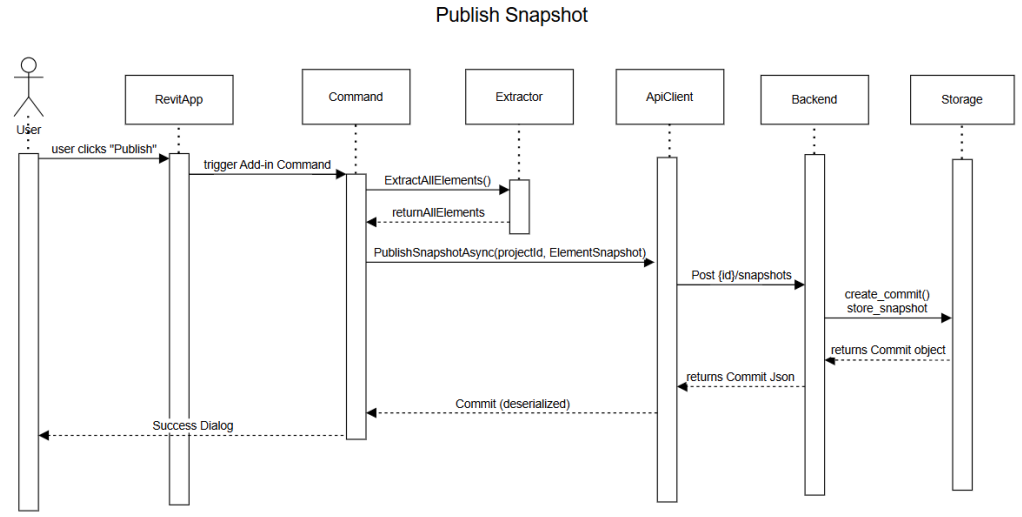


Figure : Publish Snapshot (Push) Sequence Diagram

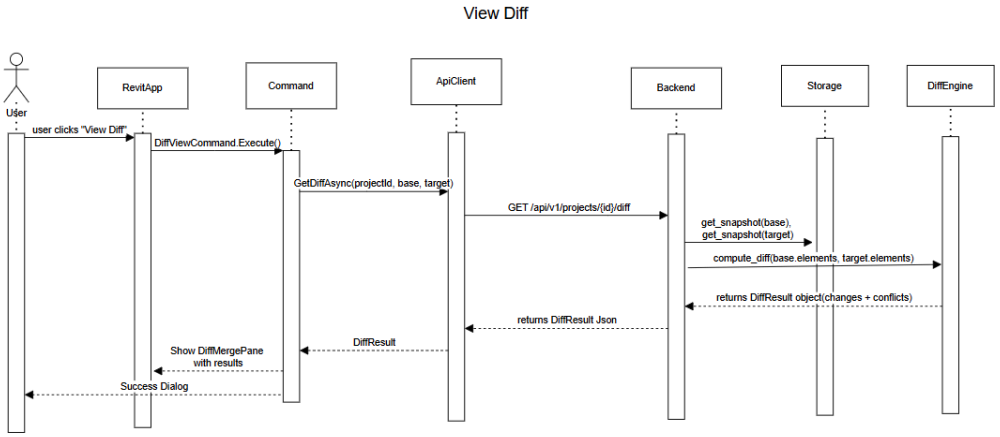


