Daniel Chen
Jay Lok
Matthew Morgan
Veronica Shei
Abel Shin

Spoiled Tomatillos Final Report

Overview of the Problem:

The client, a stealth startup, is looking to enter the movie industry through a social recommendation platform product called Spoiled Tomatillos. The main client asking for this product is the professors and the teaching staff of CS 4500. CS 4500 is a upper level undergraduate course taught at Northeastern University by professors Jose Annunziato, Nathaniel Derbinsky and Michael Weintraub. These three individuals represent the main members of the Spoiled Tomatillos Board. The client expects to see a huge adaptation of the product, fast expansion of the user base, and to aim for acquisition and profit by the end of the fiscal year 2019.

As part of the Spoiled Tomatillos team, we are an internal software development team that was tasked with creating and shipping this product. At a high level, the main features we expect to achieve is a social recommendation system for movies that bridges the chasm between services like Netflix or Amazon to Facebook. Ideally, the platform's system should comprise of these primary functions:

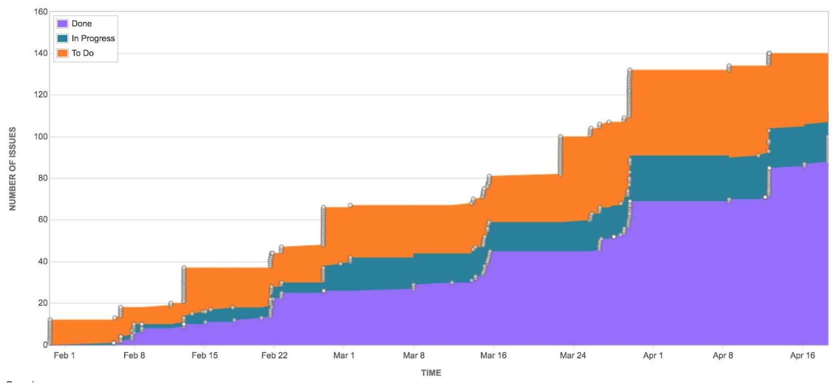
- Users should be able to create accounts

- Users should be able to search for movies, plotlines, cast, awards, and showtimes at nearby theaters
- Users should be able to identify friends on the platform
- Users should be able to recommend movies to friends
- Users should see three types of suggested movies based on average critic rating, average rating on the site, and via User-User Collaborative Filtering that utilizes advanced Unsupervised Machine Learning technology

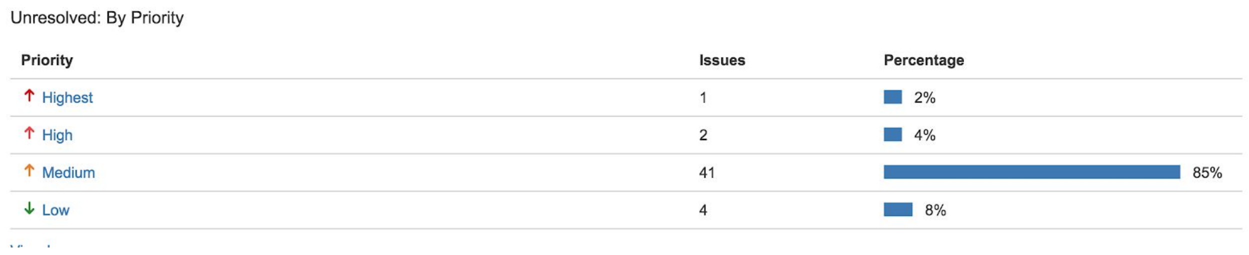
The client wants to fund and develop the Spoiled Tomatillos product because it sees a need in the social recommendation technology industry and believes there is potential for monetization and high profit. Because there is no technology that currently bridges the gap between modern digital platforms and social media, the Spoiled Tomatillos Board believes that this product would give them a big competitive advantage in the movie recommendation industry. Additionally, the three different types of movie suggestion algorithms that Spoiled Tomatillos will use gives them a big advantage over other similar movie suggestion websites like IMDb. We want to especially highlight the User-User Collaborative Filtering technology which is technology that has not yet gained high penetration in this industry. Depending on how well the Spoiled Tomatillos team does, the client is hoping that we can increase user adoption and the strength of our platform by linking Spoiled Tomatillos with affiliate codes on movie tickets and digital content providers like Netflix, Amazon Prime Video, iTunes, and more. Depending on how good the automation algorithm is, Spoiled Tomatillos may even be able to infiltrate music platforms or book platforms. Once the Spoiled Tomatillos Board decides on the ethical requirements for the product, we may potentially be able to sell all the user data that we have collected on our platform to interested companies, such as different movie production companies like Dreamworks or movie platforms like Netflix or Amazon Prime Video. Ultimately, the timeline for the Spoiled Tomatillos product is to make a Phase 1 prototype, build value via a user base, and improve the social recommendations algorithm through consumer usage, and aim for acquisition in the 2019 fiscal year, and profit.

