Developer Behaviors and Team Processes

Release 5 Evaluation, March 29, 2019

|  |
| --- |
| **1. Testing with unit tests and other tools.**  We have written unit tests for the hardware code to ensure proper operation of our code as it is written. Our unit tests are written in python, which is the same language as the code itself, using the “unittest” library. We have continued to adjust the unit tests as the code changes, which is quite often.. We also have a few unit tests for the communication code. These tests have to do with deleting files that are not selected by the user on the front end. We have also written some integration tests that test the cameras and the camera control code working together. These tests can only be run on the hardware tower, and are intended to ensure that the cameras are connected correctly and can function and capture images.  **Documentation:**  Unit tests can be found in the “test” folder in our code in Box: <https://unl.app.box.com/folder/70415669887> |
| **2. Always have a working build of the product with tests running.**  On Spreetail’s GitLab repository, we have a branch named “master” which contains our working build of the product. When all the features in “develop” are thoroughly tested, reviewed, and approved by the Development Manager, the branch can be merged to “master”. The master branch is where we monitor our working build. Master will always contain working, tested code from the previous release. We have updated master with the code from Release 5. We use GitLab’s CI to build our code every time something is merged or pushed to master. We also perform these builds on the develop branch, so that we can be assured that the code being pushed to develop is tested and build as well. GitLab runs a script called “.gitlab-ci.yml” that allows us to customize how we build our code. We are using “Docker” to place our code into containers that can be run on almost any operating system. For our project, the hardware code must be run on a linux based machine so that gphoto2 can be installed and used. Our CI script runs docker build commands that build the camera control and front end docker containers. In this docker container build, our unit tests are run, and the build will fail on any test failure, which allows us to ensure that all code is built and tested successfully.  **Documentation:**  The Gitlab CI run for our merge into master for R5 can be found in box:  [**https://unl.app.box.com/folder/70415669887**](https://unl.app.box.com/folder/70415669887) |
| **3. Run static analysis tools to remove errors and warnings.**  Hardware and Communication:  The hardware and communication code warnings and errors are caught within the PyCharm IDE environment. We are using the static analysis tools provided by JetBrains in this development environment, which follows the PEP (Python Enhancement Proposals) standards. This provides active syntax highlighting whenever a warning or error shows up. There are some warnings/errors that exist that we will not change, such as import issues. These issues arise due to PyCharm having a different method of finding modules than running the Python from the command line. These errors are known and will not be fixed due to this difference.  Front End:  The Front End is being built in React, NodeJS, JavaScript, SASS, and HTML using Visual Studio Code Editor (VSCode). VSCode has a vast amount of plugins that can be installed to provide code analysis. The libraries we’re using are called ESLint and Prettier. ESLint extends the Prettier extension, as well as Airbnb’s open source config files for formatting (installed through node package manager). We’re also able to add our own configurations on how we want to format the code.  **Documentation:**  Hardware Code: Pycharm IDE    Front End: VSCode -- ESLint and Prettier |
| **4. Maintain a story map using the Senior Design story format.**  We have used StoriesOnBoard to maintain an up-to-date story map for the project. This story map represents the high level tasks that we need to accomplish in order to complete a specific iteration. We use this story map to derive user stories and do sprint-planning later in Zenhub.  **Documentation:**  **A PNG of the story map can be found in Box**  [**https://unl.app.box.com/folder/70415669887**](https://unl.app.box.com/folder/70415669887) |
| **5. Maintain up to date definition of done and backlogs for each milestones and sprint.**  Our definition of done can be found on our github wiki page. This page describes the criteria for code to be “done” in our project context. We were able to develop this set of criteria by meeting with our sponsor and discussing what review, testing, and building needs to happen before code can be considered done. We have also updated all of our Zenhub stories to contain “Acceptance Criteria”. These criteria are mainly the developer tasks for the story rewritten as completed events. This allows us to look back when completing the work for a story to check and see if all criteria for being done is met by the development progress made. We have also discussed the handoff criteria with the sponsor and this is documented in the Transition plan located in Box.  **Documentation:** Zenhub: <https://app.zenhub.com/workspaces/spreetail-5bbceb414b5806bc2bec3047/boards?repos=145894767&showPipelineDescriptions=false>  Github wiki:  <https://github.com/cseseniordesign/spreetail/wiki/Definition-of-Done>  Box:  <https://unl.app.box.com/folder/68453247045> |
