Spoiled Tomatillos Final Report

Authors: Justin Vincelette, Alice Nin, Nicole Pristin, Kathleen Newcomer, Yuliya Smilyanski
aka Team J.A.N.K.Y.

April 18th, 2018

I. Overview of Problem

Anyone with access to the internet knows how popular social media and streaming services have become over the past 10 years. In fact, social media companies like Facebook and streaming services such as Netflix and Amazon have taken the web by storm, with each of these three sites holding a place in the top 10 most visited websites in the United States. Despite their popularity, these type of sites have remained separate, as there is no service that combines the best of both worlds. That's where Mystery Startup, an aptly named stealth startup, comes into the picture. Seeing the continued success of both social media and streaming, Mystery Startup aims to bridge the gap between the two web giants by developing *Spoiled Tomatillos*. This new website takes the form of a movie recommendation service where movies can be reviewed, discussed, and recommended to friends in a social media based structure. Being the first of its kind, Mystery Startup believes that this new space can not only be monetized, but also become the next big thing.

Mystery Startup comes out of Northeastern University, and was founded by three computer science professors. Being funded by the university, they have approached Team J.A.N.K.Y. to develop their latest and greatest idea, *Spoiled Tomatillos*. Although part of Northeastern, Mystery Startup is completely controlled by the founders, so the team will only be dealing with the company and not the university as a whole. Additionally, since *Spoiled Tomatillos* is not their only project, they have hired another team in order to help direct the structure of this website. This team, the Tomatillos Advocates (TAs), is a second resource, and will help guide Team J.A.N.K.Y. towards the best results possible. The TAs mostly know what the founders want in terms of design and quality, but information from the founders will always be prioritized. Since these founders are computer science professors, the level of quality that will be expected from this project is quite high, especially with a chance to make a big splash on the web with this new idea.

At a basic level, Mystery Startup is looking for a website where end users will be able to make accounts, and then search and review movies. Furthermore, the website will give users the power of social media, by allowing them to identify friends, who they can then recommend movies to. The site also uses an algorithm to provide additional recommendations based on their user's friends and their own tastes in movies. The combination of all of these features has never been accomplished before, and it is why Mystery Startup believes they can create a business around such a website.

Other websites have focused on the movie reviewing aspect, such as *Rotten Tomatoes* and *IMDb*, but have only tried to add social media aspects as afterthoughts. The amount that users can interact with each other is limited to reading and "upvoting" each other's reviews, or going into separate, designated forums to discuss a movie amongst each other. *Spoiled Tomatillos* intends to put the social media of finding and reviewing movies at the forefront, creating a whole new type of movie reviewing system. Being the first full fledged social media

site that is centered around movies and recommendations is an idea that Mystery Startup truly believes could be very lucrative for them, and be groundbreaking on the web.

II. Overview of Result

After three months and five sprints of work, Team J.A.N.K.Y. was able to make huge strides towards Mystery Startup's game changing movie recommendation site by completing a fully functioning phase one prototype of Spoiled Tomatillos. This was accomplished by first creating a strong backend foundation in Java and Spring Boot, that allowed the team to create database tables for several vital classes of information, including Users, Reviews, Recommendations, and Friends. Additionally, the outside source of the Open Movie Database (OMDb) was used to obtain an enormous amount of actual movie information. All of this information was then leveraged on the frontend through API calls and the help of jQuery. Furthermore, both user interface design and user experience functionality were successful thanks to the heavy lifting of React, while basic HTML, CSS, and Bootstrap also provided additional assistance.

When all was said and done, our team succeeded in implementing all of the main functionality that Mystery Startup had requested. This involved Spoiled Tomatillos incorporating the five major ideas that were stated for the system. Firstly, users could sign up for an account by creating a username, password, and email combination. After registering, they are able to search for movies to see various information, including the the movie's year, genre, parental rating, director, actors, and IMDb rating. Secondly, users can rate these movies on a 1 to 10 star rating scale, and also can write an optional review comment. Thirdly, users are able to identify friends on the site through the use of a follow system. Similar to sites like Twitter or Instagram, Spoiled Tomatillos operates through the use of followers since the team believed a one way relationship would better suit the site in cases such as large movie critic accounts, while still giving the opportunity for smaller and more intimate friend connections. In either case, the more users that a user follows, the better their movie recommendations are. Which brings up the fourth main point, a user can send movie recommendations to any of their follower/following connections. This is an important part of the social media aspect of the site. Also tied into this point, is the fifth major client idea, which is having the system produce autogenerated movie recommendations. These recommendations are calculated by User-User Collaborative Filtering, or in other words, an algorithm that factors in the movies a user and their followers have reviewed in the past.

On top of these five most important features, we were also able to achieve several other use cases that the team deemed significant for the first Spoiled Tomatillos prototype. This covered use cases such as searching for users, viewing recommended movies in a concise way, editing a user's profile, and upvoting movie reviews. Altogether, the team had come up with 24 total use cases at the beginning of the project, with 18 relating to end users, 4 relating to admin users, and 2 relating to marketing users. In the end, 13 use cases were fully completed, all associated with the end user standpoint, which is what we believed was the number one priority for an initial version of the system. This is because by implementing a functional end user focused prototype, the client not only has a working demo of all the main features, but also

has the ability to extend the system easily with what they believe would be the best admin functionality.

