Monitoring/Deployment
Senior Project Technical Report

Hot Tub Time Machine

Andrew Filipski  Greg Gonyea  Andres Ruiz

Zach Berger

Sponsored by Spectracom

John Fischer  Jeremy Onyan  Denis Reilly

Faculty Coach

Rick Weil
Table of Contents

Project Upgrading and Monitoring

Project Overview
  Purpose
  Scope
  Desired Results

Basic Requirements
  System Requirements

Development Process
  Process Choices
  Communication
  Roles

Project Schedule: Planned and Actual
  Planned Schedule
  Actual Schedule

System Design
  Process and Product Metrics
  Product State at Time of Delivery

Project Reflection

References
Spectracom Monitoring and Upgrade Service

Project Overview

Purpose

This project aims to create an enterprise monitoring and upgrade system for Spectracom’s Time Servers and Hardware. This intends to solve the problem presented by big distributed enterprises that do not have just one Time Server. The monitoring system will be responsible for aggregating the data for the entire network of time servers not just an individual box. This will also be providing the user with a selectable granularity. All captured and process data will be provided to the end user in a dashboard that may be easily integrated into existing monitoring systems. The dashboard will provide a notification system that may be customized to fit the specific needs of the customer. To empower the administrators of these systems we will provided a highly configurable suite of key performance indicators (KPIs) and historical statistics.

The second focus of our project is geared towards creating an update system. Companies have many time servers so when an update is released by Spectracom, they have to manually run the updates on each box. The upgrade system will allow the user to enter the location and information of boxes in their system and perform updates from this central location. The system shall allow a user to apply an update to all boxes in the system in one step. While updating, the user will be able to view the logs associated with the specific upgrading timeserver as well as be able to view it state.

Scope

The Upgrade and monitoring system will provide a group aggregation. Our focus is not on one box but the management of multiple boxes all contained within a customer’s network. This tool centered around clustering is set to address two major ask’s of Spectracom customers. First people were complaining that the system was not “Enterprise Class”. Through elicitation we learned that this means if a customer owns 50 time servers they have to login into each one separately to monitor and perform upgrades. Secondly in a large systems there is a need to understand where a process breaks down but also how it compares to every other box. Information like how often one box needs to be corrected is a key performance indicator and can be used to find a fault in a box earlier. The group monitoring solution will provide a one central location to monitor all time servers in a given network. Accordingly the upgrade system will be operating on the same basic principal.

Initially, we broke this overall project into two separate projects. This allowed us to focus on the needs of that specific use case and system. Therefore what we deemed as in scope or out of scope was based on essentially these two projects. Having two projects and a fixed
amount of time to complete it we made it clear to our sponsors that they would get two half systems since we would not be focusing on just one. They agreed to the following scopes:

**Monitoring In Scope**
- Define and aggregate data from groups of time servers
- Allow customers to define their own KPI's and measurements
- Provide both a standalone dashboard displaying historical statistics and real-time KPIs and a plugin
- Provide an easily integratable API for consumption of aggregate data by other services

**Monitoring Out of Scope**
- Automation including auto discovery of TimeServers on a network
- Providing a mechanism to fix/edit a TimeServer. IE if something breaks or needs to be changed providing post request to edit the box
- Visualization of data through Graphite or any live metric analysis system.

**Upgrading In Scope**
- Allow customers to configure where their boxes are on the network
- Provide a few mechanisms to upgrade boxes either serially or in parallel
- Allow customers to visualize upgrade process and progress of boxes
- Provide a view into the logs of each time server

**Upgrading Out of Scope**
- Automation including auto discovery of TimeServers on a network
- Allowing use of customer upgrade patterns. (Desirable for automation)
- Automation of upgrade process
- Polling for Spectracom upgrades to help ensure the customer is up to date

**Desired Results**
The desired results of the system will be a functional monitoring system and upgrade system. Each will be “Enterprise Class” but the monitoring system will afford customers important strategic insight into how their TimeServer network is functioning. Releasing the system to their customers should only strengthen their relationship as this addresses a very popular customer want. Along the same lines the upgrade system will also be addressing a very popular customer problem. We would hope that upon release this would also only improve customer relations.
Basic Requirements

System Requirements

Monitoring Requirements
We identified three users of this system. The three users are discussed in detail in our use case document which is referenced below. The users identified were Data Center Field Worker, Data Center Manager, and an Analyst. Each class had their own motivations and requirements. A field worker would work with the system the most as just a pure monitoring tool. Both the analyst and manager would be interested in how extensible the system could be and asking questions like; How deep the analysis could get? Could the monitor and compare by region? A manager would be also interested in configuration and administration of KPI’s?

