Software Requirements Specifications

Home AC Monitoring & Reporting System

Software Engineering, CSCE 3444, Summer 2020

The Official Group

Eric Easley
Cole Fellers
Matthew Fredericksen
Ian Ocasio
Luis Roman

Table of Contents

1. Introduction

1.1. Purpose

The purposes of this document are:

- To confirm specifications and requirements with the developers and the consumer.
- To ensure developers are on track with creation of the product.
- Provides a baseline to perform future improvements from.
- To refer back to so that full product functionality can be ensured.
- Allows consumers to confirm the final product matches specifications.

The product described in this document is:

- Designed to inform the user of changes in air-conditioning quality.
- Using a sensor placed in the A/C system, declining operations can be detected.
- Past data is recorded to use as a baseline and to inform the user.

1.2. Scope

As air-conditioning systems are used, they become worn down and do not perform as they should. This process is a slow decline that leads to a system shutdown. It is identifiable through recording the performance of the A/C system. When this happens, the downward trend is seen. However, software and hardware must be installed to monitor these changes.

Our software will accomplish this goal. It will record the running time of the air conditioner, the change in output temperature, and the air filter quality. It will detect when the difference between the output temperature and the input temperature is too large or small. When this is detected it will alert the user that something is not running correctly or is abnormal. If this abnormality continues it will let the users know so that it can be repaired before the whole system shuts down. This has the potential to save the user money in repair and labor costs, especially if the failure is predicted to occur during the summer.

1.3. Definitions, acronyms, and abbreviations

Definitions:

- Prometheus – The Prometheus time-series database, https://prometheus.io/.
- TSDB – Time-series database, a type of relational database that stores all values with an associated timestamp.
- A/C – Air conditioning
- API – Application programming interface
- RPi – Raspberry Pi

1.4. Overview

This document is broken up into 3 main sections, which are each broken up into multiple sub-sections:

- Section 1: The Introduction to the document. Gives preliminary information.
    - 1.1: Describes the purpose of this document. Both for the user and the developer.
    - 1.2: Generally describes the scope and use of the project.
    - 1.3: Provides any definitions, abbreviations, or terms used in the document.
    - 1.4: In this section, the structure of the document is shown.
    - 1.5: Lists any references used in the creation of this document or project.
- Section 2: Gives a more detailed description of the product.
    - 2.1: States the reasons behind the need and creation for this product with a diagram showing major components.
    - 2.2: Summarizes the main uses for the product and includes a state transition diagram to help visualize the steps of the product.
    - 2.3: Discusses who the product is meant for and who will have access to each product and for what reasons.
    - 2.4: States the limitations of the developers.
    - 2.5: States dependencies that can change the requirements of the product.
- Section 3: Gives specific project requirements.
    - 3.1: Describes the different interacting interfaces which include: hardware, software, communication, and user interfaces.
    - 3.2: Describes in detail the functional requirements of the software. What the product will accomplish and how it will do this.
    - 3.3: Gives the requirements of the system as a whole.
    - 3.4: Lists any constraints the developer may run into with the hardware and/or software involved.

1.5. References

References shall be contained within their relevant sections.

2. Overall Description

2.1. Product perspective

Replacing an air conditioning unit can cost up to almost four thousand dollars when done professionally. Catching failures in these systems early can help prevent the enormous cost of replacing the entire system. These early warning signs can allow for preventative maintenance at a much cheaper total cost.

This product could potentially pair well with a product offered by AlertLabs called Sentree that has a similar function when installed in the outdoor radiator component of the air conditions unit. These two would not interfere because of their different placement, each could complement the other, rather than replacing the other.

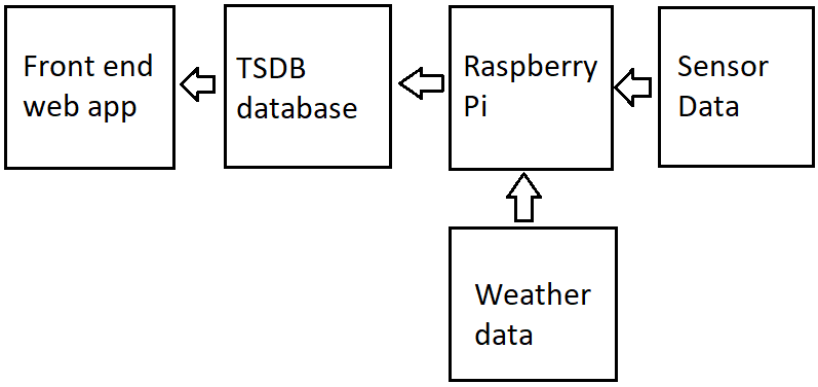

Figure 1: Component diagram

2.2. Product functions

The software will:

- Collect all relevant statistics on the AC system via sensors or third-party APIs.
- Record statistics in a database for viewing and analysis.
- Provide an interface for a user to view the recorded statistics.
- Alert the user when statistical trends are detected which indicate an approaching system event (e.g. cooling failure, filter replacement, etc).
- Provide an interface for a user to configure which alerts they receive, how they receive them, and the frequency at which they are received.

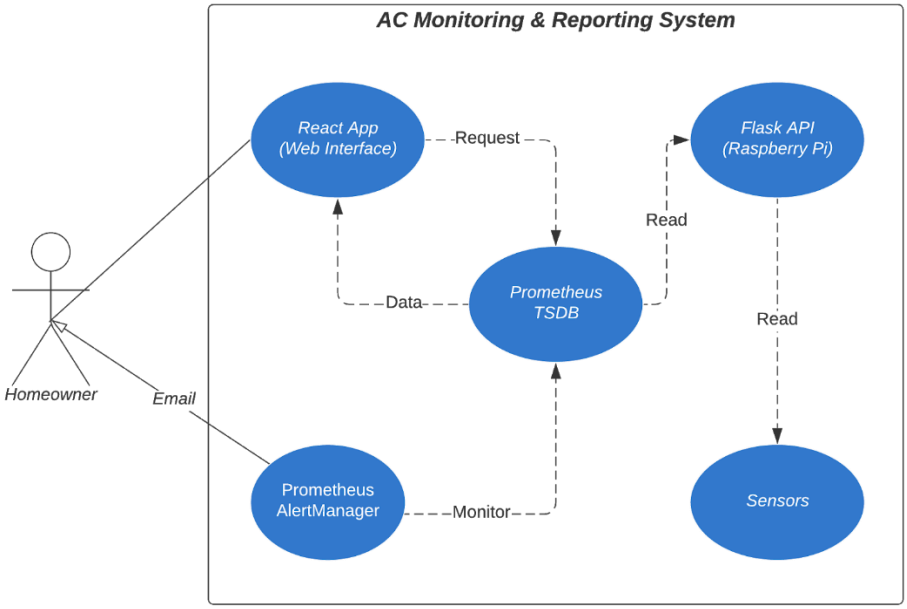

Figure 2: Use case diagram

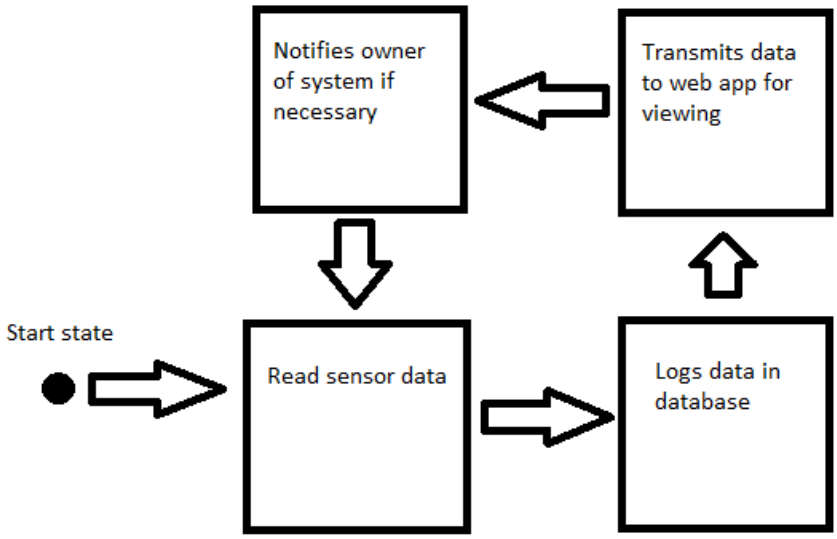

Figure 3: State transition diagram

2.3. User classes and characteristics

The only characteristics they would be required to have are their ability to log into the web app and check on the system, or log onto their email to see if the system sent any alerts.

