# Software Requirements Specification

## Project Name: Augur

| | |
|---|---|
| Author | BLAKE RUPRECHT |
| Pawprint | BCRF53 |
| Class | CS_4320 |
| Instructor | GOGGINS |
| Assignment | ASSIGNMENT 9 |
| Previous Revisions | 2020-MAR-29<br>2020-APR-01<br>2020-APR-04 |
| Final Revision Date | 2020-APR-05 |

# 1. Introduction

This document is intended to partially specify the requirements and detail the software architecture of Augur. This Software Requirements Specification (SRS) will describe the software system scope, the system architecture, the features, system usage, actor surveys, system requirements with use cases, functional and non-functional requirements, design constraints, purchased components, and interfaces.

Augur is focused on building *human centered open source software health metrics* defined by collaborations with the Linux Foundation's CHAOSS project and other open source stakeholders. This SRS will go over how this will be achieved.

# 2. Software Product Overview

## 2.1 System Scope

Augur is a collection of human centered open source software health metrics based on collaborations with the Linux Foundations CHAOSS project. Augur is available as a web application that automatically reads data from open source software projects and utilizes and displays health metrics based on those projects, with the goal of making sense of data.

The web application includes features necessary to perform health analytics on software projects, including specific insights about the project, and the abilities to track different repos and groups. The key strategies used by Augur in terms of making sense of data are 1.) Enabling comparisons, 2.) Make time a fundamental dimension in all metrics, 3.) Make all data driving visualizations downloadable as .csv, 4.) Make all visualizations downloadable as .svg.

## 2.2 System Architecture

The system is designed as a web application that gets data from the projects that are to be analyzed. The data is collected, organized, and kept up to date by Augur. Datasources are exposed as REST APIs, and Augur also caches, registers plugins, and reads configuration files. A Vue frontend is used to display controls and visualizations. Visualizations are made by Vega-Lite for the metrics exposed by the backend.

## 2.3 Feature Overview

This is a high level overview of the features that can be used by a user of Augur, and the developer who can add repos/groups to Augur.

User

- F1: Select Repo Group - multiple different repo groups can be viewed in the augur web application, allowing for the user to choose which one to view
- F2: Select Repo - allows user to choose which repo in the group to view
- F3: Overview - see analytics like lines of code added by different members, license coverage, top 10 authors, reviews per week, new issues per week, etc.
- F4: Compare Overviews - compare different repos to each other in the overview section.
- F5: Risk Metrics - view analytics like forks count by week, committers by week, licenses declared, CII best practices, etc.
- F5: Compare Risk Metrics - compare different repos to each other by risk metrics.
- F6: Insights - not fully implemented, but will provide insights about different repos and allow comparisons.

Developer

- can utilize all features of the user
- F1: Add Repo Groups - add groups of repos to the list of stored groups
- F2: Add Repos - add repos to a specific group
- F3: Run - control the starting and stopping of the server
- F4: Shell - create an iPython shell with an instance of Augur's application class
- F5: Make Install - install all Augur dependencies
- F6: Make Clean - remove logs, caches, and other junk
- F7: Make Dev - start frontend and backend servers together
- F8: Testing - run testing on all aspects of Augur
- F9: Make Docs - make and edit documentation of Augur
- F10: Create a Metric - perform the next three features to build a metric visualization
    - F10.1: Create a Metric Function
    - F10.2: Create a Metric Endpoint
    - F10.3: Build a Visualization

# 3. System Use

The main way that a user will use the system involves installing Augur and creating the backend dependencies for whichever open source software project the user wishes to analyze. They will also install the frontend, and Augur itself, to create all of the analytics and visualizations of the data. The user will also be able to build custom metrics to include in Augur.