Overview of the Result:

Overall we were able to accomplish the majority of the goals we set out for ourselves at the beginning of the project. We were able to fully develop most of the features we had in our initial plans, including account creation with email verification, searching for movies and users, external and internal movie ratings, user profiles with friend relationships, and automated movie recommendation. As displayed on the chart be below, we were able to complete over 80 Jira issues, with steady progress throughout the 5 sprints. We we still had some to-do issues open, many of there tickets were stretch goals and additional features like playlists and comments that go beyond the required use cases:



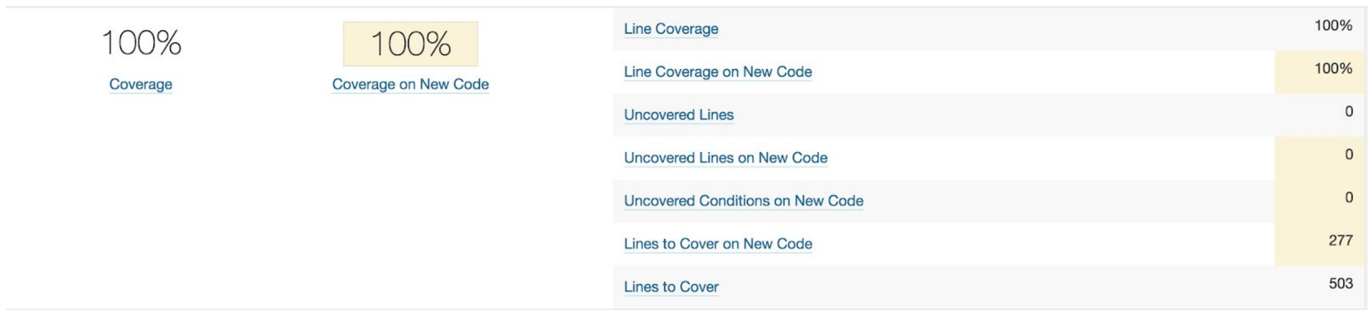
Also, as the chart below shows, only 2% of our highest priority tickets remained open, with 4% for the next highest tickets remaining. This shows that we were able to accomplish our main goals for the system, without getting side tracked by features that would be "nice to have". This kind of prioritization and focus is important to the success of any project.



Unresolved: By Priority

Priority	Issues	Percentage
↑ Highest	1	2%
↑ High	2	4%
↑ Medium	41	85%
↓ Low	4	8%

Finally we made sure to maintain the quality of our code with in-depth testing using the PyTest library. This was a learning experience for us as we had never tested in python before, but our unfamiliarity with the language only made testing more crucial, to ensure we would not make rookie mistakes. As displayed below, we achieved pull code coverage and high branch coverage.



100% Coverage

100% Coverage on New Code

Line Coverage	100%
Line Coverage on New Code	100%
Uncovered Lines	0
Uncovered Lines on New Code	0
Uncovered Conditions on New Code	0
Lines to Cover on New Code	277
Lines to Cover	503

Overview of the Team's Development Process:

For our tech stack, we decided to use the Flask microframework and Python on the backend, connected to a MySQL database, and regular HTML, CSS and Javascript on the front end. We selected Flask and Python for several reasons. Most importantly, we felt that it would be a great learning opportunity for all of us as most of us were already extremely comfortable with Java and wanted to try something new. During our research, we also found that Flask was built to be extremely scalable, which is important for our client as the demand for the application grows. Flask and Python also allowed for quick implementation and testability, which we felt reflected our selected agile workflow very well. We wanted to be able to build fast, and easily backtrack if necessary.

Our team followed the Scrum framework of an agile workflow. Our sprints lasted two weeks. Before each sprint, we'd decide on what to add to our Jira backlog based on the sprint requirements from our ScrumMaster. We made sure to give priority to the requirements and add a few stretches that we felt were within reach. We split up the stories based on both our strengths as well as our interests.

During the sprint, we used Slack to handle daily standups where we shared our current status for our respective jobs, as well as whether or not we were waiting on work from other members. We also communicated over Slack alongside our own private social media group when we needed help from others. This included things such as asking for help from the team member managing the AWS server hosting our database, or our Jenkins server, or even if we were stuck on an aspect of our technology stack.