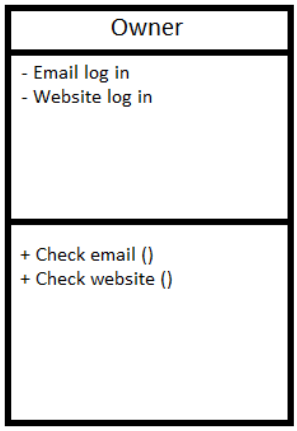

Figure 4: Class diagram

2.4. Constraints

Processing power – Raspberry Pi processors max out at the most 1.5 GHz, so any computational loads on the system would have to be light. This would prevent using any computationally expensive machine learning algorithms to detect potential error with the system.

Warranty restrictions – Depending on a potential warranty on the air conditioning unit, this might prevent us from installing the unit in certain places. We would have to identify relevant tampering clauses of any individual air conditioning unit the system is installed on.

2.5. Assumptions and dependencies

Installation fit – We are making the assumption that the system we are installing the device on has a way to install it.

Device durability – We also are assuming that the device will be able to survive the environment inside the air conditioning unit.

3. Specific Requirements

3.1. External interface

3.1.1. User interface

The user will interact with the software with a web application made using React. The user can view data from sensors live through the web application. The user can view data during a certain time frame that will be presented as aggregated values. The user will have access to a configuration menu. The User Interface has not been designed yet.

3.1.2. Hardware interface

A Raspberry Pi will be used to read pressure and temperature sensors. These items total at about $60 and have already been purchased by our project sponsor.

The Raspberry Pi will use the RPi.GPIO library to read from at most 3 GPIO pins (ground + at most two others for a one- or two-wire bus). These pins will be connected to a breadboard with analog-to-digital converters for reading voltage output from the temperature and pressure sensors.

The Raspberry Pi will have a single network port opened exposing a Flask-based REST API for retrieving live sensor readings.

The user's home server will require two network ports, one for the React app (port 80) and another for Prometheus. Prometheus will pull data at 15-second intervals from the RPi's sensor API, and the user will access the React app running on their server to interact with the system.

3.1.3. Software interface

React will be used to make a web application. Flask will read data from sensors. Prometheus will store years of data and evaluate. Prometheus alert manager system will send alerts when certain criteria are met. Open Weather is used to gather weather data. All software will be portable across all major operating systems (Windows, Linux, macOS).

3.1.4. Communication interface

Communication from the user to system (configuration, requesting data) will occur entirely through the React web interface. Communication from the system to the user (alerts) will occur via email, as configured by the user. The database will gather data from the Raspberry Pi's sensors using HTTP requests.

3.2. Functional requirements

Based on each function listed in section 2.2, state the functional requirements. The software will:

- Collect all relevant statistics on the AC system via sensors or third-party APIs.
    - Relevant statistics include incoming air temperature, outgoing air temperature, air pressure, outside air temperature, count of on/off cycles, and any others which are later determined to be reliable predictors of system events.
    - 2 temperature sensors and one air pressure sensor will be used. These sensors will be read using a Raspberry Pi.
    - The user will be responsible for installing the sensors on their AC unit. Detailed instructions will be created for the installation process.
    - Weather data will be gathered using OpenWeather or another free API.
- Record statistics in a database long-term trend analysis.
    - Data will be recorded with timestamps at 15-second intervals.
    - The database will be hosted by the user.
    - Detailed instructions on database configuration will be created for the user.
- Provide a web interface for a user to view the recorded statistics.
    - The web interface will be hosted by the user. Detailed instructions on how to set up and run the web interface will be created.
    - The web interface will provide a live view of data being gathered from sensors and other sources.
    - The web interface will provide a historical view of statistics consisting of aggregated values (e.g. min/max/average/sum) over an arbitrary range of time selected by the user.
- Alert the user when statistical trends are detected which indicate an approaching system event (e.g. cooling failure, filter replacement, etc).
    - The initial product will only detect system events based on current state. E.g. an alert would be created if air pressure rises too high (indicating a dirty filter) or if temperature differentials become too small (indicating a cooling failure).
    - Later versions will be able to detect trends and predict when a system event will occur.
- Provide an interface for a user to configure which alerts they receive, how they receive them, and the frequency at which they are received.

- Alerts will be sent via email.
- The user will be able to set a single email address at which to receive alerts. Later product versions may incorporate the ability to add several email addresses.
- The user will be able to select from a list which alerts they would like to receive and how often the alerts should be sent until their issues are resolved.