Throughout the process, our team also integrated SonarQube assessment into Jenkins in order to constantly build and test new features when they were added to the project. We started by having a quality gate mandating our tests have 85% code coverage, and then gradually worked towards the more acceptable and sufficient 100% line coverage and 85% condition coverage quality gate. This meant that we were constantly adding tests for old and new code, with the purpose of making sure our SonarQube metrics were met and our code could be built and merged into the master branch. With the final result of our project, we were able to successfully reach 100% line coverage and 96.4% condition coverage, giving us 99.2% combined overall coverage from our tests. Therefore, on top of accomplishing all of the necessary functionality of the site, we were happy that the quality of our code matched the same level of success.

III. Overview of Team Development Process

The team mastered the development process by the end of the sprint. There were a number of different tools that aided us in the software development process, including Github, Git, JIRA, Jenkins, and Sonarqube.

We began our project, as well as every single sprint, by first determining the requirements from the client. This involved not only just reading the necessary requirements provided to us, but also reaching out to the client and asking questions for clarification. As a team, we'd determine which requirements were absolutely necessary to have done by the end of the sprint and prioritize those, followed by other stretch goals that would potentially make the client happy to see done.

To help us in organizing the requirements for the sprint and the project as a whole, the team used JIRA for project management. At the very beginning of the project, the team filled the backlog with different use cases that were to be completed by the end of the project. Before the start of sprint, the team would pull from the JIRA backlog the requirements and features that the team planned to achieve during the upcoming two-week sprint. This was in the form of stories, and each story would have a priority ranking from *Low* to *Highest*. The team members would split up the work in the sprint relatively equally, and would keep track of completion via the team board.

At the beginning of the entire project, we also built a series of wireframes that would allow us to all have a common vision for how we want our system to look. While the team deviated from the wireframes in regards to design, the expected functionalities remained. However, while initial wireframes can be insightful, they are often revised throughout the software development process, especially in an agile work environment. As the team added more features, we recognized that some designs needed to be revised and implemented in a different way. This is a crucial part of agile, as it is important to take time during each sprint to evaluate and revise.

The step in the software development process that actually involves writing and implementing code was aided greatly with several extremely important tools: Git and Github. Every team member used Github as the central repository for the code base, as well as git as a

form of version control and concurrency control. Each member would create a local branch using git when they wanted to make changes or additions to the existing code base. When they were satisfied with their work, they would push all their changes to Github and submit a *Pull Request*. It was then mandatory that another group member reviewed the code in the pull request, and either approve it or leave comments about areas of improvement. There was never a time that any team member would push their new code directly to master, ensuring that the quality of the code being sent to master was high.

The team also made use of notification systems about opened, merged, and closed pull requests. Primarily, the team configured github to send an email to every group member when a pull request was opened, reviewed, approved, merged, or closed. This was extremely helpful in notifying team members when a pull request was opened and was in need of a review. Furthermore, if an alert was sent that a team member had already approved the pull request, this notification would insight the rest of the team that they do not need to worry about reviewing this pull request as it has already been taken care of. This allowed for members to prioritize their work and organize the code review process.

The part of the software development process that involves verification and maintenance was greatly aided through the use of Jenkins and Sonarqube. Part of the team's development process involved that the build and quality check on a pull request pass before being merged. The team configured Jenkins to run builds on master and on any active branch, and would notify github if the build passed or failed. If the build failed, the merging of the pull request on that branch would be blocked until a team member addressed the build issue. While we used Jenkins for builds, we also integrated Sonarqube to run quality checks on the code base; in order for a Jenkins build to pass, the pull request needed to pass the quality check first. The quality check consisted of many different components, but the one that the team used greatly in tracking code quality was the test coverage, as discussed earlier. If a certain level of code coverage was not reached, the build would not pass, and a group member would need to address the issue and add more testing before merging the pull request to master.

Another aspect of the software development process that was extremely helpful was weekly meetings. The team met several times outside of class and lab to determine the expectations for the sprint and who would be comfortable tackling which requirements. The team would also meet during the lab section and work together to answer any questions and make clarifications. However, in regards to stand-ups (or slack-ups), the team did not always do a daily stand-up, and it seems that this was the greatest issue that we faced during the process. While the team maintained a level of communication through Facebook Messenger, we did not implement consistent stand-ups, making the development process more difficult and communication much less consistent than it should have been. This is the only aspect of the process that did not work for us, and something that we would have needed to work on and strive to improve on in the future.

Another aspect of our team development process was specialization of skill sets. It was obvious from the start which group members were more comfortable with front-end, UI, back-end, and system-related work. Due to the teams complementary skill sets, there was one person who handled the majority of the system-related work involving Jenkins and Sonarqube, one person who handled most of the back-end Java work, and then three people who focused

on front-end web development and UI using Javascript and React. The specialization of tasks based on complementary skill sets is what separates a team from a group; a group is just several people who have a common goal. On the other hand, a team is a group with a common goal and where each team member has a particular skill set that is crucial to the teams success.

