One of the biggest design challenges I faced was deciding how to best bring everyone who wanted to split a bill into the process. I bounced from having one person in charge of the splitting and then simply making a payment request, to social media integration, to a code that code be shared to join a bill, and for a location-based signal that could be sent. The problem really was a matter of ease and speed. Our contextual inquiry participants I retired of splitting a bill by hand or sending out multiple payment requests in different apps. Thus, I wanted to choose the two solutions that seemed easiest to use and most technologically realistic. So, I decided the best options are social media integration via bill invites and a share code that users could tell or text their friends. Certainly I could imagine ourselves just linking our Facebook account and then selecting the friends I wanted to share the bill with. I also thought simply telling a friend a four-digit code so the friend could join the bill themselves was simple to design, easy to use, and allowed additional freedom. The importance of freedom or flexibility is something I thought of only at the end of our designing. I realized that while the Facebook or Twitter invites could potentially be quicker and would allow additional ways to notify users (say a Facebook alert or Twitter DM), this method was restrictive on who the bill could be split with. What if a user wanted to split a bill with someone, they weren't friends with on Facebook? Would they have to add the person on Facebook out of the app or could I allow that in the app? That might take a considerable amount of time and both users may not be interested in connecting via social media. In this case, a four digit share code that could be given across the table or through a quick text message would likely be less intrusive, easier to use, and quicker. However, while this scenario would definitely influence our final solution, I  wanted to test both designs in a perfect situation. Both designs would be tested as if the user was amongst close friends, all connected on social media and in the same location. With this perfect scenario it's hard to judge which design would be faster. On one hand the social media invitations would be an immediate notification on the other user's phones and I believed would prompt quicker action than simply telling friends a share code. On the other hand, while I couldn’t account for this fluidly in our design, there would be a searching period to find the correct Facebook friends. Sharing a code would be simple and would only require other users to open up the app and enter the code to join the bill, which in a perfect scenario that I am testing wouldn’t take much time. Still, I have to recognize the possibility of an imperfect scenario where a share code might not be used right away or forgotten later. Ultimately, in a perfect scenario I believe the social media integration would allow the main user to invite other users and those users to join quicker than a share code.

Once I finished the high-fidelity outlines for both our designs I was  ready to implement them into CogTool. For both designs our main form of user input was the touch-down of a finger, but I also had to take into consideration the time taken for the main user of the app to communicate with their friends to get them to join the app, the other users to join the app, and a short confirmation on how the bill is being split. Starting with the the “Share Code” design, the main user is given the share code right away and they can start selecting the items they bought, but in order to correctly split with friends they had to wait around 15 seconds for the other users to open the app and type in the code to join the bill. Thus, the main user didn’t have to spend much time on the actual invites, but the wait for others was a bottleneck. One improvement here could be allowing the main user to not only choose their own items, but add people even if they haven’t accepted the invite and then pick items for that person. Then upon joining the bill the other user could select the entire profile of the previously added user and take over all of the previously added items. The “Facebook Invite” design on the other hand allowed the other users to simply get a push notification that would automatically open to the bill, which made the wait for users to join much quicker. However, with the Facebook invite, as it is quick for other users, it turns out to be very slow for the main user of the app. Instead of telling the friends sitting around him the 4-digit code, which takes about 6 seconds, they had to scroll through all their Facebook friends to find the ones they are sitting right next to, then make sure they have everyone selected and send out the invites. This process on average took the user over 30 seconds to complete for only 2 users and will get exponentially longer the more users they are splitting a bill with. At this point I realized this step could be drastically improved by adding a search functionality to the friends list search and possibly even adding a “frequently split” tab that lists frequent friends. Once all users were added into the app selecting what everyone ordered had similar times for both designs averaging at about 25-30 seconds, however it can only be completed once every user finish so this time mainly depends on the response time of other users or how quickly they can get into the app once they have a code or invite. With all of these times taken into consideration and computed through the CogTool the “Share Code” design finished in 53.5 seconds and the “Facebook Invite” design finished in 83.7 seconds, which was the opposite of our original guess, primarily due to the long searching times to find other users in the Facebook invite screen.

Reflecting upon our experience with user performance modeling, I found it easy to discount its usefulness before using it, but after working through the process it became obvious that it’s a valuable process. Certainly I could have run tests ourselves with users and gathered data that way, but using a digital tool such as CogTool allows the entire process to be cheaper, faster, and more flexible while still maintaining quality insights. Without knowing how our design might compare to other alternatives in user performance, I may prioritize our design for beauty, personal preferences, and pre-conceived notions over the hard facts of usability and performance. Going through the process of user performance modeling and working with CogTools shifted our ideas on interface design away from pure aesthetics to a combination of looks and performance. Usability was always a concern in our designs, but I realized performance could be tested and while various other factors such as flexibility, aesthetics, and ease of use play a role in a design decision, performance is one of the few that can be unbiasedly tested.

Where a design should be optimized for task execution times is a tricky proposition. In an ideal world every design should be optimized for task execution times because I as interface designers should strive to make the quickest, most usable systems I possibly can. However, it’s more complex than that because as noted earlier, there are other design considerations that need to be taken into account. Thus, designers have to pick and choose where to optimize for task execution times and where it isn’t worth it. Given our short experience with user performance optimization it seems that there are a few situations that should definitely be optimized for time. Any task that could be time intensive such as manually scrolling through a list of friends in a social media application (where a search bar could drastically reduce this operation) would be a good place to optimize for task execution. In a similar vein, any task that is time sensitive where users need to quickly complete the task and performance may be the primary design goal would definitely benefit from task execution optimization. Ultimately, tasks that have performance as a primary design goal would benefit from task execution optimization. Whereas, tasks that might value flexibility and ease of use over pure performance might not need execution optimization because there could be other design goals that could be optimized. The overall cognitive model that CogTool employs is the basis for our performance testing, which is both interesting and impressive!