We used Github for source control, and smart commits to connect our source control to our backlog, making it extremely simple to find exactly when a specific feature was implemented, or bug fixed. We also setup our repository to ensure that nobody could push directly to the master branch. Instead we created our own branches for each feature we intended to implement, and then put in pull requests when we completed the task. Each pull request had to be reviewed and approved by another member of the team to ensure that our master branch was always stable. After the first sprint, our continuous integration system was also setup. We used Jenkins to manage our continuous integration, testing, and deployment. Everytime a pull request was put in, Jenkins would automatically run our tests and ensure that the new version could be successfully built. We then also used SonarQube to do ensure continuous code quality. SonarQube helped to generate reports on the status and health of our application, as well as the thoroughness of our testing.

Finally, at the end of every two week sprint, we had a sprint review with our ScrumMaster. We showed off our accomplished goals from the sprint and discussed both how we can improve, as well as what we needed to focus on next.

Overall, our team's development process worked very well, and as we all settled into it, allowed us to be extremely productive, and cooperate with ease.

Retrospective of the Project:

The project was a great introduction to the software development life cycle and gave us exposure to being a part of a technology team operating on two week sprints in an Agile environment is like. Each member of the team was able to improve their web development skills, learn more about the Spring Boot and Flask technology stack, improve on our source control skills, learn how to use Jenkins, learn how to deploy on Amazon Web Services, and also learn more about working as a team. Ultimately, this project was a great way to expose students to software development principles, especially since the majority of our team member's had never been on a software development co-op. Furthermore, for those of us who had gone on a technical co-op before, this project provided a great opportunity for them to take on more of a project lead role and develop their management skills.

Our team was really effective about starting work early and dividing up tasks between all the team members. Because the way we operated was very task-driven, we were able to mostly work on our respective parts of the project without impacting another team member's work. When our work did impact the other tasks, we were good at communicating via Slack or Facebook messenger to convey any roadblocks we faced. Additionally, the task-based nature of the work meant that each team member was forced to learn how to use JIRA effectively, do smart commits, learn how to do branching in GitHub, and utilize Pull Reviews as an opportunity to see what our other team members were working on. The only downside of this task-driven approach is that for parts of the code we did not directly work on, we did not have as good of an understanding of the code base or know how aspects of the project worked. However, due to the more specialized nature of what each person worked on, we were each able to really develop our skills in one key arena, like front end development, UI/UX design, back end development, testing, and more.

Because none of us had much experience working in Flask before for such a large project, we all learned a great deal about the Flask framework. Furthermore, learning how to test in Python, integrate auto deploys in Amazon Web Services, utilizing smart commits to link JIRA and GitHub, and doing Pull Reviews was something most of us had never been exposed to. This project taught us a lot more about each of the respective tools listed above and seems to reflect a lot of the applications used by technology teams in the workforce, which really made the content we were learning feel applicable to the real world. Not only that, but because we were a team of five who were each working on independent things in the project, good communication was really key for us to be successful. We added a daily standup integration to help inform other team member's of our progress on our work and also as a barometer for us to ensure that we were completing our work in a timely manner. We learned to play to each team member's strengths and interests and how to work the most effectively as a team of five.

Overall, while we felt that the Spoiled Tomatillos project was a good learning experience, we feel like there could still be improvements to make the experience a great one. We feel that the amount of work required to complete each Sprint objective treats the students as full time employees. However, students have other classes and activities to attend to, making this an unfeasible task. Because our team had group members with a strong coding background, we were able to do most of the work in a timely manner, but I would imagine that other groups with less experienced members struggled to finish all the objectives on time. We recommend reducing the number of tasks needed to be completed for each sprint or to provide more guidance on how to achieve the functionality asked for in the sprint. The majority of the project required team member's to teach themselves how to code and deploy the functionality in the project with little assistance from the teaching staff. Even the lectures presented in class related to general software development principles, but did not feel like it related directly to the project, which made it hard for us to find guidance if we were stuck on any part of the project. If we had been given more guidance in our project, we most definitely feel like there were better design choices or technology stacks we could've chosen to use. Furthermore, we would include a more numerically defined grading criteria. While the sprint expectations tell us the general grading criteria, often times it felt like our grade was on the whim of our teaching assistant and we weren’t always given feedback for why we received the grade we did. However, overall this project taught us a lot about the software development life cycle and we feel that by taking our recommendations into mind, this project could turn into an even better experience for students who take this class in the future.