CS4500: Software Development
Spring 2018

Team 26: Coders Unlimited

Spoiled Tomatillos Final Report

George Abinader, Joseph Donovan, Benjamin Faucher, Madeline Leger

The Problem

Our client, CCIS, recognized a market need for a social platform providing consumers a one-stop-shop to find and discuss movies with their friends, family, and other acquaintances. To fulfil this need, we at Coders Unlimited were tasked with creating the website Spoiled Tomatillos -- a social media platform dedicated to movies and the responses that they generate from users.

Our design specification included the following:

The service would require users to create accounts in order to interact with the features of the site. Users would be presented with an interface that allows them, first and foremost, to look up movies at will and learn about their plot, cast, awards, availability at local theaters, and possibly more. The interface should be similar to IMDB, Netflix, Amazon, Fandango, and similar existing services.

As a social media platform, the service should mimic the “friends” feature seen in many social media applications today; it should incorporate a way to find friends they know who use the platform and identify that they are friends on the platform. Once a user has friends, users should be able to recommend movies to their friends, either just to watch in general or to go see together. Users should also be able to create groups for the purposes of interacting with other users on the site over special interests.

Personalization is also a key feature. Users should be able to rate and review movies they’ve seen. From this data, the platform should be able to then offer recommendations for movies they haven’t seen. The recommendations should also factor in data such as external critic ratings, site-wide ratings, and other users on the site with similar taste in movies.

The client, CCIS, wants to be able to maintain Spoiled Tomatillos using members of their own staff. They will be responsible for moderating the user data and content that is posted and generated as well as acting as system administrators to keep the service working as intended. To this end, the platform should also include an administrative back-end that shows telemetry on site functions such as recommendation effectiveness, user retention, as well as direct control over user account data and statuses.

The Results

We codified the project specification into a set of 34 discrete, testable use cases that were then recorded in Jira (these are detailed in their own report). We divided the use cases into 5 categories: End user profiles, End user movie interactions, Group usage, Site moderation, and Administration. The breakdown of completed use cases out of total use cases by category is as follows[1]:

End User Profiles	4/7
End User Movie Interactions	3/7
Group Usage	1/12
Site Moderation	0/3
Administration	0/5
Total	**8/34**

Qualitatively this means we implemented only a fraction of our original intended functionality in the time we had available. What this doesn't account for though is all the devops work that went into making this prototype happen. We completed 21/23 devops tasks intended for this release[2]. This included setting up server instances on AWS, maintaining documentation on Jira, setting up our own SonarQube server, hooking up integrations between Jira, GitHub, Jenkins, and SonarQube, and lots and lots of time figuring out Jenkins. 12 of those 23 tasks were Jenkins related.

In terms of quality, we achieved a reasonable quality standard based on SonarQube metrics of the Java portion of our codebase. We achieved A ratings in reliability, security, and maintainability, which amounts to no bugs or vulnerabilities and relatively few "code smells" for the size of our codebase. We achieved 98.1% statement coverage and 60.6% branch coverage[3]. Qualitatively, our code base is extremely organized, obeying the standard Java organization scheme of folders mirroring the java package scheme of the code and taking advantage of object oriented paradigms to modularize functionality.

---

[1] Sources:
https://cs5500-jira.ccs.neu.edu/secure/DataplaneReport!default.jspa?report=5bda7860-a571-42e5-af04-1878b24b752b,
https://cs5500-jira.ccs.neu.edu/secure/DataplaneReport!default.jspa?report=6f6aee3e-cb9c-425d-ae00-f8468a60472f

[2] Source:
https://cs5500-jira.ccs.neu.edu/secure/DataplaneReport!default.jspa?report=23327c7c-49c7-435d-a63e-841a02645281

[3] Source: http://qube.codersunltd.me/component_measures?id=edu.northeastern.cs4500%3Ateam26-st-app-master

The Team's Development Process

As a team, we were able use a development process that worked well based on everyone's comfort level with certain technologies, as well as each individual's personal strengths. We were fortunate enough to have each team member have some solid background knowledge in at least one main area of the development pipeline: Maddy had a solid working knowledge of front-end technologies like jQuery, Ben had experience with Amazon Web Services, George was knowledgeable with SQL, and Joe had some previous experience with Java web development as well. Our preliminary discussions with one another revealed this diverse background knowledge and experience that we had as a team and that set the tone for the development process that we would implement once Sprint 1 began.

We decided it was best to, at the beginning, have each team member focus on developing within their area of expertise, per se, and later on in the semester we would consider trying out some new areas of development once the application's roots had been established. In hindsight, we are happy we chose this strategy, as we were able to get the application up and running relatively quickly while each team member utilized their strengths as a developer.

