CSC 490 – Spring 2022

Software Implementation and Testing (Delivery No. 3)

Due Date: 8 / 5 / 2022 (soft copy uploaded to Blackboard)

In the third and last phase of the "CSC490 Software Engineering" project, you are required to implement the "**Virtual Learning Assistant**" software that you designed in Phase # 2 by writing the code using a programming platform and language of your choice. In addition, you are required to **Validate** and **Verify** (V&V) your software. The main operations that are required from you in this phase are described below:

1. For the implementation, you are free to choose the application type of your VLA, i.e., you can implement the VLA as a desktop, web, or mobile application. In addition, you are free to choose the programming language that you are familiar with, under the condition that it is a well-known and widely used language. Do not implement the VLA in a non-familiar language. For example, if you decide to implement the VLA as a desktop application, you can use any famous language such as Java, Python, or one of the C family. If you want to implement the VLA as a web app, you can do that using a mixture of JavaScript, PHP, React, Angular, Ajax, Node.js … For mobile app, you can choose between Java, Kotlin, C#, or Swift. If you want to choose a language that is not mentioned here, check with the instructor before you start the implementation.
2. Based on the Context Diagram that you created in Phase #2, your VLA may need to interact with other systems, such as the LMS. If you create and implement the interface and connections between the VLA and the other system(s), you will receive bonus points. If you are not able to create such connections, you are required to simulate the connection by creating and using hypothetical data that resembles the data that the VLA would obtain from the other system. For example, suppose that one of the features in your VLA requires that the VLA obtains the class roster from the LMS. In such case you can create the rosters manually using virtual data and input them to the VLA as if they are coming from the LMS. Similarly for other connections between the VLA and any system in its environment: you can simulate them instead of implementing them with real systems (such as Blackboard, Moodle, etc.)
3. You are required to write the code for the Use Cases of the VLA that you selected in Phase #2 (As usual, one Use Case per student). You base your code on the VLA Class Diagram from Phase #2, and you deduce the various variables and methods that are needed in each class from the Phase #2 diagrams (Activity Diagrams, Sequence Diagrams, and State Diagram). While you are writing the code, you might discover that one or more of the requirements and/or design diagrams require updating (based on the details of the implementation code), hence you go back to the Requirements Document and the software design and update it/them. This is a good practice that equips you with the experience of the software engineering process for a real-life system, in which change could occur at any stage in the process, and the various phases of the process should be updated accordingly. If you

encounter such a case, you need to document the changes that you made by creating a new version of the Requirements Document and/or the Software Design Document.

4. You are required to test your code as you progress in the implementation. For each class that you create, you design unit test cases that test the methods (functions) of the class. Note that this should be a UNIT testing process, meaning that selected functions of the class should have separate test methods in the unit test cases. If you are implementing the VLA in Java, you can use JUnit as your testing framework. If you wrote your code in Python, you can use PyTest (https://pytest.org/) as a testing framework. If you implement the VLA as a web application, you can use Jest (https://jestjs.io/) which is an excellent JavaScript unit testing framework that supports Node.js, React, and Angular. Finally, if you implemented the VLA as a mobile app, you can use Junit or Appium (https://appium.io/) as a unit testing framework. For each major method in your class, you identify the tests that will ensure that the method is working properly and with no errors (based on Section 8.1 in the textbook). You identify for each test the inputs to the test and the expected outputs. After you finish the code you execute the test in which you compare the actual outputs and the expected ones and produce the test report. If the test report reveals errors, you should debug the corresponding function, discover, and correct the errors. Don't forget to document each test that you make since you are required to add the details of the tests in the delivery submission of this phase. Note that you are required to test major functions only in each class. Simple functions that are called from within major functions are not required to be tested separately. Normally, you could have between 2 to 5 major functions in each class. Also, note that the tests that you execute should include both validation testing (i.e., testing normal inputs) and defect-testing (i.e., testing abnormal/unexpected inputs). After you test each method separately, you are required to test all the methods in the class together using the concept of regression testing. You can design as many class-level tests as needed to ensure that all the possible operations of the class have been tested. There will be grades on both the function-level tests and the class-level tests.
5. After you finish the whole code of the system, you are required to generate (design) and execute four system test cases that represent the utilization of the system as a whole (**for groups of four students, you design and implement five system test cases**). A system test case should be designed and implemented to test each Use Case that you implemented. In addition, a system test case should be conducted to test the whole system (the three/four Use Cases together). The combination of all the system test cases should ensure that the whole system (i.e., three/four use cases) has been tested. In each system test case, you write down the feature(s) of the VLA that you are testing, the steps of the test, the inputs to the system in each step, and the expected outputs. Each system test case can be conducted by combining the corresponding unit test cases and executing them together. You should try to download test data from online datasets that can be used as your input data if possible. If you are not able to find an online dataset that contains suitable data for your inputs, you need to simulate the input data by creating it. If you simulate your input data, you should make it as realistic as possible. After you define the above details, you execute your system test case and record the actual outputs that you obtain. After that, you compare the obtained outputs with the expected ones. If there is a considerable deviation between the actual and expected outputs, you need to search for the reason of the deviation and correct the system accordingly. Don't forget to document each system test case separately in order to add it to the submission report.

