System Verification and Validation Plan for Software Engineering

Team #22, TeleHealth Insights
Mitchell Weingust
Parisha Nizam
Promish Kandel
Jasmine Sun-Hu

November 1, 2024

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

1 Symbols, Abbreviations, and Acronyms **iv**

2 General Information **1**
- 2.1 Summary 1
- 2.2 Objectives 1
- 2.3 Challenge Level and Extras 1
- 2.4 Relevant Documentation 2

3 Plan **2**
- 3.1 Verification and Validation Team 2
- 3.2 SRS Verification Plan 2
- 3.3 Design Verification Plan 3
- 3.4 Verification and Validation Plan Verification Plan 3
- 3.5 Implementation Verification Plan 3
- 3.6 Automated Testing and Verification Tools 3
- 3.7 Software Validation Plan 4

4 System Tests **5**
- 4.1 Tests for Functional Requirements 5
  - 4.1.1 Data Processing and Display 6
  - 4.1.2 Video and Audio Data Analysis 9
- 4.2 Tests for Nonfunctional Requirements 11
  - 4.2.1 Performance 11
  - 4.2.2 Area of Testing2 18
- 4.3 Traceability Between Test Cases and Requirements 18

5 Unit Test Description **18**
- 5.1 Unit Testing Scope 18
- 5.2 Tests for Functional Requirements 19
  - 5.2.1 Module 1 19
  - 5.2.2 Module 2 20
- 5.3 Tests for Nonfunctional Requirements 20
  - 5.3.1 Module ? 20
  - 5.3.2 Module ? 21
- 5.4 Traceability Between Test Cases and Modules 21

6 Appendix **22**

6.1 Symbolic Parameters 22
6.2 Usability Survey Questions? 22

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	**description**
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS]

2.2 Objectives

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

2.3 Challenge Level and Extras

[State the challenge level (advanced, general, basic) for your project. Your challenge level should exactly match what is included in your problem statement. This should be the challenge level agreed on between you and the course instructor. You can use a pull request to update your challenge level (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

[Summarize the extras (if any) that were tackled by this project. Extras can include usability testing, code walkthroughs, user documentation, formal proof, GenderMag personas, Design Thinking, etc. Extras should have already been approved by the course instructor as included in your problem statement. You can use a pull request to update your extras (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

2.4 Relevant Documentation

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. You can create BibTeX entries for your documents and within those entries include a hyperlink to the documents. —SS]

Author (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

3.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates (like your primary reviewer), or you may plan for something more rigorous/systematic. —SS]

[If you have a supervisor for the project, you shouldn't just say they will read over the SRS. You should explain your structured approach to the review. Will you have a meeting? What will you present? What questions will you ask? Will you give them instructions for a task-based inspection? Will you use your issue tracker? —SS]

[Maybe create an SRS checklist? —SS]

3.3 Design Verification Plan

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.5 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walkthrough. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

3.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

The following are the automated testing and verification tools to be used during the validation and verification process for the software being tested in this VnV plan:

- Unit Tests: Jest, Pytest
- Linters: Flake9, Prettier, ESLint
- Continuous Intergration: GitHub Actions

For our code coverage, we will use Istanbul and Coverage.py. Istanbul is a code coverage tool that works with JavaScript testing frameworks like Jest. It helps developers see how much of their code is tested by creating reports that show untested lines and functions. Coverage.py is a code coverage tool for Python, which we use for our machine learning model. It measures how much of the code runs during tests and generates reports in different formats, helping developers find untested parts of their Python applications and improve test coverage.

3.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

Our plan for validating the software includes review sessions with stakeholders and extensive user testing. Dr.Yao Du, one of our key stakeholders, will provide the software to clinicians and patients for real-world testing in clinical settings. This will allow us to gather valuable feedback on the software's functionality and usability in actual healthcare environments. In addition to this field testing, we will conduct structured user testing sessions where participants will simulate the experience of being a patient. During these sessions, users will navigate through the software, interacting with its features, and afterward, they will share their insights on what worked well and what didn't. Overall, using both field testing and targeted user testing, the information gathered will help us refine the software and ensure it meets the needs of its intended users.

4 System Tests

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Data Processing and Display

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

The test cases below focus on ensuring reliable storage and retrieval of multimedia data, privacy compliance by excluding PII, accurate grouping under unique identifiers, and long-term report accessibility, meeting all data storage and organization requirements.

- FR-ST-DSC1

  Control: Automatic

  Initial State: Database is empty or initialized with test data.

  Input: Multimedia files (video, audio, and structured data files).