Process-wise, we came to an agreement that it was important for us to utilize JIRA as much as we could in order to have a better sense of what each team member was to work on. Early on, once we had our use cases established, we were able to add them all into the JIRA system and design a relatively elaborate timeline and schedule for our development process. The team agreed upon creating a variety of epics that our use cases, which were implemented as stories on JIRA, would each fall under. When it came time to select a specific feature to work on, the team came to a strong understanding as to how we would indicate, using JIRA, what each specific team member was working on. Each story (use case) would have their own subtasks that would be specific to the implementation details that we desired. For instance, we had one use case that stated "As an end-user, I can login to the application." These use cases were intended to span a multitude of layers within our application and represent functionality that would run through each layer so we could create subtasks that we could work on individually. So, in this use case example, we had subtasks such as "Front-end support for logging in a user" and "Back-end support for logging in a user" that were each assigned to different team members. An area of improvement that we noticed was that we could have been much more descriptive in our JIRA subtasks as to how exactly we intended certain implementations to go. We also should have used JIRA's "blocking" relationships in certain cases. Sometimes, we would be concurrently working on subtasks with a story that depended on one another's implementation in order to work correctly. For example, the front-end implementation of a API endpoint cannot be done until the API has been created in the back-end. We frequently ran into issues like this and could have utilized JIRA in a better way in order to enhance our development process.

Some of the team's strongest development process ideas came via Slack. We were able to use Slack's nifty integration tools with other systems such as Github, Uptime Robot, Jenkins, and Standbot in order to keep everyone up-to-date on the status of the system and the team as a whole. Each team member would be prodded with a message from Standbot each day, asking them to perform a virtual stand-up via Slack. Unfortunately, we did not do the best job of consistently inputting our statuses on a day-to-day basis, which is something we regret as a team. However, our frequent communications with one another via Slack was the strongest aspect of our team. We would be alerted when Jenkins builds would run, when team members would create a new pull request on Github, and when the site would unexpectedly go down. These handy notifications seemed to remind the team about the other members' status and would frequently generate useful discussions amongst the team.

The place where our team believes that we could have done considerably better was in the timeliness of our working being done. Frequently, we would find ourselves late in the sprint with a lot of requirements yet to be fulfilled and a sense of confusion as to what our plan would be. Most of the time, fortunately, we were able to get back on track in the final few days of the sprint and deliver what was asked of us. However, had we been on the same page earlier on in the sprint, we think that we could have been able to implement more features and save ourselves some stress overall.

Another part of the development process that we had intended to do, but unfortunately were unable to really accomplish, was give each other a chance to work on a different part of the system that we had not worked on yet in the project, or even at all in our respective computer science backgrounds. A few team members were able to expand a bit into areas they were a bit unfamiliar with, but overall we would have liked to have done that more. We, as a team, saw that as we went on further into the semester we were finding ourselves busier with other classes and schoolwork as well, limiting our time to consistently work on the project throughout the sprint. Although this disappointed us, we all understood one another when it came to making sure everyone was getting their work done amongst their variety of classes.

Overall, the team believes that we were able to set up a solid development process throughout the life-cycle of this project and that were were able to do well acting in an Agile environment, as we constantly were changing minor requirements and details while responding quickly and resoundly. We now understand, as we move forward in our careers, how important communication and the development process is as a whole towards the success of the project at hand.

Project Retrospective

The model of the class project was a great representation of the typical software development process many students will experience as a software engineer. Our group grew our skills in agile development, development operations, and UI/UX design through our work on the project. Our main concerns were with the project-related homeworks, lack of continuous integration/deployment instruction, and overall requirements of the project.

The homeworks spaced throughout the first half of the semester were clearly meant to prepare students for work on the project itself. However, feedback on the homework was not given quickly enough for us to integrate it into the related work. For example, the UML diagram homework feedback was given well after the UML diagram for the system was due. The homework assignments that were tutorial-style were particularly useful for reference while setting up the system, though it felt like busy work to complete it as a homework assignment and then yet again for the project.

Although the homeworks had tutorial-style steps for continuous integration and deployment, there was no overview of Jenkins that could provide a roadmap for customizations past the base setup. Our team struggled to implement our ideal CI/CD scenario for multiple sprints because there was very little guidance given, particularly for deployed dev branches or fixing the build steps in the Jenkinsfile. Had Ben not dedicated himself entirely to dev operations, we would not have had a robust CI/CD system in place for testing the more complicated features like Google Authentication.

The project was clearly pitched to be much larger in scope than what could be accomplished in 5 sprints. However, no guidelines were given on what parts were considered the “MVP”, and instructors regularly gave conflicting answers on what was more important. It was often that our recitation TA would say to implement one feature for the next sprint, our professor would say another is important, and a third team would be working on something completely different. Although this was meant to model the real-world challenges of working with a client, it was inappropriate for an academic setting. For future iterations of this course, the professors need to come up with a defined way of answering these common scope questions and define what kind of product they want to see at the end. One implementation of this would be for the TAs/professors to have their own use case document that would reference when asked questions about scoping.

One issue the team wanted to readdress was the fact that we were short-handed compared to the rest of the other teams. Most teams had five or six members, while we were a team of four. Around the end of the first sprint, the team was made aware of a potential fifth member who was supposed to join our team, Abdullah Allowain. We were instructed to “wait and see” whether or not he would make the effort to include himself in our team. Therefore, for a few days, the team was put into a bit of an unusual situation, as

we were trying to plan the workload for both scenarios of him joining or not. The instructors gave us no guidance beyond their initial advice, and his name was never actually removed from our Teammates page. As a team, we would like to express our recommendation for future instances of this class that short-handed teams are acknowledged and not compared to other teams' projects who have additional members, as well as not being put into a situation like we were with Abdullah.