Snap on Windows

IT 446

Phillip Anderson

McKade Clements

Devin Durtschi

Mathew Kuhn

Jesse Millar

Coach: Dr. Jay Ekstrom

Sponsor: Taylor Thomas

# Table of Contents

[Table of Contents](#_7vn9fvbdc6fq)

[Abstract](#_fjdt6vkyq1ul)

[Introduction](#_j7wci9ff0fce)

[Concept Definition](#_a7ugbtbeeld0)

[Explanation](#_wh4n88z1yv17)

[Stakeholders](#_ohpgk9a1bqc8)

[Intel](#_webm1tb3qxo)

[Companies with Windows Systems](#_ai185e49yobg)

[Operators of Windows Systems](#_6acj4prdbjyh)

[Open-source Maintainers of the Snap Project](#_ob1asef5gh4z)

[Members of the Snap Community](#_bzcq5e89thzq)

[Stakeholder Requirements](#_3bstj8nprbsc)

[Validation](#_5u4793drpsh)

[Verification](#_8yact62egvuu)

[System Definition](#_g7508nb77l6d)

[System Requirements](#_avgdfvx9it7t)

[Logical Architecture](#_mqhfx7uvnbt7)

[Traceability](#_g3mf7j8v3sr)

[Realization Plan](#_i2w8qpwn4e8k)

[Learn and Use Go and Powershell Environments](#_6vw9l5t0gcte)

[Build Windows Test Environment](#_efogv3bl9gfy)

[Write Perfmon, SysInternals, Active Directory Plugins](#_ik87w8vfde3w)

[Test Perfmon, SysInternals, Active Directory Plugins](#_7xzyx583e3x3)

[Build a Snap Executable for Testing](#_vk24zn41jnnb)

[Create an Automated Build Script for Snap](#_3ajfahmaylv4)

[Merge Code with Main Snap Repository](#_1v27eo3hs4zj)

[Critical Path](#_yibv8rgcm5k2)

[Verification and Validation](#_34dwwfoofzvm)

[Project Management](#_ge26xd9cb6a7)

[Objective Statement](#_y04gdyt5b4q8)

[List of Deliverables](#_jhjzyb49rum1)

[Conclusion](#_vpm81vpmsfqq)

[References](#_ivdhs8eh0rrp)

[Appendix](#_p556xuugiavr)

# 

# 

# Abstract

Snap for Windows is an Brigham Young University Information Technology 2016-2017 capstone project. Currently, Snap does not function on Windows systems and the initial portion of this capstone project revolves around the Windows porting process. Our plans are to port Snap to Windows and create an easy install process. We then will make a small collection of Windows-specific plugins: Perfmon, Sysinternals, and Active Directory.

# Introduction

Snap is an open source project started in July of 2015 with the intent of creating a single API for collecting telemetry data in Unix-based systems. Telemetry is the gathering of system metrics from a distance. For example, one might want to measure the number of established TCP connections there are on individual machines forming a cluster. This data could be analyzed and then applied for load balancing or troubleshooting purposes.

