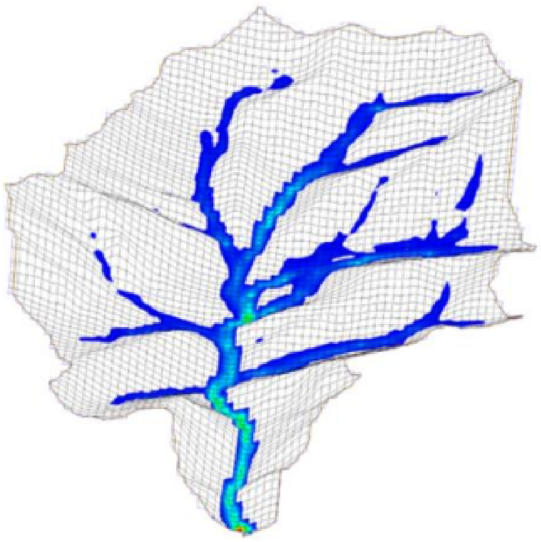
CSCI 370 Final Report

Water Walkers

Umberto Gherardi
Dylan Nichols
Joe Archibold
Caden Swartz

Revised November 24, 2024



CSCI 370 Fall 2024

Mr. Caleb Bartel

Table 1: Revision History

Revision	Date	Comments
New	9/1/2024	Completed Sections: I. Introduction II. Functional Requirements III. Non-Functional Requirements IV. Risks V. Definition of Done XIII. Team Profile References Appendix A – Key Terms
Rev – 2	9/15/2024	Completed Sections: VI. System Architecture
Rev – 3	10/18/2024	Completed Sections: VII. Software Test and Quality VIII. Ethical Considerations
Rev – 4	11/8/2024	Completed Sections: IX. Project Completion Status X. Future Work XI. Lessons Learned
Rev – 5	11/19/2024	Completed Sections: XII. Acknowledgements Updated Sections: XIII. Team Profile
Rev – 6	11/24/2024	Updated Sections: Incorporated Final Report Feedback into All Sections

Table of Contents

I. Introduction

Effective water management is vital to our modern society, particularly in areas of relative water scarcity like the western United States. Water accumulates in reservoirs and flows through various sources, such as rivers and creeks. This is especially relevant in Colorado, where much of the water supply comes from snow runoff in the mountains. Because of the complexity related to the sources of water and our limited supply, water management is a difficult task. In order to facilitate water management, Walker Water is developing software to visualize the geographic traversal of water over time, referred to as the Walker Water Irrigation Management System (WWIMS). The set of tools provided by WWIMS allows for efficient water supply administration by providing a thorough understanding of the Front Range's water sources. WWIMS provides a detailed map of the Front Range's streams and reservoirs. Flow rate and water level data are obtained from sensors and displayed in the WWIMS UI. Another notable feature of WWIMS is its collection of sensor data over time, allowing the application to provide historical data charts for a stream or ditch of interest.

As a team, our task is to enhance software previously developed by another Field Session team which simulates snow runoff from mountains and is known as the Snow Runoff Simulation (SRS). Before our involvement, the SRS could already fully simulate snow runoff in the front range in fairly high detail. However, some features have not been implemented – for example, rainfall is not being included in the simulation, and many parameters are not easily configurable.

The simulations produced by the SRS system must be easily validated and open to accuracy adjustments. Thus, the team will implement tools for users to compare the simulated water data (generated by the SRS) to the corresponding sensor values observed over a similar timeframe. Furthermore, the application toolset must allow users to adjust simulation parameters such as runoff coefficients to improve the accuracy of the simulations. Additional factors will also need to be added to the SRS to increase its realism, including precipitation (rain) data. A final, lower priority goal for the team is to integrate the SRS software into WWIMS, as the two applications are currently separate. The improvements made to the SRS may allow for water supply predictions to be generated based on snowmelt levels, thus providing another tool for water administration. In doing so, we hope to contribute to more effective water management for our community and Colorado at large.

II. Functional Requirements

For our enhancements to deliver the most value to hydrologists and hydrogeologists, we provided a method for easily tuning and rerunning simulations, incorporated more natural elements into the simulation via precipitation data, and designed the system to be integrated into WWIMS. The specific functionality for these enhancements included the following:

- Developing UI elements in the SRS for users to tweak individual simulation variables, such as the runoff coefficient, to help close the gap between simulation and real-world results.
- A comparison interface between water level SRS results and the corresponding sensor data from WWIMS for a basin of interest.
- Utilizing real-time rain data from weather stations near the basin of interest to further increase the accuracy of the SRS.
- A user interface for fetching API data and running the SRS with that data.

III. Non-Functional Requirements

This project's nonfunctional requirements are used to support the aforementioned functional requirements by providing details that constrain the project's implementation and design. A few of the most notable nonfunctional requirements are listed below.

- The application must be designed with the intention of ultimately running within WWIMS.
- Updates to the SRS code will be written in the C++ programming language.
- Under the circumstances that SRS integration is feasible, the integration must be written in C# to align with WWIMS.
- Code must be published to a private GitHub repository which the client will be able to access.
- Final code and outputs will be delivered to the client in a zipped folder.
- The use of various libraries is permissible if they support the development of the SRS code.
- Proprietary libraries or other paid pieces of supportive software should not be downloaded or used without prior client approval.
- SRS simulations should be performed efficiently to allow for timely data analysis.
- The software must remain stable and crash-resistant when running in a Microsoft Windows 10 environment.
- SRS results must be of the same format or translatable into the same format as the corresponding real world data to ensure that results are comparable.

IV. Risks

While working on this project, it’s crucial to understand the potential risks associated with our work. The two main categories of risks we have considered are technology and skill based. Technology risks are those which can disrupt a business and issues that can arise due to our implementations. Skill risks are gaps in our team's knowledge that may lead to future problems.