### 3.1 Actor Survey: User

The user will install Augur, including Augur's backend, the data collection workers, generate documentation, generate the configuration file, and optionally install the database schema and sample data, and the Augur frontend and it's dependencies. From this point, the user can:

- add repo groups
- add repos
- start and stop the server
- run tests on Augur
- view metrics and analytics
- download data .csv
- download visualization .svg
- create new metrics

# 4. System Requirements

## 4.1 Use Cases

Use Case UC01 - Add Repos

| | |
|---|---|
| *Use Case Name* | UC01 - Add Repos |
| *System or Subsystem* | Augur |
| *Actors* | User |
| *Brief Description* | This use case describes how a user can add repos to their instance of Augur |
| *Basic Flow of Events* | The flow begins when the user is in their Augur directory:<br>1. Create a repos.csv with all repos URLs and the groups they must belong to<br>2. run `augur db add_repos repos.csv`<br>3. The repos are added to the respective repo groups |
| *Alternate Flow of Events* | |
| *Special Requirements* | |
| *Pre-conditions* | The user must be in the Augur directory to run the augur db command |
| *Post-conditions* | The .csv file must be in the specific format of:<br>`<repo_group_id>,<repo_url>`<br>`...`<br>`...` |

Use Case UC02 - Run the backend server

| | |
|---|---|
| *Use Case Name* | UC01 - Run the backend server |
| *System or Subsystem* | Augur |
| *Actors* | User |
| *Brief Description* | This use case describes how a user can run just the backend server, and show the logs in the terminal |
| *Basic Flow of Events* | The flow begins when the user is in their Augur directory:<br>1. run `augur run`<br>2. The backend server is started up<br>3. The logs will be displayed in the terminal |
| *Alternate Flow of Events* | |
| *Special Requirements* | |
| *Pre-conditions* | The user must be in the Augur directory to run the augur db command |
| *Post-conditions* | The logs will be displayed in the terminal<br><br>`ctrl-c` will kill the operation |

### 4.2 System Functional Specification

User Terminals

UT1: run augur backend servers
UT2: add repos
UT3: add repo groups
UT4: get repo groups
UT5: util shell
UT6: install project dependencies
UT7: clean project
UT8: rebuild project
UT9: start frontend and backend servers together
UT10: log frontend and backend
UT11: start the frontend server

UT12: start all installed data collection workers
UT13: check status
UT14: test
UT15: make docs
UT16: create a metric

Frontend

UT17: users must be able to compare repos
UT18: time must be a fundamental part of all metrics
UT19: all data used to drive visualizations should be downloadable as .csv
UT20: all visualizations should be downloadable as .svg

### 4.3 Non-Functional Requirements

NFR1: Augur must be open source and readable by users
NFR2: Augur must be readable and make sense to users
NFR3: Augur must not fail to run backend servers
NFR4: Augur must not fail to run frontend servers
NFR5: Augur must allow users to read and edit documentation
NFR6: Augur must allow users to create new metrics
NFR7: Augur must function on all modern browsers
NFR8: Users must be knowledgeable of command line, directories, Python

# 5. Design Constraints (at least 5)

- The major ethical standard to adhere to is to be Open Source Software, with all proper documentation and licenses
- The system must run on all major web browsers
- The system must be able to be run and edited from Windows, Mac, and Linux OSes
- All code for the system must be open source and on a public github
- The system should utilize Python, Pandas, Flask, Requests, MySQL, Gunicorn, Docker, pytest, tox, Sphinx, apidocjs, VueJS, Vega-Lite, Brunch, Stylus, Babel
- Data must be brought together from Github, issue trackers, mailing lists, library dependency trees, the Linux Foundation's badging program, code complexity and contribution counting, and more!
- The system should format data using PostgreSQL 10 or 11
- The system shall minimize the number of hardware constraints wherever possible.

# 6. Purchased Components

Since most of the software run by Augur is open source and free, the only things that must be purchased are as follows:

- Computer for editing - - - - - - - - - - - - - - - - - - - - $ 100
- Web Server (if needed) - - - - - - - - - - - - - - - - - - - $ 300
- Data Storage (if needed) - - - - - - - - - - - - - - - - - - $ 200

# 7. Interfaces

User Interface: the main user interface will be a terminal emulator, allowing the user to use shell commands and specific augur commands to interact with data and Augur itself.

Software Interface: the web application interface will communicate with the internal database server, and display web applications with VueJS, Vega-Lite, Brunch, Stylus, and Babel