After determining these users we came up with the following functional requirements:

1) Notification System
   i. SMS
   ii. Email
   iii. Scheduling Alerts
   iv. Subscribing

2) Key Performance Indicator
   i. Defining KPI’s
   ii. KPI’s Threshold
   iii. KPI’s Priority

3) Integratable and Extensible plugin
   i. Nagios plugin allows it to be pulled into existing systems
   ii. Extra configuration can be added through Nagios
   iii. Docker can be used to set up and deploy the system completely

4) Reporting Suite
   i. Able to view one’s machines statistics vs the whole network
   ii. find trends and problems before they occur Notify of changes in the average
   iii. Exportable data in various formats including CSV

5) Customizable Monitoring Dashboard
   i. selectable granularity (single box vs whole network)
   ii. view and create reports
   iii. View current status of boxes in system
   iv. Able to view configured alerts

6) Provided Templates (out of the box)
   i. KPI’s set to standard thresholds, with set priorities
   ii. Reports to watch the system defaults
Upgrading Requirements
We identified only two users for this upgrade system - regular users and administrators. It made the most sense to have a regular user who could login, view current upgrades and configs, but could not start any upgrade process. Where as the administrator could handle users and would also be the one performing upgrades. Understanding the roles and the motivations of the system we created the following functional requirements in our sprint zero:

1) Configuration Management
   i. Docker deployable
   ii. Ruby on rails

2) Login and User management
   i. secure
   ii. admin account allowing the user to upgrade
   iii. restricted regular account that may view system

3) Configuration of System
   i. Able to enter details and configuration information for each box in the system
   ii. Able to create rules indicating how to upgrade boxes

4) Monitoring
   i. Able to view all information of upgrading boxes
   ii. Able to pull version of box from box.
   iii. Shall be able to view the logs of each box upgrading

5) Upgrading
   i. A user shall be able to select a specific update to apply to one or more time servers in the system.
   ii. A user shall be able to select a method in which the updates should apply either serial or parallel.
   iii. A user shall be able to define their own update method

6) Logging
   i. Able to select of level of logging for this system and its logs locations
   ii. Shall be able to view the logs which will contain a timestamp and the box/state

*** items in grey were identified but shelved since it violated the agreed upon scope.
## Constraints

### Monitoring Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Component</th>
<th>Description</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SecureSync Hardware</td>
<td>The system needs to be connected to a GPS signal</td>
<td>High</td>
</tr>
<tr>
<td>2</td>
<td>SecureSync Hardware</td>
<td>Limited computing resources (Spectracom box cannot have Nagios running on it)</td>
<td>Med</td>
</tr>
<tr>
<td>3</td>
<td>SecureSync Software</td>
<td>Undocumented API makes it quite hard to foresee adding new features</td>
<td>Low</td>
</tr>
<tr>
<td>4</td>
<td>Runtime</td>
<td>Unrealistic testing environment</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>SecureSync Hardware</td>
<td>Limited access to option cards to evaluate and test</td>
<td>Med</td>
</tr>
<tr>
<td>6</td>
<td>Nagios</td>
<td>Limited to 3 status (low, med, high levels) to represent system state. This could impact our ability to allow lots of configuration</td>
<td>Low</td>
</tr>
<tr>
<td>7</td>
<td>Spectracom</td>
<td>Inability to update or change the API</td>
<td>Low</td>
</tr>
<tr>
<td>8</td>
<td>Spectracom</td>
<td>Incomplete API impacts our ability to post to the system</td>
<td>Low</td>
</tr>
</tbody>
</table>

Spectracom was very open and wanted us to produce something without too many constraints. Therefore the constraints we did have were mostly imposed by the capability of the hardware and software at the time. For example we wanted to add the ability to set information like adjusting the clock internally for a box that may have gone wrong however there are no posts configured currently in the TimeServer. After speaking with Spectracom it became clear that they would not be updating the current API and implement new features that our team would request. Our chosen technology stack was not an issue but what we were confined to was what the TimeServer API exposed.
### Upgrading Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Component</th>
<th>Description</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Runtime</td>
<td>Unrealistic testing environment</td>
<td>High</td>
</tr>
<tr>
<td>2</td>
<td>Spectracom</td>
<td>Unable to get all state data from TimeServer software</td>
<td>Low</td>
</tr>
<tr>
<td>3</td>
<td>Spectracom</td>
<td>Inability to update or change the API</td>
<td>Med</td>
</tr>
<tr>
<td>4</td>
<td>Spectracom</td>
<td>Unable to add RSS feed</td>
<td>Low</td>
</tr>
<tr>
<td>5</td>
<td>Team</td>
<td>Learning Ruby on Rails</td>
<td>Low</td>
</tr>
</tbody>
</table>

