Sharing Framework Final Report

Bobby Kearns
Ian Saad
Chris Spalding
Greg Finch
Wesley Stedman

December 20, 2013

1 Project Summary

The sharing framework consists of the behind the scenes server work of the Edith project, a 2D web-based computer science teaching tool. In short, our module acts as the middleman between other modules and the database, providing the ability to validate a user, save data, and load data. The actual user of Edith will not directly interact with us, nor will he/she be aware of our existence without knowledge of web oriented operations. Other modules could be considered users of our system as they will be directly calling our functions. However in considering the involvement of the actual user with our system it breaks down into the following key functions. If the user wants to register an account to begin working or wants to login to an existing account he/she enters info on that page and clicks register. Once that button is pressed, we make it happen via validation in our database. If the user wants to save a project he/she's been working on and presses save to do so, it's our module that puts that project into the database. If the user wants to load an existing project he/she presses save and it's the sharing framework that makes pulls the project from the database and gives it to the main page. In interacting with other groups we provide similar functionality, if a group has data to save to us, say an object made by the user, that module calls a save function and inserts the object into the proper table. Essentially our framework is just a closet that other teams store and take from.

2 Development Procedure

While our group didn't follow any one particular software model, we used an iterative form of development consisting of creating checkpoints and before beginning each step, planning out how best to go about implementing and testing once implemented. We went about this by sitting down as a group and discussing what our module would have to look like and how best to break the product apart. Testing wasn't really planned out, as we just continually tested as functions were made. This process was repeated for each checkpoint and each testing round.

Initially we began work on how to set up a database for us to access locally. We used WAMP/MAMP on our laptops to simulate the environment we would eventually need for the system. This involved learning both the WAMP framework and php in order to properly communicate with the server.

Doing so was a challenging step for most of us luckily Bobby had some experience in the matter and helped us all move forward much quicker than we would have otherwise. Once we had a server to talk to, we began working on all our key functions; save, login, load, and share. Save and login were the first targets, as they were simple in concept and could be demonstrated for Integration 1. These were completed on track primarily by Bobby and Ian, and we then progressed to loading and integrating with other teams. Greg and Chris provided the load function for the other teams, and it was at this time we realized our functions had to differ depending on which groups were calling them. We spent a lot of time at this point understanding the structure of the project as a whole and figuring out how teams would need our functions to look/how we were going to give them data. This was a challenge both in concept, due to grasping a larger complexity, and in implementation, due to the need to figure out how AJAX worked. Ajax is required for html to communicate with a server/utilize functions. At this time we were also extending our server from laptops to a local server; Wesley helped out greatly with this part by setting up his desktop at home to work with Edith. After this was set up all of our work was refactoring and syncing our functions with other groups, making sure the data was going into the proper tables, and assisting other groups with integrating to our network.

Testing in general was done rapidly; we could easily tell when a function worked or didn't work as data was being stored in a database since WAMP allows us to visually look at entries in the database, tables, and everything else involved with the DB and gets updated instantly. A lot of debugging had to be done when integration was going on due to the constant changing of functions and tables that occurred as groups were finishing up/restructuring their module. The process as a whole, not just testing would have gone much smoother had inter-group communication been more existent through all stages of the project. Most groups, ours included, planned out how their module would work and focused on getting that done without grasping the scope of integration. Our planned out process involved too, too large milestones that while reached kept us from really following a plan. The overall development was planned out, however each milestone felt as if we just went right into the coding process with the next checkpoint in mind. More frequent, smaller milestones would have aided development greatly. Gluing the modules together was also a very difficult task given the time and communication. Additionally, writing reports was in and of itself helpful, but perhaps enforced group isolation and could have been tailored in a way to have us to report on other groups or how our groups should integrate. Forced or more incentivised communication would have led to a smoother development process. Communicating with and organizing a team of 5 was challenging enough for us students who, for the most part, hadn't ever done any project near this scope or involving anywhere near the amount of people involved. This caused groups to isolate and never really meet with others until it was demanded of us through the later Integrations. There definitely was a vibe amongst the modules of, "well our stuff works, they just have to use it" and no real attempts to make that happen. It was only at the end that this idea was dropped. Had inter-group interaction been pressed more strongly, perhaps through labs where every team is actually working on the system together, we felt that Edith would have been developed more smoothly and effectively.