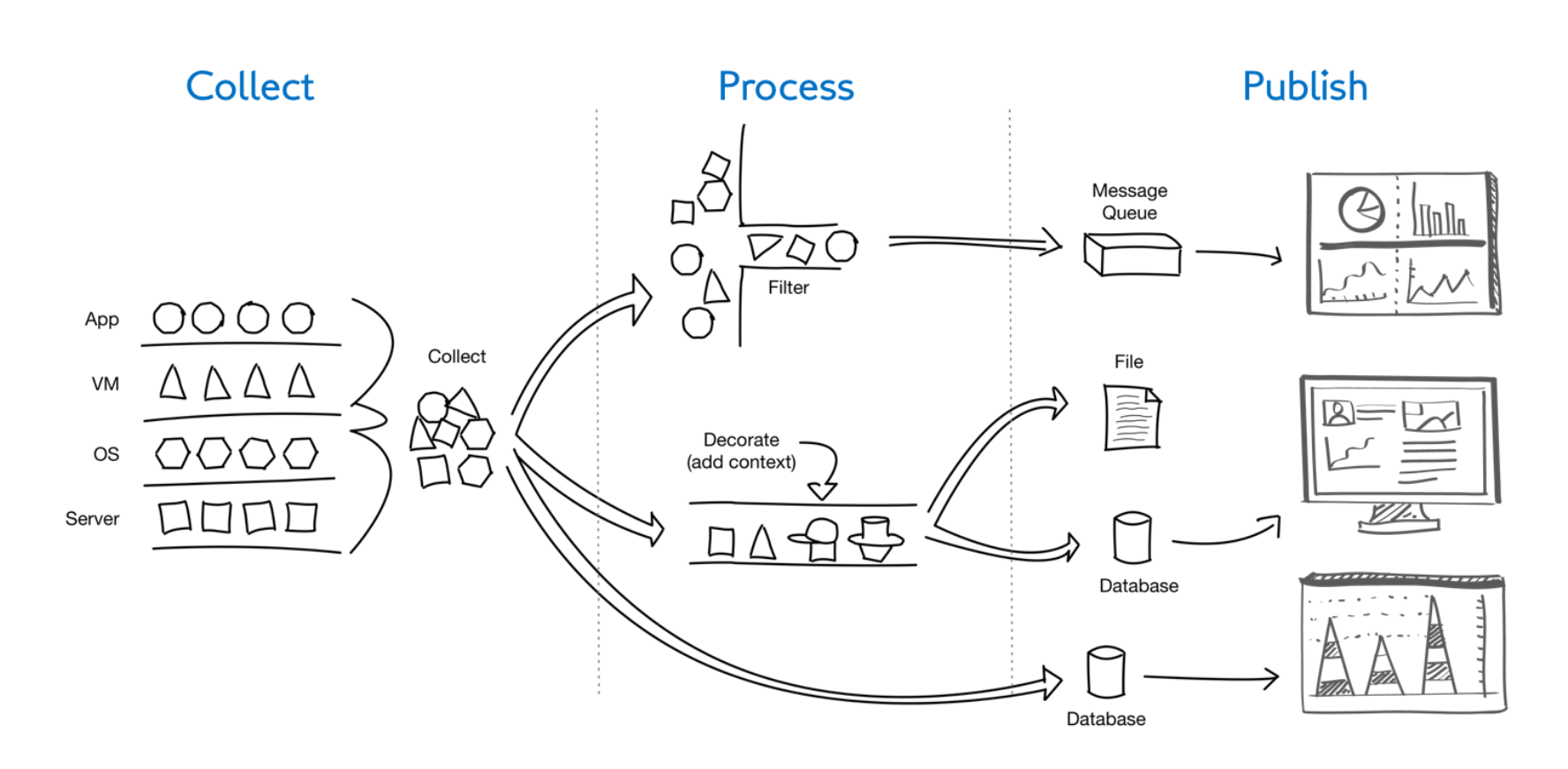
Snap allows for collected data to be processed as well as published. Processing data involves adding context, or “data about data,” to the collected information. It also allows for generating new information from the data collected, such as a calculated average of a certain metric across all devices in a cluster. By publishing the data, the information collected and processed can be analyzed by the user in the form of text files, database outputs, and reports.

This document was created as a part of the Brigham Young University Information Technology capstone class of Fall 2016-Winter 2016. It outlines the process and finished products that will be used and turned into our coach and sponsor at the completion of the Winter semester.

# Concept Definition

## Background

Plugins are the method by which the collecting, processing, and publishing of data takes place in Snap. Plugins are modular programs, written in Go, which can be mixed and matched to allow for customized metric outputs. Plugins are grouped into collector, processor, or publisher categories and run on top of the main Snap process. The data from the plugins is accessed through the Snap API. Plugins can be built and ran as needed to collect whatever type of telemetry data system administrators may need, making them a critical component to Snap.



