Lendr

Bringing Communities Together

Customer: Martin Stryffeler

Development Team: Michael Byrne, Lunkai Chen, Alisha Crawley-Davis, Sara Sakamoto, Brandon Swanson, Matthew Walz

Objective: Use eXtreme Programming to implement a web application which allows people to create neighborhood groups for sharing things that are useful but not always in use.

Live Demonstration of Lendr Implementation:
http://web.engr.oregonstate.edu/~swansonb/Lendr/login.php

Cycle 1

User Stories

After reviewing the vision statement provided by our customer, we developed the following user stories.

Create a group
A user of the app can create a group for sharing items locally

Track an item
A user can check which other user currently has a borrowed item

Search for an item
A user can search to see which items are available for borrowing

Register/log-in
The users of the app can register and log in to the site

Join a group
A user can join a local group for the purpose of sharing items

Offer an item
A user can offer an item to be borrowed

Borrow an item
A user can borrow an item by selecting it from a search

Return an item
A user can return a borrowed item

Estimate of Tasks and Effort

In our first group meeting with decided that one hour would be equal to one unit of time. We then worked together to estimate the time it would take for each task based on the average experience level.

Create a group
Total Cost: 5 units
A user of the app can create a group for sharing items locally
Tasks:
- Set up database to store people and groups (Cost - 2 units)
- Create a web page to query database and create a new group (Cost - 3 units)

Track an item
Total Cost: 8 units
A user can check which other user currently has a borrowed item
Tasks:
- set up database schema of items and people and Item-people (Cost - 2 unit)
- query database for items belonging to owner and currently lent (Cost - 1 unit)
- create item html Builder (JS receive JSON, make html element) (Cost - 3 unit)
- display owners items highlighting the borrowed items (Cost - 2 unit)

Search for an item
Total Cost: 6 units
A user can search to see which items are available for borrowing
Tasks:
- Create webpage for search entry forms and results (Cost - 3 units)
- Query offered/available database for desired or similar items 1 (Cost - 1 unit)
- Display items or negative result 1 (Cost - 1 unit)
- Offer new search 1 (Cost - 1 unit)

Register/log-in
Total Cost: 7 units
The users of the app can register and log in to the site
Tasks:

- Set up database to store the user’s information (Cost - 2 unit)
- Create a web page for the user to input personal information (Cost - 1 unit)
- Upload the information to database (Cost - 2 unit)
- Create a web page for the user to input username and passcode (Cost - 1 unit)
- Display login status (Cost - 1 unit)

Join a group
Total Cost: 5 units
A user can join a local group for the purpose of sharing items

Tasks:

- create web page listing groups with button to join (Cost - 2 unit)
- setup query to join group and update database with user membership to group (Cost - 2 unit)
- display joined groups to users (Cost - 1 unit)

Offer an item
Total Cost: 5 units
A user can offer an item to be borrowed

Tasks:

- Validate that user exists (Cost - 1 unit)
- Create web page with item entry form (Cost - 2 units)
- Validate that item exists or create item (Cost - 1 unit)
- Check item in or add item to database (Cost - 1 unit)

Borrow an item
Total Cost: 2 units
A user can borrow an item by selecting it from a search

Tasks:

- add button to search results with query to request borrowing (Cost - 1 unit)
- change state of item to borrowed (Cost - 1 unit)

Return an item
Total Cost: 6 units
A user can return a borrowed item

Tasks:

- Create web page where user can view all items they are borrowing (database query of items being borrowed by user) (Cost - 3 unit)
- Add button to each item in list on web page which ‘returns’ the item (Cost - 2 unit)
- Change state of item to available in database (Cost - 1 unit)

Priority List

After determining the effort required for each story, we decided on a timeline for the first cycle and estimated we could accomplish 24 units of work. We contacted the customer via email with this information. The following are the user stories the customer wanted us to implement in the first cycle in the order of priority that he assigned:

Register/log-in
Total Cost: 7 units
The users of the app can register and log in to the site
Tasks:

- Set up database to store the user’s information (Cost - 2 unit)
- Create a web page for the user to input personal information (Cost - 1 unit)
- Upload the information to database (Cost - 2 unit)
- Create a web page for the user to input username and passcode (Cost - 1 unit)
- Display login status (Cost - 1 unit)

Offer an item
Total Cost: 5 units
A user can offer an item to be borrowed
Tasks:

- Validate that user exists (Cost - 1 unit)
- Create web page with item entry form (Cost - 2 units)
- Validate that item exists or create item (Cost - 1 unit)
- Check item in or add item to database (Cost - 1 unit)

Borrow an item
Total Cost: 2 units
A user can borrow an item by selecting it from a search
Tasks:

- add button to search results with query to request borrowing (Cost - 1 unit)
- change state of item to borrowed (Cost - 1 unit)

Return an item
Total Cost: 6 units
A user can return a borrowed item
Tasks:

- Create web page where user can view all items they are borrowing (database query of items being borrowed by user) (Cost - 3 unit)
- Add button to each item in list on web page which 'returns' the item (Cost - 2 unit)
- Change state of item to available in database (Cost - 1 unit)

Summary of Pair Programming

For the first cycle, we organized into three sets of pairs: Brandon and Sara, Matthew and Alisha, Michael and Lunkai.

