CS 361 - Software Engineering I

Winter 2015

Project B - Avoid Drunk Driving Through Safe Parking[1]

David Adams

Yong Lee

Andrew Kasnevich

Jasn MacManiman

Dustin Phelps

Anthony Smreker

[1] Available on Piazza at "Expanded Vision Statement: Avoid Drunk Driving through Safe Parking," https://piazza.com/class/i4buwx20j8h5b7?cid=171

First XP Cycle

List of All the User Stories Collected:

During the first XP cycle for our project, we reached out to Kevin Parks, the author of the "Avoid Drunk Driving through Safe Parking" vision statement and our client for the project. The main user stores that were a result of a dialogue with Kevin are below.

Web Page: Users can navigate via the web to the system's homepage.

User Account Creation: A user can create a username and associated login password. The credentials can be used to access their account from this point forward.

Moderator Account: A special account has privileges to erase Messages, Users, and Destinations.

Destination Creation: A registered user can create a destination and fill in attributes/properties concerning the destination.

Destination Research: A user can search for a destination and display basic information about the destination when selected. For example: Name, Address, Business Hours, Parking Lot size/availability, Parking alternatives, Message Board, etc.

Destination Message Board View: A user is shown the top comments/messages (by # of votes) per destination when the Destination's Message Board is viewed.

User Messages: Users can leave/post messages on a destination's page. Other users can reply to these messages. Messages are organized in threads.

Voting: Registered users can add confirmation points or some other means of verification to the destination itself and message threads about the destination.

Estimate of Tasks and Effort Required for Each User Story

As a team we went through our list of user stories and talked about the specific tasks that the user story would contain along with the amount of effort that we thought would need to be applied based off of our level of programming experience. We decided that it made the most sense to make hours our unit of measurement for effort estimates since we are doing one week XP cycles.

Web Page
Tasks:

- Consists of creating a working website landing page that the user can navigate to.
- Working with HTML and CSS to create the website and styling template design for the other pages
- Having something set up on the OSU server so that it is a live web page

Effort Estimate: 6 hours

User Account Creation
Tasks:

- Create a registration page that the user can navigate to in order to create a username and password
- Requires a back-end MySQL database to store the username and password
- Working with HTML and PHP to create the register page
- Handle the user input and store the username and password in the database
- Ensure that each user is assigned a unique username by checking the database to see if the user has already

Effort Estimate: 8 hours

Moderator Account
Tasks:

- Allow the creation and sign in of a special account. Requires adding another level of user creation to the register functionality
- Would require different page views throughout the site for the moderator in order to use the functionality to erase data
- Mostly creating PHP pages that are viewable only by the moderator

Effort Estimate: 10 hours

Destination Creation
Tasks:

- Allow only a registered user to create destinations

- Validate if the user is registered and then show them a separate page that has the added functionality of being able to add destinations
- Create a page to handle the form for adding a destination
- Would be all PHP and HTML and validating info from the database

Effort Estimate: 4 hours

Destination Research

Tasks:

- Allow any user to search for a destination and view its details whether they are registered or not
- This will be functionality built into the main page and accessible to all users
- Adding some PHP functionality to query database and show results for search

Effort Estimate: 4 hours

Destination Message Board View

Tasks:

- This is done after the voting system is set up as it relies on votes.
- Creating a dashboard view that would allow users to see top voted data based on comment data stored in database
- Have to create a whole new view for the website to view the message board

Effort Estimate: 20 hours

User Messages

Tasks:

- This is creating a reddit like thread system for each destination where users can comment and reply to messages all for that destination
- Would have to set up back end database to accommodate for all of the new entities in order to set this up

Effort Estimate: 10 hours

Voting

Tasks:

- Would only be for registered users and continues with the reddit style message board where users can upvote or downvote
- Database would have to be set up to accommodate this sort of data for the votes for each destination

Effort Estimate: 10 hours

Priority List of User Stories

After conferencing with Kevin Parks and establishing the initial user stories, we reached out to Kevin again so that he could determine the user story priority. Below is what he provided.