3 Requirements Evaluation

Our original requirements, as stated in the requirements document we wrote, were as follows.

Logging On

Actor: Account Holder
Flow: First the account holder must enter their username and password, and then click 'login' or an equivalent button. Server authenticates the account holder's information, allowing them to view, make, and share their projects.
Alternatives: If the user doesn't have an account then the system will prompt the user to make an account. If the user enters the wrong username/password the system will tell user and prompt them to enter username/password again.
Post Conditions: user is logged on and can access their profile and upload/save/share docs.

Storing Animations

Actor: Account Holder
Preconditions: Must be logged on.
Flow: User names animation and then selects save.
Alternatives: if the animation has the same name as another animation, system prompts user "overwrite old file or rename?"

Post Conditions: the animation is saved on the server under their username.

Sharing Animations

Actor: Sharer
Preconditions/Assumptions: User must be logged on and have a saved animation.
Flow: first the Sharer logs on, then select share, then select animation to share, and finally the sharee receives URL to the animation page. The sharee cannot edit the project that has been shared.
Alternatives: If the sharer is not logged on the system will prompt him/her to log on. If the user has no animations to share the system will throw an error mesage telling the user "No saved projects to share."
PostConditions: return a link to animation the sharee can go to in their browser to view the animation.

Loading From The Database

Actor: Account Holder
Preconditions: The user must have an account and be logged in, have a network connection of some sort, and they must have a saved animation or project to load.
Flow: The user must log on and then select which animation or project to load.
Alternatives: If the user is not logged in, prompt them to do so. If there is no animations to load prompt them to start a new project.
Postconditions: Display the animation and all the tools needed to share or edit.

Nonfunctional Requirements

Network Connection

A strong network connection between client and server is required for the functions to properly occur.

What Systems We Will Support

This program will not support mobile devices.

We were able to meet all of our functional and non-functional requirements except for the sharing requirement. The failure to meet this requirement, due to the fact that it shares its name with our team name, makes it sound much more dire than it is in actuality. In reality, the saving and loading functionality, which were both successfully implemented, are far more fundamental to Edith than is sharing projects between users. Furthermore, we added another functional requirement during the semester, registering a user, that we were also able to implement.

Because of the straight-forward nature of our module (we knew from the beginning that we needed to make a database that would allow other modules to communicate with one another), most our requirements were very well defined in the beginning of the project, and were thus very easy to follow and it had made it quite clear throughout the project what we needed to implement and what had been successfully implemented. Along with this, the sharing functionality that we had in mind wasn't well formulated until far too late in the semester. This is because, in order to have a clear vision of how the sharing functionality would work, we needed to see how the save and load functionality worked when fully implemented. Along with this, in order to create the sharing functionality we had to work closely with Story Creator, and in order for both of our teams to be far enough on the project to work on the sharing functionality we need to wait until around integration 2. This clarity of our initial requirements made them very effective, and made it more simple for us to implement our system throughout the semester.

One of the beneficial side effects of writing well defined requirements in the beginning of the project was that our requirements did not change a lot in the course of the implementation of Edith. During the course of setting up the logging in functionality we realized that there would be no way for a user to log in to the system if we didn't support some way of registering new users to the system. This realization led to the only major addition to our requirements, which was a register user functionality. Because logging in and registering are fairly similar functions, this addition was useful for the project's implementation as a whole, but very minor in terms of difficulty. The other major change we made to our functional requirements was that we were not able to implement a sharing function that would allow users to showcase their projects with one another.

