**Completeness**

**Arming** – there is an arming controller handling the arming of the system. It contains a changeAlarmStatus method that will activate an alarm if it is listening for one. Similarly, when armed the controller will initiate an entry locking sequence.

**Locking –** The arming controller has a lockEntries method that runs through each lock in the Locks set and locks it. Locks can be locked or unlocked individually through manual turning of the lock. Otherwise, through lockEntries, all of the locks are handled together.

**Sensor readings –** every sensor has an issue it is constantly trying to detect via a looping detectIssue method. Gas leak sensors are checking that the currentLevel stays below a permitted level, motion sensors listen for motion, smoke detectors wait until smoke is detected, and cameras wait until something shows up on its feed.

**Report –** Reports written for issues (or for status updates) by sensors should be stored and readable by the home-owner. These reports should be viewable and editable. The report controller allows these functions. The reports are stored in the report list.

**Call for help –** when an issue is detected, after appropriate steps are followed, an emergency service should be called to assist the home-owner. The sensor has a detectIssue method, and the emergency controller uses this to call for help after verifying that help is needed by the home-owner. After calling for help, the emergency controller verifies that help has arrived.

**Feeds –** When the home-owner wants to see what a sensor is reading; they can choose to view a feed for the sensor. Through the interface, the home-owner requests the feed for a sensor. The feed controller creates a feed for a specific sensor. Then, the feed itself has a showFeed method that sends the created feed to the interface for the user to view.

**Consistency**

**Arming** – The system arming is consistent behavior. The alarm has highly cohesive behavior. It cannot be changed by anything besides the changeAlarmStatus method. This is directly handled by the arming controller. The changeAlarmStatus will always flip the status of the alarm. This reduces variability within the arming protocol.

**Locking –** Similarly, the locking system is consistent. The locks can only be unlocked manually. Physically, if the lock is already unlocked, then it cannot be unlocked. The lock just detects that it is being unlocked and changes the status. For the locking protocol, since it is always handled the same way regardless of the initial status of the locks. Since all the locks can behave as a single unit, variability is reduced in this case as well.

**Sensor readings –** Each sensor has its own way of ensuring that they work consistently. The smoke detector has a wait time that allows it to be sure that there is an actual issue and it is not a false reading. This prevents problems from being caused by emergency services being called too soon. The gas leak detector is constantly checking the current level with a standard permitted level. This allows there to be a constant that controls when the issues are detected and thus when statuses change. Camera and motion detectors work similarly. Whenever there is movement detected for them, they detect issues. However, since they will detect more issues than the other sensors, that does not automatically call for help. Instead, they consistently report it. Then, if the home-owner decides that help should be called, the emergency controller handles the rest.

**Report –** Report consistency is ensured through reportID’s. Since each report has a reportID, this allows a specific report to be stored uniquely. Altering a report (through the saveContents method) edits in in place. The sensors call the updateReport method which utilizes this functionality. Hence there is only one, consistent method for editing the contents of a report.

**Call for help –** Since emergency contacts are saved under account data, there is a single store for the emergency contact information. Thus, the same contacts will be called each time a specific issue comes up.

**Feeds –** Feeds are dependent on the sensor that they are made for (using a specific serialID). For a specific type of sensor, a feed will be built the same. The only thing that will change between instances of a feed for the same sensor are the data that the feed displays. This allows consistency for all feeds of the same sensor.

For all these system intentions, we ensured that the methods in the collaboration diagrams is in the design class diagram. As the model has evolved, we checked that there was strong transformational accuracy between components. These methods from the collaboration diagram come from messages that we used in our detailed system sequence diagrams for the use cases that we designed. To help guarantee this accuracy, statecharts were built to express the states and activities corresponding to the methods.

**Design Quality Criteria**

For most of our classes, we have a very low number of **methods per class (MPC)**. This reduces complexity and reduces the amount of testing required to ensure correctness. Also, the **average method complexity (AMC)** is low. Most methods are simple accessors and mutators. This keeps the system easy while allows for good object-oriented practice through encapsulation.

Since the deepest inheritance tree level is one, the class behaviors are much more predictable due to the **Depth of Inheritance Tree (DIT)** and **Inheritance Dependencies** criteria. Since the only tree is not deep, this reduces design complexity considerably.

We used many of the size metrics to allow reduced complexity in many classes. Most methods only require zero or one parameters. Also, since many of them are for accessing and mutating, the number of operators and operands used per method is low. Most classes have few member variables to simplify them and keep them cohesive.

The system as a whole is designed to reduce coupling by adding controllers and following pure fabrication guidelines. Doing this allows good design quality as it reduces complexity. These controllers help split up system responsibility to the classes that best represent them. This helps reduce the risk of failure due to coupling issues.