Where's My Damn Bus
Final Reflection
Joey Goode

Introduction

Where's My Damn Bus is an Android application designed to help users coordinate bus arrival times with their schedules. Since MBTA buses rarely arrive exactly at their scheduled time, most people just head to the bus stop and wait for the next bus. Where's My Damn Bus attempts to assist travelers by letting them know the current predicted arrival times, based on calculations done by the MBTA within their real time API, so that they can better decide when to leave for the bus stop or seek alternative routes to their destination.

Problem

Bus stops are not places where many people want to spend their time. There is nothing worse than turning the corner and seeing your bus pull away from the stop as the realization dawns that one would have to wait at the stop for 15-45 minutes for the following bus, except if it was also raining on that day. Travelers often wait extensive periods at the bus stop, especially during off-peak hours. Many MBTA bus stops are also uncovered, creating problems in typical Bostonian weather. Finally, many buses travel through, or to, potentially unsafe neighborhoods. These neighborhoods often house many honest, hardworking people who use public transit to get to work each day, or to go into the city for a night out. However, not all buses go to all destinations, and sometimes a transfer is required. Transferring buses in the dead of night in a known unsafe neighborhood is not a pleasant experience and it's much safer to be moving than sitting still at a bus stop.

These problems manifest themselves in different ways. When people get robbed or mugged at the bus stop, it increases the crime rate with the slew of economic effects that has. When people are forced to wait in the snow, it decreases ridership. This means the buses themselves are less likely to recoup their expenses through fares and have to take money from taxes. It also means these would be riders are taking alternative forms of transit that often contribute more to traffic and pollution. Lastly, waiting at the bus stop is far from the most efficient use of a person's time. It makes them unhappy like a commute and during that time they contribute nothing to their own personal life or society. That time can be better spent elsewhere and these problems can be fixed by providing people with a quick, easy, and accessible application for planning the trip that they take frequently.

A solution like this would benefit many groups of people. The primary beneficiaries of such a solution would be commuters, students, retirees and anyone who uses the MBTA more than once a week. However, bus drivers, police officers, local employers and the MBTA itself would also benefit from the lowered crime, increased ridership and increased productivity that such a solution would create. Finally, public officials and taxpayers would surely relish the opportunity to spend tax dollars on things other than saving a public transit system that continues to lose money. But how to create such a solution...
Design

The design of Where's My Damn Bus is primarily centered around correcting the problems of two existing solutions that try to solve this problem. First however, we should discuss the problems that Where's My Damn Bus is not trying to solve, and why we're not trying to solve these problems. The main problem that Where's My Damn Bus is not trying to solve is the routefinding problem. People who know their at point A and want to get to point B can use one of dozens of online and mobile services such as Google Maps, which includes accurate data for travel by public transit. In order to accommodate all the diverse needs of the people who have such a problem, the Google Maps solution includes a lot of necessary overhead in both time cost and complexity cost.

In Where's My Damn Bus, we try to strip out a lot of that time cost and complexity cost by narrowing our goal. Since our goal is specifically to tell people when their bus is coming and get walking directions to a nearby stop, we nip a lot of the possibilities at the bud. If a traveler doesn't know what bus they need to take, they are better off using another sufficient solution, because the solution will need more data and input from the user to make that judgment. If a traveler doesn't know (at least generally) where a specific bus line goes, they should also use another solution, since we would have to display an entire MBTA map to them. We are targeting users that know what bus they want and have a general idea of where it stops. We expect most of our users to have a specific bus route, terminus, and stop in mind before they even get started.

There is, however, one existing solution that solves this problem. There is an Android and iPhone application called “MBTA Alerts” that solves the problem. However, MBTA Alerts fails in the intuitiveness and efficiency of its interface. MBTA Alerts requires users to first scroll through a list of route numbers to find the route they want. Then, users select a terminus for their bus, and lastly, choose the stop the want. We think this process could be a lot simpler by using information that we don't need to ask the user for. For example, instead of making the user sort through all the possible stops of a bus, why not use the user's location to determine what is nearby? If the user doesn't want nearby spots, they can enter a location to search near. This way our solution doesn't have to display every single bus route, inundating the user with information. The only holdover from the MBTA Alerts system is that the user must select a terminus from a list, but this list is not usually long enough to run onto the next screen. The result is that users can navigate the interface much more quickly.