| **6. Estimate stories and assign points in the context of design.**  Our sprints are 1 week in duration, and we do sprint planning every Thursday at the end of our team time. We estimate our stories when adding them to the backlog by applying a generalized tag that represents a small, medium, or large task. When we plan stories into the sprint from the backlog, we then use a form of planning poker. We look at a story, and we all write down how many points we believe the story would take. We then look at all of the point values and discuss the story if points do not correlate. This method of estimation has been very successful and easily adopted. For release 6, we have decided to include the entire release as one long sprint of 3 weeks, as most of the work we will be doing is closing practices and documentation.  **Documentation:**  Zenhub: <https://app.zenhub.com/workspaces/spreetail-5bbceb414b5806bc2bec3047/boards?repos=145894767&showPipelineDescriptions=false> |
| **7. Release milestones to the sponsor.**  At the beginning of release 5, we met with our sponsor to nail down what is expected for the end of the release. We decided that the main tasks to be completed during release 5 were implementing the output page in the front end, making a connection to the Spreetail Storage, adding integration tests to the hardware, adding a progress bar, painting the turntable white, selection ability, and overall rounding out and completing the front end. We were able to accomplish all of these feats, although there were some issues during the demo. We found fixes to some of these issues as well as performed user testing to ensure that users cannot break the front end processes. We found some very useful information from this testing, and plan to make some fixes to the front end by the end of R6.  **Documentation:** |
| **8. Have a security plan for the highest security risk areas of project.**  Our two highest risk security issues for the project are access to authentication keys from the web application and complete access to our web app from anyone. Before, if a user accessed our web page, they were sent the javascript code that displayed our front end components as well as connected to the Azure message service. Users could inspect the web page and get the authentication keys that allowed access to the Azure message Bus. Also, anyone can access our page by entering the url for our web app.  To solve that issue, we created an environment variables file (.env) to store all of the keys and secrets from APIs. These variables can be pulled from frontend and won’t display when inspecting the interface through the web browser. Below you can see that we have stored the keys in variables such as ‘REACT\_APP\_SBS\_SHARED\_ACCESS\_KEY’ and these are being pulled from the env file.  For the issue of users being able to access the web app from anywhere, Spreetail will handle adding sign in security when deploying the web app to their Azure environment. This will require users to sign into spreetails network before being allowed access to the web-app.  **Documentation:** |
| **9. Measure performance and use telemetry to inform decisions.**  We have taken consideration into the performance of our cameras, communication techniques, as well as computer hardware. Switching from a Raspberry Pi to a Linux tower has granted us with better overall performance within the CPU as well as transfer speeds from camera to hard disk.  Cameras are plug and play as long as they are compatible with the gphoto2 library. However, switching from the Nikon D3400 to the Canon EOS 6D Mark II allows us to snap pictures and auto focus faster and more consistently. The Canon EOS 6D Mark II has more options for auto focus, being able to determine the size of the focus. We’ve also updated the lens to a prime lens instead of a standard stock lens. Prime lenses (50mm f/1.8) can capture more detail, compared to a stock lens(18-55mm f/3.5-5.6), which focuses on the bigger picture. However, the lens are also interchangeable based on the situation.  Communicating to the tower and sending an image back took an extreme amount of time (15 seconds to take, transfer, downsize, and send image) when doing it via direct connection. This was reduced to around 5 seconds when we sent it through a new communication layer, Azure Service Bus. We tested sending 50 images that were already on disk and were able to send them all out within 20 seconds. Since it’s a message queue, the front end (or whoever wishes to receive it) is able to pull an image right when it arrives in the queue.  We’re now displaying the entire capture process in the frontend with a progress bar. This progress bar displays the percent of the entire process. The first 70% will display the number of pictures that the camera has taken and at 70% it will have finished. The next 10% tells the user the process of downloading all those images from the camera to the tower. It also tells us when it will send those images to the service bus. After that, 80%-100% is the images being pulled on the frontend from the service bus and displaying it. The number of images for each side is also listed, and the progress for them as well.  **Documentation:**  Github Wiki:[**https://github.com/cseseniordesign/spreetail/wiki/Telemetry-and-Testing**](https://github.com/cseseniordesign/spreetail/wiki/Telemetry-and-Testing) |
| **10. Maintain cadence iterations.**  We do our sprint planning at the end of team time each Thursday, which is the end of the week for our team. This allows us to look back and see what we’ve completed as well as what may need to get pushed into the next iteration. We generally plan for the number of team time hours available that week, but we sometimes plan more if development speed needs increased. Overall, our burndown charts for each sprint were very vertical, as most stories were completed on Thursdays at the end of the week. We have labeled all of the R5 stories in Zenhub to be a part of a new milestone called R5. This allows us to see the burndown for the entire release, which can be seen below.  **Documentation:**    Zenhub: <https://app.zenhub.com/workspaces/spreetail-5bbceb414b5806bc2bec3047/boards?repos=145894767> |