Figure : View Diff Sequence Diagram

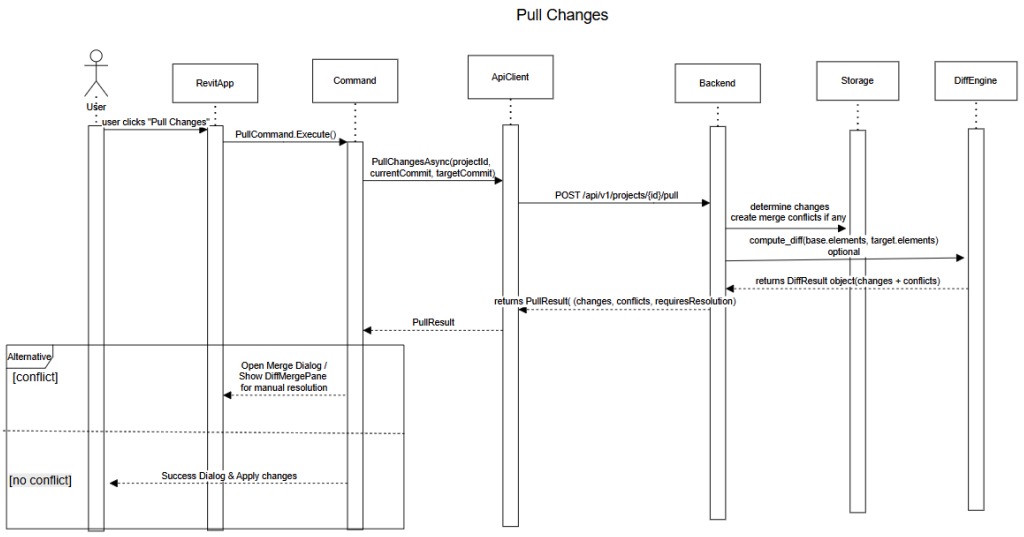
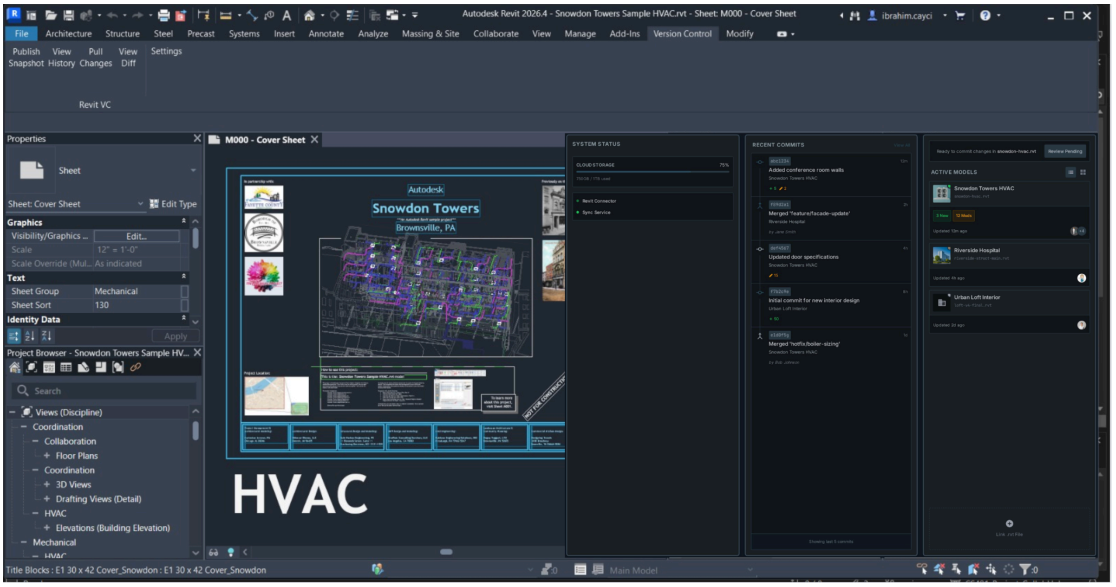
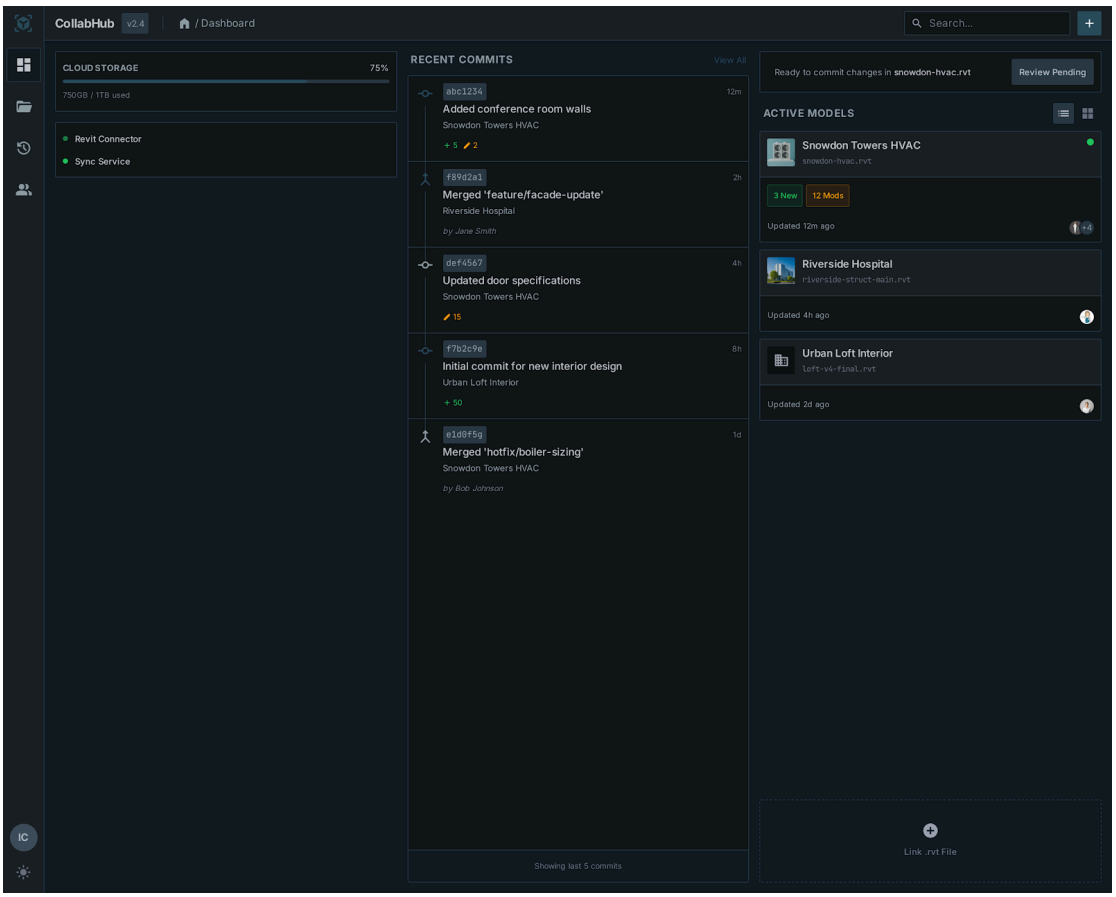