Plugins are run using “tasks.” Tasks define the data collector, the amount of time it should run (whether infinitely or for a certain length), the plugin used to process the data, and the publisher plugin which outputs the data in a certain format. Tasks can also define the metrics gathered from a plugin. For example, the netstat plugin may include metrics for open TCP connections and established TCP connections, but the task may only require the collecting of open TCP connection information.

There are a number of products similar to Snap. Nagios, which performs monitoring of network devices, is one example of a Snap alternative. However, these tools are not as streamlined or modular as Snap, which makes collecting data from large corporate networks less than ideal. Similarly, Windows has its own telemetry collection tool, but it is limited to Windows-based networks and is thus not as scalable as Snap.

Because the Snap project is open source, the code behind Snap is available in a Github repository where a thriving community actively works to make Snap better. Common contributions are bug fixes, additional plugins, and sharing of information through searchable discussions. The repository is well documented and contains deployable Snap binaries.

Currently, Intel systems mostly run Windows, but Snap plugins have only been built for the Unix environment. There has been a large push by the Snap community, which not only includes individuals but also large corporations like Intel and Staples Inc., to make it possible to use Snap on Windows-based systems. In order to make a Windows port fully featured, plugins need to be written to gather telemetry data from programs like Active Directory and Perfmon. Some of the existing Snap plugins, like netstat, can be cross-compiled for Windows. Since Snap is written in the Go programming language, this cross-compilation is trivial.

Snap can currently be run on Windows, but is not very useful without built-for-Windows plugins. Our task, as defined by our Objective Statement, is to automate the Snap build process and port at least three Snap plugins to Windows, including Perfmon, Sysinternals, and Active Directory.

## Stakeholders

The following table is a summary of each of the stakeholders, including the role of each party within Snap (interest), and the benefits received by each party upon completion of our project. Below the table are brief, high-level descriptions of each stakeholder.

|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Interest** | **Benefit** |
| Intel | Creators of Snap and sponsor of project | Increased sales; double market share by porting to Windows |
| Companies with Windows Systems | Building and using plugins for their own monitoring needs | Easier monitoring and metric-collecting; Less need to hire for monitoring |
| Operators of Windows Systems | Contributing to Snap plugins on behalf of companies | Makes managing systems easier |
| Open-Source Maintainers of Snap Project | Participating in Snap creation and maintaining | Economic benefits include selling more Intel products which run Windows and monitoring environments more easily |
| Members of the Snap Community | Participating in Snap community; May be members of above stakeholder groups as well | Learn more about telemetry |

### Intel

Intel holds a direct business interest in getting Snap to work on Windows because Windows is run almost exclusively on Intel processors. Having more functionality added to Windows helps Intel sell more of their own products. By improving upon a telemetry framework specifically designed to monitor Intel devices, customers can use Snap to reduce the labor needed to monitor systems, thus reducing their total cost of ownership.

### Companies with Windows Systems

Companies with existing Windows systems could benefit greatly from a more concise tool for telemetry. Utilizing Snap allows companies more flexibility and adaptability by making it easier to integrate with Unix based systems should they desire to “mix and match” or eventually migrate from Windows systems to Unix systems.

Native Windows support in Snap reduces the total cost of ownership for third-party companies by limiting the need to train operators in more than one telemetry system or having several operators to maintain separate systems.

### Operators of Windows Systems

Distinct groups of operators such as app developers, system administrators, lab managers, and database administrators can all use Snap to monitor system performance of the products they build and administrate.

### Open-source Maintainers of the Snap Project

Maintainers of the open-source Snap project are the main shareholders. The maintainers are the ones that control which plugins and features are added to the Snap code base. Project maintainers have an interest in seeing that the project functions properly as a whole.

### Members of the Snap Community

Members of the Snap community use Snap and are not actively involved in the development of the project. Their interests revolve around the ability to more easily and quickly gather, process, and publish data from their systems in a user-friendly format. Regardless, these users have a vested interest in keeping the project going because Snap is a part of their systems.