- Technology Risks
    - While improving the SRS, we could have made a change that breaks other parts of the SRS.
    - There could have been issues with setting up API access to the various sensors the application is pulling data from.
- Skill Risks
    - Since we were working entirely on existing codebases, the team needed to spend time understanding the different parts before we began making changes or additions. If we didn’t take this time, we could have potentially modified or removed code that we thought was wrong, breaking certain parts of the app.
    - Only two team members had experience with C# prior to this project, our other team members needed to spend time learning the language before they could make contributions to the app.

There is also one final risk associated with the use of the SRS and its impact in future decision making. Since the SRS may be used to predict future water availability, inaccurate predictions could lead to poor decisions regarding water access. It will be important for our team to keep this primary use case in mind as we make new additions to the simulation.

V. Definition of Done

The MVP for this project is to have a closed feedback loop within SRS using sensor data. The main features to fulfill that MVP are as follows:

- The application will need to run a simulation from a given map of water levels.
- The application will need to fetch sensor data from the various APIs provided.
- The application will need to provide some method for a user to compare simulation data with sensor data.
- A user must be able to select and tune the runoff coefficient for a desired region.
- The application will allow for the user to easily run a simulation with specified parameters.

The client will be given the opportunity to test the application, and especially the tuning process, to ensure it works as desired. The client will have access to our work via a GitHub repository. Once the project is complete, the final code and outputs will be delivered in a zipped folder.

VI. System Architecture

Technical Issues

Our team has experienced a few technical issues while exploring the codebase thus far: chief among them are the build difficulties encountered when using the build tool. In its current state, the build tool developed by prior Field Session teams isn't user-friendly and provides insufficient documentation for the team to easily create a successful build. It's easy to make a mistake when choosing build options and there is very little indication of whether the user selections are valid. In addition to the existing build problems, a few potential blockers also stand in the way, namely WWIMS integration and the reliance on multiple sources of truth for sensor information.

As a stretch goal, the client has expressed interest in integrating the simulation display into WWIMS. The SRS is being run in C++ and the display component is being run in Python. Our client also has a MySQL database that collects current sensor data over a set interval and does not contain historical stream and reservoir sensor data. Combining previous sensor data collected from the sensor APIs with current data provided by the MySQL database may be an issue for our team. The database contains infrequent sensor readings. If the original data is too old, the application may not be able to make accurate comparisons between simulation outputs and their corresponding sensor measurements.

High-Level User Flow

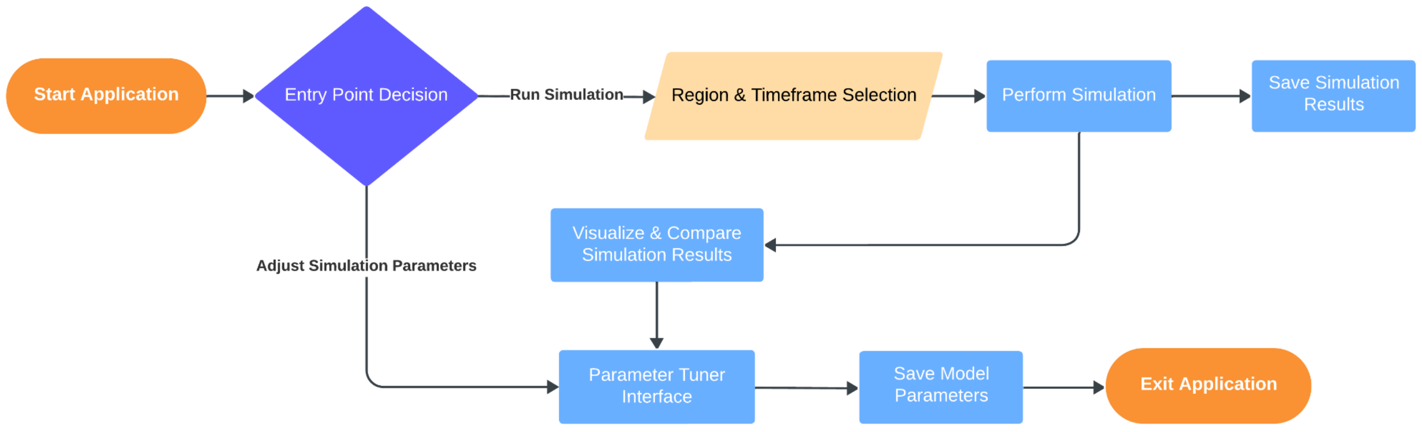


Figure 1 - Application User Flow

The main user interaction flow of our application is shown in Figure 1 above. Upon entering the application, the user will have the choice to run the simulation or update its parameters. If they choose to run the simulation, they will be prompted for the timeframe and region they want to observe, and the program will prep and run the simulation. The user will then be able to visualize the simulation and compare its results with the measured sensor data. From here, the tuner interface will appear, allowing the user to modify parameters to achieve higher simulation accuracy. Finally, the user can save the potentially modified simulation parameters and exit the application.

