Nicholas Papacostas
Capstone Documentation:

Overview:

Consensus decision making is a process used to promote equitable participation in discussions and meetings. Much like Robert's Rules of Order, it provides a framework for guiding and facilitating decision making in a group setting. The differences between the two formats generally stem from consensus' strict adherence to non-hierarchy. For a full list of rules see http://www.ic.org/pnp/ocac/ .

This problem has been tackled in some forms (i.e. Google Wave, GoToMeeting etc.) however these platforms attempted to answer broader problems of group communication with video conferencing for more business oriented environments. My application was intended to be a web-based, text-based (at first), chat-room with which an organization could join, invite members, and hold meetings according to consensus process. My personal motivation for creating this application derives from my involvement in a number of organizations that use consensus process. I constantly hear claims of how great consensus is but that it cannot be scaled out on a large scale because it becomes incredibly inefficient. My hypothesis is that if certain procedural aspects of consensus were automated it would be able to be used on a larger scale.

Objectives:

Initially I set out to have a working chat application with a number of meeting process features to enhance discussions. However, the amount of difficulty I encountered setting up the infrastructure for such an application was too great. As such, I cut the scope of my project to only include a chat application with different channels and an organization/user backend structure.

This was made more difficult by trying to use new tools and experimental frameworks which were inconsistent with documentation and examples. That being said, it was also a very good learning opportunity as I familiarized myself with a number of new technologies including the SpineJS javascript framework, web-sockets, Ruby on Rails and Twitter Bootstrap (for styling).

I did not complete my objectives because my scope was too broad and my focus was too spread out. Had I initially tried to just create a simple javascript chat application I probably would have been able to finish with a more polished final product.

Skills Gained:

Git:

In building this application I wanted to use industry standard tools. This included Git for source code version control and Heroku for free hosting. Learning these tools was a substantial roadblock in my progress but also extremely important to my career in software development as they are (especially Git) ubiquitous in software industry. I also immediately saw the benefits of using something like Git when I would make a mistake or want to see what my project looked like in the past. I also now have the advantage of my code being public on the web.

Heroku:

Heroku is a web application hosting site which has great free opportunities. It is also very popular for enterprise solutions especially within the Ruby on Rails community. This project was a great opportunity for me to familiarize myself with the Heroku deployment process. It was especially difficult, however, because my application requires a number of other services besides the standard rails server (e.g. redis-server and juggernaut).

Ruby on Rails:

While I have worked with RoR in the past, this was the most substantial undertaking I have done on my own. I became intimately familiar with creating a database structure in RoR and serving up dynamic pages with database content. Rails is a very popular web application framework and I now feel quite proficient in it.

SpineJS and Javascript MVC:

This was by far the most challenging aspect of the project as I had very little experience with javascript prior. SpineJS proved to be a more fickle framework than I was expecting and it caused me lots of problems. That being said, although I utterly failed in almost every aspect of using this framework, I have learned quite a bit about the MVC design pattern and javascript/ coffeescript. Hopefully as I continue to work on the project in the future I can further develop this skill.

Web-sockets / juggernaut:

Getting asynchronous chatting to work proved to be extremely difficult as I had to work with a number of various components (Spine, Juggernaut, WebSockets, Redis) of which some kept up to date and some were not. In the end I finally got Juggernaut working but at the expense of lots of functionality.

Twitter Bootstrap:

Twitter Bootstrap has become a wildly popular CSS framework which provides functionality for styling web pages. While it initially gave me some trouble it proved to be a very useful resource in rapidly changing the look of the website.

Evaluations Section:

Performance and Efficiency:

These are slightly less important features for a modern web application of this size because it pales in comparison to most major web apps. That being said, there are things that could be done to improve the efficiency of the app. Most obvious would be to limit how often assets are shipped to the client via caching. Right now there is quite a bit of redundancy in how/when assets are shipped. That being said, it is highly unlikely that this inefficiency would give any modern browser a difficult time given a reasonable internet connection.

Scalability:

The app is actually quite scalable once deployment to Heroku is finalized. All of the tools being used have a capacity far greater than Consensus demands.

Maintainability:

Consensus uses some tools which are rapidly becoming outdated. Juggernaut can be replaced with HTML5 Sockets and is no longer being supported. This would/will make maintenance very difficult. Similarly, SpineJS seems to have lost the javascript MVC battle to Backbone.js which will make it unlikely to be supported and upgraded in the future also hurting the application's maintainability.

Ruby on Rails and Bootstrap are very popular with extremely active communities making them much easier to maintain and find guidance on and in this regard the app is somewhat maintainable.

However, above all, the reason the app is not maintainable is because of the lack of *tests*. Test-Driven Development(TDD) has become extremely popular in the web development world and an industry standard skill. Because of my poor time management and difficulty learning ruby and javascript testing suites I have not supported my application with unit or integration tests. This is a real weak point that I wish I had done a better job on. In the future, adding a testing suite to the application would greatly increase its durability and make any additional features much easier to implement.

Reusabililty:

Most of the code in this application is not reusable. It is mainly dependent on specific technologies and specific application needs. The knowledge gained from creating the application, on the other hand, is extremely reusable and widely applicable. Creating database relationships and ensuring proper asset compilation is an essential skill for a modern web programmer and thus it will undoubtedly serve me well in the future.

Usefulness:

The idea behind the project is quite useful. A consensus based chat application would be awesome and helpful to many groups I am personally involved with and thus I would imagine with many others. The state the application is in now though, it is almost completely *useless*. In getting Juggernaut to work I destroyed most of the actual chatting functionality.

Limitations:

As stated above, the app is quite useless as it stands today. There is a lot of potential for improvement and real-world applicability. Also, because of its many dependencies it is more fickle and unwieldy, this makes deploying to different environments quite challenging.

Conclusion:

Although the application isn't what I wanted it to be, I am still proud of my work. This was the hardest programming endeavor I have ever taken on, and even though I would say I failed to reach my objective, I learned a great deal. I learned about a number of development and deployment tools and further familiarized myself with ruby and javascript, two of the most prevalent languages in web programming today. I also learned that I need to better manage my time and have a more realistic and concise scope for projects in the future.