Visual Editor Final Report

Graham Baker, Walker Bohannen, Jessica Penick,
Steve Marx, Vikram Nilakantan, Eli Spiegel

December 18, 2013

1 Project Summary

2 Development Procedures

3 Requirements Evaluation

4 System Design and Architecture

5 Reflection: Vikram Nilakantan

5.1 Challenges

5.1.1 Technical Challenges

The biggest technical challenge that I faced while on the Visual Editor team was my personal inexperience with the HTML5 Canvas or any of the graphical JavaScript libraries. After learning that the visual editor was going to be based in an HTML5 Canvas, I started to go through the process of figuring out how it worked and how to use it. I was told that the Canvas was very similar to doing graphics in Java, but since I had little to no experience with Java graphics, that advice did not help me as much. Two things mainly helped me overcome this challenge. The first was my prior experience using JavaScript and HTML. Even though the Canvas was unfamiliar, the language used to control it was very familiar and I was able to use my prior knowledge to accelerate the learning process. The second learning tool was simply trial-and-error. No one on the visual editor team had ever used the HTML5 Canvas or KineticJS before so what really helped us was several test documents trying to figure out how object were created and how we could interact with them.

5.1.2 Organizational Challenges

Unlike other groups working on the Edith project, our group had six members, opposed to five. Since none of us had any familiarity with the technologies we were working with, our organizational challenges revolved around slow progress. Another challenge that surrounded the visual editor is that it was very difficult to separate work out until we had a solid foundation of what we were doing with Canvas and KineticJS and how we were going to execute such actions. Until we got to a point where we had a plan of action in which we knew how things were going to be done, we were unable to work separately and the only time progress was made was when all of us could meet up and surround one computer while we tried things out. Fortunately, we soon figured out what we were trying to do and were clearly able to divide tasks among group members.

5.2 Applied Software Engineering

Many aspects of the visual editor were designed and implemented according to software engineering techniques, however, I thought that the biggest lesson from software engineering we used was on creating the user interface. We estimated that the visual editor component of Edith is where the typical end-user would spend the majority of their time. Because of this, we wanted to spend some time

on designing the user interface and become aware of exactly what actions users would be able to perform and make sure that certain elements of the editor are presented in a consistent manner.

5.3 Different Approach

One of the largest over-arching challenges that the visual editor team faced was that we were unsure of our requirements for the project and I felt like each of us had 'an idea' of what we were doing, but no one's ideas actually matched up with anyone else's. This posed a big problem at the beginning of this project and until we were sure of what we were doing, our progress stalled and at some points, came to a halt. If I were to do this project again, I would make sure that I was completely aware of the project/section requirements. If I were given more time to continue to work on this project, I would want to improve some of the graphical elements and transitions on the visual editor. The boxes and 'method containers' used currently are functional, but I feel like with more time, I could make those boxes more visually appealing. Another feature I would implement is ease-of-use for the end-user. That is, making sure it is clear how to operate the visual editor and making sure the user is not confused when they are trying to operate the application.

6 Reflection: Graham Baker

6.1 Challenges

Some of the main challenges I experienced during development were learning a new language, dealing with GitHub, and trying to find dividing lines between our group and others. I really enjoyed learning to use JavaScript. While it was pretty difficult at the beginning to work out how to access certain attributes, using global variables and other general points of confusion transitioning from traditional object oriented programming languages, I felt just pounding through trying to get my code to work was extremely beneficial and valuable. GitHub was also a source of confusion and frustration early on. I used the terminal for the first half of the semester and struggled to accomplish what I wanted to do. After I switched to the GitHub application, things got a lot easier. I am glad that I was introduced and got experience working with this important software development tool. Finally, the most difficult challenge this semester was settling on what each group was supposed to do. There were a few times where we were not really sure what we were supposed to be doing because our requirements overlapped with other groups. It was a good experience for us to have to grapple with the miscommunication and complexity that is so often the downfall of many projects. I learned that it is vital to get the requirements and direction of the project as early as possible to make life easier as the project goes on.