Detailed Information Flow

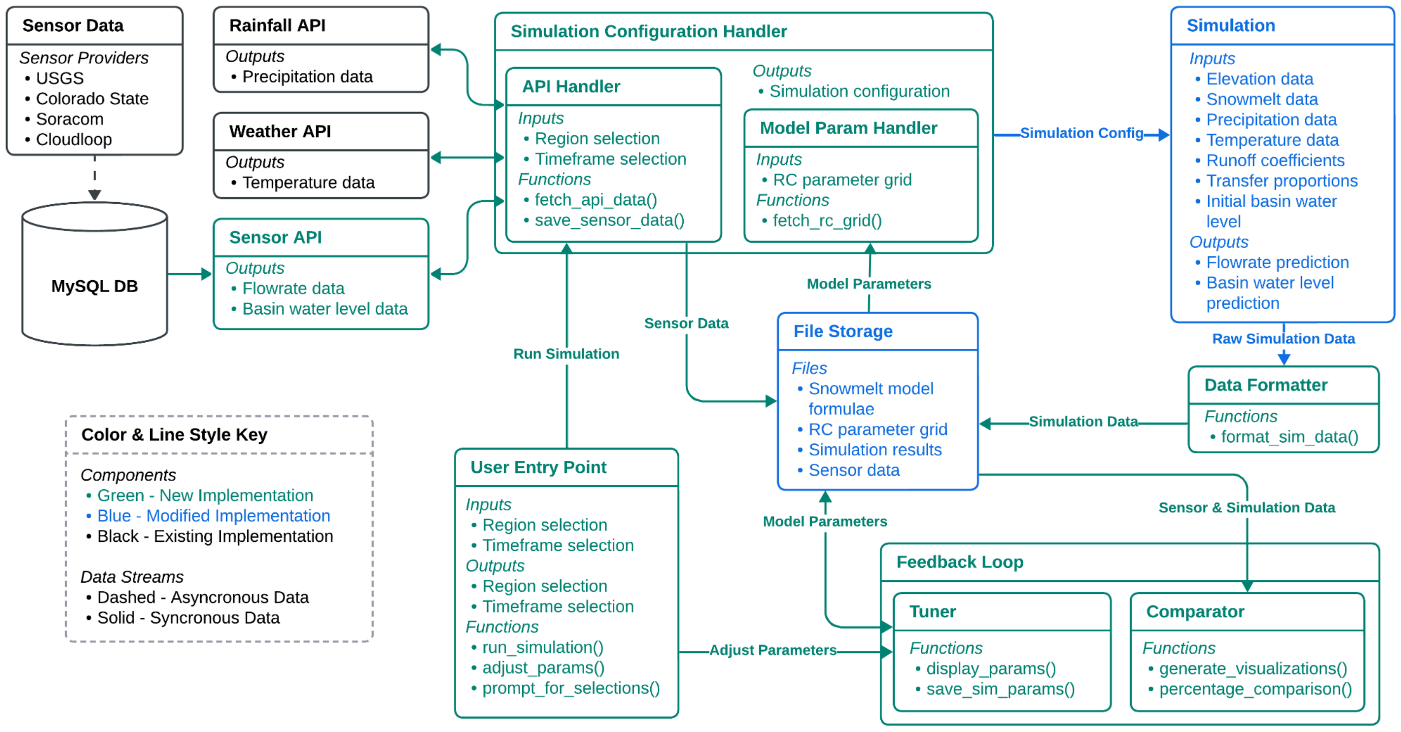


Figure 2 - Detailed Information Flow

User Entry Point

This module marks the start of the application for the user, where they can choose to run the simulation or adjust parameters. If running the application, the user will be prompted for a region and time frame selection for the simulation.

Simulation Configuration Handler

The simulation configuration module is responsible for making synchronous API requests to each of the data endpoints and loading the necessary model parameters (e.g., the snowmelt model formulae and RC parameter grid) from the filesystem. Once the parameters and data have been obtained, the simulation configuration handler will aggregate and standardize this information into a format that will be used to execute the C++ simulation.

Sensor API

This API collects stream and reservoir data from several sensor providers for the purpose of comparison to simulation results. Additionally, the API makes synchronous requests to the MySQL database, where asynchronous sensor data is stored.

File Storage

Various input and output files from the system will be stored locally. As there is no database within this system, this is the only persistent storage which the system is writing to. The file storage will contain several pre existing file format types, including those used to represent elevation data, reservoir and sensor data, climate data, and the simulation output. File formats new to our system will be the RC parameter grid to adjust runoff coefficients spatially.

Simulation

Most of the simulation will not be altered as there already exists a functioning snow runoff simulation with a solid structure. However, some improvements will be made. Firstly, the simulation will be modified to use a geographic, grid-based specification for each cell's runoff coefficient. This allows for full customization of the runoff coefficient in every grid cell of the simulation. Then, the simulation will be improved by factoring in rain data to the calculation of runoff, rather than only simulating water runoff from snowmelt.

Data Formatter

The data formatter component will format the simulation data to be compared with sensor data. After standardization, both types of data are fed into the comparator.

Feedback Loop

The feedback loop is the interactive portion of this system. Feedback loop functionalities include the displaying of simulation results and sensor data in the comparator, and the ability to adjust spatial RC parameters in the tuner. The comparator will display simulated versus measured (by sensors) fill level and flow rate for each reservoir/stream. The simulation data comes from the data formatter module. The measured sensor data comes from the file storage, as this data would have been fetched by the sensor API previously. The tuner component will display a large grid of RC values mapped to the geographic coordinates for each cell in the grid. From this display, the user will be able to modify cells' RC parameters to make the simulation more accurate in the future. This modified RC parameter grid could be saved back to the file system.

Component Communication

To summarize how each component communicates with each other, the simulation configuration handler component fetches all API data, which is immediately sent to the simulation. The simulation also gets data from the tuner component to run the simulation. Finally, the simulation's output is passed to the comparator to display the results.

Simulation Improvements

This new structure also helps us to improve the simulation's accuracy. In particular, we are fetching hourly geographical rain data and adding this precipitation to the simulation in the same way that snowmelt adds water to the simulation.

VII. Software Test and Quality

Our testing plan includes a variety of unit, integration, usability, and functionality tests. These tests, including their expected and actual results, are listed in detail in the table below.

