Problems we came across and what we did to resolve them:

**Choosing the Project Framework**

Our first major decision was which project framework to choose: SPA, Web Forms, MVC or Web Pages. With a little research (http://www.codetails.com/2013/11/08/web-forms-mvc-single-page-app-web-pages-comparison/20131108/) we quickly determined that the tight coupling between the UI and business layer would make our planned parallel development approach more difficult. Web Pages were also ruled out due to planned scale of the project. SPA seemed to be the hot new approach and very appealing, but MVC is a recognized software engineering approach. We had Learned about MVC in our classes, seen it in job descriptions and thought it would fit with our view of how we expected to share implementation responsibilities. It was unanimous! We chose MVC.

**Doing the first walkthrough of deploying to Azure**

We did this very early in the semester hoping to gain experience and generate a little confidence. We spent most of a day doing the walkthrough. There was a slight euphoria when it turned out to be pretty straightforward. While a great group bonding experience, it was a little demotivating in that it made it seem like everything in the project was going to be that easy…and it wasn’t.

**Integrating JQuery plugins into MVC project**

Each of the JQuery plugins specified their own set of .js, .css files. Some files were .min files. We had to understand where the files lived, what code loaded them, when and how and to integrate them into the project file structure. In addition, we had to learn about minification and choose a tool, then minify code changes we had made to .css, and .js.

**Integrating javascript code into MVC/Razor/Html Model**

Circliful, Tree and schedule, all JQuery plugins are provided with static demonstration objects.

Integrating them into an MVC project required providing the model data to the plugin so that it could be rendered in the view.

For Tree this is the provided static data code:

**var** items = [

**new** primitives.orgdiagram.ItemConfig({

                    id: 0,

                    parent: **null**,

                    title: "Root Project here",

                    description: "Project Tree",

                    image: "demo/images/photos/a.png"

                }),

**new** primitives.orgdiagram.ItemConfig({

                    id: 1,

                    parent: 0,

                    title: "CSC 311",

                    description: "Ugh! C#",

                    image: "demo/images/photos/b.png"

                }),

**new** primitives.orgdiagram.ItemConfig({

                    id: 2,

                    parent: 0,

                    title: "Senior Project",

                    description: "Fabulous",

                    image: "demo/images/photos/c.png"

                }),

**new** primitives.orgdiagram.ItemConfig({

                    id: 3,

                    parent: 0,

                    title: "Mom",

                    description: "Visit and Love",

                    image: "demo/images/photos/d.png"

                }),

**new** primitives.orgdiagram.ItemConfig({

                    id: 4,

                    parent: 3,

                    title: "Bake Cake",

                    description: "More Fabulous",

                    image: "demo/images/photos/d.png"

                })

            ];

To this:

var items;

var newItem ;

var items = [ ];

@foreach(var item in Model)

{

@:newItem = new primitives.orgdiagram.ItemConfig({ id: '@(item.ID)', parent: '@(item.ParentID)', title: '@(item.Title)', description: '@(item.Note)' });

@:if('@item.ParentID' == '0') //needed to make javascript implementation compatible with Project Model

@: { //specifically, project model has int for parentid and sets root to zero

@: var headparent = 0 ;

@: newItem.parent = null ;

@: }

@:items.push(newItem);

}

You have a mixture of C#, razor and javascript, figuring out what needed to be done and the syntax of it was quite time consuming. Circliful and schedule had similar issues. After completing the first, experience made subsequent integrations a little easier. Despite how challenging it was, it saved timed and allowed greater functionality in the project.

**The GIT Bash Learning curve - Living in GITHub and working in GIT**

GIT Bash is not easy to learn. We had a significant number of walkthroughs. There were a large number of emergency Skypes to our resident expert. Version control is conceptually complex and absolutely necessary on a group project. Adding to the confusion is that we used github for our repository. It took a couple of weeks to realize that git and github were not the same thing.

**Database**

MVC5 code first has a lot of moving parts. We ran into some issues understanding how to include the account table in the projects database. Without a full understanding of all the parts, data migrations didn’t work as expected and each time we made an entity change, we had to delete the database. It was fairly late in the project that we had an adequate understanding of how the database interacted with the code. This is a really complex area of study and would need a lot more time and effort to have a full appreciation for the features.

**Integrating the MVC “free” User Authentication**

MVC5 framework comes with a fully implemented account management system. In theory, it should be pretty straightforward to restrict access to certain functionality and it is. Finding information that allowed us to store the account userid in our project data table turned out to be more difficult to research.

**No change is a standalone change**

While we each had areas of the project we were more familiar with, our efforts overlapped into other areas frequently. There were quite a few instances where a change made to one area of the project had unintended consequences in another or an existing implementation weakness was made worse when a change was made elsewhere. In one case, we didn’t notice that the code was broken until a few days later. Git made it easy to pinpoint which commit introduced the problem and we got it fixed quickly. Because we have a relatively small project, it was easy to regularly (about once a week in the second half of the project) do a quick feature sanity check and identify issues quickly. On a larger, more complex project, it would be important to have dedicated regression testing after each incremental feature set.

**Sprints, deadlines, formal testing and documentation – not so much….**

Pretty much none of these turned out the way we expected. It is really hard to be disciplined about these items. Nobody likes to do documentation, especially when there is real work to be done. Deadlines require an enforcer. The sprints felt contrived except at the very end. In a small group, it helps if one person is willing to be the point person, for us, it evolved out of necessity and we kept a very lightweight process. What seemed to work best, is to let a process issue develop just to the point we could all acknowledge it, then find a solution we could all live with. It slowed things down at the beginning of the project, but in the longer run, we became more efficient. The risk is that a problem goes on too long, unrecognized. Since we are all very process oriented, we did pretty well with this. We can say that more in retrospect. At that time, it sometimes felt stressful and uncertain.

**Scheduling, Teamwork and meetings**

We were brought together because we are Salem State students. Our lives and commitments outside of Salem State puts us on very different schedules, often making meeting impossible. We all had different productivity/involvement cycles. Because we are friends and had started by building a foundation of working together, we were able to keep pushing forward. We used Skype meetings, IM’ing, emails to stay in regular touch about the project. Towards the end of the project, we started using Visual Studio Online to track bugs and backlog items. This turned out to be a really good tool. It would have been great to use it from the start.

As a team, there were times when we were stuck, spending long hours for minimal progress, all of us insisting on learning and participating in everything. As time went on, we were able to develop trust in each other and the team. We had some amazing moments, when 2 or 3 of us sat down to solve a problem and were able to move forward rapidly. The last few weeks of the project were so focused on the outcome that most other issues disappeared.

**Beta Testing by Family and Friends**

After our “final” code release, we sent the link to a few family and friends. We found they were registering, logging in, but not creating any projects. After registering, it wasn’t obvious where to go next. We quickly updated the register/login page to redirect to the project tree view and added text in the top of the tree instructing the user.

**What would we do differently?**

Start implementation sooner.

Waste less time ~~arguing~~ discussing details before we’ve actually implemented anything.

Use backlog tools sooner.

After picking MVC approach, OO design efforts were not as coordinated as they could have been. Work hard to prioritize the work effort towards getting functionality before increasing design complexity.

We spent a lot of time upfront, developing a common vision of what we were doing. That early effort sustained us through some of the more challenging decisions.