## Stakeholder Requirements

The following is a list of requirements as obtained from each of our stakeholders:

* **Snap executable compiled for Windows**
  + The executable is needed so that pre-build binaries can be distributed to those who would like to use Snap, rather than having them figure out the compile process. This initial executable also serves as an initial proof-of-concept that Snap does build for Windows.
* **Automated build script to compile Snap for Windows** 
  + The process to compile Snap for Windows should be automated. This will allow for members of the Snap community to more easily compile Snap, rather than going through a more tedious and complicated manual process.
* **Plugins written in Go** 
  + The plugins should be written in Go to provide greater maintainability for the project maintainers, as they are most familiar with this language.
* **Plugin functionality written in language native to Windows**
  + The gathering of system metrics should be done using either PowerShell or Batch scripting, as these languages provide native interfaces into the Windows API’s and allow for greater ease in gathering the metric data.
* **Follow plugin-authoring framework**
  + The interfaces of the plugin libraries must be observed in order for a seamless merging of our project with the main repository once our project is complete. The guidelines already created in terms of building plugins should be followed so that the plugins can communicate with the Snap daemon.
* **Sysinternals plugin**
  + The plugin should provide metrics associated with common Sysinternals tools. It should integrate seamlessly with the Snap framework, providing data in a form which the Snap daemon understands.
* **Perfmon plugin**
  + Perfmon was specifically requested by a member of the Snap community. It should provide common metrics associated with the native Windows Perfmon graphical application, such as CPU utilization, memory available, and memory used. It should integrate seamlessly with the Snap framework, providing data in a form which the Snap daemon understands.
* **Active Directory plugin**
  + The plugin should provide metrics which can be gathered from Active Directory, and should be tested within a server environment with Active Directory. It should integrate seamlessly with the Snap framework, providing data in a form which the Snap daemon understands.
* **Testing of each plugin**
  + The plugins should be tested within a server environment, simulating a real-world corporate environment. This development environment should contain a domain controller, Active Directory, and multiple Windows devices to gather data from.
* **Issue logging system** 
  + In order to look back at issues we may have faced during development, an issue logging system should be used to track all issues we come across. This makes the project more maintainable after we leave.
* **Project documentation** 
  + As with the issue logging system, any special bits of information needed to work with our plugins should be recorded, for maintainability purposes. Existing documentation may need to be modified as well, to include instructions for using Snap on Windows platforms in addition to Unix-based platforms.

## Validation

We have spoken with our sponsor, Taylor Thomas, and have discussed each of the above requirements during the course of several video-conferences. As we gained a better understanding of the requirements and presented them to Taylor during these meetings, he was able to approve each of them. We feel comfortable with Taylor’s approval of the requirements as he himself works for Intel, who is the main benefactor of our project. He has also been in contact with other stakeholders, including Staples (company with Windows systems), who have confirmed many of these requirements themselves.

## Verification

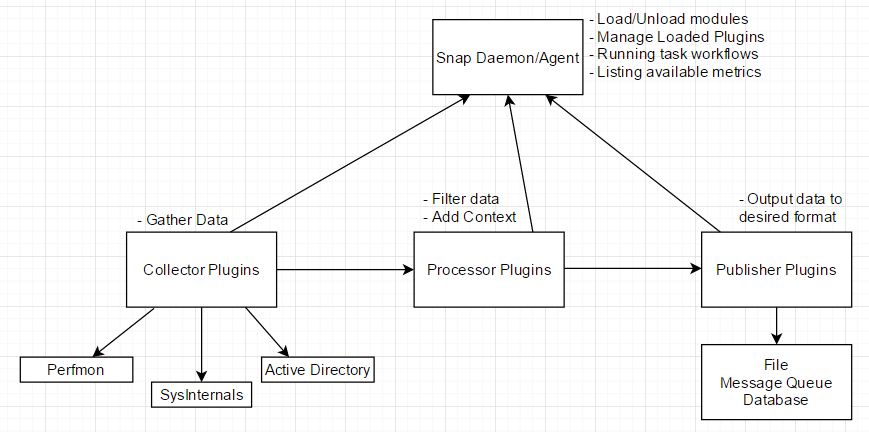
Once we complete the initial requirements outlined above, we will meet with several Intel representatives (our sponsor) to review the requirements and request feedback in terms of what we may need to modify or if they feel we should move on to our stretch-goals. We will also address the other stakeholders via the dedicated Snap Slack channel and verify our work. If any corrections or revisions are requested, we will make goals to fulfill each before the project deadline.

