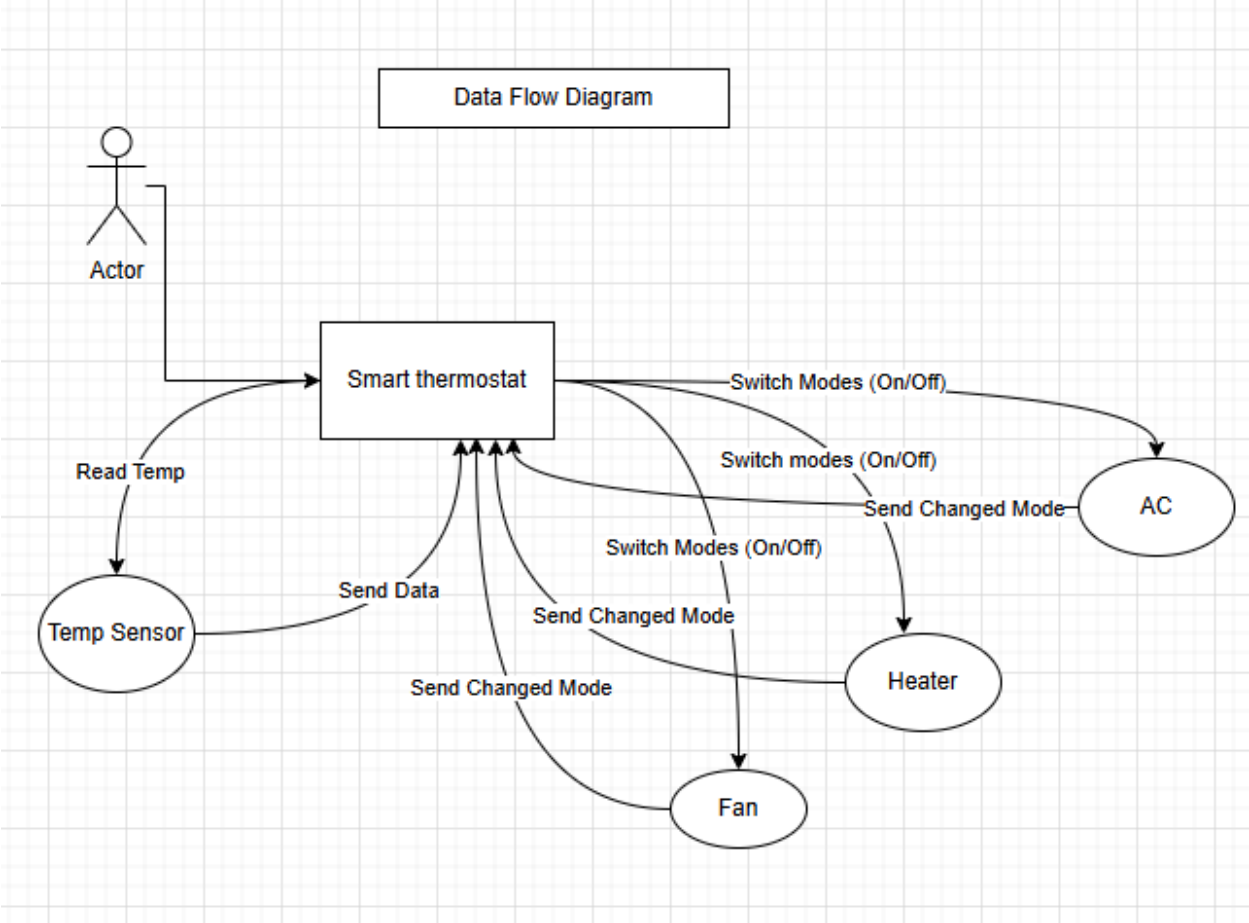
Software Engineering Project

This paper serves to show the design process and practical use of a developed Smart thermostat. In this paper, a Data Diagram, State Chart, and Activity Diagram will all be included to show the reader how the Smart thermostat was designed and what pieces go into its implementation. Furthermore, the Activity Diagram will demonstrate how the Smart Thermostat will work with being used in a functional setting. There will also be a UI showing how the app itself would work on a mobile device.

Alongside this, will be explanations on the different elements, and justification for their uses. This will include a list of the classes used for the class based design, as well as what choices were made to encourage efficiency and simplicity, while providing a quality product