There were a few major reasons for this shortcoming. The first was a lack of communication between our module and the Story Creator team. We realized towards the end of the project that we needed Story Creator to make a way to allow the Sharing Framework to grab the canvas on which the animation is made and pass that from user to user via URL. When we asked them if this was possible they were very willing to help us work on it, but as we began to help them, the Sharing Framework module decided that it would be highly unlikely for us to be able to have a good enough implementation of the sharing functionality by the time the final implementation was due. So the main reason for us not developing the sharing method was that our module decided it would be too primitive to be worthwhile. The other major reason was that we simply did not have enough time to implement everything we wanted to. This was due to everyone's course load, and the fact that we were really only working on this project for 10 weeks. We put sharing low on the priority list early on in the project, and by the time it surfaced at the top of Sharing Framework's to do list, we realized that it simply wouldn't be possible. We prioritized the sharing functionality so low because, as mentioned earlier, we thought that the most important functionalities of our module were the save and load functions. Connecting these to a database that was allowed to be accessed by other groups easily was functionality that would be visible immediately to developers, which is why we thought these were the most important functions. While there were 5 people working on each module for 10 weeks, there were 5 people who had two to three other courses to worry about that were at least equally demanding as this course, so while we had a lot of people power on paper, in practice everyone's effort was spread very thin. Failing to meet this requirement suggests that our development process could have been more organized throughout the semester. It was likely less organized than it could have been because it was every person's first time working on a major software engineering project, as well as the fact that everyone in every group had other major curricular and extracurricular time commitments. Overall we thought that this was a minor failure and we were okay with it, but we would certainly implement it if we were able to continue working on the project.

4 System Design And Architecture

High Level Design Our system is simply a database that has three tables, a table for holding the user data which includes user id, passwords, and emails, a project table that saves projects based on username, and lastly a table that stores the objects users create. All the tables in the database can be written to and read from. We provide functions that allow other modules to save, load, and validate data to and from our database. Low Level Design

dbEdith.php This php function creates and initializes the tables in the MySQL database. This script is only ran once and is for the initial set up of the edith database. These tables are “user,” “project,” and “objectCreator.” If the tables already exist and the script is ran it will drop the table and all the information stored within them.

function.php
This php function prevents SQL injections so malicious users cannot drop our database. This script uses a php function that essentially safeguards the strings that are being passed through to the server so a user cannot accidentally or purposely drop table informatiion

dbLogin.php
This php function is used to login to the database in every other function so we can update the login credentials to work with our database and not have to update each file.

valUser.php
This php function checks the information given to it by the registration page (such as index.html or registerpage.html) for empty fields or the existence of another user with the same username or email address. If all checks are passed, the user is added to the user table.

login.php
This function is called when the user tries to login to the edith website. When the function is called from index.html it post the information from the login form to the php file so that it can reference the users credentials in the database. If the username and password is found within the database the system will validate the user and create a cookie session.

logout.php

This php script logs the user out of the website. It simply ends the cookie session.

save.php

This php script saves the code of a project. The username making the save request, code, and project name are received from the main editor page. The script then checks if there is already a project under the given name, deletes one if it exists, and saves the project with the given username and project name.

objectSave.php

This php script saves the code and name of the object created. The username making the save request, code, and object name are received from the object editor page. The script then checks if there is already an object under the given name, deletes one if it exists, and saves the object along with the all the layers with the given username and object name.