Test Name	Category	Env	Setup	Action	Expected Result	Actual Result
Simulation Parameter Loading	Unit Test	Dev	Premade elevation data file w/ set RC params	Simulation parse and load params for each cell	Each cell has the specified parameter value	Passing
Simulation Parser	Unit Test	Dev	Simulation file in .dat format that aligns with the expected configuration used in the display.py file	Parse the simulation output file in C#	Ensure the data extracted is the same as the data extracted in the Python script used to display the simulation	Passing
Tuning Grid Testing	Unit Test	Dev	Various premade tuning grids and coordinate lists (elevation data file) provided	Tuning grid and coordinates lists parsed and rc parameter assigned for each coordinate based on the tuning grid	RC parameters are correct for each coordinate within the grid. Coordinates outside the domain of the grid remain with unset RC parameters	Passing
Comparator Testing	Usability Test	Dev	Configure the comparator component with simulation and sensor data	Walk through a typical user flow to ensure the comparator results are presented in an insightful way for the client	The client is able to understand the data being displayed, and can take appropriate action to modify the simulation for closer future approximations	Comparator displays sensor data, simulation data, and reservoir capacities in a bar chart format for easy comparison
Simulation Config Handler	Integration Test/ Functional Test	Dev	Create a sample input from the user entry point to send to the API handler	Call API handler with the sample input	API handler should properly use inputs from user entry point, calling the APIs at the correct time based on the interval provided	Passing
Tuner Integration Testing	Integration Test/ Functional Test	Dev	Provide existing RC param grid format	User uses a tuner to modify the parameter grid. Simulation is initialized	Simulation uses modified parameters that the user specified via the tuner	Passing
Feedback Loop E2E	Integration Test/ Functional Test/ Usability Test	Dev	Feedback loop has been configured	Walk through the entire feedback loop user interface	The feedback loop is intuitive for the user, and contains all the features necessary to understand and adjust the simulation	Passing

Table 1 - Software Quality Plan

Due to the scope of our project, we are unable to perform traditional user acceptance testing. However, the team is continuously receiving feedback from the client throughout the development process; feedback is obtained through live functionality demonstrations during bi-weekly client meetings. Furthermore, we are employing additional quality assurance through our code review process. All changes to the software are reviewed by another team member with a pull request to ensure that code quality meets the standards of the team.

VIII. Project Ethical Considerations

Some particularly relevant ACM and IEEE Principles to consider are highlighted below:

- **ACM 1.1** - *Contribute to society and human well-being, acknowledging that all people are stakeholders in computing.*
    - Ultimately, our software aims to benefit the general public by providing knowledge regarding the flow of water. As a team, we should keep this mission in mind throughout the development process.
- **ACM 2.5** - *Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of their risks.*
    - Our product will be used to inform the actions of water managers. Specifically, one of the reach-goals for our project is to alert water managers when reservoirs and ditches are predicted to overflow: this alert will be produced based on the results of a programmatic simulation. It is therefore paramount that our simulation is able to provide accurate and timely predictions to ensure water managers have adequate time to address water distribution issues.
- **ACM 3.6** - *Use care when modifying or retiring systems.*
    - Most of our work this semester required building on previous field sessions' work, which required modifying or removing certain parts of their code. We needed to ensure that these modifications were warranted and necessary before they occurred.
- **ACM 3.7** - *Recognize and take special care of systems that become integrated into the infrastructure of society.*
    - While this software is still in its early stages, the end-goal for this product is to be a tool used by water administrators.
- **IEEE 3.07** - *Strive to fully understand the specifications for software on which they work.*
    - Our regular meetings and product demos with our client ensure that our software design decisions are in line with our client's needs and objectives for the project. If we fail to observe this principle, our product will not meet the requirements of our client nor our end users.

Michael Davis tests are a pertinent aspect of ensuring software meets certain ethical standards. Two of these tests are discussed below:

- **Mirror Test**: *Would I feel proud of myself when I look into the mirror?*
    - One of the reasons why our group was interested in this particular project was because we recognized the potential good it can do for society. As our client expressed to us, water management is a generational issue that impacts the lives of millions. Our group believes that contributing work to a system that alleviates the burden on water managers and permits them to make productive decisions is a worthwhile pursuit. We are proud that we have the opportunity to tackle such a complex and universal issue.
- **Professional Test**: *What does the Software Engineering Code of Ethics say about this problem?*
    - The Software Engineering Code of Ethics calls for work to be aligned with public interests, maintains that the work must be of a high quality, and finds that the work should be conducted in a collaborative manner. It's clear that the values and intentions behind this project will aid the public; should the team maintain a high quality of software and practice effective communication, our solution will pass the Professional Test.

Finally, there are significant concerns if our software is completed without proper quality assurance. In particular, the comparator component is expected to provide accurate information on the state of water in reservoirs. If this information is faulty, users could make misguided decisions. Additionally, it is critical that the accuracy of the snowmelt simulation is made clear to users, and that users have the ability to increase this accuracy. This ensures that water administrators do not use the software in potentially unprofessional ways.

IX. Results

We are proud to have implemented all of the functional requirements outlined in this report. Our notable features and accomplishments for this project include:

- An interactive graph window comparing simulation results to the most recent actual measured reservoir water levels.
- A parameter tuner, where users can modify runoff coefficients for areas of the simulation quickly and easily, and pass these modifications into the simulation.
- A unified user interface containing the above two features as well as the ability to fetch all API data and run the simulation at the press of a button.
- Improvements to the simulation, including bug fixes and the addition of hourly rainfall data to the snowmelt runoff.

Images for the user interface that we have implemented are provided in Appendix B of this report.

There were also some features that our team planned on implementing, but fell short in some regard.

As mentioned in the detailed description of our application's data flow, there are two factors of interest when it comes to modeling snowmelt: water level and flow rate. Our application successfully provides a feedback loop tool for viewing and modifying the simulation's water level predictions. Our group initially assumed that the simulation modeled flow rate in some capacity. Unfortunately, this assumption was incorrect. The team was primarily focused on validating the accuracy of the simulation with their feedback loop work, and they lacked the capacity to implement an accurate and robust flow rate model. However, the team wanted to ensure that the flow rate was easy to implement if the desire arose in the future, which prompted them to make a few key design decisions, e.g., creating an abstract "comparison graph" class that can be instantiated to either compare reservoir water levels or stream flow rates.