The first problem that we decided to tackle was the problem of letting the user select a stop from those near them. Our first solution entailed a system that we would display the MBTA names for those stops, text strings like, “South Huntington Ave @ Huntington Ave.” These would be familiar to users as they are what is displayed and announced on the bus itself. However, this quickly proved to become confusing. First off, what order would we display the stops in? If they were ordered by distance, two stops on opposite sides of the users location could appear right next to each other. If they were ordered alphabetically,
we still had the same problem of users haven't to sort through a long list of stops. Displaying a list defeated the purpose of our solution: speed and efficiency.

We scrapped that solution in favor of a graphical approach. Android users are likely already very familiar with map controls, and so we decided to solve the ordering problem by not ordering them at all. By using a map with street names, we had to give up the MBTA text strings, but we avoided the problem that they solved entirely. We simply highlighted the stops themselves on the map with little icons. Initially, these buttons were red pins, because this is a familiar interface, but users found in paper prototyping that these indicators were unclear to their purpose. To solve the problem, we switched to a little blue icon of a bus.

The rest of the process was then simple, users would select a bus line and a terminus and we would display a list of times. Here again, we found another problem, which was that users didn't know which bus lines stopped at which stops, and so would have to search through the available stops to find their destination. This obviously detracted from the efficiency of the application, as we discovered when users tested our digital prototype. Our first solution was to overlay the MBTA provided bus map, with all the colored lines onto the top of our route map. However, this was very confusing for users as they had to consult a legend to indicate which bus route was associated with each color. We had gone from not having enough information to providing too much information, and so we instead added route numbers to each stop on the map display. Users could now know at a glance which bus lines stopped at each stop, without having to consult any sort of table or look-up.

The final big design problem that we encountered is that users weren't instantly familiar with behaviors that were intuitive and rational once the user knew they were available. Two key cases were selecting the bus icons, as users didn't know they were selectable, and returning to the map after they had opened a pop-up dialog. In these two cases, we opted to present the users with a tutorial. Initially, the tutorial only showed up once, never to be seen again. But we found that users often didn't read the tutorial text and then wondered about the functionality and couldn't get it back. So we instead opted for the more annoying approach of showing the tutorial every time until the user made a conscious action to disable them. By being more annoying, we found that users were more likely to read the tutorial before disabling them. Since they explained key functionality, it was important that the users read them.

**Implementation**

Our implementation centers around a system of pop-up dialogs and overlays. When the user starts up the application, they are immediately launched into a dialog. After selecting the location they wish to search from, they are presented with a map and asked to select a stop. When a stop has been selected, we then present them with another dialog to find out which bus they have requested. Once we have determined which bus the user wants, we provide a list of times. At any point after they select a stop, they may select a button that will take them immediately to walking directions to the stop.
One of the major flaws with our implementation was that, using Java Swing, we couldn't do multi-line text boxes with transparent backgrounds. This caused a lot of development time to be spent trying to align text-boxes, which didn't always work out well. Thus, some things were shown in horizontal sequence rather than the more readable vertical sequence. Other than this, however, there were no real implementation problems, and the implementation itself was really quite simple.

Evaluation

Before we evaluate testing itself, it is important to take in the effect of design discussions made before the application was even built, because there were no major overhauls before testing. One major change that showed up before the prototypes were built was a change from a transparent pop-up dialog to a solid one. Discussions with colleagues rapidly revealed that the lines of roads behind the text made the text more difficult to read, especially to people with poor vision. To illustrate the point, we made life-size mockups of the application and tried to navigate them without glasses on. If a user was in a rush, stopping to squint at our application was not what we wanted, and the transparency was just visual candy, not an essential part of the application. It was cut. We additionally cut a feature that caused the bus route number to show up between the termini of the bus route, with northbound/eastbound routes above the number and southbound/westbound routes below the number. This proved only to confuse users and make the list more difficult to read and scan, as they had a hard time figuring out the directionality from it.

With these changes, we created a paper prototype of the application. We wanted to see how quickly users could learn our interface, and so we created three tasks that walked through similar steps. We did see noticeable improvement in things like selection speed between the start of the testing session and the end of it. Most users, by the end of the test session, were only limited by our ability to manipulate the prototype quickly. In addition, most users liked the application and thought it fulfilled a need that they had.