Figure : Pull Changes Sequence Diagram

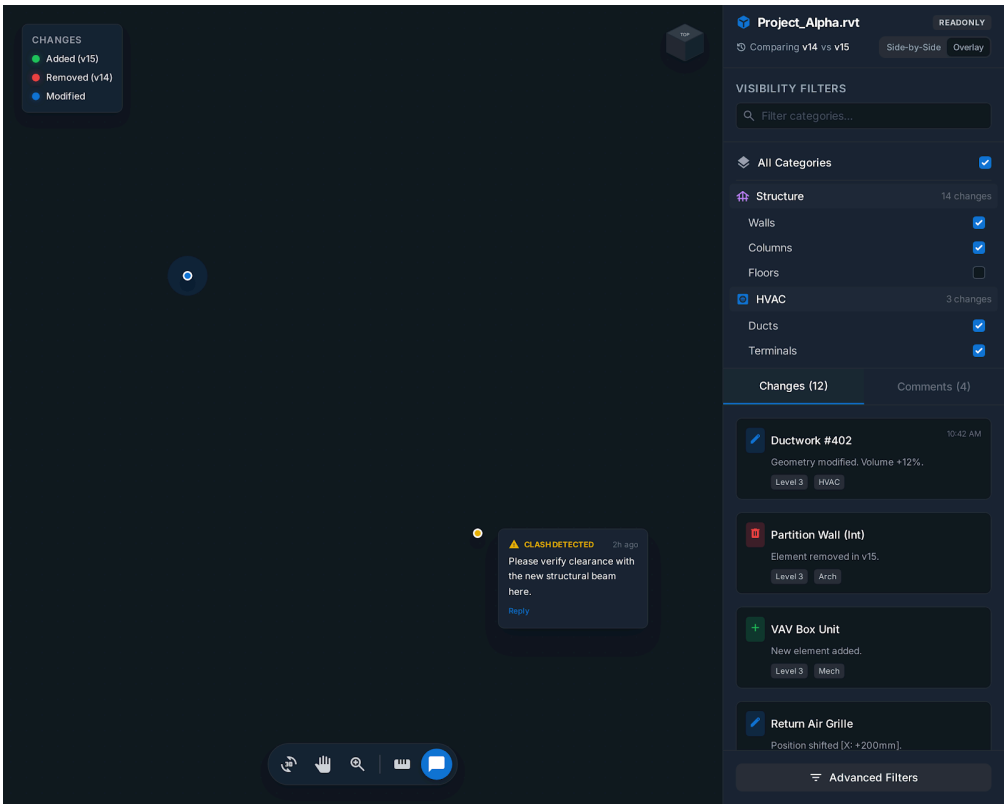
3.5.5 User Interface



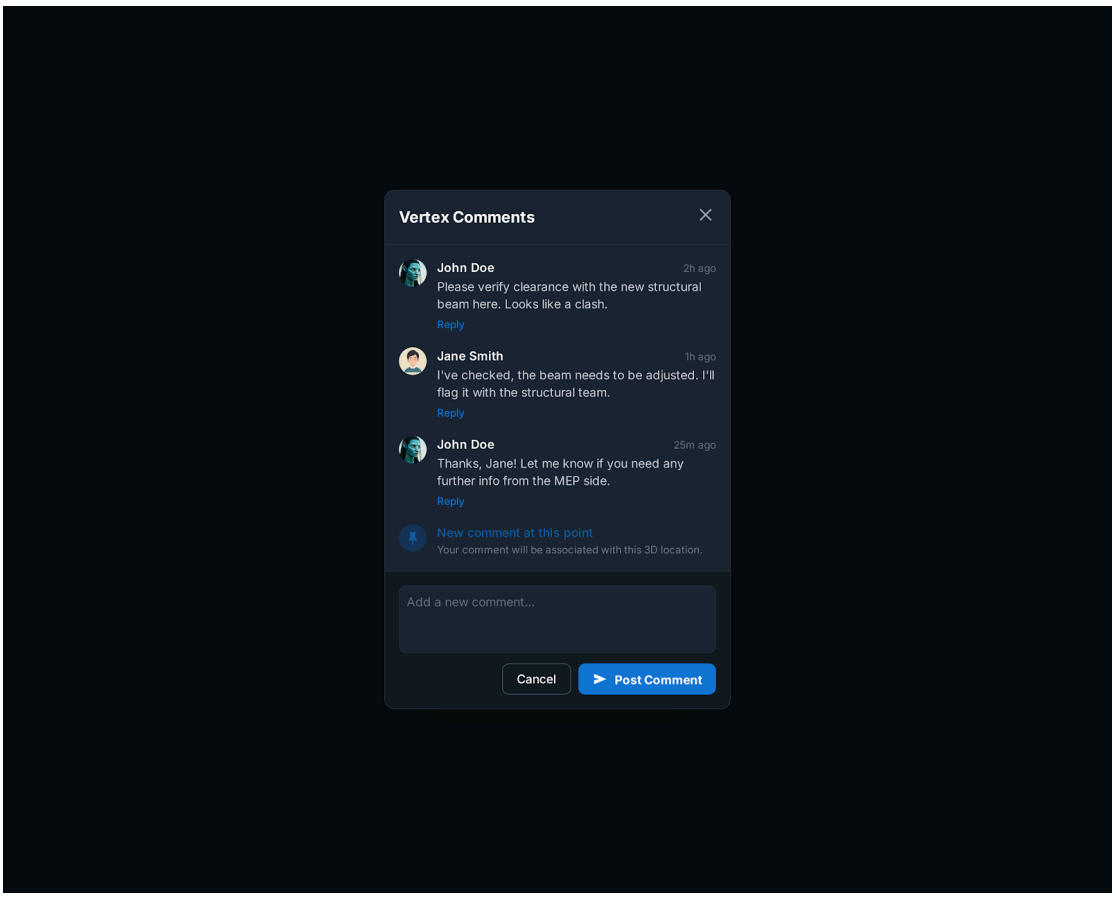
Dashboard panel



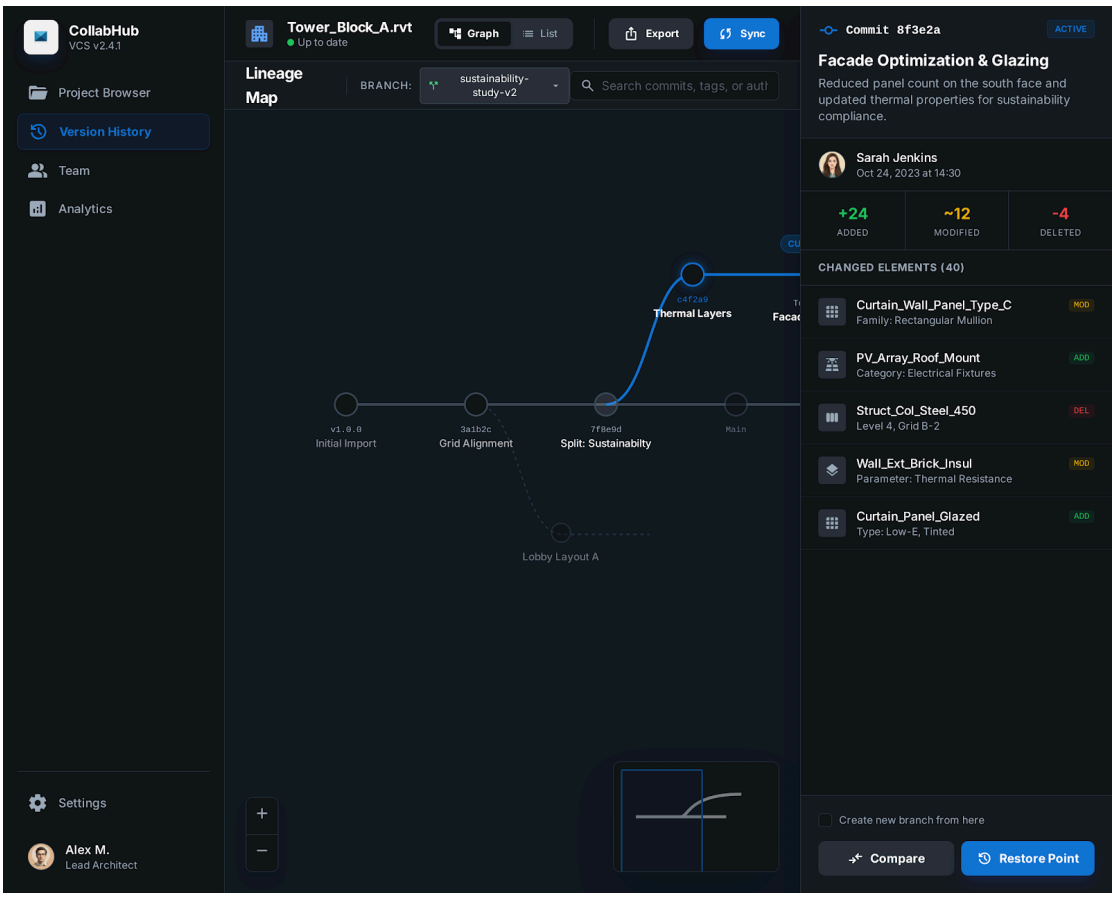
Dashboard panel close-up



Diff view panel



Vertex comments screen



Timeline Page

4 Other Analysis Elements

4.1 Consideration of Various Factors in Engineering Design

4.1.1 Constraints

4.1.1.1 Implementation Constraints

- CollabHub will be developed primarily as a Autodesk Revit Plugin using C# and the .NET framework to ensure integration with the Revit environment.
- The backend will be built using Python (specifically with FastAPI framework) as it is a framework that most of the team are experienced with.
- PostgreSQL will be used as the relational database for storing metadata (commit logs, user info, branch pointers).
- The system must comply with the Autodesk Revit API limitations; specifically, it must handle main-thread restrictions when interacting with the active model to prevent freezing the user interface.[4]

4.1.1.2 Economic Constraints

- The project must utilize open-source libraries and free-tier cloud services where possible to fit within a project budget of zero.
- Since .rwt files are significantly larger (Gigabytes) than text files of code, Cloud storage costs are a major constraint. The system must implement an efficient way of storing the .rwt files in its database.

- The development requires access to Autodesk Revit licenses. The team relies on educational licenses that can be accessed through university..

4.1.1.3 Ethical Constraints

- **Intellectual Property (IP) Protection:** Architects' designs are their trade secrets. The system must ensure that proprietary design files are not accessible to unauthorized users.
- **Data Integrity:** In the construction industry, a corrupted file can lead to structural errors in the real world. The system is ethically bound to verify the integrity of every uploaded and downloaded file to prevent losses and potential safety hazards.(eg. The system must clearly inform the user if a "merge" operation resulted in any data loss or if a conflict requires manual intervention.)
- **Privacy:** User activity logs must be stored securely and only visible to authorized team members, not exposed publicly.