6.2 Applied Software Engineering

The technique that I most frequently used over the course of the semester was the idea of iterative development. Since our requirements and goals were constantly

shifting, we had to adapt every time we got a new direction. It was also nice to see that is consistent with how most software engineering firms operate.

6.3 Different Approach

If we started over, I think we would probably seek out another graphics package to use for our user interface. While kinetic.js offered quite a bit of useful tools, I think one that supported text input would have probably suited our needs better. We landed on kinetic.js after trying to get things to work with JQuery UI as well as other graphics packages. As we approached the first implementation, we just had to go with kinetic.js. If we wanted to continue development we could find a better package that would suit our needs more appropriately. Now that we are at a decent position, I think there is a lot that could be done. It would be really great to add method box encapsulation like we wanted to do. Now that things are up and running, it is easy to see things that could be improved on and how to make the software easier to use for the user.

7 Reflection: Jessica Penick

7.1 Challenges

7.1.1 Technical Challenges

I met a number of challenges in the entirety of this software engineering course, mostly due to the project itself. Firstly I spent the whole semester working intimately with languages that I had no previous familiarity with: JavaScript, LaTeX, and to some extent HTML (though I'd had a little past experience, and probably knew enough). I struggled with git at first. For the most part it made sense conceptually, but using git seemed to be an entire other story, and I one time deleted everyone's lab assignments on accident, which, luckily, could be undone due to the nature of the git (that is, version control). It seemed like everyone got everything faster than I, which was, to say the least, frustrating. My group in particular had technical experience beyond mine, so I took on a supporting position to compensate. I read a bunch of resources on JavaScript, but I think I learned the most about the language from examining their code. Overcoming the challenge of LaTeX was a matter of online inquiries (read: Google), templates, and tutorials.

7.1.2 Organizational Challenges

The project some organizational challenges over the course of the project as well, both in the code and the group. For example, we never really designated a leader, which at first wasn't an issue, but later when disagreements arose proved problematic. We argued over which library to use, and later what functionality to focus on, but all of these came to peaceable ends, and never got particularly out of hand in the first place. The lack of a leader also lead to rather unsustainable agile module development on our part; often we would meet up then divide different tasks to each person to solve on their own, when I believe a pair system might have benefited us better in the long run. Even though pair programming is somewhat frustrating, two heads are, in fact, better than one.

7.2 Applied Software Engineering

There was a lot to take away from this class, too. Chiefly process models, testing methods, architectural patterns, and design patterns. More notably, process models and testing methods were some of the applied techniques of our project. We prepared and implemented a plan-driven development, but I think in the end our project, and more specifically my group's module in it, became a kind of agile development process, which isn't necessarily a bad thing; both models have advantages and disadvantages. Testing, too, played an instrumental part in development. Of course testing should always be instrumental, but linting the code proved especially important since clean code is easier to parse and revise later.

7.3 Different Approach

If we were to start over, I would do a few things differently: 1) I'd choose a group I could be more helpful in, 2) I would be more involved in the coding process, or, barring that, I'd have taken more of a stand on the organization of the code/communication within and without the group. There is more to do than coding in a development team, so I would have operated in a position of administration rather than facilitation. If we were to continue developing the code, I would work to optimize it for efficiency—-maybe make it more easily maintainable. That includes documenting it thoroughly, observing the interfacing between modules and seeing how them may be improved by the reallocation of responsibilities, or standardization of the information being passed around. Mostly re-imagine it as a project conceived as a single idea from the top down (or more likely, the bottom up). Also, over time, add more features, clean up/unify the interface, and generally expand it (maybe a message board to discuss created stories, or the ability to annotate them? Possibilities are endless).

