Assignment 9

Individual Requirements Analysis

Spring 2020 CS4320 Software Engineering

Tristen Harr

Tjhm9c

1. Introduction

1.1. Purpose

The purpose of this document is to provide an analysis of the requirements for the software involved in building an open source software health and sustainability metrics tool. This document will describe the purpose of the system, as well as the system use. This document will also examine the functional and non-functional requirements of the system. The document will describe the design constraints, and briefly examine the purchased components and the interfaces of the system.

1.2. Scope

Open Source software projects are an extremely important part of the software development environment, probably the largest example of which being the Linux kernel, which has thousands of contributors. Many open source projects are started by a core group of people who believe that the software could be useful to others and therefore release it as an open-source project with varying degrees of licensing, some requiring credit when used, and others offering complete free-use commercially and otherwise. Many of these projects can be great for the software development community, however when key contributors abandon the project for any number of reasons, the projects oftentimes stop being updated, and therefore become irrelevant. An OSS health and sustainability tool can be used to examine open source projects and help the contributors ensure the project is maintained moving forward, even when some of the contributors are no longer working on the project.

This can help open-source projects to ensure that they have a healthy project by allowing them to analyze data regarding their project and work towards hitting key metrics that historically produce sustainable projects that don’t lose support.

2. Software Product Overview

This section of the document provides an overview of Augur as a software product. It contains a summary of how Augur works.

2.1. System Scope

Augur is an OSS health and sustainability tool that is used to allow projects to be examined using a data-centric approach. This allows open source software teams to compare their projects to other projects using metrics such as commit history, top contributors, and other data available on GitHub. It allows teams to analyze a project over time, and provides insights as to how the projects compare to other projects historically. It provides users an easy way to download the data, as well as a way for them to download the visualizations of the data so that they can be used in presentations, and requests for funding.

3. System Use

3.1. Informal System Use Cases

Use Case: Creating a metric

- User writes a PostGreSQL query against Augur's schema to retrieve desired data
- User wraps and parameterizes the SQL query in a metric function
- User defines an endpoint for the metric to be accessible remotely

Use Case: Creating a visualization

- User connects to an endpoint
- User adds a new page

3.2. Actor Survey

Augur Development Team:
The Augur development team is responsible for the development and maintenance of Augur. This actor is responsible for keeping Augur up and running, and providing users a way to use Augur. This actor may also be a user actor.
System Features:
Build/Maintain Database
Design/Maintain Endpoints
Write/Maintain Documentation

Augur User:
The Augur user is the main user of the system. This actor is responsible for utilizing Augur to analyze software projects.
System Features:
Create/Extract Data
Create/Download Visualizations
Compare Repositories

4. System Requirements

4.1. System Use Cases

System Use Case: Creating a Metric

Use-Case Name	Creating a metric
System or Subsystem	Augur
Actors	User
Brief Description	This UC describes how the actor will create a metric
Basic Flow of Events	1. Write a complicated PostgreSQL query based on the poor documentation 2. Wrap and parameterize some function and hope it works because it probably won't (I've tried) 3. Define an endpoint
Alternative Flow of Events	1. Everything Falls apart and refuses to work. No supporting documentation, and then Augur just goes offline.
Special Requirements	Uptime: Augur should be up for at least 7% of the time each month. Load Time: Metrics should be able to be successfully created at least 1/10 attempts
Pre-Conditions	Augur must not be down
Post-Conditions	None
Extension Points	

System Use Case: Creating a Visualization

Use-Case Name	Creating a visualization
System or Subsystem	Augur
Actors	User
Brief Description	This UC describes how the actor will create a visualization
Basic Flow of Events	1. Be someone who built Augur and knows everything about it 2. Connect to an endpoint 3. Add a new page and build the visualization
Alternative Flow of Events	1. Go to the documentation and then bang your head against a wall for at least 2 hours. 2. Admire the broken links in the documentation: https://oss-augur.readthedocs.io/en/master/getting-started/create-a-metric/frontend.html (Click the link on the page, it redirects to itself) 3. Find the actual link to the right part of the documentation. 4. After multiple hours give up, and resolve to never use Augur again.
Special Requirements	Uptime: Augur should be up for at least 7% of the time each month. Load Time: Visualizations should be able to be successfully created at least 1/10 attempts
Pre-Conditions	Augur must not be down
Post-Conditions	None
Extension Points	

4.2. System Functional Specification

User Terminals:

UT1: User should be able to create a metric

UT2: User should be able to create a visualization

UT3: User should be able to update data

4.3. Non-Functional Requirements

Usability:

NFR-1: User interface lags constantly and is terribly designed.

NFR-2: All Users will be required to decode the documentation and it will feel like trying to crack the enigma machine.

NFR-3: Users should be able to use Augur with a minimum of 50 hours of training.

Reliability:

NFR-4: Augur should be down at least 93% of the time

NFR-5: The Mean Time Between Failures should be 7 minutes

Performance:

NFR-6: Augur should not allow a page to load faster than in 12 minutes.

NFR-7: The amount of overhead to run Augur should crash most standard PC's

Operating Systems:

NFR-8: Augur should only run on Linux because all other operating systems are stupid and should be erased from existence.

5. Design Constraints

1. Augur will run as a Flask application on a REST server.
2. It will only run on Linux only.
3. The system will be developed using a long list of requirements.
4. Documentation of the system is prohibited because good code documents itself, and thus only vague misleading documentation is allowed.
5. Augur will be hosted on servers that are slower than dial-up in 1998.