Brandon & Sara
Brandon and Sara decided to work together as their schedules matched up the best for the first cycle. They met to do the pair programming using google hangouts with the screen sharing option. Brandon started out as the pilot and Sara was the copilot. The biggest issue faced was with a gap in experience and comfort levels with PHP and database. The majority of the team has completed Web Development using PHP, but Sara is in it now and using Node.js. This was a difficult problem to overcome, but one solution was to thoroughly review code from a completed user story before meeting. The other problem encountered was switching roles between pilot and copilot. Google docs has remote desktop control, but it is a little cumbersome to use. To solve this issue, Brandon kept control of his desktop, even when Sara was piloting.

Matthew & Alisha
Matthew and Alisha worked together as their schedules matched up and they both live in the same area. They decided to meet in person. Matthew was the pilot and Alisha was the copilot. This worked out well since they were in the same place and were able to easily discuss any issues during coding. One problem they had was just with logistics - the work took longer than they had the room they were working in scheduled for. The problem was solved by finding

another location to finish up! One thing they discovered that was nice about pair programming was that debugging was easier with two people both working on the same problem at once.

Michael & Lunkai

Michael and Lunkai were the third pair to match up. They worked together to do pair programming using google hangouts with the screen sharing option. Lunkai started out as the pilot and Michael as the copilot. Because they could imitate other groups work to write the offer item page, it was very easy to finish it. However, Lunkai faced a problem that he could not modify his engineering account. It meant that he could not debug. At the end, he found a way to fix the problem by using XAMPP to set up a local server and database.

Summary of Unit Tests

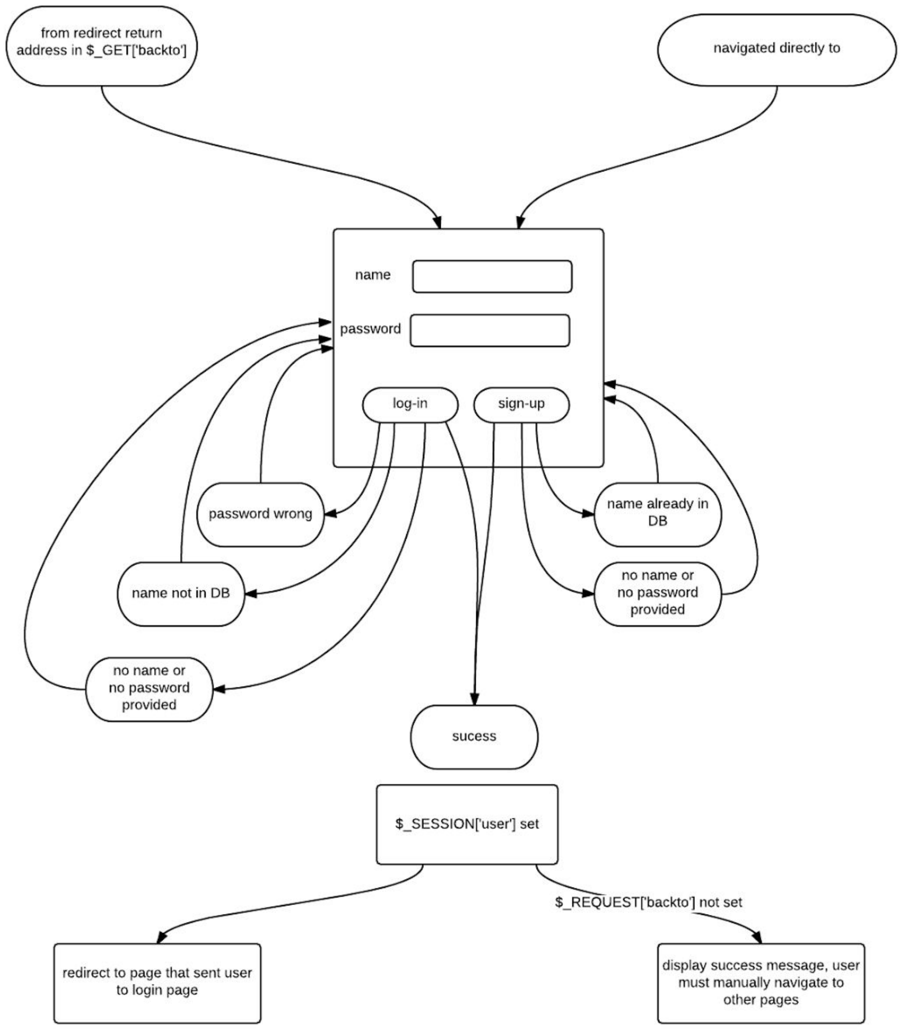
We wrote unit tests to ensure that the features/stories we implemented worked well. In order to do this, we first wrote tests that would break/not work if the feature was not correctly implemented but would if it was.