Again for this part of the project Spectracom was pushing for us to do whatever was best. We decided to use Ruby on Rails since all but one of us was familiar with it. The constraints we did encounter were again limitations of the current software. We made note of these limitations for Spectracom and in most cases this did not mess up the core of the design it just took away a nice to have feature. One such example is the idea of having the tool always show the current update available from Spectracom. For this to work we would have needed them to create an RSS feed but this was considered out of scope. We were able to design the system in such a way that if they do add the RSS feed it will not be too hard to wire it in.

### Development Process

#### Process Choices

#### Monitoring Phase

Since Monitoring was the first phase we wanted to do Scrum because of the timeboxing and accountability it places on the individual however the requirements elicitation process dictated us using Waterfall. So what happened was a hybrid. We performed a scrum RUP hybrid where we did sprints, but also did requirements, design, development etc. This aligned with what Spectracom wanted as they wanted to sign off on each phase before we moved on. This actually worked quite well given the nature of the project. However it had a noticeable lack of metrics since it was created. Therefore when it came to improving we realized reinventing the wheel although it worked should not be continued. We termed this successful but unimprovable process Greg-gile.

#### Upgrading Phase

The upgrading phase of our project was during the 2nd semester. At this point we were quite familiar with Spectracom and the TimeServer. We were also quite comfortable with each other and our assumed roles. We had entered the performing phase of our teams relationship. We
decided given the previous experience that a true Scrum approach would be best. It would improve our communication with Spectracom and also let us focus on working software. We were all relatively familiar with the process so after talking with Spectracom we made it official. We would be using a Scrum with 2 week sprints.

Communication
For the first semester of Senior Project we used a couple of different methods of communicating to each other, the sponsor, and the coach.

To communicate to each other as team members, we mainly used a private Facebook group. This allowed us to send group messages and send updates on any findings we might have come across. Because of the lack of functionality (missed notifications, endless scrolling) that Facebook had, we decided that moving into semester two we would be switching over to use Slack. Slack worked quite well as we were able to also connect it to our Jira, Jenkins build, and GitLab affording us one app to see everything.

Communication with Spectracom was primarily through email. In the first phase we would meet only a few times face to face although we did go and visit them. We met to demo and give them an idea of what we were doing. This was fine but going into Semester 2 we wanted to improve communication and with Scrum that was imperative. We got them to commit to a hangouts meeting weekly on friday as well as being able to schedule meetings at other times if need be. This has worked wonderfully as everyone is on the same page and no surprises have popped up. We also started sending bi weekly email reports to anyone that could not attend the meetings.

Communication with our Coach, Rick, has been consistent with what we as a team were doing. Rick continued to participate in the email chain and also utilized Slack. The communication has always been good. We meet in person and Rick also makes it to the remote meeting.

Roles

Product Manager - Greg Gonyea
Hardware Lead - Zach Berger
Software Lead - Andres Ruiz
Configuration/Documentation Lead - Drew Filipski
Denis Reilly - Hardware Engineer for Spectracom
Jeremy Onyan - Salesman provided customer perspective
John Fischer - Chief Technology Officer
We were able to fall into our roles relatively easily. Both Drew and Greg were trying for the PM role. Greg and Drew did well splitting out the PM and configuration roles that were needed for the project. He was also a back PM making sure everything was running smooth.