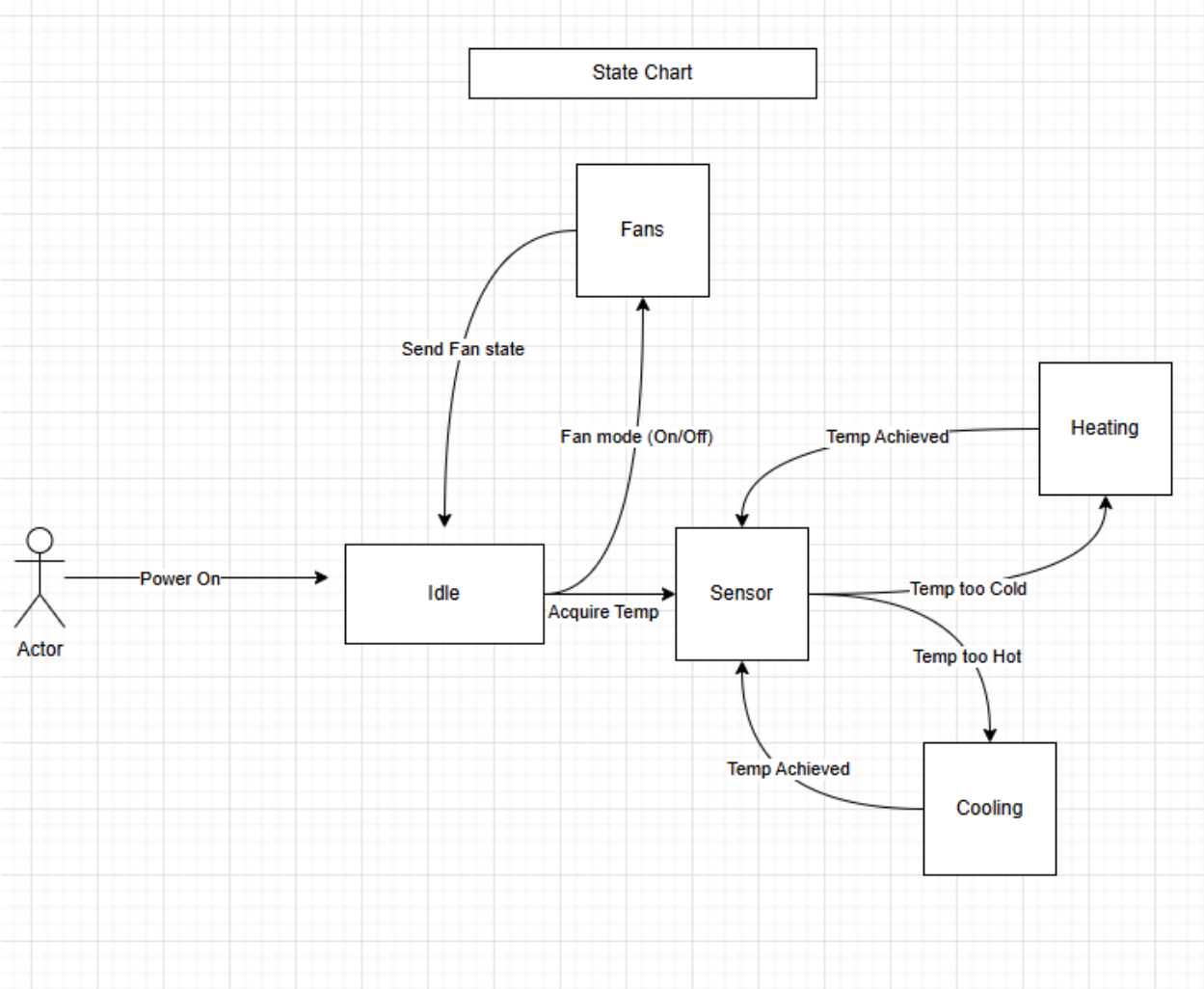
Design Flow Chart:



The Design Flow chart was created to show the user's interaction with the Smart Thermostat. In turn, the Smart Thermostat interacts with its separate components to achieve the outcome the user hopes for.

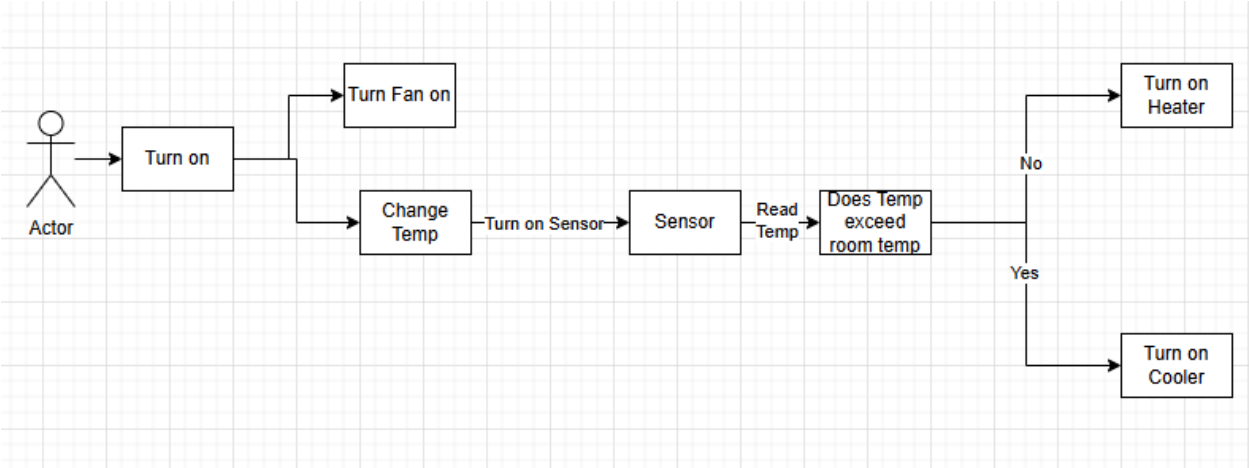
Software Engineering

State Chart:



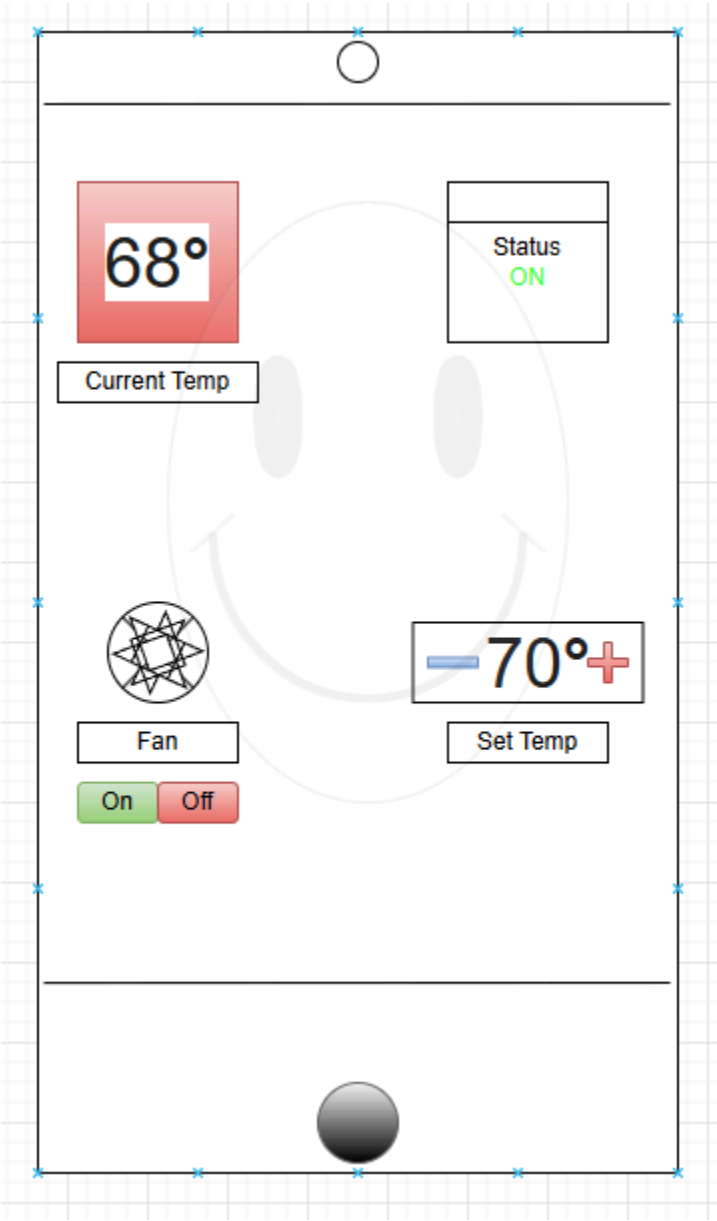
The State Chart was built to show the individual states that the Smart Thermostat could be in. The chart above shows many different states that the Smart Thermostat could be found in. This includes “Fan On/off” or “Acquire Temp” turning the Sensor on.

Activity Diagram:



The Activity Diagram shows how the user may interact with the Smart Thermostat, and the potential influence their choice would make on the Smart Thermostat. This could include the changing of “Temp” or “Turn Fan On”

UI:



The UI was designed to be simple to navigate, all while achieving the desires of the user. All functions have been included. The user can view the current Temp, while making adjustments to the Temp. They can also turn the Fan on or off, and see the status of the system.

Class Based Component Design has many uses and benefits. It allows for the creation of classes that can then be used throughout the app's development. This is beneficial because it allows for “one and done” solutions. Optimizing efficiency and allowing for children classes to be created should the need arise. For example, the class Heater can be used to encapsulate everything regarding the parent class. The children's classes could include Heaton() or Heatoff(). This makes for a seamless workflow.

Testing Plan:

Unit Tests:

- Ensure each separate element works individually
- Feed Smart Thermostat different scenarios to ensure desired outcome

Integration Tests:

- Ensure the separate classes interact well with each other
- Ensure the integration of separate classes work well in the Smart Thermostat

Functional Tests:

- Ensure that all components and functional on the GUI
- Ensure that the desired outcome is achieved with the Smart Thermostat is interacted with

Acceptance Tests:

- Ensure the client is satisfied and all requirements are met
- Ensure all components interact the way the client intended them to work

Performance Testing:

- Put the Smart Thermostat under certain loads to see how the application handles them

- Review outcomes and adjust as needed

Justification for Testing plan:

The plan above is laid out as such to cover all bases and ensure a smooth and effective launch. The separate sections cover comprehensively every area of concern, ensuring that not only the product itself works, but works in a way conducive to the requirements set by the client. The plan covers interactions within the app in a natural state, as well as when stress is applied in different and unique ways. Allowing the developer to see how the Smart Thermostat may react in different conditions.