The second piece of work that the team did not have time to implement was a fully-featured UI for the tuner. Based on our conversation with a hydrogeologist, the team believed it would be beneficial for a user to remain in the application while making changes to the tuner files. The team initially conceptualized this feature as embedding an Excel file within the WPF application, which they believed would give the user traditional file modification functionalities while still allowing for a more seamless user experience. However, a more robust UI would be ideal: regional areas could be selected by clicking and dragging over a geographical area, and these areas could have their parameters modified. The team determined that this feature would make the most sense to be implemented after the WPF application had been implemented into WWIMS.

The final feature that the team was unable to accomplish was incorporating their work into the WWIMS application. Although this was a stretch goal for the team given the scope that the client defined, it is recommended that future field session teams prioritize this work to provide maximum value to Walker Water.

As outlined in section VII, the team completed a variety of tests to verify the quality of the developed application. The first form of testing the team successfully completed was unit testing. While writing code, the team implemented several small tests to ensure the basic functionality for several different components was being met. The tests were successful, allowing the team to quickly catch minor bugs in their code before integrating these components into the larger application framework.

Additionally, the team found that providing mandatory code reviews before merging new features was extremely beneficial. On several occasions, the team was able to catch errors introduced from the different program setups each group member was using. This was especially important since future users may be running slightly different software, and ensuring that the code does not behave differently depending on those factors is crucial to the usefulness of the application.

After merging each individual component into one complete application, the team performed a final series of integration tests to verify that information was being properly passed between the components. To simplify these tests, the team created a configuration file that contained all the information being shared. Each component could then write its information to the file before navigating to the new component. Using this method, the integration tests proved to be successful. There were no issues of lost or incorrect information when switching from one component to another.

X. Future Work

Firstly, we did not have the time to add all potentially useful features to the user interface of our application. Some recommendations for future features are listed below. These features would also serve as good starting points for future teams to make quick, meaningful improvements:

- Not all parameters and files can be set within the user interface – some must be set by directly modifying the configuration file. A settings menu for the application could implement these features.
- The simulation has many flags and parameters that can be set, such as how many simulation frames should be skipped. These flags are currently set to a specific setting and cannot be modified by the user without modifying the code. Allowing the user to modify these parameters allows for more customization and flexibility in the simulation.
- The previous team created a Python application which plays a timelapse of the simulation: this timelapse cannot be accessed from our application's user interface. Adding this Python script to the application would make it easier to interact with as a user.

Our team's contributions were aimed towards determining the accuracy of the simulation. While we did add an extra accuracy factor, namely precipitation data, we did not introduce many new features that drastically increased the quality of the simulation results. Future field session teams should use our work to improve the accuracy of the simulation itself by adding more data factors and work closely with hydrogeologists to verify its accuracy. The goal of the simulation project was to give end users the power to run meaningful simulations, and thus, to be able to make more informed decisions about stream and reservoir water management based on the results. Adding further richness and accuracy to this service will aid in driving this goal forward.

Some particularly noteworthy improvement recommendations for the simulation have been highlighted below:

- Flow Rate – This was discussed more thoroughly in the previous section, but adding flow rate calculations to the simulation will provide another output metric which will be useful both for verifying accuracy of the simulation, and for providing water administrators with more information about water's behavior in the simulation.
- Sophisticated Handling of Reservoir Overflow – Firstly, it should be possible for reservoirs to have a current water volume greater than their capacity while they are overflowing. Secondly, knowing when a reservoir is overflowing would be highly useful to a water manager as this may signal that they should be taking more water out of this reservoir.
- Runoff Coefficient Handling – The simulation treats runoff coefficient as being water lost per unit of time, while the hydrogeological version of runoff coefficient is unitless. As such, there is not a clear way to translate between what the simulation and tuner uses, and how hydrogeologists understand runoff coefficients. So, while users can currently tweak RC parameters, this does not correspond with real runoff coefficients. Being able to somehow translate between these or use a different, potentially more relevant unit of measure could improve the clarity of the simulation from the user's perspective.

- Setup Speed - The process for fetching weather data and preparing the files for the simulation is currently very slow. The Herbie Python library our team used has built-in functionality that allows for multi-threaded API fetches, which could drastically improve the time it takes to initially prepare files before running the simulation.
- Simulation Speed – The current iteration of the simulation is, in general, very slow to run. Optimizing the simulation could significantly improve it. Our team has not substantially analyzed the code in terms of improving its speed. If a team were to do so in the future, the usability of the software could be substantially enhanced.
- Time Frames – There is currently little flexibility in the simulation for the time frame that the user selects. It would definitely be useful to have an option for the simulation to run at a higher level, over more time and with less precision. This could also improve the speed of the simulation by simulating less data points and providing more general estimates.
- Additional Tuning Parameters – Currently, only runoff coefficients can be adjusted by the tuner. It is very likely that there exist other parameters per simulated point that could be tuned by a user to make the simulation more accurate. For example, the client has identified factors such as vegetation density being influential to runoff.

In the previous section, we discussed how we could not meet our stretch goal of integrating the SRS and user interface into WWIMS. Accomplishing this in the future may be a significant task, but it will substantially increase the usability and functionality of the application. Doing so would mean that the SRS application will be able to communicate with and easily share data with the WWIMS application, which has a vast array of geographical and hydrological data. As such, integration with WWIMS should be a primary goal for future work on this project.