# System Definition

## System Requirements

* **Snap executable compiled for Windows**
  + The Windows executable should have the same functionalities as the binary build for Unix systems
* **Automated build process to install Snap on Windows**
  + The installation process should be as simple and standard as possible to allow as many system administrators as possible to install Snap quickly
* **Plugins written in Go**
  + The code for our plugins needs to be readable and easily maintainable
  + Our code should be simple to cross-compile
* **Plugin functionality written in language native to Windows**
  + Our code should work well in Windows environments
  + Metrics should be gathered and manipulated in the most efficient way possible to avoid introducing latency into client systems
* **Follow plugin-authoring framework**
  + Plugins must integrate well with existing Snap framework
  + Plugins should be completely modular
* **Sysinternals plugin**
  + Must pass PsUtils data to Snap
* **Perfmon plugin**
  + Must gather CPU and memory utilization metrics
* **Active Directory plugin**
  + Build and maintain a documented development environment which implements Active Directory
  + Retrieve, parse, and send Active Directory usage data to Snap
* **Testing of each plugin**
  + Write small unit tests as mandated by the
  + Development environment which implements a domain controller
  + Development environment implements Active Directory
* **GitHub issue log**
  + When we finish the capstone project, we should leave behind enough documentation and searchable discussion about our project, code and the issues we hit as possible
* **Project documentation**
  + Documentation is stored in public and searchable GitHub wiki format

## Logical Architecture



## Traceability

The traceability of this system has been defined by relationships between system components and the system requirements and how those requirements relate to stakeholder requirements, as seen in the sections above. The Snap and plugin components fulfill the system requirements of being able to collect metrics in a distributed fashion. The plugins also fulfill the requirement of being modular and integrating well into the existing Snap framework. In terms of stakeholder requirements, the system requirements allow for the stakeholder requirements to be fulfilled in the most effective way possible. The stakeholder requirements of maintainability of code are also fulfilled by the way we have defined the plugins to be built and the languages we have chosen to implement them in.

## Realization Plan

In order to fulfill our objective of porting Snap from Unix-based environments to Windows in a way which will be maintainable by the project maintainers, we will need to complete the following tasks. The specific task and resource requirements for each system component are defined below:

### Learn and Use Go and Powershell Environments

We have already learned much of the Go syntax and the understand basic powershell cmdlets needed to write our plugins. We will need environments, however, to run and test Go and powershell code. We are currently using our personal laptops or virtual machines for our Windows platforms, and are using various Integrated Development Environments which have their own plugins for each language, respectively, for our plugin writing and testing.

### Build Windows Test Environment

We will need to build a testing environment which implements a domain controller as well as Active Directory in order to fully test each plugin in a cluster setting. This will also allow us to test the Active Directory plugin as well. We will also require a Windows 2016 server if time allows us to build and test a Windows 2016 Docker plugin. As Kade works in the Cage, he has access to the server resources needed to build this development environment.

### Write Perfmon, SysInternals, Active Directory Plugins

We will need to build fully-functional collector plugins to mimic the functionalities of common Windows-metric collecting tools. This will require an understanding of how the plugin libraries function, as well as time on our parts to write and test the code. We will also need a GitHub repository so that we can collaborate effectively on code with version control.

### Test Perfmon, SysInternals, Active Directory Plugins

In order to make sure that our plugins work in a way which will be maintainable by Intel, we need to understand the process used by the Snap community to correctly write and test plugins, so that the plugins conform to the pre-conceived Snap standards. We have read over the plugin authoring documentation, and will need to continue checking in with our sponsor and the documentation to make sure that we are following correct procedures. This will be important to successfully merge our code with the main Snap repository. This will take time and collaboration with Taylor to gain a full understanding of the authoring process.