6. Purchased Components

1. The first purchased component is a server built by a hobbyist in their basement back in 1993.
2. The second purchased component is Tylenol for the headache you're going to have.

7. Interfaces

Software Interfaces:

The Application Server will communicate with the internal PostgreSQL database server.

8. A Note to The Grader

These assignments frustrate me. The descriptions for the assignments tend to be extremely vague, and it feels as if this entire class is focused around Augur. The problem I have with this is that Augur is at best, still very early into development. (Or based on its usability and uptime I would really hope it's in early development)

The documentation written for it is horrendous. Even the purpose of Augur is unknown, as it is not stated in the What is Augur? section of the documentation. Why should I use Augur? What benefits does it have to me? It allows me to "compare my project to others" and to "see visualizations of the comparisons" but why would I want to do that? I can make some assumptions and guesses as to why that might be useful. I would expect to be explicitly told why this is helpful to me. As a software developer, why should I dedicate my time, and effort to using Augur? This entire project seems to scream "I'm an Academic project and nothing else!" I've contributed to open-source-software projects. In my opinion, Open-Source projects die because the need isn't strong enough. Linux won't die because the need for Linux won't die. Open source projects that do die, die because while they may have been useful, they weren't useful enough to attract the attention to build a strong base of contributors.

Augur seems to be down more than it is up. The process for using Augur is complex and convoluted. Why can't I just paste in a link to a GitHub repo and Augur magically takes care of things for me? Give me a list of checkboxes for what visualizations I want. In order to successfully use Augur, I first have to go through a complex installation that takes a decent amount of trial and error as the instructions seem to only work half of the time, and the outputs are not what the instructions state. Then I must work with a series of PostgreSQL queries to work with the database, before registering endpoints, and then something involving Vue? I can't quite be sure, as none of this is clearly stated. The overhead for this project is massive. I could, in ~2 hours link up to the GitHub API and pull in data and throw together a few plots with a basic plot library like Matplotlib. Why is Augur better?

The documentation is terrible, both inside the codebase and in the readthedocs.io
I've read through some of the code on GitHub and the documentation is almost non-existent. If I'm lucky a method will have a single comment at the top-level that's still extremely vague. It's honestly a bit mind-blowing to me that a class about Software Engineering, would be so focused around such a poorly designed piece of software.

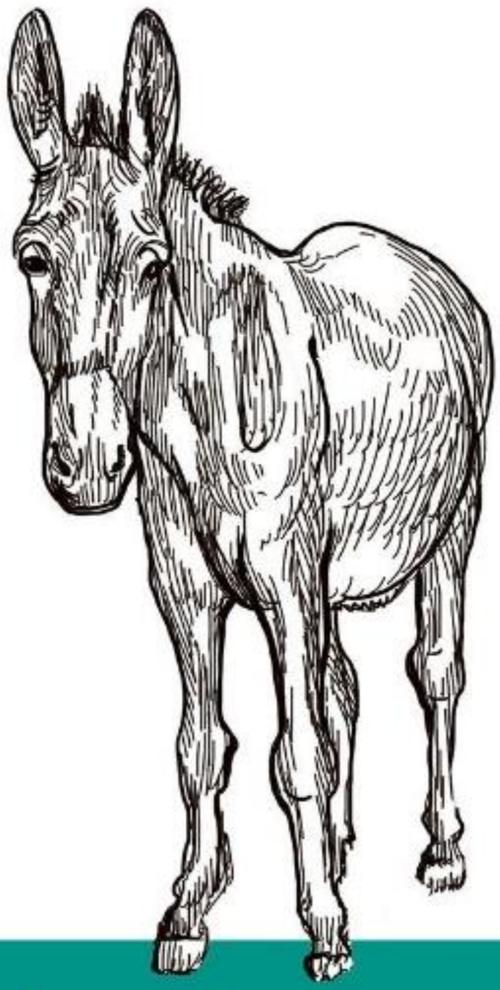
Granted, I work in the medical field writing software, and the standards I must adhere to are extremely strict. Even then, the fact is that this class is disorganized, it has no direction, the assignments are so vague it's nearly impossible to understand them, and the professor who says "I'm only going to use Slack for communication, I won't even be checking our school’s official system for communication” (Canvas) then proceeds to leave questions completely unanswered, both in the group chat, and when direct messaged on Slack.

I get it, Augur is his baby. I understand how exciting it must be to be given funding for a project like that. That doesn’t mean that the entire class should be focused around writing a bunch of papers for a half-working complex to use piece of software that many of the students have no interest in. If something is built that is fully functional, has great documentation, is easy to install, and has very low downtime then absolutely; center a class around it.

It's easy to name-drop about having friends at Google, and how you’ve gotten a whole bunch of funding for your projects. I have a friend who works at Google too, but my friendship to them says nothing about me as a programmer/software engineer. This class feels more like CS 1000 than any other class I’ve taken at Mizzou so far. Pointless, directionless, and a waste of everyone’s time.

The single **MOST IMPORTANT** thing a software engineering class should teach is writing good documentation. Never have I had a single class teach this. My boss told me that the one skill everyone who starts working in software development for him are lacking in is good documentation practices. It’s one of the biggest things that differentiate a software engineer from a programmer. It took me months of on-the-job experience before my boss didn’t have to tell me to go re-write my documentation when I pushed a commit. It’s just very difficult to have any respect for someone who teaches a Software Engineering class, yet the assignments and codebase is an example of terrible software engineering.

Where's the fun in just knowing what the code is supposed to do?



Essential

Excuses for Not Writing Documentation

O RLY?

@ThePracticalDev