XI. Lessons Learned

As a team during this project, we engaged with aspects of software development that we weren't particularly familiar with before starting the project. Additionally, working for a client in the setting of a four-person team was a unique structure for us. Because of these challenging and exciting aspects of the project, we have learned multiple valuable lessons, including the following:

- An initial obstacle for the team was understanding and implementing the WPF MVVM design pattern. As only one of the team members had experience with this technology, there was a great deal of research done to ensure that code quality and maintainability was adhered to by all members of the team. We also found WPF documentation to be either difficult to find, or overly verbose and confusing in most cases, which made it difficult to make significant progress early on. These challenges were overcome by following a set of foundational WPF video tutorials, and by making use of modern and well-documented libraries, like LiveCharts2, to accomplish more complex pieces of functionality.

- A major source of volatility for us was our client relations. Ultimately, we feel fulfilled with our client relationship, but there were various roadblocks that came up throughout development. John is our main client and the head of Walker Water, but in early meetings we were able to speak with some of his colleagues. However, for various reasons, these colleagues stopped associating with Walker Water or were unable to meet with us. This created some inconsistency in the feedback we were receiving throughout the semester. Ultimately, this has taught us to be flexible in dealing with stakeholders. It is still up to our team to put out the best product we can with the feedback the clients can or can't provide us.

- In a similar vein, particularly as we are college students working in a professional setting, it can be easy at first to over-promise on meeting the requests clients make for a particular feature or improvement. However, we have learned that it is more valuable to both us and to the client to be realistic with what we can accomplish, and re-iterate the requirements we plan on meeting. From our understanding, clear expectations will lead to more satisfaction even if some client ideas need to be rejected or placed on hold.

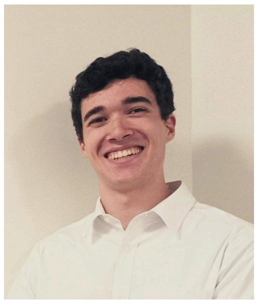
- A final challenge our team faced was picking up the project from previous field session groups. As more groups contribute to the project, it can become increasingly difficult to hand off the work and knowledge to new teams.

Our team encountered several cases where the existing code had strange errors and identified several instances where the documentation and inline comments failed to explain certain decisions and functionalities. This has taught our team the importance of clear documentation, and the role it plays in this project. If we want future teams to be able to carry on work for this project, they will need to understand the decisions and contributions our team made.

XII. Acknowledgments

The team would firstly like to thank Trevor Hirsche, the team's hydrogeologist contact. Early on in the semester, Trevor helped to define the scope, requirements, and underlying purpose of the project. The team would also like to express their immense appreciation to their client, John Walker, for his continued support and advice throughout the project development. John provided access to several data sources that were necessary for the success of the project, including a personal MySQL database, and worked closely with the team to ensure all members understood the basic physics of snowmelt and the value behind the application. John's hydrology expertise allowed the team to move forward with confidence when faced with uncertainty. Finally, the team would like to thank their advisor, Caleb, for helping to conduct sprint retrospectives and ensure the team remained on-track for every class deliverable.

XIII. Team Profile



Umberto Gherardi
Computer Science + Computer Engineering
Hometown: Fort Collins, CO
Work Experience: Software Engineering Intern, Technology Solutions Consulting Intern, Undergraduate Researcher, CSCI403 Teaching Assistant
Interests: Hiking, biking, misc. outdoor activities, spending time with family and friends

Umberto Gherardi led the design of the C# WPF UI components, wrote parsers for the simulation output and sensor data files, and worked to integrate each of the individual components into a cohesive application. Umberto was also responsible for assisting with the development of the MVVM project structure.

Dylan Nichols
Computer Science + Computer Engineering
Hometown: Las Vegas, NV
Work Experience: Software Engineering Intern
Interests: Competitive Dance Games, Game Development, Music Production

Dylan Nichols was responsible for leading the development of the MVVM structure inside of C# WPF and served as a technical advisor to ensure scalable design from other team members. Dylan also assisted with frontend development and UI implementation.

Caden Swartz
Computer Science (General)
Hometown: Littleton, CO
Work Experience: Software Engineering Intern
Interests: Listening to music, hiking, gaming, reading

Caden Swartz led the design for simulation tuning component, added rainfall to the simulation, and also provided some finishing touches and quality-of-life features to the WPF application.

Joe Archibold
Computer Science (Data Science)
Hometown: Longmont, CO
Work Experience: Data Science Intern, Data/Software Engineering Intern
Interests: Video games, board games, hiking, and disc golf.

Joe Archibold primarily worked on the simulation configuration handler and implementing the API access to the database and weather systems. He also helped add minor changes to the simulation to fix inconsistencies.

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	**Definition**
Runoff Coefficient	A representation of the amount of rainfall or snowmelt that ends up in ditches, expressed as a decimal value between 0 and 1.
WWIMS	Walker Water Irrigation Management System (Our client's existing application)
SRS	Snow Runoff Simulation (The software developed by previous field session teams and improved upon by our team)
WPF	Windows Presentation Foundation (A C# .NET format used to create Windows Desktop Applications)