6. In addition to the system test cases in the previous point, you need to execute at least one system performance test case. This is a test that focuses on testing the performance of the VLA as follows: you select one of the non-functional requirements that you wrote down in the Requirements Document (Phase #1) and that is related to the three use cases that you implemented (can be a user or a system requirement). Next, you design a system test case, similar to the four test cases in the previous point, that aims at testing the accomplishment of the selected non-functional requirement by the system. For example, suppose that you selected to test the response-time requirement that states "The response time of the system shall not exceed 2 seconds for any operation". In order to test such requirement, you identify the most complex operation that the three use cases contain, and you test it using simulated data, while recording the start and end times of the operation. You compare the total time (end time – start time) and see if it is less than 2 seconds to check whether this requirement is fulfilled by the VLA or not (the calculation and comparison should be done automatically). If you find that the requirement is not fulfilled, you should update the system (add code, rearrange existing code, split classes, combine classes, etc.) in such a way to make the requirement fulfilled. You should document each step that you make while executing this test.

After you finish the implementation and testing, you need to write the Delivery document for this phase. The Delivery document should contain the following **main sections**:

1. **Introduction**: A summary of what you did in this phase: You describe the platform and tools that you used to implement and test your system, the features that were included in the system implementation, the importance of the tests that you conducted, and the assurance (expectation) that the client can be given based on the results of the tests that you made.

2. **System Implementation**: In this section you present the details of the implemented system. **DO NOT** copy and paste the code of the system here. Rather, you describe each part of the code and its role in the system. For example, suppose that one of the packages in your code is "searching" and it contains five classes. Then you describe the functionalities and services that are offered in each class. In addition, you describe the functionality and purpose of each major method in each class. Moreover, you are required to describe how your code implements the various objectives (features) of the VLA when it is executed. The best way to describe this is by using use case stories/scenarios, and describing the parts of the system that are being executed when each story/scenario is run by the system. For example, you write down a story for "Track Study Hours" that includes the parts of the system (classes/methods) that will be executed when this objective is being used by the student and instructor. Your description should depict what is happening in the system during the implementation (sequential method calls within the objects and between different objects/interfaces, parallel method calls, time-based method execution, etc.).

3. **Unit Testing**: In this section you present and discuss the Unit Test Cases that you conducted in Step 4 above. For each unit test case, you describe the Unit(s) that you are testing, the

input data that you use, the expected outputs that you defined, and the test report. If you made modifications to the system after you conducted the test case, you describe them too. If you re-executed the unit test case due to errors or bugs, you describe the process. You are required to provide a screenshot that shows the test case execution and output report (for both function-level tests and class-level tests).

4. **System Testing**: In this section you present and describe the System Test Cases that you conducted based on Steps 5 and 6 in the previous section. Similar to the Unit Tests, you describe for each system test case the features (and/or constraints) of the system that you tested, the overall inputs for the system test case (data that you used), the expected outputs, the code of the test case (for example, the combination of the unit test cases that were run to execute the system test case), and the obtained test report(s). Any changes to the system that you made due to the execution of the test case should be described. For each system test case, you should provide screenshot(s) that show the test case execution and output report (for the test cases in both steps 5 and 6 above).

5. **Conclusion**: In this section you summarize the implementation of the system that you detailed in Step 2, and highlight the importance of the various tests that you made. You focus on how the conducted tests and the corresponding changes enhanced the software and what the client can expect when executing and using the various features in your VLA.

The deliverables of this Phase are:

1. Delivery Document that includes the sections described above.
2. Complete code of the system implementation and testing (i.e., implementation code and testing code).