Priority 1

Web Page

Priority 2

User Account Creation

Priority 3

Destination Creation

Priority 4

Destination Research

Priority 5

User Messages

Priority 6

Voting

Priority 7

Destination Message Board View

Priority 8

Moderator Account

Pair Programming Summary

To decide how we organized into pairs for writing code for this project we looked at two factors. We first looked at what kind of experience each group member had in specific areas of writing code and dealing with systems that we were planning on using for this project. We each expressed our experiences and comfort levels with what we were trying to accomplish. The second factor we took into account was where everyone was located in the country, because we wanted to set up pairs in a way that would facilitate pair interactions that reduced time zone differences to the extent possible. We felt that this made the most sense because it makes it easier to schedule times to meet when a pair is in the same time zone or close to it. In our group we have one person in the Eastern Time Zone, three people in the Central Time Zone and two in the Pacific Time Zone. Based on where each person was located and comfort level of tasks, we were able to organize into three pair successfully.

Pair One (Andrew and Anthony)

Andrew and Anthony paired up based on Andrew being in the Eastern Time Zone and me being in the Central Time Zone. Also we both had some experience in the same areas but based on other experiences we didn't share we felt we could be able to learn from each other. We worked on the database creation along with user registration and the destination creation page. When we used the driver and co-pilot method we used video chat via Google Hangouts to be able to communicate while we worked. We also used email and GitHub to be able to share code. We did find it a little difficult to be able to find syntax error and to follow each other while being on video chat. Sometimes being able to communicate where we were stumped or confused about something was difficult and would probably be easier in person. We were able to overcome some of these problem by showing patience with each other and being descriptive about where we wanted our partner's attention.

Pair Two (Dustin and David)

David and Dustin paired up based on both of us being on Central Time. We had similar experience writing HTML and PHP from previous classes, but David had experience using Twitter Bootstrap which I had never used.

The first hurdle we had to overcome was sharing code to work on as a driver/co-pilot. We used Google hangouts to video chat, but the screen sharing limited us to having only one person being able to actually edit code at a time. We found a website called codeshare.io that worked perfected to allow us both to clearly see and edit our code in real time. Another problem we had when writing code split up among multiple programming pairs was hosting

code for our web site. To test functionality when different php files had been written by different pairs we had to post all of our code to GitHub and then each pair had to rehost all the code on their own.

We were also not really sure how to handle unit testing for a web site. We settled on using manual tests to check functionality for various fields on the web page and charting expected output vs. actual output for each field.

Pair Three (Jason and Yong)

Jason and Yong paired up based on both of us being on the west coast. Due to both of our schedules being incongruent we weren't able to meet for any pair-programming, and decided that the next best alternative would be to keep each other updated by email on any progress we made or questions that we had.

Neither of us have had experience writing unit tests before, so we started by researching our options. Since we are making a web-based application, the strongest contender initially seemed to be PHPunit. After extensive effort, we weren't able to get any tests to work with this and decided to try an alternative that Yong had found, called SimpleTest. By the end of the first XP cycle we had completed several test functions comprised of 29 tests total. The tests focus on ensuring the web page has the form fields it's supposed to and that when data is entered, it's correctly stored. Links are also tested for functionality.

The main struggle we had was being asked to start unit testing without any real introduction didactically. We did our best to find our own solution and begin implementing it, and the group's next unit testing programming pairs will continue to add more unit tests in addition to manual unit tests in the upcoming XP cycle.

Unit Testing Summary

The group discussed unit testing in one of our meetings during the first test cycle, and Jason and Yong were the programming pair assigned to unit testing in this cycle. Upon reviewing and attempting to employ the PHPunit and SimpleTest testing frameworks, the SimpleTest unit testing framework was selected for use on the group's website, see http://www.simpletest.org/.

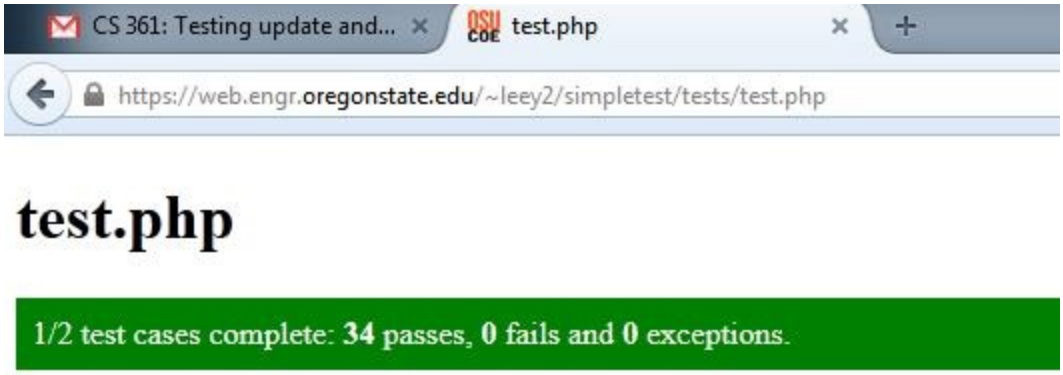
The initial testing focused on the current implementation of the website near the end of the first cycle, and was directed to testing for the presence of the required input form fields and that each for correctly stores the information it receives when data is entered, and the links to each page were also tested. To this end, each testing function takes the general form of a "$this->get('website URL')" request, and is followed by a series of "$this->setField" and "$this->assertField" commands.