3.3. Non-Functional requirements

3.3.1. Usability

The requirements for the front end of this software is a simple, not-crowded menu that allows the user to easily switch between a historical view to a live view of their temperature allowing them to clear the history of temperatures to save space if needed as well as allowing different people to view this information via the software itself or notifications from the software.

The goal of this software is for it to be easy for the users to easily access, read, and understand what temperatures are in and outside of their house and be able to make decisions based off of that information (i.e. cooling the house down before it gets hot, changing the temperature when they get a notification saying the temperature difference is too high, etc.)

3.3.2. Reliability

There should not be too much of a range for errors for this software but the biggest failures that could happen if not programmed carefully is that the back end could misread or store false/incorrect temperatures. For the front end the biggest problem could be not showing the information correctly even with correct temperatures.

Any incorrect information shown to the user could come with mild or large drawbacks depending on how wrong the information is.
A minor flaw would be if the temperature read is a few degrees off from the actual. In this case if the user changed the temperature of the house accordingly then it won't affect the cost of using the AC or heater.
A major flaw could be if the temperature was 20 degrees off of the actual. If the user were to adjust to this, it could cost them a summable amount of money over time especially if it keeps happening.

Not only does the software need to be reliable, but the hardware as well. If the sensor cannot correctly read the information, out of place, or broken, the user will need a notification that it is not working so they can provide a quick fix or order a replacement if needed.

Without a notification the user will have misinformation for days, weeks, or even months.

Due to these reasons it is important that our code runs reliably and the hardware is treated with care so as to not waste the users time checking why the software is wrong and to save their money as much as possible.

3.3.3. Performance requirements

Our software requires a constant daily reading throughout every day while it is in use to provide accurate information for the user showing them temperatures that are not outdated. To provide this the weather reading API will take the temperature every few seconds or minutes depending on how well the software can run or according to user specifications to scale correctness, memory, and/or speed to the user's liking.

With the constant collection of data we can have an excellent database of previous and current information that the user can use to compare each day just out of curiosity or to have the AC/heater reader for a certain part of the day that they notice is the same temperature every day.

3.3.4. Safety requirements

As our software is strictly a monitoring process it is already safe for users to use as it doesn't require any personal information other than the temperature of the inside of their house. The only safety issue with this software would be the previously discussed temperature mis-readings.

3.3.5. Security requirements

While unlikely, a malicious third party may want to read the user's home temperatures to find patterns that may indicate that nobody is home (e.g. the AC is off for a period of time). This could be a risk when using an unsecure serv like an HTTP connection, and for this reason, HTTPS must be considered to protect the users information.

The user may also specify who may receive notifications from the software like broken sensors as to limit how many people can see their in-home systems.

3.3.6. Legal

As this software is a product that needs to be installed by the user themselves, if their AC or heater were to break in the process of installation the user might blame the damage on us. Due to this reason we will have a disclaimer to carefully read the installation instructions and any damage caused due to not following the instructions is on the user.

A user might also fail to check their actual AC systems relying on our software to provide a notification and blame us when they have had a dirty filter for a few weeks without getting a notification. A solution would be to specify that our software is strictly a temperature monitoring software that tracks its own errors and failures, stating to the user that they should often check their systems every once in a while to make sure it is working correctly along with our product.

3.4. Design constraints

Describe The system will operate using 2 machines:

Raspberry Pi 2 Model B:
- OS: Raspberry Pi OS (version: August 2020)
- RAM: 1GB
- Programming Language: Python 3.6+
- Software running:
  - Flask (Python)

User's home server:
- OS: Windows/Linux/OSX
- Disk space: 105MB
- Memory: 2GB
- Programming Languages:
  - React JS
  - Python 3.6+
- Software running:
  - Prometheus (version: 2.21)
  - React app for web interface

Based on the formula provided by the developers of Prometheus, the system should only require 105 MB to store 5 years of AC system statistics.

```
needed_disk_space = retention_time_seconds * ingested_samples_per_second
                                            * bytes_per_sample
                  = (5 * 365 * 24 * 60 * 60) * (5/15) * (2)
                  = 105,120,000 bytes
                  ≈ 105 MB
```

The user's home server may be any computer with at least the required resources available. The user must be willing and able to install the necessary software using provided instructions, and to leave the computer on at all times to maintain the integrity of the system's data.