We have chosen to structure the software with 4 classes, each of which has one main task. The four dividing categories are: testing suite, listeners class, main logic class, and the event logger class. Each of these following has a very distinct job, where it has been observed that one communicates with one another frequently. This frequent communication is executed with the goal of having high comprehensibility in mind. In our group’s finding, we have agreed that if all the listeners and logic were in a place of its own, complications will be lessened for the user in examining certain tasks that is computed/completed. Additionally, the event log is created in a class of its own to maintain this high level of changeability. Thus, our group has created our event log by utilizing a string event system that changes at ease. Given in the case if the machine were to have a software update, then it is required to record this new event by simply writing the log. Furthermore, any changes in communication with the user, or the event log, are done in parallel for all cases. The display and event log are created in a manner that anything can be written to them, hence, allowing the software to not change formats for different cases.

In addressing changeability, the most realistic feature allowing this code to evolve would be the ways in which a user’s credit is being incremented. Currently we are keeping track of the user credit in one variable with the credit being incremented when a valid coin is inserted. With this method, which one variable is keeping track of the credit, we can add features that allow a user to partially pay coins, and pay the remainder with any other form of payment that may be implemented at a later date. The credit is incremented in the listener class. If a feature for other forms of payment were to be added, this portion of the code would not have to be changed very much, if at all, thus allowing an ease to build on top of the pre-existing software. The structure of the software is designed in respect to support alternative hardware for future uses. This is evident through the style used to code the methods such as event log, valid coin inserted, buttons pressed, and so forth. The method used to determine what buttons are pressed is simplified using an array list to benefit hardware changes. This array list comes in handy when the number of buttons has changed in the hardware, thus, allowing buttons that are added to vending machine hardware to be easily represented in the software.

The setup of our group’s array list is arranged in a manner that is highly dynamic, and does not pose a difficulty for changes. Necessary boundaries have been checked to insure reliability. Without these checks, bugs would arise, thus causing the machine to crash or cause probable extreme physical harm to users. Finally, our software is able to adapt in multiple ways, such as the ability to recognize different currencies, advancing to touch screens due to the adaptability and simplicity of our message display, and so forth. Thus, the software is designed in mind to easily integrate any future changes.