### Build a Snap Executable for Testing

The Snap executable will be used to test our Snap plugins in a Windows environment. It will also be used to distribute Snap to end users in a form that runs on Windows without having to be compiled. We have been able to successfully build a Snap executable for Windows manually using built-in Go compile features. This process will need to be converted into an automated build script for greater compile ease in the future.

### Create an Automated Build Script for Snap

A script will be needed to automate the build process so that future users who wish to compile their own executables do not have to follow a complex manual process, thus minimizing risk. This will require the CMake application, which allows for Unix’s. Make-like capabilities within a Windows environment.

### Merge Code with Main Snap Repository

In the end, we will need to merge our code so that it can be used by the Snap community. As stated earlier, this will require us to strictly follow the plugin authoring guidelines, and we will need to collaborate with our sponsor to ensure that it is done without breaking any other bits of code.

## Critical Path

The critical path for our project begins with gathering a basic understanding of the existing Snap framework. From there, we must compile Snap for Windows, build an automation script to compile Snap automatically, write the three plugins, develop a testing server environment with necessary components, perform plugin testing, and document necessary data relevant to maintaining our plugins. Finally, we will merge our code with the main Snap repository.

## Verification and Validation

In order to verify and validate the fulfillment of each system requirement, we will submit our completed automated build process for Windows, as well as our completed plug-ins, to the Snap community via Pull Requests on GitHub for approval. This will include documentation and unit tests for each feature. In the case that adjustments need to be made, the Snap community will notify us via the GitHub Pull Request and we will make the necessary adjustments. Upon approval, the community will merge our contribution into the existing Snap repository signifying it is up to the community’s standard.

# Project Management

## Objective Statement

Automate the Snap build process and port at least three Snap data collection plugins for Windows, including Perfmon, Sysinternals, and Active Directory by March 20th.

## List of Deliverables

Our project of porting Snap and a collection of plugins to Windows will be complete when the Snap project maintainers accept our code into the master branch of their GitHub repository. As part of our pull request that presents our work to the project maintainers, we will need the below deliverables.

* Plugin documentation describing the functionality of our newly-built plugins
* Unit tests for our new plugins to automatically verify that the plugins work as intended
* A script to automatically build Snap for a windows machine
* A collection of Snap plugins built for Windows systems

In order to complete the academic aspects of our capstone project, we will also need the below deliverables.

* Project presentation slides for our final presentation
* A project video outlining our project in a concise and engaging way

# Conclusion

Through Snap’s improved way of collecting metrics through a single API, we believe porting this functionality to Windows systems will greatly improve troubleshooting and monitoring on those systems. After publishing data, the information collected and processed by Snap can be analyzed by the user in the form of text files, database outputs, and reports. We look forward to improving upon the existing Snap framework and completing the development and testing of our planned plugins.

# References

[1] Cheney, D. (n.d.). Cross compilation with Go 1.5. Retrieved October 05, 2016, from <http://dave.cheney.net/2015/08/22/cross-compilation-with-go-1-5>

[2] Command go. (n.d.). Retrieved October 05, 2016, from <https://golang.org/cmd/go/>

[3] Develop Perfmon windows performance monitor plugin · Issue #1175 · intelsdi-x/snap. (n.d.). Retrieved November 05, 2016, from <https://github.com/intelsdi-x/snap/issues/1175>

[4] Go by Example. (n.d.). Retrieved October 05, 2016, from <https://gobyexample.com/>

[5] How to Write Go Code. (n.d.). Retrieved October 05, 2016, from <https://golang.org/doc/code.html>

[6] Intelsdi-x/snap. (n.d.). Retrieved November 05, 2016, from <https://github.com/intelsdi-x/snap/blob/master/docs/BUILD_AND_TEST.md>