As none of us had experience with test driven development, we tended to focus on unit tests that we had to run manually instead of those that could be done automatically.

We wrote test cases about the returning an item story. Using these test cases, we found a bug where, when the item was returned, it was correctly removed from the borrower but not added back to the list of items owned by the original owner, as was the intention.

For the offer item page, we were able to modify code that other groups had already finished. Because there was not much to change, our unit tests ended up not finding any bugs!

For the log-in and sign-in story, we created the following diagram to illustrate the possible user actions and the desired outcomes. We tested the component against each action and found a bug in the redirection of the user after logging in. The desired outcome was that the user would return to the page they had come from, but it was not redirecting correctly.


from redirect return address in $_GET['backto']
navigated directly to
name
password
log-in
sign-up
password wrong
name not in DB
no name or no password provided
name already in DB
no name or no password provided
sucess
$_SESSION['user'] set
$_REQUEST['backto'] not set
redirect to page that sent user to login page
display success message, user must manually navigate to other pages

Summary of Acceptance Tests

As per the requirements of the assignment, we ran the acceptance tests ourselves (as opposed to having the customer run them.) Like the unit tests, we started with acceptance tests that could be run manually.

For the Return an Item story, we ran acceptance tests that showed what should be in a user's list of items before and after they decided to return an item. The acceptance testing did not yield many bugs, per se, but did show us things that would have been unacceptable to the client. For example, we found a typo and we found a way to enhance the display in a way that was not in the original design but made the website more usable.

For the Sign-Up and Log-In stories, we did the acceptance testing after our unit tests. As with the Return and Item stories, we did not find any bugs, but we did find issues with the visual layout of the user interface. To fix the problem, we added CSS styling to increase visual appeal and make it more consistent with the other pages. We felt that we were limited by doing our own acceptance testing because we didn't get an outside perspective.

For the Offer Item story, we did acceptance testing. Again, because we were able to modify already completed stories, our acceptance testing did not find any bugs or issues!

Cycle 2

Summary of Changes

User Stories

In the first cycle, we were able to complete three of the four customer-prioritized stories. Therefore, the list of user stories presented to the customer for the second cycle was shorter than the first. We presented the user with the following stories:

Borrow an item
A user can borrow an item by selecting it from a search

Create a group
A user of the app can create a group for sharing items locally

Track an item
A user can check which other user currently has a borrowed item

Search for an item
A user can search to see which items are available for borrowing

Join a group
A user can join a local group for the purpose of sharing items

Estimates

We ended up not changing our overall estimates for the second cycle, as what we learned from the first cycle showed that we both overestimated and underestimated some tasks involved in each user story. For example, we found that we assumed that some stories would be implemented before other stories when figuring out the estimates, but the customer did not necessarily choose the stories in that order. So we had to implement some stories earlier in order to finish the stories that the customer wanted. This made our estimates incorrect in that we needed more units than we thought. However, we also found that once we implemented some user stories, the rest took much less time than we estimated. It ended up that our overall estimates for each story were valid, but how much time we thought we would take on each task were incorrect. Because of this, we presented the same overall estimates to the customer.

Priority List

We were unable to complete one of the prioritized stories in the first cycle, Borrow an Item. We weren't sure if the customer's priorities had changed for the second cycle so we did not make the assumption that it was prioritized again. In the end, the customer did give high priority to Borrow an Item, but did not give it the highest priority. The stories chosen by the customer in order of priority for Cycle 2 were:

1. Search for an item
2. Borrow an item
3. Create a group

Summary of Pair Programming

Because the experience of meeting in person was nice but unique for this program, Alisha and Matthew paired up again for the second cycle. Sara and Michael paired up for the second cycle using Google Chat and Lunkai and Brandon were a pair also using Google Chat.

One negative difference was the lack of time with the holiday to find times to meet. The second cycle was more difficult from a scheduling standpoint because of the Thanksgiving Holiday. It was much harder to find times when people could meet. It was evident that everyone had less time for this cycle than the first which put a lot of pressure to accomplish more in each session.