8 Reflection: Eli Spiegel

8.0.1 Technical and organizational Challenges

The project started trying to use the waterfall method, however as it progressed the lack of direction made the waterfall method challenging. This led to our team and a few others moving to a more agile approach. Agile helped us get a product that worked in a shorter amount of time as we were able to progress and find different problems instead of spending all of our time planning and then quickly finding new issues. Other issues included communication between people with diverse schedules, and working with tools and languages I was unfamiliar with, mostly JavaScript and GitHub.

8.1 Starting this Project Over

I think writing requirements as a class with Joel as the "client" could have made for a better use of the waterfall method as everyone would have a better understanding of the overall project, which would help with moving forward on the individual modules and solve division of labor/dependency issues. Another issue the project had was a lack of experience with the languages/tools groups

would be using took the focus off of the software engineering problems and made them more implementation problems. I believe that a stronger focus on learning the tools and languages at the start of a project will make it clear what problems may arise and possible ways to solve them.

8.2 Learning

In the end some of the more valuable things I learned were how to write code as a group, and how to use tools such as GitHub. I also learned that knowledge is the most important tool in software development. If a group comes in knowledgeable about the tools and about what the goal is that they will build even complex projects in fairly short about of time.

9 Reflection: Walker Bohannan

9.1 Challenges

Developing the Visual Editor platform for the Edith project involved a tremendous amount of work, and creating boxes with which to create and edit code with was not without its difficulties. Some specific challenges that I encountered in the beginning was on deciding which framework to use. Some options we considered were jQuery UI, RaphaelJS, finally settling with KineticJS, which offered the tools to manipulate what we needed to on a HTML5 canvas. Organizational issues often arose with cross-team integration of using GitHub, and resolving conflicts was not always as simple as we might have hoped. In the end, we figured out how to solve these problems, and came away from the problems with renewed expertise and troubleshooting skills regarding GitHub. That being said, I was always extra careful of any changes I made when pushing to Git, and being sure if I needed to modify another group's code I was explicit in what I did and commented my changes so they would not be deleted. Another challenge which required being overcome was a mid-project reassessment of what our role in the project as a whole was. Our requirements were not explicit until this point, and until then there was a lot of, "well the other group is taking care of that," going on not just in our group but in others as well. Other challenges was learning a new language - Javascript, which I knew none of prior to this course. The syntax quickly came in, and past coding experienced helped, and soon I was enjoying using a new language as I had used others, which only minor hiccups along the way.

9.2 Applied Software Engineering

A software engineering technique that helped us during the process was using rapid application development, where we constructed prototypes, and would test them (lining up with the deadlines for our integration tests). We iteratively developed our prototypes. We also fulfilled one of the main principles of rapid application development by emphasizing the "business need," which was being able to create and write code using blocks rather than focusing on the technological excellence. We used ourselves to act as the users, testing continuously. In continuity with our integration deadlines, we focused on reducing implementations and making our deadlines with what we had working well. Another useful

technique was to use pair programming, or even group programming, with one person "driving" while the other members looked on. This was especially useful in imparting knowledge of Javascript (which started at zero), as we started this project. We would switch roles often, with different combinations of different group members taking different roles, so nearly everyone had a chance at the controls.

9.3 Different Approach

If I were to start this project over again, I would probably create my own framework, rather than use KineticJS or something else similar to it. Creating our own framework would enable us to manipulate objects and variables in a way which was efficient and could be used in a way to how we could optimize it. Often I felt as though I was spending a majority of the time figuring out how to make the KineticJS work, figuring out correct syntax rather than being able to go through my code and edit and implement new features. That being said, having the KineticJS framework probably did save us an enormous amount of time, rather than discovering halfway through the project that our own implementation was not going to work. As far as continuing work using the existing code we have, I think method encapsulation would be the next step, as well automatic end bracket creation. However, with how we are implementing it, I think requiring the user to add end brackets is a very relatable aspect of coding, and will teach the user how to code better. That being said, implementing a "compile" function, letting the user know they forgot something syntactically would also be useful.