**Quality Assurance Procedure.**

For the quality control of our product —UVsim— multiple factors were considered. One of the factors initially established is the verification of correctness, since it is of crucial importance to define UVsim specifications related to the needs of the end user. With the specifications established and defined it was necessary to create a testing routine to verify the functionality according to the customer's requirements. The tests carried out were used to make sure UVsim adhered to the previously established specifications and to verify that our product was doing what it is supposed to do correctly. UVsim proved to be within the parameters and specifications initially defined and it fulfilled the user's mission objectives. The correctness test considered product characteristics such as size, complexity, design features, performance, and quality levels. An important aspect included in the correctness test was the use of unit tests for functionality verification. Also, during the development process of UVsim reliability tests were performed. This aspect within quality control was designated for performance verification, to test how reliable the system is. It was necessary to know whether UVsim performed its functions appropriately and accurately. Simultaneous and repetitive tests demonstrated that UVsim met the needed performance requirements, executing its functions correctly and consistently. Although it failed while running at first, we were able to quickly identify the problem and correct it.

Additionally, efficiency tests were included and verified. UVsim maintained the code size requirements initially established by the client. The code size remained within the required levels following strict design guidelines —high cohesion and lose coupling. This kept UVsim's design within the necessary efficiency requirement.

High cohesion and lose coupling design guideline followed:

1. No class can have more than 5 coupling with other classes.
2. No class can have more than 5 attributes.
3. No class can have more than 10 methods. Constructor and accessors are not counted.

The integrity of UVsim was built by designing a product that limits insider access only to staff responsible for making changes or internal maintenance of the program. UVsim will protect itself from unauthorized access using the available resources installed on the computer host.

UVsim was properly modified to allow users to get an easy-to-operate product. UVsim contemplates a novel and fresh design that provides the user with a comfortable environment. The new design decreases the time spent to learn how to use the program. The operation of UVsim was designed with the idea of making it user friendly and easy to use. The output that UVsim generates is deployed in the GUI (Graphical User Interface) and easily accessible for interpretation. All of this makes UVsim a final product easy to use and program, improving its quality as a final product.

Regarding maintainability our program was built with maintenance in mind. As such, all the classes we implemented were built to account for the change that might occur in the future. UVsim is not only maintainable but adaptable. As more requirements might come up, our programmers are easily able to adapt to new conditions and circumstances. UVsim's testability features demonstrated that it is a program that performs properly and within operability requirements. The integration of unit tests demonstrated the functionality of each part of the program. To achieve an appropriate level of testability several test cases needed was defined and was followed by the UVsim team for program functionality and verification. UVsim is defined as testable by supporting testing procedures. UVsim has great testability properties since this follows a design guideline with high cohesion and lose coupling. Since UVsim is testable at a high level, the testing efforts are small. Having great levels of testability capabilities makes UVsim a testable program and all parts of the program can be tested for quality control. Testability capabilities helped us find potential system failures, dramatically improving UVsim's overall performance. Flexibility. As stated earlier, UVsim was developed with adaptation in mind, and we believe that the way we implemented our classes was one of the best ways it could have been done. If the client ever specified more directions or clarified past requirements, we would be able to easily adapt our program to handle these new cases. The way we implemented our interpreter allows for easy addition of function-like classes that make adaptation a natural and pain-free process.

UVsim was developed using Visual Studio 2019, a powerful IDE capable of generating upgraded versions of UVsim. A GUI was implemented using c++ as the coding language. It was decided to add a nonfunctional requirement dedicated to UVsim portability with the vision of creating a product with even greater capabilities. It created an executable without additional dependencies which gave UVsim the ability to run on other machines with different hardware and Windows desktop operating systems. The effort required to move UVsim from one environment to another is an easy task that can be accomplished with little effort, and one important aspect to mention is that visual studio is not needed to run the executable which simplifies portability even more. The size of the executable file is considerably smaller than the entire solution block generated by Visual Studio and this is definitely a great improvement.

UVsim’s GUI and user input field could be used in other applications to take in input with only a few altercations to our original program —reusability was greatly improved. We could also reuse the structure of the program I.e., the interpreter to manage a lot of classes seamlessly. We can also use the program to manage 100 items, that do not necessarily have to be the memory, you could make it into a shopping list that keeps track of the items on your list in the memory section with just a few alterations. If you start thinking creatively like that, there is a ton of stuff that can be done with this sort of program that has nothing to do with the main memory or anything of the sort. We could couple UVsim with other systems making it interoperable. Again, we would have to make some changes to the original code. But, with just a few changes we could implement something similar in a mobile application, which could be really cool because then the students would be able to work on their programs wherever they find themselves. If we combine reusability and interoperability the possibilities for adaptation become almost limitless. If we take my shopping list idea for example and combine it with the mobile app idea, you see that we can turn this UVsim into a shopping list app without much change. And that is something that is really awesome.