4.1.1.4 Social Constraints

- **Usability for Non-Programmers:** Unlike software engineers, architects may not be familiar with usage of command line tools. The application must provide a Graphical User Interface (GUI) that abstracts complex version control concepts into understandable terms.
- **Collaboration:** The system should foster a collaborative environment by allowing multiple architects to work on different parts of a building simultaneously without "locking" the entire project. This constraint also reflects the core functionality this project aims to achieve.

4.1.2 Standards

4.1.2.1 IEEE 830 The IEEE 830 standard provides the guidelines for our Software Requirements Specifications (SRS). By following this standard, we ensure that our requirements for "RWT Versioning" and "Visual Diffing" are clearly defined, unambiguous, and verifiable, serving as a solid contract between the developers and the stakeholders.[3]

4.1.2.2 ISO 19650 (BIM Information Management) While primarily a construction standard, ISO 19650 defines the concepts of a Common Data Environment (CDE). CollabHub aligns with these standards by managing the "Work in Progress" and "Shared" states of information containers (RWT files), ensuring the project adheres to industry-standard information lifecycles.[5]

4.1.2.3 UML 2.5.1 - Unified Modeling Language We utilize UML 2.5.1 to visualize the complex interactions of our system. Specifically, Sequence Diagrams are used to map the synchronization process between the Client Plugin and the API Gateway, and Class Diagrams are used to structure the internal logic of the proprietary file parser.[6]

4.1.2.4 AES-256 (Advanced Encryption Standard): All data storage will be encrypted using AES-256 to prevent unauthorized access [2].

4.2 Risks and Alternatives

Potential Risk	Likelihood	Impact	Alternative / Mitigation Strategy (Plan B)
Data Persistence Failure (Currently using in-memory storage.py)	**4**	**10**	**Alternative:** Implement a containerized PostgreSQL or MongoDB instance. Use an Object-Relational Mapper (SQLAlchemy) to ensure data is written to disk immediately upon a "Publish" command.
Revit Version Incompatibility (C# API changes between Revit 2024/2025)	**4**	**8**	**Alternative:** Develop a "Version Wrapper" or abstraction layer in the C# plugin. If the latest API fails, the system falls back to a basic IFC-based (Industry Foundation Classes) extraction method.
High Latency/Network Bottleneck (Large JSON snapshots slowing Revit)	**5**	**5**	**Alternative:** Implement "Delta-Only" transfers. Instead of sending all elements, the ElementExtractor compares hashes locally and only sends elements that have changed since the last CommitID.

Geometry Hashing Collisions (Different shapes producing the same hash)	**2**	**9**	**Alternative:** Move beyond simple hashing to a "Multi-Factor Verification" where the DiffEngine checks bounding box dimensions, volume, and XYZ coordinates in addition to the hash.
Authentication/JWT Expiry (Users losing connection mid-sync)	**2**	**3**	**Alternative:** Implement a "Local Cache" or "Draft Mode" in the C# plugin. If the FastAPI server is unreachable, changes are saved to a local SQLite file and pushed automatically once the connection is restored.

4.3 Project Plan

Factor	Effect Level	Effect
Public Safety	**9**	Ensuring the DiffEngine accurately identifies changes in structural elements to prevent collapse risks during BIM updates.
Public Welfare	**6**	Improving project delivery speed, which reduces public infrastructure costs and delays.

Global Factors	**8**	Compliance with international BIM standards (ISO 19650) for interoperable data exchange via JSON.
Environmental	**4**	Tracking material quantity changes in the VCS helps monitor the carbon footprint of the design.
Economic	**9**	Reducing labor costs associated with "manual" file merging and data loss in architectural firms.

Risk	Likelihood	Effect	B Plan Summary
Data Loss (In-Memory)	**2**	**10**	Replace storage.py in-memory logic with a persistent PostgreSQL database immediately.
Geometry Hash Conflict	**4**	**10**	Implement a "Strict Mode" that flags complex geometry for manual verification if hashes match incorrectly.
Revit API Breaking	**2**	**8**	Use a version-agnostic wrapper in the C# plugin to support Revit 2023, 2024, and 2025.