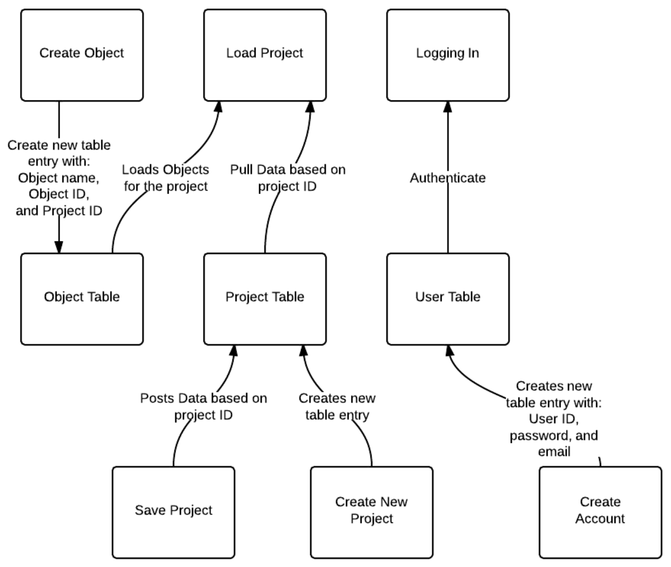
load.php The php script will load the project code onto the mainpage.html for the user to pick up where the user left off. The user will click the load button and will be prompted for the project name and from there the script will check for the name of the project within the project table. If the table finds a project that matches the name given by the user and matches the username it will return back the code in a json format.

objectLoad.php

When the user creates an object, they can then load the object onto the mainpage.html object canvas. If the table finds a object that matches the name given by the user and the project being worked on, it will return back the code of the object in a json format.

display.php

The display script will grab all the project files associated with the user currently logged in. This script is used so that if the user cannot remember the name of the project they are currently working on so they can pass that name into the load function.


Create Object
Load Project
Logging In
Create new table entry with: Object name, Object ID, and Project ID
Loads Objects for the project
Pull Data based on project ID
Authenticate
Object Table
Project Table
User Table
Posts Data based on project ID
Creates new table entry
Creates new table entry with: User ID, password, and email
Save Project
Create New Project
Create Account

The Sharing Framework databse UML Diagram is shown above.

5 Individual Reflections

5.1 Greg Finch

Developing Edith has been an interesting challenge. First off, it was by far the biggest project both in the scope and the number of people participating. One of the biggest challenges technically was the fact that a majority of our group has minimal PHP and JavaScript experience. This was difficult because it took a lot of time to try and write the code. Normally a bulk of my time is spent in actually trying to think about how the program should run, and actually coding that tends to take less time (besides debugging). This was the exact opposite; I spent significantly more time trying to get my code to do what I wanted, than actually coming up with how I wanted it to work. Organizationally, the challenges including managing time well. Because this project was so large in scale and the deadlines were pretty far apart (not that this is bad) it was easy to end up having to pull some really long coding sessions near the end of deadlines, and sometimes we didn't meet our goals because of this. During our development we used more of an iterative development process. Initially we planned what we wanted to have done by each checkpoint, and at the beginning of each checkpoint we spent ample time trying to plan and develop a solid understanding of how we want to move forward with the next step of implementation. We then went forth with development, followed by testing. This process was repeated after every checkpoint.

In the future, there are a few things I would do differently. First off to fix the issue of struggling with code, I think I would spend more of the beginning of the project to work on basic tutorials so I would have had a better understanding of how the language works, because this lack of understanding led to some pretty late changes in the workings of our code, which led to us recoding things in a different manner, while if we were PHP experts we probably would have done them right the first time. Organizationally, personally I think our group didn't make small enough goals. I felt we just had the large goals due at the checkpoints, and this made it so we tended to procrastinate a bit. If we would have given ourselves smaller goals that we did throughout the whole period we probably would have ended up do a lot more earlier and leaving less of a load during the last week or so before a checkpoint. If we were to keep developing for this project the first thing to do would be to come up with a better method of sharing projects. Secondly, we have almost no security on our server at all, that would be something that we would need to implement if we were to try and ever release this piece of software.