Appendix B – UI Screenshots

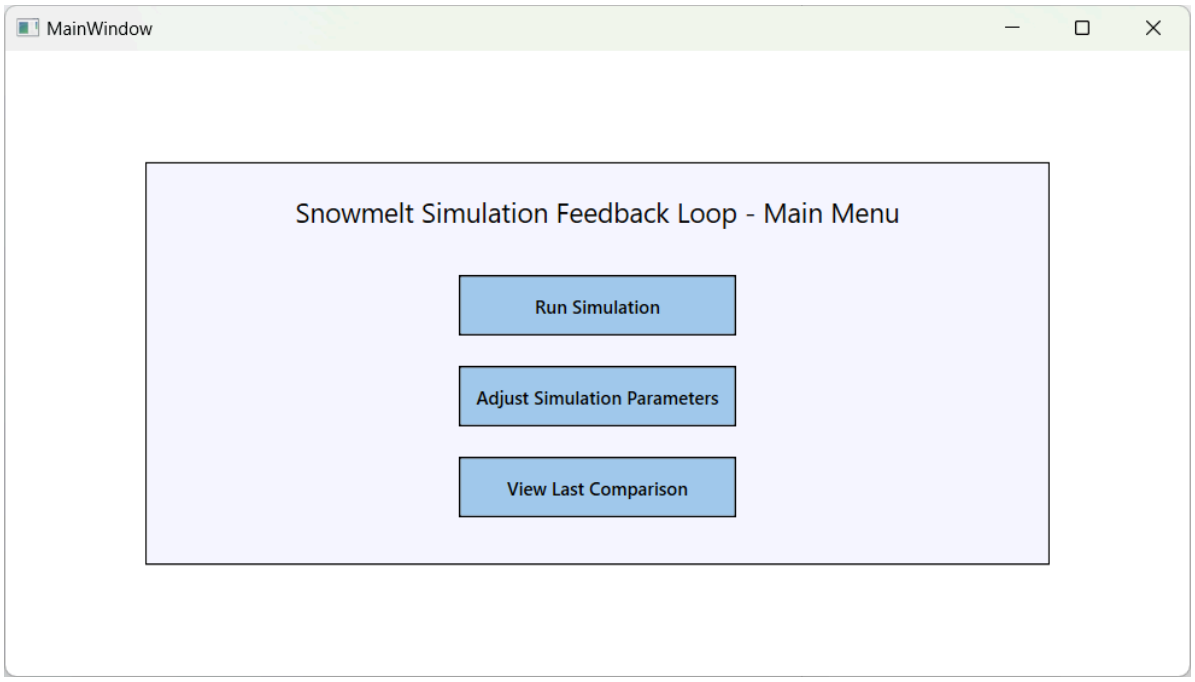

MainWindow
Snowmelt Simulation Feedback Loop - Main Menu
Run Simulation
Adjust Simulation Parameters
View Last Comparison