  Output: The expected result is a success message in console for both storing data and retriving data

  Test Case Derivation: Ensures the database meets storage capacity and integrity requirements as per FR-DSC1 in SRS document.

  How the test will be performed:

  1. Insert a session containing multimedia files (video, audio, JSON) into the database.
  2. Retrieve the session files and check for data integrity by comparing size of stored data with retrieved data.
  3. Verify that the retrieved files are uncorrupted and correctly match the original files.

- FR-ST-DSC2

  Control: Manual

  Initial State: Database is set up to store assessment session data.

  Input: Video, audio files, flagged occurrences, and timestamps for each assessment question.

Output: The expected result is the creation of a JSON file that contains flagged occurrences and timestamps which is stored alongside the data.

Test Case Derivation: Allows the system to store all important information for clinicians to use.

How the test will be performed:

1. Insert a test assessment session with video and audio files, flagged occurrences, and timestamps.
2. Query the database to retrieve the session's data and verify the presence of a JSON file with accuracy of flagged occurrences and timestamps.

- FR-ST-DSC3

Control: Automatic

Initial State: Database configured to prevent storage of personally identifiable information (PII).

Input: Attempted insertion of a record containing personally identifiable information.

Output: The expected result is that the database rejects any PII-containing records and stores only anonymized data.

Test Case Derivation: Ensures compliance with privacy standards, verifying that no PII is stored in the database.

How the test will be performed:

1. Attempt to insert a record with PII (e.g., name, address).
2. Verify that the system blocks or anonymizes PII, preventing its storage.
3. Retrieve all clinician-accessible data and confirm the absence of PII.

- FR-ST-DSC4

Control: Manual

Initial State: Database initialized and ready for storing user session data.

Input: Multiple sessions, each with unique user identifiers.

Output: The expected result is that all session data is stored and grouped correctly according to the unique user identifiers.

Test Case Derivation: Confirms database's grouping and retrievable capabilites, ensuring accurate data organization.

How the test will be performed:

1. Insert multiple sessions into the database, each tagged with a unique user identifier.
2. Query the database for each user identifier and verify that all associated session data is correctly grouped.
3. Confirm no data is incorrectly associated or left unassociated.

- FR-ST-DSC5

Control: Manual

Initial State: Database initialized and ready to store reports with unique identifiers.

Input: Assessment report linked to a patient's unique identifier.

Output: The expected result is that the report is successfully stored, linked to the corresponding patient identifier, and retrievable for at least 7 years.

Test Case Derivation: Verifies long-term storage and retrieval of assessment reports, supporting patient progress tracking.

How the test will be performed:

1. Insert a test report with a unique patient identifier into the database.
2. Retrieve the report using the identifier and confirm the report's accuracy and timestamp.

3. Check when database will be wiped out and when backups occure

4.1.2 Video and Audio Data Analysis

The test cases below focus on ensuring the video analysis model can reliably access session recordings, accurately detect and log speech disturbances, and correctly flag disturbances with timestamps, questions, and user responses to support efficient clinical review.

- FR-ST-VDA1

Control: Automatic

Initial State: Completed assessment sessions are available in the database, with video and audio recordings accessible for processing.

Input: Request by the analysis model to access video and audio data from a completed session.

Output: The expected result is that all videos requested are processed with a success message in the logs

Test Case Derivation: Verifies that the model has reliable access to stored multimedia data, which is critical for processing and analysis.

How the test will be performed:

1. Retrivee the video and audio recordings from several completed sessions.
2. Check that the model successfully accesses the multimedia files for each session without data access errors.
3. Verify that the files are correctly loaded for analysis with no corruption or access issues.

- FR-ST-VDA2

Control: Automatic

Initial State: The analysis model is initialized and ready to process test video and audio data.

Input: Video and audio data containing speech disturbances, interruptions, and other irregularities for analysis.

Output: The expected results is an accuracy of 95% in a JSON file for number of disturbances found by the model

Test Case Derivation: Confirms that the model's disturbance detection meets the accuracy requirement, reducing bias in the analysis process.

How the test will be performed:

1. Run the model on a test dataset containing known speech disturbances.
2. Compare the disturbances identified by the model with human observations for accuracy validation.
3. Verify that the model achieves at least 95% accuracy in identifying and logging disturbances.

- FR-ST-VDA3

Control: Automatic

Initial State: Video and audio data with disturbances has been processed by the analysis model.

Input: Disturbances identified by the model, requiring flags with associated timestamps, assessment questions, and user answers.

Output: The expected result is an accuracy of 95% in a JSON file for timestamp accuracy

Test Case Derivation: Ensures clinicians can quickly access relevant parts of the assessment with accuracy, aiding in efficient diagnosis.