For the first cycle round of unit testing, we tested the register page, the index page, and the add a destination page. For the register page, the tests make sure that all of the register input fields are there and are able to be set to the value that a user would enter in. The fields that were tested for were a username, password, first name, last name, and email. Then, it tests to make sure that the form submit correctly. For this testing, we were not able to test the email field because we were using the HTML5 email input type for the form, which the simpletest testing suite was not compatible with. For the main index page, we tested to make sure that the destination field and search form performed correctly and also tested to make sure that the register and sign in links worked. For the add a destination page, we tested to make sure that the fields for adding a destination were there and working correctly. Then, we tested to make sure that the form submitted correctly.

Our unit testing in the first cycle did not uncover any bugs in the code as implemented at the end of the first cycle. We believe this to be due to the fact that the developers knew what we were testing for and were able to make sure that they had all of the correct inputs for the forms that they were creating. Since, most of our unit testing dealt with testing forms and links within the site, this showed the benefit of creating unit tests because it helps the developers know how to lay out the site so that they will pass the tests. However, as discussed below in Andrew and David's second cycle pair programming summary, as the unit testing continued to develop, the unit tests did uncover bugs in the website.

For the second cycle, the search and login functionality is what was tested because those were the user stories being implemented. The unit tests for the login page were written very similarly to the register page mentioned above, focusing on the form inputs and the form submission. The login functionality created a new page that need to be tested, which was the index_loggedin page and so tests were also written for this page. For the search functionality, this was implemented within existing pages, so testing needed to be added to the main index page and the index_logged in page to test the search form.

Below is a screenshot showing the test file being run and the result of the 34 unit tests that we wrote for the project.



Acceptance Testing Summary

Our acceptance testing was primarily carried out by setting up a live version of the software system on our OSU web server, which was also connected to a database. Therefore we were able to see exactly what the web interface would look like to the end-user, which would then allow us to perform acceptance testing on the site in order to make sure that it does indeed do what it is supposed to. Since we don't have a real client which would perform the acceptance testing, we did it ourselves while maintaining a client or end-user's perspective.

The acceptance testing that we did after the first cycle consisted of testing the overall usability of the website, the ability of a user to register, and the ability of a user to add a destination.

One of the bugs that was found through acceptance testing of the functionality of the website was that the register button was too hard to see originally and the user was not able to easily see it. Therefore, we made the button similar to the sign in button so that it was easily recognizable.

Another bug that was found during acceptance testing was that when a user clicked on the register button and was taken to the register page and successfully registered, there was no way for them to get back to the main site page. Therefore, a back button was implemented in order to allow the user to get back. This was also a bug found within the sign in page and the destination creation page and the same fix was applied.

Similarly, another bug we found was that if a user clicked the sign in button, but hadn't already registered, then they would have had to go back the main page and then click the register button in order to register. This series of steps was unnecessary and the fix was to add a register button within the sign in page that would allow the user the ability to go straight to the register page.

While doing acceptance testing on the register functionality, a bug was noticed that the error messages were being placed at the bottom of the screen and were hard to see. Therefore, the error messages were moved so that they would be at the top and were in a larger font so that the user couldn't miss them. This same thing was also found in the destination creation page and the same fix was applied.

Second XP Cycle

During the second XP cycle, we went back and spoke with Kevin and gave him our new list of user stories, which was the same list we originally had, but we removed all of the completed user stories. We had a couple of issues that we talked with Kevin about in regards to the user stories and he agreed that all of our points were valid. The main point was that there should have been a separate user story for a Login functionality versus register functionality. We told Kevin that based off of how we implemented the register functionality, we didn't yet have a fully working Login system where I user could come to the site and login if they were already registered. Therefore, we added this as a user story and Kevin agreed that this was the right choice. The new list of user stories re-ranked by priority are listed below.