WP#	Work Package Title	Leader	Members involved
WP1	Backend API & Diff Logic	Tuna Göksal	Ömer Edip Aras, Moin Khan
WP2	Revit Plugin & Element Extraction	Moin Khan	İbrahim Çaycı, Ömer Edip Aras
WP3	UI/UX & Conflict Resolution	Yiğit Özhan	İbrahim Çaycı

WP 1: Backend API & Diff Logic			
Start date: Week 1 **End date:** Week 4			
Leader:	Tuna Göksal	**Members involved:**	Ömer Edip Aras, Moin Khan
Objectives: Create the bridge between the Revit modeling environment and the VCS server. This package focuses on extracting geometric and parameter data from Revit elements and converting them to JSON.			
Tasks: ***Task 1.1*** **Server Setup :** Configure main.py with FastAPI, CORS, and JWT authentication routers. ***Task 1.2*** **Diff Engine Development :**Program diff_engine.py to calculate differences between two ElementSnapshot objects. ***Task 1.3*** **Storage Persistence: :** Transition storage.py from temporary RAM to a structured database.			
Deliverables ***D1.1:*** Functional REST API Documentation (Swagger). ***D1.2:*** DiffEngine logic supporting selective change application			

WP 2: Revit Plugin & Element Extraction			
Start date: *Week 2* **End date:** *Week 6*			
Leader:	Moin Khan	**Members involved:**	İbrahim Çaycı, Ömer Edip Aras
Objectives: Build the client-side bridge between the Autodesk Revit environment and the CollabHub backend. This involves developing the C# logic to query the Revit database, extract element geometry and parameters, and serialize them for transmission.			
Tasks: ***Task 3.1*** **History Pane:** Create the HistoryPane.xaml to display a list of previous commits and authors.			

Task 3.2 **Visual Diff UI:** Develop the DiffMergePane to highlight changes directly in the Revit 3D viewport using colors (Red/Green). ***Task 3.3*** **Conflict Dialogs:** Implement DiffSelectDialog.cs for manual selection when two users change the same element.
Deliverables ***D2.1:*** Compiled .addin file for Revit**.** ***D2.2:*** JSON schema for architectural element snapshots.

WP 3: UI/UX & Conflict Resolution			
Start date: *Week 4* **End date:** *Week 8*			
Leader:	Yiğit Özhan	**Members involved:**	İbrahim Çaycı
Objectives: Build the visual interface within Revit that allows architects to manage versions. The focus is on making "Diffing" and "Merging" intuitive for non-technical users.			
Tasks: ***Task 3.1*** **History Pane:** Create the HistoryPane.xaml to display a list of previous commits and authors. ***Task 3.2*** **Visual Diff UI:** Develop the DiffMergePane to highlight changes directly in the Revit 3D viewport using colors (Red/Green). ***Task 3.3*** **Conflict Dialogs:** Implement DiffSelectDialog.cs for manual selection when two users change the same element.			
Deliverables ***D3.1:*** WPF-based Version Control Dashboard. ***D3.2:*** Viewport Overlay System for visual change tracking.			

4.4 Ensuring Proper Teamwork

The way you plan/method to establish teamwork, can be described. You can have roles and responsibilities matrix or some other plans to ensure teamwork.

Task / Deliverable	Tuna	Moin	Yiğit	İbrahim	Ömer
API Endpoint Security	A	R	I	R	I

Element Extraction Logic	C	A	I	R	R
Conflict Resolution UI	I	C	A	I	R
Database Migration	A	I	I	I	I
Project Documentation	R	R	R	R	R

Key: A=Accountable, **R**=Responsible, **C**=Consulted, **I**=Informed

To ensure efficient project management, we investigated various collaboration tools, including Jira and Trello. We ultimately selected GitHub as our primary management tool due to its seamless integration with our codebase, allowing us to link tasks directly to pull requests. For real-time communication, the team utilizes Zoom. To ensure proper teamwork and equal contribution, we will rely on the objective data recorded in our project board and version control history. The commit logs and task completion status will serve as the primary evidence of individual performance and shared leadership throughout the development process.

4.5 Ethics and Professional Responsibilities

Developing a VCS for the built environment carries unique ethical obligations regarding safety, transparency, and data ownership.

- **Algorithmic Transparency (The Diff Engine):** Our diff_engine.py is designed to be deterministic. We have an ethical duty to ensure that "Conflicts" are never resolved automatically by the AI/Code in a way that compromises structural integrity; human intervention is always required for geometric clashes.
- **Data Verifiability:** By storing snapshots as JSON in storage.py, we create an immutable audit trail. In the event of a structural failure or legal dispute, the system provides an honest record of who modified a specific element and when.
- **Security & Privacy:** While the current build uses a demo JWT (auth.py), we acknowledge the professional responsibility to implement TLS and salted password hashing to protect proprietary "Trade Secret" architectural details from industrial espionage.