However, we did discover some problems during this prototyping session. The vast majority of the problems we discovered were wording problems. We made changes. “Stop” became “bus stop.” The formatting on the times display changed from “mm:ss” to “mm minutes, ss seconds.” Users also kept trying to select the arrival times like buttons. We tried changing the appearance to be distinctive, but couldn't find something comfortable and intuitive. Failing that, we spoke to users and discovered a problem with the wording of the task. We changed the wording on the route selection screen to be more indicative of it's purpose. This seemed to alleviate the problem. We also made changes to the pin icons here, as mentioned above. Some users had difficulty selecting the pins precisely with their fingers and so we made the pins larger and more responsive.

After this, we moved to a digital prototype. We also made some changes due to feedback and discussion between the iterations. One discussed change that we wanted to test was using bus icons similar to Google instead of little red pins. We thought this would convey their purpose more meaningfully, and allow us to drop the tutorial with less pain. For this iteration of the application, therefore, we
dropped the tutorial. Tutorials often feel forced and cumbersome, and we wanted to see if the application can stand on its own without it. In addition, we moved the return to map functionality to the back button, as we thought this would be more familiar for Android users who are already familiar with that button.

We then put this version to a panel of heuristic reviewers. These heuristic reviewers did a fantastic job and were a major component to the success of the application. We immediately got complaints about removing the tutorial. One reviewer thought it was very unclear that the user needed to click on the little bus icons. We tried a variety of solutions relating to making the buttons look more click-able including beveling, but they failed to induce users to click them and so we put that tutorial pop-up back into the application. This related to one of the failings of a touch screen interface. In a computer based application, I would have simply had the button highlight when the user mouses over it, but there is no way to hover your finger over the screen of an Android device. Another reviewer asked for help and documentation. Knowing how much users hate reading through documentation, we decided to prefer tutorials to documentation. We were more likely to reach users with tutorials and leave them less frustrated. Users are likely to go to documentation after they are frustrated, at which point we've already failed. The tutorials solved the problem in a simple and Android-compatible way.

A reviewer also complained how long it took to get the application to offer you walking directions. This was one of the few complaints that took us totally be surprise. The idea that a user would want to get directions to a bus stop without knowing when the next bus was coming wasn’t something we thought users would want to do. However, clearly there was a need for this, and as we began to implement it, we began to see why. Suppose a user used our application to find out when the next bus was coming, got directions, and then put their phone away and set off. They get to an intersection, confused, and want to ask again. That seemed like a valid use scenario, and so we moved walking direction buttons on to each screen. [See Illustration 3] Users could now get walking directions as soon as they had selected a stop.

The remaining fixes were all fairly minor. We reimplemented the return to map functionality, while keeping the back button functionality, because some reviewers commented on the niceness of the back button, and others didn't like it and wanted a quick way back to the map. We had another case where a reviewer thought our color choice for the “You are here” arrow blended too well with the map itself. We responded by averaging all the colors in the map and inverting it for use as the arrow color, resulting in maximum contrast. That wasn't too appealing to look at, so we tweaked it into something nicer looking while still maintaining the contrast.

With these fixes, we took the application in for user testing. It turns out that we had done a good job so far, as our test users came back with very few complaints. However, they did do a good job finding bugs for us, and we were able to squash a lot of bugs that the heuristic reviewers had missed. However, there was one major complaint that had kept coming up and came up again. Users didn't like opening every stop to find out where the bus stopped. We tried overlaying the color bus map onto our map, but it turns out that there was a lot of cross-referencing to do in order to figure out what color corresponded with which bus. Instead, we elected to simply display a list of route numbers near each stop. Finished with user testing, we are ready to move on to implementing the back-end.

Reflection

We found the paper prototyping stage to be the most inherently useful stage of the design process. The combination of quick iteration time and the amount of feedback we got from it enabled us to quickly redesign problematic elements. If anything, we would have liked to have done a second paper iteration in order to have the chance to put our findings from the first session to the test. Once we built the digital version, it was far too costly to throw out entire systems.

We also found that scenario generation and development of tasks and personas didn't really help refine our focus on our goal, but rather served to side track us. We think a better course of action
would have been, since we already had an idea of what the application would look like when we had the idea, would have been to instantly prototype it and see if it worked. Then we could have gone into scenario generation with a better idea of what we were going to make and look for use cases there.

**Conclusion**

All in all, we thought the solution successful solved the problem of finding the arrival time of a bus with the proper combination of efficiency and learnability. Test users of our digital prototype were able to complete each task in less than 30 seconds having never seen the application before. Their times improved as they moved down the task list, implying that the 30 seconds was not the speed limit for our application. Test users confirmed our theory that there was a need for such an application, and we are pleased with the results of the development.