Hızal Çelik

Bill Scherlis

17-413

December 10, 2016

There comes a time in a company when there is a need for some reorganizing, re-planning and even refactoring of their products or codebase, and every organization needs different methods, solutions and goals to successfully complete this task. Unification of code, both for back-end and front-end, is one such example of reorganizing that involves all of those factors, including refactoring and re-planning. These tasks involve developers, whether they are individuals or teams, to meet together to figure out why unification is needed, which codebases should and shouldn't be a part of this process, what the unified codebase will look like and what technologies will be used for the unified future.

This was the case during my few months of interning at LiquidNet, a financial technology company based in New York City that provides a trading market/platform for institutional (high volume) traders. For this company, about a year or two ago they decided they need to unify their various products and services, both internal and external, to minimize excess code or needlessly replicating code, such as when two isolated developer teams solve the same issue independently, resulting in two possibly similar solutions that could have been done by just one team had they communicated and shared code in the first place. For LiquidNet, this unification involved both front-end visual synchrony as well as a shared library/code-base for backend structure and services for their webapps. As far as I know, this unification also involved core services and software that their webapps use as well, many of which date back to the late 90s. To do so required refactoring the old languages/code, unifying similar code and services and for their webapps, deciding on a singular framework to base the front-end on. However, many issues arose from this seemingly straightforward task.

Thrown in the middle of the efforts to unify the seemingly endless list of ever-aging products mixed with brand new apps, my own job as an intern was to focus on the front-end. The two major projects I worked on over the summer both involved redesign. My actual internship was through the Design team, although I worked heavily with the software side of the company for my projects. One of those projects was the redesigning of the user interface and (more importantly) the user experience of a core internal app used by many internal divisions, but mainly product services. Although this internal app was relatively new, it already was a mess in terms of usability and visual appearance, and I worked with the users, the developers and my own design team to create a better experience and good looking platform that also matched the look of the rest of the new webapps (in order to continue the trend of unification). My second major project was to create the unified styleguide for the company to use, whether the user is part of the design, marketing or software teams. This styleguide included webapp components, visuals, layouts and technologies that ideally all internal and external products should be using to create this unified vision the higher-ups strived for.

I worked on my own, jumping between various developer teams with questions, tasks or help while I worked on a central style library to be used by all developers in the company. However, during my back and forth trips between developer teams I witnessed a lot of discrepancies and lack of communication between them regarding choices of frameworks, refactoring methods and other generally big decisions. It seemed that the higher-ups wanted unification of all products, both in terms of visuals and familiar functionality, and yet all the teams split up to start working on their versions of this unified future, rather than actually working together to create one truly unified version of backend/front end code. Each team had their own choice of the perfect framework to build the future on, and they chose these frameworks based on the previous projects and products they've had to build in the past. With each team developing different products in the past, each team had a different framework choice that fit their needs best, but instead of the leaders coming together to finalize their decision to one framework, they would leave meetings in disagreement and continue to work independently with their teams on their vision. The result, as of now, is like a "refresher" for the company that only ends up in the same place. Instead of a bunch of disconnected and different old products, we have a bunch of disconnected and different new products using webapp technologies.

Regarding my own experience specifically, I was pulled between three different directions when creating this styleguide and the app components within it. My design team boss wanted me to create a styleguide that all developers can use easily, as simple as drag-and-drop code into their webapps. The developer I worked with closely over my other project, the redesigning of an internal app, told me that the styleguide should be using UI-Grid and other components that require an angular-JS backend. The other major development team that focused more on webapps already had a relatively unified framework, both visually and backend, complete with their own drag-and-drop library of components and templates. However, their apps used an older, outdated version of angular-JS that the first developer I mentioned refused to use (as well as other developers in the teams he was on). So unfortunately in my efforts to please all, I ended up doing exactly what was the original problem; I created a visual library that *looked like* the unified library the webapp team had, but was created as vanilla as possible (using JQuery and other simple Javascript components) to look, act and be used for the same thing but wasn't actually a part of any Angular-JS. I pretty much created a third option rather than a final library.

There could be ways to mitigate this for the future, which I may bring up to my boss this coming summer (as I have secured another round of interning with them). One such possible solution to the lack of agreement and general independence of the teams is to have a new managerial position or leader that will make the ultimate decision *and enforce it*. From what I understand, there is no general project manager that leads these multiple teams. At LiquidNet, there is a product manager, but the position handles more feature and content based queries, not technical issues. This was an issue brought to my attention towards the end of the internship, and I will see if there has been any changes in leadership by summer. Another mitigation strategy would be to a long, productive meeting between all developers and team leaders. Until now, it has just been the team leaders and they only meet occasionally, for short periods of time. There should be just one big, long meeting with plenty of discussion, comparisons, proposals of a plan or process and critiquing of each other's suggestions. This would be done between developers and their leaders all together, which will hopefully bring about a unanimous or majority-vote solution and each team will work to realize these visions. Having a structured process in then going about refactoring and unifying is also key, and this process should include enforcement. That enforcement could come from the currently nonexistent overall development leader, or from other higher-ups in monthly or quarterly scrum meetings. Unfortunately, the biggest reason why I don’t see this discussion between developers happening is because the teams are relatively small as it is, and I don’t think LiquidNet can afford to lose developers for more than an hour, if at all, despite the fact that this would help improve and streamline future development. Some investments of time and resources are harder to do than others, I suppose.