Kevin thought that the newly added login functionality should be the new first priority user story, which we agreed upon. Aside from this, Kevin did not see any reason to change the priority of the user stories. So, the remaining user stories kept the same sequential order as before except they were updated to account for the newly added user stores and the completed user stories.

A few of the effort estimates were revised as well to account for some areas that we thought would end up being harder than we originally planned. For instance, the messages and voting functionality would have required a very detailed re-working of our back end database structure to accommodate this. After talking about it, we realised that the more time consuming user story would actually be implementing the messages as this would require the back end side of the system to be mostly worked out. Therefore, we reduced the time slightly for the destination message board view and increased the time for the user messages and voting due to the increased workload of setting up the database infrastructure that these would take.

Completed User Stories:

Web Page

User Account Creation

Destination Creation

User Stories Listed by Priority:

User Login: A user is able to successfully log in to the site and the site shows that they are logged in. Logging in to the site allows the user to create a destination.

Revised Effort Estimate: 5 hours

Revised Priority: 1

Destination Research (Search): A user can search for a destination and display basic information about the destination when selected. For example: Name, Address, Business Hours, Parking Lot size/availability, Parking alternatives, Message Board, etc.

Revised Effort Estimate: 4 hours
Revised Priority: 2

User Messages: Users can leave/post messages on a destination's page. Other users can reply to these messages. Messages are organized in threads.

Revised Effort Estimate: 16 hours
Revised Priority: 3

Voting: Registered users can add confirmation points or some other means of verification to the destination itself and message threads about the destination.

Revised Effort Estimate: 14 hours
Revised Priority: 4

Destination Message Board View: A user is shown the top comments/messages (by # of votes) per destination when the Destination's Message Board is viewed.

Revised Effort Estimate: 14 hours
Revised Priority: 5

Moderator Account: A special account has privileges to erase Messages, Users, and Destinations.

Revised Effort Estimate: 10 hours
Revised Priority: 6

Pair Programming Summary

To decide how we organized into pair for writing code for this project we looked at two factors. We first looked at what kind of experience each group member had in specific areas of writing code and dealing with systems that will use what we need for this project. We each expressed our experiences and comfort levels with what we were trying to accomplish. The second factor we took into account was where everyone was located in the country, because we wanted to set up pairs in a way that would facilitate pair interactions that reduced time zone differences to the extent possible. We felt that this made the most sense because it makes it easier to schedule times to meet when a pair is in the same time zone or close to it. In our group we have one person in the Eastern Time Zone, three people in the Central Time Zone and two in the Pacific Time Zone. Based on where each person was located and comfort level of tasks, we were able to organize into three pair successfully.

Pair One (Yong and Dustin)

Yong and Dustin paired up to implement the user story for “Destination Research”. Being in different time zones and with different availability, it was difficult to find a time we could meet up, but we were able to meet long enough to implement the user story. Lessons learned from pair programming in week 1 such as using codeshare.io to be able to both view code and already having everything in place to test changes really made it much quicker to get things done this week than the previous. In the process of implementing the user story we were also able to find a bug in one of the other user stories previously implemented and we were able to communicate it with other members of the group to get the bug fixed. After the user story was completed, we uploaded our changes to GitHub so they would be accessible for the entire group.

Pair Two (Jason and Anthony)

Jason and Anthony paired up to work on the user story for implementation of the sign in button. We paired up based on our schedules matching up the best for each of us. We used Google hangout and codeshare.io while we did our pair programming. We met for a couple of hours to work on implementing the sign in button on our homepage along with setting up the session once the user is signed in. We ran into a couple of problem while coding but being able to share our code via codeshare.io which was a big help. We were able to find solutions to our problems fairly quickly because we were able to see exactly what the other was talking about. This second cycle was probably a little easier than the first because we had a better idea of what we were doing and what our expectations were. The whole group having a better idea of the vision of the project has also made the second cycle a bit easier. We are all more

on the same page about what we expect the end product to be. This has made the second pair programming cycle a little bit different than the first and has allowed us to be more productive.