**Project Schedule: Planned and Actual**

**Planned Schedule**

**Monitoring Phase**

<table>
<thead>
<tr>
<th>Week (Monday-Sunday)</th>
<th>Deliverables Due at the end of the week (Green weeks require status update)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Week 4</strong></td>
<td>Initial SRS Section 1 completed Begin playing with assigned tool</td>
</tr>
<tr>
<td><strong>Week 5</strong></td>
<td>Finished rough draft of SRS V1 continue researching assigned tool</td>
</tr>
<tr>
<td><strong>Week 6</strong></td>
<td>Gain familiarity with Spectracom HW and SW present tool findings and conclusions</td>
</tr>
<tr>
<td><strong>Week 7</strong></td>
<td>Peer Evals Updated SRS V2 with new familiarity with HW and SW</td>
</tr>
<tr>
<td><strong>Week 8</strong></td>
<td>Meet with Customers and Jeremy</td>
</tr>
<tr>
<td><strong>Week 9</strong></td>
<td>Select tool set Rough code architecture draft</td>
</tr>
<tr>
<td><strong>Week 10</strong></td>
<td>Schedule midterm presentation</td>
</tr>
<tr>
<td><strong>Week 11</strong></td>
<td>Start Authentication Fix Proof of concept Plugin (combine curl and nagios in plugin)</td>
</tr>
<tr>
<td><strong>Week 12</strong></td>
<td>Investigate Kpis and provide prioritized list</td>
</tr>
<tr>
<td><strong>Week 13</strong></td>
<td>Actual presentation, Implementation of 4 KPI’s completed</td>
</tr>
<tr>
<td><strong>Week 14</strong></td>
<td>Off filler/catch up week</td>
</tr>
<tr>
<td>Week 15</td>
<td>Clean up Retrospective</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Week 16</td>
<td>End of terms evals</td>
</tr>
<tr>
<td></td>
<td>Self assessment</td>
</tr>
<tr>
<td></td>
<td>Presentation?</td>
</tr>
<tr>
<td></td>
<td>Update website</td>
</tr>
<tr>
<td></td>
<td>Course Eval</td>
</tr>
<tr>
<td></td>
<td>Plan for next semester!</td>
</tr>
<tr>
<td>Week 17</td>
<td>Reflection possibly at Mcgregors?</td>
</tr>
<tr>
<td>Vacation for Winter BREAK</td>
<td>Enjoy time off</td>
</tr>
</tbody>
</table>

**Upgrading Phase**

<table>
<thead>
<tr>
<th>Week (Monday-Sunday)</th>
<th>Deliverables Due at the end of the week (Green weeks require status update)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 9</td>
<td>● Attend session on making a poster and writing the technical report</td>
</tr>
<tr>
<td></td>
<td>● Sprint 4 concluded</td>
</tr>
<tr>
<td></td>
<td>● Coordinate with sponsor for attendance at final presentation</td>
</tr>
<tr>
<td>Week 10</td>
<td>● Final presentation scheduled</td>
</tr>
<tr>
<td></td>
<td>● Project poster concept</td>
</tr>
<tr>
<td></td>
<td>● Begin Sprint 5</td>
</tr>
<tr>
<td>Week 11</td>
<td>● End sprint 5</td>
</tr>
<tr>
<td></td>
<td>● Preliminary project poster</td>
</tr>
<tr>
<td>Week 12</td>
<td>● Project poster delivered to Senior Project Coordinator</td>
</tr>
<tr>
<td></td>
<td>● Begin Sprint 6</td>
</tr>
<tr>
<td>Week 13</td>
<td>● Poster presentation during Software Engineering Project Day</td>
</tr>
<tr>
<td></td>
<td>● Draft final presentation</td>
</tr>
<tr>
<td></td>
<td>● Technical report outline</td>
</tr>
</tbody>
</table>
| Week 14 | • End Sprint 6  
|        | • Give final presentation this week, or next week  
|        | • Attend required final presentations  
| Week 15 | • Attend required final presentations  
|        | • Deliverables  
|        | • Individual peer evaluations  
|        | • Team final self-assessment  
|        | • Summary of Project Reflection meeting  
|        | • Course evaluations  
| Week 16 | • Finals  
|        | • Deliverables  
|        | • Final project artifacts  
|        | • Final technical report  
|        | • Summary of final reflection meeting  
|        | • Senior survey  
| Week 17 | • Graduation  

**Actual Schedule**

We deviated from the original project plan a little based on when we got the hardware in the monitoring phase. We identified these risks early on and had contiguous plans that we were able to enact. In this case we focused on what we had and combed through documentation we were sent in a zip. Otherwise we were able to stick to our schedule and produce the deliverables we wanted to by the end of phase 1. For the Upgrading Phase, we actually did our sprint Zero over Christmas break in order to hit the ground running when Semester 2 started. We did not create a schedule until week 9 since there were no deliverables outside of our sprints beginning or ending to track. This worked well as we had no real deliverables or things to schedule. During week 8 we got together and created this schedule. We have not deviated yet from the schedule in Semester 2/upgrading phase. The sprints themselves served as pseudo milestones where at the end of each one we were able to see how we had been progressing.
System Design