[7] PLUGIN CATALOG. (n.d.). Retrieved October 05, 2016, from <https://github.com/intelsdi-x/snap/blob/master/docs/PLUGIN_CATALOG.md>

[8] SNAP REST API. (n.d.). Retrieved October 05, 2016, from <https://github.com/intelsdi-x/snap/blob/master/docs/REST_API.md>

[9] Support flow collector like sflow or netflow? · Issue #1030 · intelsdi-x/snap. (n.d.). Retrieved October 05, 2016, from <https://github.com/intelsdi-x/snap/issues/1030>

[10] Windows support · Issue #671 · intelsdi-x/snap. (n.d.). Retrieved November 05, 2016, from <https://github.com/intelsdi-x/snap/issues/671>

# Appendix

## Constraint Matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | Scope | Schedule | Resources |
| Most Constrained |  | **X** |  |
| Moderately Constrained | **X** |  |  |
| Least Constrained |  |  | **X** |

The schedule of our project is by far its most constrained aspect in our constraint matrix. We simply must have the project completed by mid-March in order to be prepared to present our work at the end of the capstone class. The scope of the project is moderately constrained because we are making a port of existing code, not adding sweeping new features.

Our resources are nearly limitless within the scope of the project. We have access to an active communication medium with the creators and maintainers of the project, a direct line of communication to our sponsor, full-featured Snap documentation, and, hopefully, far more time than we will need to complete our additions.

## Governance Framework

In order for the project to progress as smoothly as possible, we have agreed upon a governance framework. Utilizing this governance framework to decide on individual roles inside the team now will make enforcing team responsibilities easier than if conflicts were resolved when they arise later.

Any and all conflicts among team members or pertaining to project decisions will be resolved by team majority vote. These decisions become final when voted upon and written down in the GitHub project manager. If the conflict involves the open-source code we are contributing to, the issue should be resolved by leaders of the open-source project via Slack or discussion in a GitHub issue thread. As contributors to the project, we want to ensure that we are following the procedures already established by the Snap community and not stepping outside the bounds provided to us.

We will meet as a team twice a week, once with only the students on the team, and the other with the students and coach. The first meeting will happen on Wednesday evenings and the second will happen on Friday mornings. The Wednesday meeting must be attended by all members of the team either in person or via Google Hangouts. The Friday meeting should be attended in person by all team members. We will meet with our sponsor bi-weekly via the Zoom team collaboration application. As member availability may vary for these Zoom meetings, only members who are able to attend will be required to participate. If members consistently miss any of these meetings, we will speak as a group and decide upon a time that works better for everyone.

## Communication

Frequent, clear, and searchable communication is integral to the success of this project. For questions regarding the needs of the Snap community or general Snap discussion, we will use Slack. Slack is the main form of communication currently used by the project maintainers at Intel. Google Hangouts will be used for capstone team communication, remote meetings, and general discussions between team members. As an alternative means of communication, a Slack channel has been created specifically for our team’s use.

Our meeting notes, personal notes, and other class-related documentation will be kept in a shared Google Drive folder. All other documentation will be housed and maintained on our GitHub project’s wiki. GitHub’s project management features will be used to keep track of our task list and critical path. GitHub provides a number of tools that make tracking the progress of fixing bugs and adding features very fluid.

## Acceptance Documentation

As part of the contribution guidelines, the Snap project maintainers request a single GitHub pull request with one commit per feature added. To accomplish this, we have created a GitHub organization that includes every member of our capstone team and forked the Snap project into our organization. Forking a project in GitHub makes a copy of the code as it currently stands and enables us to modify files without restriction.

When we are done adding and testing a specific feature, we will use GitHub’s interface to create a pull request. Creating a pull request sends a notification to the Snap project maintainers informing them that there is new code for them to review and potentially merge into the master branch of their project. Once our code is accepted and merged, our changes will be made available to the community in the precompiled Snap binaries.

Additionally, we will need to draft documents outlining our feature addition proposals to give to our sponsor. This step can be seen as a preliminary requirement approving of our plans before we set off in a potentially wrong direction.