Sara and Michael ran into technical difficulties because of bad weather affecting Michael's internet connection. After trying to make the voice chat work, they resorted to screen share with instant messenger. This made it very difficult for dynamic collaboration because the driver could not code and chat at the same time. Because of time-zone differences and scheduling conflicts, it was not possible to try again when the connection was better.

A positive difference from the first cycle was that after one round of pairing up, there was less of a gap in experience levels. Already having the general look and feel of the program down made the work quicker and easier to share across pair programming teams.

Because Matthew and Alisha paired up both times, their pair programming experience went very smoothly the second time. They each better understood the other's strengths and weaknesses, leading to more efficient use of the time.

Brandon and Lunkai were the third pair to match up. They worked together to do pair programming using google hangouts with the screen sharing option. Brandon started out as the pilot and Lunkai as the copilot. Their job was finishing other's work. They focused on the search page, adding new features and improving other's work. The most difficult part was searching items by users' group. Because a user can have multiple groups, it was hard to declare how many groups that the user could choose to search. They tried multiple ways to fix syntax and logical bugs. Finally, they ended up using a group_selected counter and foreach to implement the search feature.

Summary of Unit Tests

During the second cycle we were able to create more automated and repeatable tests by loading the pages within iframes. Each test case is attempting to test a specific expected behavior of the part of the system being tested. The relevant parameters and expected state of the system are displayed along with each test. This allowed us to quickly ensure that changes we made had not compromised the expected behavior of the site.

The pages we were able to make such automated unit tests for where the Search and Login pages and they can be seen here:
http://web.engr.oregonstate.edu/~swansonb/Lendr/testsearch.php
http://web.engr.oregonstate.edu/~swansonb/Lendr/testlogin.php

Summary of Acceptance Tests

Using a similar technique as the automated unit tests we were able to create a page in which an interaction that uses all of the features of the Lendr system could be observed. By logging in two different users and loading different pages one after another in iframes we were able to demonstrate the following sequence of events:

- MrTester offers a new item
- MrTester can see the new item offered in his list of offered items
- MissesTester searches for items from people in the group ‘MrTesters’s Club’
- MissesTester borrows the newly offered item
- MrTester can see that the item is borrowed by MissesTester in his list of offered items
- MissesTester can see this item in her list of borrowed items
- MissesTester returns the item to MrTester
- MrTester can see that the item has been returned in his list of offered items

This acceptance test suite can be seen here:
http://web.engr.oregonstate.edu/~swansonb/Lendr/testofferborrowreturn.php

Running this acceptance test allowed us to quickly ensure the functionality of a majority of the prioritized user stories. We also managed to spot at least one bug or unexpected behavior using this test. Upon borrowing an item from the search screen, the same search is performed but the group filters that had been selected were not maintained, making it difficult to quickly verify that the item was borrowed. After this was noticed during acceptance testing, we added this step to the borrow request and corrected this problem.

Reflection

During the course of this class, we found many advantages and disadvantages to both XP and waterfall. For XP, we found the main advantage was the speed at which we were able to deliver user stories to the customer. It was nice to be able to have something to quickly show the customer. It was also nice to have the customer be a member of the team. It meant that we would not accidentally go too far down a path that the customer did not agree with, as might happen when using a less agile method.

One disadvantage of XP we found at the beginning is that none of us were very comfortable with pair programming. At first it seemed inconvenient and inefficient. However, as we proceeded with the project, our feelings about pair programming became mixed. On the one hand, it was hard to have to match up schedules, especially as we were in different locations. On the other hand, it was really nice to have two sets of eyes on the code, which was definitely helpful for debugging. The pair programming was definitely more of an advantage for copilots who were not as knowledgeable about the technologies we were using, as they were able to learn from the pilots. If this project were to last longer than two weeks, this would have been an advantage as the copilots would have more time to implement the newfound knowledge.

However, since we only had two weeks, the pair programming felt more like a burden, where it slowed down the pilots for no reason. So overall, the pair programming part of XP was both good and bad.

The advantage of the Waterfall method is that once the documentation is finished, you spend less time changing things - you think everything through before you start. So, while it takes more time before you start coding, once you do, you should know exactly what you should be doing and things should not change very much. Because this particular system was pretty small with simple requirements that were easy to understand, Waterfall probably would have been a better choice.

To sum up, we found Waterfall preferable to XP when the system was relatively small because there were fewer changes/surprises once everything was designed. In addition, we did not feel comfortable with pair programming, at least not at first. We found XP preferable to Waterfall in that we thought it was really nice to be able to deliver something quickly to the customer and we could see the benefits in designing tests before we began, as evidenced by the bugs we caught during unit tests.

Live Demonstration of Lendr Implementation:

http://web.engr.oregonstate.edu/~swansonb/Lendr/login.php