Figure 3 - Feedback Loop Main Menu View

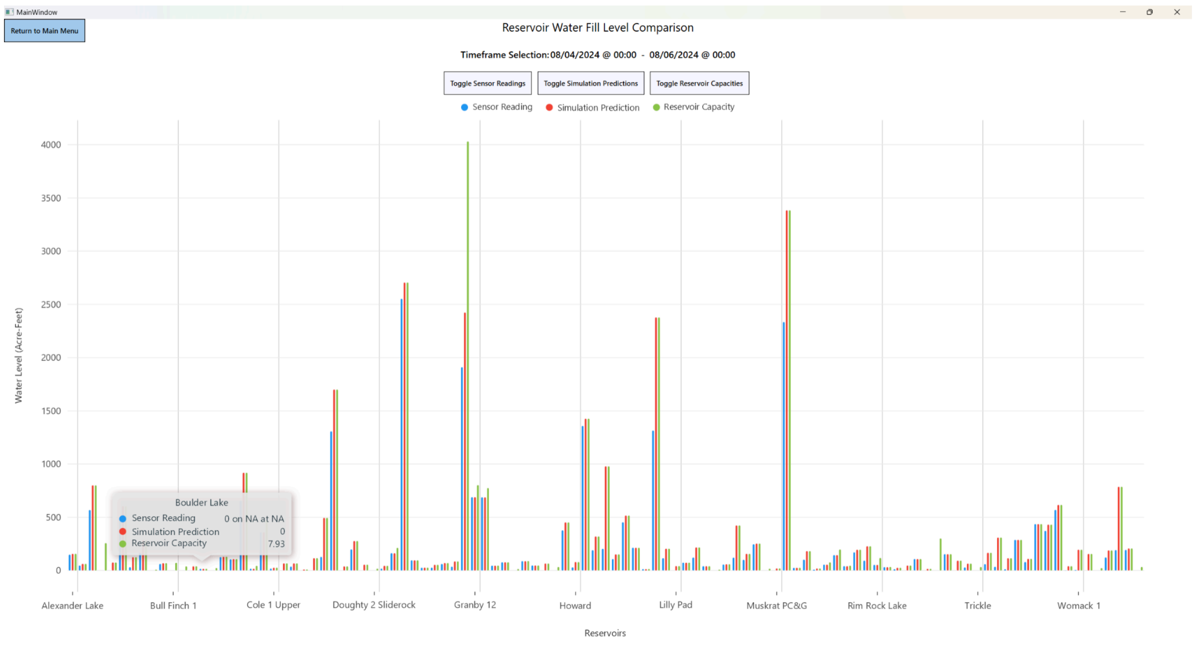


Figure 4 - Comparator View

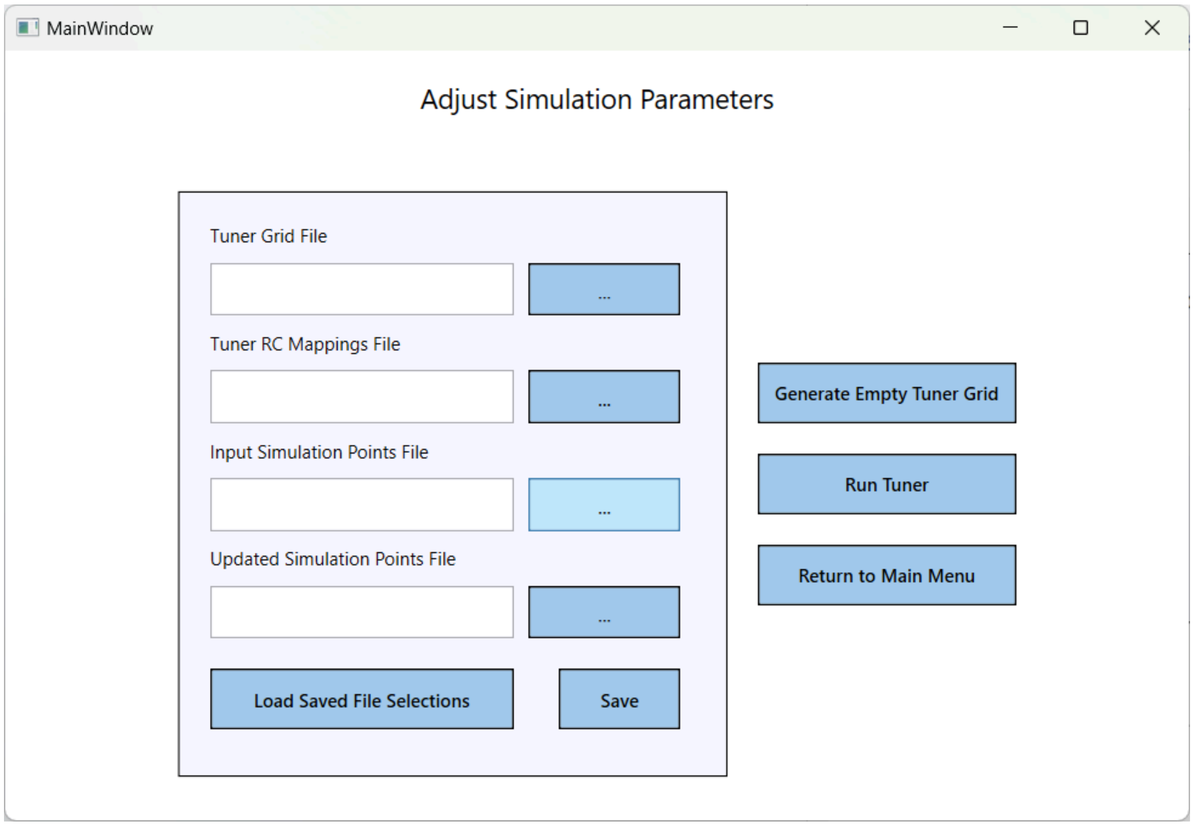
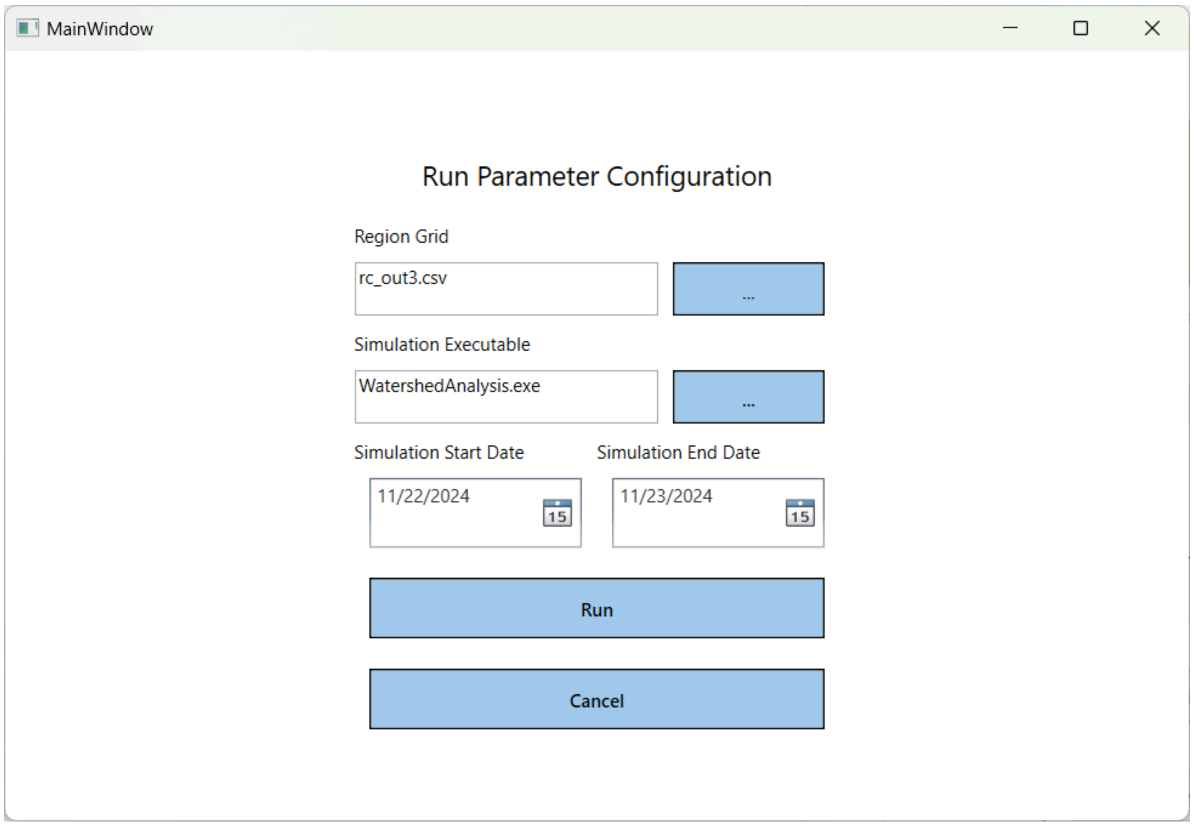

MainWindow
Adjust Simulation Parameters
Tuner Grid File
...
Tuner RC Mappings File
...
Input Simulation Points File
...
Updated Simulation Points File
...
Load Saved File Selections
Save
Generate Empty Tuner Grid
Run Tuner
Return to Main Menu

Figure 5 - Simulation Parameter Adjustment View


MainWindow

Run Parameter Configuration

Region Grid: rc_out3.csv

Simulation Executable: WatershedAnalysis.exe

Simulation Start Date: 11/22/2024

Simulation End Date: 11/23/2024

Figure 6 - Run Parameter Configuration View