5.2 Ian Saad

The Edith project as a whole presented multiple challenges, most of which boil down to problems with organization and communication. The sharing team was no exception to this. Most of the time at the beginning was spent on learning the neccesary languages and anticipating what other teams would require from us/figuring out how the entire system worked and while we had planned out concrete and specific milestones for our group, looking back they were far too large and spread out. Additionally, communication between other groups virtually didn't exist until late in the semester; when everyone had mostly figured out what their module would look like. On a personal level, I struggled with php and specifically AJAX formatting for which online resources were either overly specific or unclear. Understanding the entire system/how the modules interacted also eluded me for some time; spending more time initially with other groups or within my group discussing and talking about the flow of the entire system would have been greatly beneficial.

In developing our system we had no official software design pattern in mind, however I'd say we loosely followed an iterative process. Our approach was followed our pre-set guidelines with a waterfall method of code to milestone and test and repeat. The techniques learned in class didn't really provide any benefit to implementation, but the testing and communication sections definitely left their mark and eased the process.

Overall, I will come out of this project truly knowing the importance of inter and intra group commincation. Without a proper venue / regularly enforced communication procedure it just won't happen or won't happen well. Communicating with large amounts of people is challenging to most, and involves a lot of effort negotiating meeting times, and putting yourself out there; a task that people don't perform unless there's an easy way of doing so, or they are provided some incentive. Looking back, if there was a forum for all the groups to interact or more time was spent between groups on the overall design things would have gone more smoothly. Towards the end of the semester however, the groups really pulled together and worked as a whole and development really sped up due to the improved communication and organization.

5.3 Chris Spalding

The semester was filled with both organizational and technical challenges involved in implementing Edith. The largest organizational obstacle was intergroup communication, while the most difficult part of putting together the software was learning an entirely new programming language. I applied software engineering techniques in order to test our code as well as other teams' code, but I did not find myself using software engineering techniques such as design patterns to implement or integrate during Edith development.

Figuring out to communicate effectively with other groups was difficult for a few reasons. For one, everyone has a busy and unique schedule, making it hard to find time for multiple groups to meet with each other. Furthermore, in the early stages of the project, each module is effectively developing independently, so I had developed a routine of meeting with my group on certain days and at certain times. Straying from this routine is difficult late in the semester when teams start to interact more and more. Electing a delegate from each group to facilitate intergroup communication was certainly helpful, but I think that making some sort of public forum on Moodle where all members of the class can update one another on progress would have been beneficial to development. But the delegate from each group helped a lot, and I feel that I was able to stay effectively updated on every group's progress on Edith. Overcoming the technical challenges that came with the project was an entirely different issue. Learning PHP over the course of a semester and writing substantial code in the language was a highly difficult task. However, the only way to really overcome the challenge was to find tutorials online and write a lot in the language in order to get comfortable using it. This method is exactly how I learned PHP, and while a lack of experience led to some inefficient code, the important aspects of the code work as well as they need to, so I would say that I tackled the challenge successfully.

In terms of the application of software engineering techniques and theories during the creation of Edith, I found that it was unnecessary to explicitly follow a design pattern while implementing our individual module, but I certainly used techniques that we learned in course readings in order to do static analysis and unit testing, and I used some guidelines I learned about from course readings in order to test our own code. For example, I think that reading a few articles about static analysis was helpful when I was writing testing reports and performing static analysis on another team's code.

While learning about design patterns may have been useful for the future, the individual modules of Edith, at least the Sharing Framework module, was pretty straight forward to implement. While Edith as a whole did follow a design pattern in order to be implemented, I don't think that the size and complexity of the Sharing Framework module necessitated a design pattern.

If I were to start this project over again I would start a forum on Moodle or make a group on some networking website, and if I were to continue working on the project I would refactor my module's code, as well as implement some sort of method for users to show their projects to others on the actual Edith website. I would make a forum or a group in order to increase the ease of communication between groups. Overall I think that the project went well, but as noted above, there were definitely a few things that could be improved in the future.