Finally, the sprint reviews were a major part of our software development process. Every two weeks, the team would meet with the client and determine if the requirements were met. The sprint reviews were also a good planning period to gauge what features the client expected to be built during the next sprint, and to determine areas of improvement on the existing product. Sprint reviews are also an essential part of agile, and allowed for the team to evaluate how far along they are into the project.

IV. Retrospective

Overall, our team felt successful in our completion of the Spoiled Tomatillos project. While all of our skill sets and past development experiences varied greatly, we feel confident that we learned a lot through this process. Many of us learned for the first time and improved our skills in JavaScript and React, and how to build dynamic web applications. We learned to how incorporate business logic in the back-end using Java, and we learned how to test API and database calls using Mockito. Besides technical skills learned, we have also learned a lot about working in an agile environment. While all of us have been on co-op, not all of us have had co-ops which employ pure agile in their software development. We found that using agile in this course specifically was very beneficial in our organization of the project. We enjoyed splitting our work in to two-week sprints and building upon each sprint with iterations. We identified bugs and changes to be made before each sprint, and were able to implement those changes during the next sprint. It allowed us to focus less on perfectly completing each functionality and instead to implement what we can, and come back later to what needs to be fixed. Obviously, this project had a lot of moving parts with a lot to keep track of, so being able to split up our work in this way was supremely beneficial to our overall execution of the project.

Despite our overall positive experience with what we learned in this course, we do think that a lot needs to be changed about the organization of the course and its expectations and requirements to create a better learning experience for students. First, we believe that the communication between the professors, TAs, and students was not great. Professors would always answer questions during class, but they seemed to be hard to reach otherwise. Many posts on Piazza went unanswered and ignored by the course staff. Since this is our main mode of communication in the course, we believe it is utterly unfair for professors to be ignoring any questions at all on Piazza, whether it was on accident or on purpose. Piazza is a great place for students to get questions answered publicly so everyone in the class can benefit from the responses. Even if a student is ignored on Piazza but then gets an answer by personally emailing a professor, this prevents all other students from knowing probably relevant information to them. In addition to the lack of responses on Piazza, we believe that the TAs were not as knowledgeable about the course and its expectations as they should have been. We spent a lot of time during lab, in sprint reviews and off-weeks as well, speaking to our TA about what exactly was meant by this expectation or this communication from the professors, and many times, our TA was not able to give us an answer. He would say that he was confused by the

requirement as well, or would be unsure of who to ask to clarify this. The TAs were our main communication point in terms of clarification and guidance, and it is not right that the TAs were not given all the information needed to appropriately guide us. After many conversations with our TA, we found ourselves even more confused than when we began.

Along the lines of confusing expectations, another thing we think needs to improve about this course is the sprint expectations. To start, we believe that the format that the sprint expectations sheets were given in was very confusing. There is definitely a better way to lay out information than by a slightly ordered list. We believe it'd be much more helpful to organize the expectations like so: one section that lists & explains the base requirements, one section that lists some basic stretch goals, and the last section that lists ultimate stretch goals or tasks for the next sprint that could be tackled early. We found it confusing that stretch goals were mixed up with base requirements in the list, and on more than one occasion, we almost missed a base requirement due to being unable to differentiate between base and stretch goals. Additionally, we believe that the requirements themselves were not conducive to having positive sprint reviews and good grades on the sprints. Our team spent a lot of time, energy, and effort working on every sprint, especially sprints #3 - 5. Despite our increase in time-spent, effort, and quality of result, we saw our grade go down from sprint to sprint. We believe that our accomplishments weren't appropriately recognized by the sprint expectations. For example, we would complete 6 use-cases, but then our TA would inform us that actually, 3 of those use cases does not count, and all of a sudden, we were in a mad dash to get something in just to hit the base requirements. On another sprint, one of the stretch goals was something that the TAs and students had deemed impossible to accomplish: connect to a streaming service API. How can this be a stretch goal if it is literally impossible for us to complete? Because of this, our team was left to think of and scrounge up whatever other stretch goals we could to hopefully bring our grade up. In the end, we were told that despite hitting all the base requirements and getting two stretch goals, we still would likely not be receiving above a B on this sprint because we were not able to complete more use cases. No where on the sprint expectations sheet did it say that completing more use cases was a stretch goal.

Overall, we had very frustrating experiences during almost all of our sprint reviews. We never felt that all the work we put in to those past 2 weeks were recognized and appreciated. We would often leave sprint reviews feeling deflated and discouraged after working so hard to complete everything for that review. We truly believe that better communication between course staff and students and a reorganization in the way requirements are presented would make the learning experience for students much more positive. Rather than spending time thinking about what the minimum requirements for the sprint even are, we would be able to spend time on actually writing good code, creating appropriate tests, dealing with Jira and Sonarqube, and completing actually good work. Our hectic learning experience would not be tainted by grade frustrations and confusion on what we are actually expected to accomplish.