Monitoring Phase
After identifying our functional requirements we decided to take time to investigate what the industry was currently using for monitoring. Each group member did a one pager on an off-the-shelf solution. This proved to be very fruitful as we wound up selecting one of the technologies we discovered. Nagios which is used in many data centers and provided a lot of the functionality Spectracom wanted. It had a dashboard that would allow you to specify where something was and what you wanted to monitor. It would also provide emails and other notifications about customer alerts like a ping not responding in a certain amount of time. Nagios was also open source so once we had the okay from Spectracom we dove in and started by identifying KPIs. From the list of available information available through the Time Server API, we worked with Spectracom to develop a list of key measures (e.g. frequency monitoring and oscillator configuration info and real time performance, PTP performance, GPS signal quality,...) Each one of these measures were deemed a KPI that we would monitor. As a team we were interested in utilizing another tool that would make deployments easier. We decided we wanted to use docker to deploy the application. The next step was to create a proof-of-concept Docker deployable prototype that utilized Nagios to query to retrieve
a few KPIs. Once we realized what a KPI was and how we could get it through the TimeServer we then just produced a few proof of concepts. The first was getting Nagios to run in Docker. Once that was complete we just wanted to monitor something in Nagios. Once that was completed we brought it together by monitoring a specific KPI. After that proof of concept we also created a Nagios plugin. This could be distributed if a customer could not use Docker. At this point our phase ended but had we continued it would have been a turn the crank kind of situation as we would be just adding more and more default KPI’s. The final product that we handed over was a polished proof of concept that would have been more refined had we had more time or another semester to focus on it. The next steps would include adding visualization through graphite which would add a layer of analysis. Nagios and Graphite play quite well together so this too would have been easy to integrate and deploy. Below you will see how our Nagios plugin works at a high level.

Upgrading Phase

The beginning of semester 2 marked the beginning of a new phase for us. We switched to a full scrum implementation with a sprint zero. During the sprint zero we built up our product backlog by eliciting requirements from our familiar contacts. Once we had a good idea of what the customer was looking for we sat down and chose our tech stack. We had good success with Docker and liked how easy it made local development synchronization so we decided to use it again. We had a long discussion about which programming language to use, narrowing
to CakePHP, Python, and Ruby on Rails. We choose Ruby on Rails because we had two experts, one competent, and one familiar team member -- by far more than any of the others. After getting the go ahead from Spectracom we grabbed our tasks for sprint one. We decided that 18 story points would be the ideal amount of work. The initial sprint was building the foundation so we wanted docker up and running and a hello version of rails running. During this time we also decided our application would have two users. An admin who could run everything and a normal user who could login and just read. Aside from users our application would also hold a TimeServers. To represent TimeServers we use a name, an ip or domain, and then what version they are currently running. The last thing we needed to model for our system was what we call a credential set. In this case a credential set is what password and username are used to login to a specific box. We use this to run the actual upgrade on each specific box via SSH. Once completed and demoed for the customer. We continued, adding logging and admin capabilities. Devise the gem we choose for Authentication afforded us a super secure system out of the box. This gem is industry tested and quite easy to use. We paired that gems like bootstrap, a async request loader, and many development gems. While coding our parallel and serial run configurations we kept it abstract. At the end of each sprint we demoed and used the feedback going forward. One of the design decisions we had to make was how to show that many boxes were updating at once. We decided we would show a group page with individual progress bars for each box that was updating. If you were updating serially you would only see one progress bar moving at a time. If the updates were happening in parallel you would see each bar moving at the same time. We showed this concept to spectracom and they greatly enjoyed this. Also with this design Spectracom would be able to easily extend this system adding the capability to allow the customer to define their own update methods which you could still easily visualize. To run the actual updates we created a script that copies the updated script to each TimeServer and then calls the associated TimeServer command to run the update. In this case if you use a force flag it overwrites everything there etc. So our system is merely an aggregation wrapper. The rest of our design decisions were based on where we see the product going. If Spectracom ever wanted to add automation of version checking from an RSS feed we wanted to have scaffolding in place to easily support such a change.