- **Accountability in Design**: The system discourages "anonymous" changes. Every Commit is tied to a user ID, fostering a culture of professional accountability.

4.6 Planning for New Knowledge and Learning Strategies

Transitioning from "File-Saving" to "Element-Committing" requires a structured change management plan.

Strategy 1: The "Visual Diff" Discovery

Architects are visual learners. Our primary learning strategy is Visual Feedback. By using the DiffMergePane.xaml, users can see deleted elements highlighted in red and new ones in green directly inside the Revit Viewport. This lowers the cognitive load of learning a new technical system.

Strategy 2: Learning Interface Driven Training

This strategy uses C# WPF UI and ElementExtractor to provide "just-in-time" learning.

- WPF Error Guidance: The PublishDialog and ApiClient provide real-time feedback. Instead of generic errors, the UI explains the ElementSnapshot logic (e.g., "Geometry hash mismatch detected"), teaching users the technical logic of the system while they work.
- Sandbox Simulation: Using the existing storage.py (which is currently in-memory), new users are given a "Practice Branch" to test Publish and Pull commands. This allows them to master the tool in a safe environment where data resets on restart.

Strategy 3: Knowledge Maintenance

- Internal Wiki: Documenting common ElementExtractor errors and how to handle "Geometry Hashing" discrepancies.
- Unit Test Education: Encouraging the team to write new test cases for diff_engine.py when new Revit categories (like HVAC or Plumbing) are added to the scope.

5 Glossary

AES-256 (Advanced Encryption Standard): A symmetric encryption algorithm that uses a 256-bit key to secure data. In this project, it is used to encrypt RWT files stored on the server to protect intellectual property.

BIM (Building Information Modeling): A process involving the generation and management of digital representations of physical and functional characteristics of places. CollabHub is a tool designed specifically to manage BIM data.

Clash Detection: An automated process in architectural software that identifies where two building elements (e.g., a pipe and a wall) physically overlap or interfere with one another.

CORS (Cross-Origin Resource Sharing): A security feature that allows the Revit Plugin (client) to make requests to the FastAPI backend domain.

Diffing (Differential): The process of comparing two versions of a file or object to identify changes. In CollabHub, this refers specifically to identifying geometric or parameter changes in architectural elements.

ElementSnapshot: A structured JSON object defined in this project that represents the state of a single Revit element (geometry and parameters) at a specific point in time, used for comparison by the Diff Engine.

Geometry Hash: A computed alphanumeric string generated from the vertex data and dimensions of a 3D object. If the hash changes, the system knows the geometry has been modified.

IFC (Industry Foundation Classes): A platform-neutral, open file format specification that is not controlled by a single vendor or group of vendors. Used as a fallback method for data extraction in this project.

RACI Matrix: A responsibility assignment chart that maps out every task, milestone, or key decision to the roles of Responsible, Accountable, Consulted, and Informed.

Revit API: The Application Programming Interface provided by Autodesk. It allows the CollabHub plugin to programmatically read, extract, and modify 3D elements within an active Revit session.

Ribbon: The main toolbar interface in Autodesk Revit where the CollabHub plugin buttons (Push, Pull, Login) will be located.

RWT (.rwt): The proprietary file extension for Autodesk Revit project files, which contain the full architectural model and metadata.

6 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 Specifica tions,” 1998.

[4] Autodesk, "Revit API Developers Guide," Autodesk Knowledge Network, 2024. [Online]. Available: https://help.autodesk.com/view/RVT/2024/ENU/

[5] Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) — Information management using building information modelling — Part 1: Concepts and principles, ISO 19650-1:2018, Dec. 2018.

[6] Object Management Group, "Unified Modeling Language (UML) Specification Version 2.5.1," Dec. 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/PDF

[7] 3D Repo Ltd., “3D Repo — BIM Collaboration Platform,” 2025. [Online]. Available:

https://3drepo.com/

[8] Autodesk, “Autodesk BIM Collaborate,” 2025. [Online]. Available:

https://www.autodesk.com/products/bim-collaborate/overview