Pair Three (Andrew and David)

Andrew and David paired in the second cycle because we reside in the Eastern and Central time zones, respectively, and we were interested in working on the unit testing tasks. This meshed well with our project teammates, who had availability earlier in Week 10 to address the additional programming tasks, and we both had time later in the week to perform unit testing once the programming work had been completed. We utilized a Google hangout for audio communication during our pair programming session, and used a joint codeshare.io session to collaboratively write the code. We also both used the screenshare option of Google hangout throughout the process to show each other relevant information that was on our screens.

As neither of us had prior experience with unit testing, we began by researching the SimpleTest framework and reviewing and assessing the unit tests written by Jason and Yong. This led us to discovering a few bugs with the already written unit tests, which we resolved. As noted above, other group members worked on the login and account creation functionalities during the second XP cycle, and we revised the unit testing to accurately test the most recent implementations. This work led us to detect a few bugs in these functionalities, which we shared with the group via email. In addition, we expanded the unit testing to address the destination creation site feature by preparing testing function

Following a consultation with Yong, our unit testing ended the second cycle with 34 written tests, and the project website passed each of these tests. Andrew and David found using codeshare.io along with a Google hangout for audio communications a beneficial collaborative programming setup, as both partners were able to implement code changes in the shared file, while verbally communicating with each other in real time.

Second XP Cycle

REFLECTION A summary of the ways in which you found XP preferable to waterfall, and the ways in which you found waterfall preferable to XP – strengths and weaknesses of each process, in your experience (1 page)

Upon consideration of the Waterfall and XP methods employed for Project A and B, respectively, the group recognizes considerable strengths and weaknesses in each approach.

The Waterfall approach allowed us to spend more time planning up front and gave us a much more thoroughly documented design to use when looking at the various features. We recognize, however, that a significant disadvantage of this approach is its lack of flexibility. For example, if the customer had evolving needs that they wanted updated or changed on a cycle-to-cycle basis, there would have been no good way to change the plan with Waterfall, short of simply discarding the already-developed plan and performing the design and documentation process again from the beginning.

The XP approach, on the other hand, allowed us to be flexible with our design. If we realized something wasn't working while we were implementing it, we could simply change it on the fly to make it work. We also believe that XP is a better fit for a small, relatively simple task such as this project, since we were able to get a basic structure created and build on it iteratively. Another advantage of XP, would be the constant testing that takes place during the design and implementation. This allowed bugs to be found and fixed immediately instead of waiting until the very end to find them all. That said, the disadvantages of the XP process are a result of its strengths. For example, the flexible nature of XP meant that the implementation we had developed by the end of our development cycles could be a great deal different than what we had decided on in our original plan. Depending on the customer, this may be a problem in the long run.

In the end, it was obvious that neither Waterfall or XP is perfect for every project. The project itself should be the determining factor in which design method is used. If there is a clear, definitive picture of what the customer wants and little to no chance those needs will change or evolve, we believe Waterfall is the better option. If an initial deliverable product needs to be produced rapidly and developed or changed over time, or if the customer's initial requirements are uncertain and likely subject to change, the XP approach seems to be a better fit.

Instructions on How to Run the Project

Our project is a live website, which is being hosted on OSU's web server that we have access to as students. Therefore, there are no set instructions to get the site up and running other than going to the link provided here:

http://web.engr.oregonstate.edu/~adamdavi/CS361/index.php

Once you follow this link to the homepage, you will have several options. You can register an account and then sign in along with searching the database for a certain destination. Additionally, you can view all of the destinations that have been entered in the database. Once you register an account and sign in, it will take you to a similar looking home page, but you will have the ability to add a destination to the database. You can then logout of the site from that page.

The GitHub repository that holds all of our code that we used throughout the project is located here:

https://github.com/CS361-ProjectB/361ProjectB

All group members are owners of this repository. This allowed us all full access to the repository and files and was an ideal way to set this up for a team project.

This GitHub repository shows the exact code that we also turned in for the project. Please note that if you are going to try and get this code running on a local testing environment, you have to first create the databases that the site uses to store the user and destination information. There is an SQL file in the code that we submitted that shows the create queries that we used to create the tables. Then, you would need to update the connect.php file to reference the database that is storing the tables that can be used for this project. For us, we each had to set this up locally using our OSU databases and then change the connect.php file to connect to our database so that we could get a local version operating on our individual OSU accounts.