**Process and Product Metrics**

The metrics we captured during the monitoring phase were not that usable. We costed all of our tasks based on how long we thought they would take in hours. This was not reliable and a lot of our tasks were ill costed. The one thing Greg-gile did was to set week long sprints which was a short period of time which was good for keeping us all accountable for our work.
Here you can see that all of our sprints varied greatly. This was because we made tasks out of formal processes, like our SRS document and project plan. Although this worked and we were successful in producing something the customer asked for, we felt this would not be the best approach for next Phase.

At the start of Phase 2/Semester 2, we put in place a pure Scrum approach. This approach requires not only us to change our process but our customer had to adapt as well. When we got them on board the changes were immediate and easy to see. The first was the visibility into the system was increased as there were never any surprises. There were fewer questions about direction etc. During this phase we captured and adjusted our velocity based on our individual schedules etc. We also used planning poker to cost our items. This worked much better time-based costing as we were able to relatively accurately cost our tasks. This accurate costing lead to more tasks being fully done done.
As you can see we were able to complete almost all tasks at a healthy rate. We chose 16 as our default velocity which we would adjust if a few of us would be gone for interview or something else. This worked very well and the added customer collaboration made this semester seem a lot easier. At the end of each bi weekly sprint an email was sent out recounting what was accomplished, what we were doing next, general info, risks, and an analysis of the burndown. For example the analysis for sprint 1:

Our burn down chart is healthy and it seems a good velocity for each sprint is around 17 sp
The big jumps down shows that we do most of our work on the weekends and when time allows after a meeting. Please let me know if you have any questions or comments

Based on when work was getting done we did not pay too much attention to the burndown unless tasks were being slipped. When tasks started to slip it became clear it was because they had unforeseen errors. Since we were only working weekends really this set tasks back. To remedy this we started enforcing status updates more strictly at each of our 3 meetings during the week. This public shaming and increased collaboration made us more agile and able to deal with these unforeseen risks.
Product State at Time of Delivery

Monitoring Phase
The state of the project was solid at the time of delivery. We have comprehensive requirements and design documents. The Spectracom APIs were undocumented from the start. Since we needed them to see what KPIs could be monitored, we created this documentation. At the time of handoff, we had Nagios configured, deployed in a Docker file, and monitoring the most important KPIs. If we were going to continue this project into the second semester we would have implemented the rest of the KPIs and some of the dashboard work that was missing, but the state of the project was at a “turn the crank” point.

Upgrade Phase
The current state of the project is one that satisfies the major asks. The system is easy to deploy and administer. Once deployed the system can allow you to perform upgrades to TimeServers that are properly configured. While the upgrade is being performed you can visualize it on our dashboard. Once handed off spectracom will also get our documentation and our full product backlog which contains tasks information to continue building the system. We are delivering exactly what was promised which may be taken and improved later on. It is another good proof of concept.

Project Reflection
Overall the project ran quite smooth all things being considered. Starting way back with team formation we all were able to come together and stay in a performing phase for quite some time. The role of PM was given to Greg based on his most recent co op experience which left Drew at a bit of identity crisis. Drew was able to mold himself into the tester and customer collaborator. We set ourselves up for success early on with a good requirements gathering round. During this time we were able to really narrow down what solution Spectracom wanted us to build. During this time we noted that Spectracom really wanted an Upgrade system but we made it clear that was not our focus but instead something we would address when Spectracom felt the monitoring project was done. In hindsight we were so focused on the document that I think we missed subtle clues that they actually were more interested in an upgrade system. This would come back to haunt us later on as they continued to ask about upgrading. It was about week 9 after we had visited Spectracom that we sat down and talked about potentially switching projects. We were not super enthused as we had signed up for monitoring not upgrading but it seemed like would really make the customer happy so we sat down and had some scope change meetings. We set the expectations of what the project shift would be like. What we would hand over for monitoring and what they could expect from us for upgrading. They agreed and we began basically a new project over intersession starting with a good sprint 0. We set up our process and product backlog initially. We carried our initial momentum through semester 2 having only
one bad sprint where we slipped some tasks and lost motivation. To fix this we sat down and talked about and reminded ourselves we were almost done! The last couple of weeks have been great as the code hit the point we wanted and we have focused on just cleaning things up and getting them ready for the final handoff. During this time I wish we had implemented more testing but since it was near the end of our senior year there was an understandable lack of interest in the project about week 14. We as a team worked very well and hard on this project and it's something we are all proud of.

References

- User Classes
- SRS for Monitoring
- Design Doc for Upgrading found in the actual code