5.4 Bobby Kearns

Developing the database with the Sharing Framework has been a great but also challenging way to further improve my communication and development skills. I was able to effectively apply my basic understanding of the PHP language to help design an efficient database with the group. One of the biggest technical challenges that came up during this project was the communication between the various teams systems. This was a difficult hurdle to get past due to the multiple factors that went into accomplishing it, learning another language on top of the languages we were already working with and collaborating with other teams made this part extremely difficult. Some of the key successes team Sharing Framework used to accomplish this task was key communication between groups to meet up and discuss problems and layout exactly what was needed to communicate with the database. The one thing that would have been beneficial to know and was a big takeaway from this project would be the knowledge of AJAX and a better understanding of the javascript language.

In terms of organizational conflicts the biggest one that we faced as a team was sticking to the goals or checkpoints the team has set up or put in place. Instead of by sticking to the goals to accomplish within a certain time period the team just dived in and started to build everything at once which made things difficult later on when new methods needed to be implemented and made things a little unorganized towards the end. While the team got everything done pretty quickly, we found ourselves frequently changing how we structured our methods and database. Looking back on this project I think it would have been more beneficial to have set smaller, more achievable goals and stick with them rather than just dive in and code away like we did. By setting smaller goals I feel like things would have gone a lot smoother and we would have hit less restructuring roadblocks and would have made communication a little easier since most of that part consisted of rearranging or changing parts within methods or the database. We overcame both of these obstacles by effectively communicating between our team and other teams to make sure that we are all on the same page and have the same goals in my mind. The biggest takeaway I got from this project in a non-technical aspect was to set smaller goals or lay out a plan and stick to it rather than just diving in and doing everything at once.

Within the Sharing Framework group we did not really stick to one particular development style, like previously mentioned we all kind of just dived in and just started coding the database. If anything the team implemented a more iterative approach to the development of the database. We set out what we wanted in a database and had somewhat of a development cycle to meet but did not really follow it once we started working on it. The only thing that we really made sure to do was testing, after hitting a “checkpoint” we would make sure to test to see if the system was working as expected and met the requirements of the system as a whole. Since we did not really have a formal design style, there was no real benefit from the techniques learned in class. The communication we had within our group and other groups in the class was key in successful developing the Edith project and helping us overcome our poorly implemented development style.

5.5 Wesley Stedman

Working on the Edith project presented me with some unique challenges and learning opportunities. The first challenge, a personal one, was learning HTML, CSS, JavaScript, and PHP. I knew nothing about these before this class. Now, though I do not know these very well, I am interested in gaining mastery of these languages to expand my skill set. The second major personal challenge was learning how to use Apache, MySQL, and PHP to make a functioning web server and database. Fortunately, I had WAMP, LAMP, my teammates, and articles on the internet to help me get started. Unfortunately, the differences between MAMP and WAMP in conjunction with poor communication inhibited the effectiveness of hosting a remote server.

Additional challenges included establishing effective communication both within my team and with other teams. Due to prior engagements, I was unable to arrive to my group's meetings at the scheduled time. This reduced the impact that I had on the coding process. Otherwise, the means of communication within my team were effective: we used social media group conversation functions to agree on meeting times. Such a system was not set up for communication with other groups, however. This led to other groups not communicating their necessary interactions with the database until late in the development cycle, requiring us to modify our functions to suit their needs. The save function is an example of this. In early versions of the code, there was only one save function. Our plan was to have the other teams pass a parameter that would determine what code to use to save the data that was also passed. We determined that this would be needlessly complicated, and refactored our code to use different save functions for different data.

Though we didn't consciously attempt to use any specific software engineering techniques, our strategy could be best described as incremental development, blending waterfall development and prototyping.

If I were to restart this project, I would get a better understanding of how my part of the system would interact with the other parts so that I could develop the right functionality properly the first time. This would involve confirming what the other modules will be passing to and expecting from the server. If I were to continue developing this project, I would improve the sharing functionality to allow users to create their own copy of another user's shared code, which they can then edit. I would also implement a way to export finished projects to either a movie format or a .gif file.