How the test will be performed:

1. Process a test assessment session with the model, identifying and flagging disturbances.
2. Retrieve flagged disturbances and confirm each has an accurate timestamp, associated question, and user response.

3. Compare the flagged data with human observations and verify at least 95% accuracy in the model's associations.

...

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy. —SS]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

4.2.1 Performance

The test cases below ensures that the system meets essential performance metrics, including quick page load times, low latency in video and audio recording, high video resolution, and efficient report generation.

- PR-ST-SL1

  Type: Dynamic

  Initial State: System initialized, strong internet connection

  Input/Condition: User navigates to different web pages within the system.

Output/Results: The expect output is that each web page loads fully with all functionalities within 3 seconds.

How the test will be performed:

1. Navigate to various web pages within the system.
2. Measure the time taken for each page to load completely.
3. Confirm that all pages load within 3 seconds.

- PR-ST-SL2

Type: Static

Initial State: Video and audio recording session initialized.

Input/Condition: Audio and video session of user performing gestures while talking

Output/Results: The expected output is the latency between actions and recorded playback remains under 1 second, ensuring synchronization.

How the test will be performed:

1. Begin recording a session.
2. Have the user perform timed actions while recording.
3. Play back the recording and measure the latency between actions and their corresponding timestamps.
4. Confirm latency does not exceed 1 second.

- PR-ST-SL3

Type: Dynamic

Initial State: System configured to store video data.

Input/Condition: Video recorded and stored from assessment an session.

Output/Results: The exepect output is the video quality is at least 720p resolution upon when retriving or storing.

How the test will be performed:

1. Record a session and store the video.
2. Retrieve the stored video and verify its resolution.
3. Confirm the resolution is 720p or higher.

The test cases below focus on verifying the system's precision in detecting and analyzing speech disturbances, ensuring timestamp alignment, and maintaining 100% accuracy in assessment data.

- PR-ST-PA1

Type: Manual, Dynamic

Initial State: Audio analysis model is loaded with sample audio data.

Input/Condition: Audio data containing speech patterns with known disturbances.

Output/Results: The expected output is that the analysis achieves 95% accuracy in detecting speech disturbances.

How the test will be performed:

1. Load test audio files with known disturbances into the analysis model.
2. Compare detected disturbances with human-reviewed observations.
3. Verify that the model correctly identifies at least 95% of disturbances.

- PR-ST-PA3

Type: Static

Initial State: Timestamp function is synchronized with real-time actions.

Input/Condition: User performs actions in the recorded session.

Output/Results: The expected output is that the timestamps delay within 1 second of the real-time action.

How the test will be performed:

1. Record a session with specific user actions.
2. Analyze timestamps and compare them to the actual timing of the actions.
3. Confirm each timestamp falls within a 1-second margin of the real action.

- PR-ST-PA4

Type: Manual, Static

Initial State: Assessment answer key is loaded.

Input/Condition: Manual verification of the answer key's accuracy

Output/Results: The expected output is that the answer key is 100% accurate.

How the test will be performed:

1. Manually review each entry in the assessment answer key.
2. Check for errors or inconsistencies in each answer.
3. Confirm all answers are correct, ensuring 100% accuracy.

The test cases below validate the system's ability to handle errors, back up data reliably, and enforce strict input validation.

- PR-ST-RFT1

Type: Dynamic

Initial State: System is operational with error handling in place.

Input/Condition: User initiates actions known to cause common errors.

Output/Results: The expected output is the system displays clear error messages for at least 95% of the common errors encountered.

How the test will be performed:

1. Simulate common user errors, such as invalid inputs or incorrect file uploads.

2. Observe system response and displayed error messages.
3. Verify clarity and accuracy of error messages in at least 95% of cases.

- PR-ST-RFT2

Type: Dynamic

Initial State: Database backup processes configured and operational in the system.

Input/Condition: Monthly data backup event.

Output/Results: The expected output is that the system performs a data backup within a 4-hour timeframe on the first of each month.

How the test will be performed:

1. A data backup is triggered.
2. Record the duration of the backup process.
3. Confirm that backup completes within 4 hours.

The test cases below ensure that the system can handle the expected user load, data storage needs, and simultaneous uploads without performance degradation.

- PR-ST-CR1

Type: Dynamic

Initial State: System initialized with maximum user capacity parameters.

Input/Condition: System loaded with 2000 user accounts.

Output/Results: The expected result is that the system operates stably and manages all accounts without issues.

How the test will be performed:

1. Create and load 2000 user accounts into the system.
2. Monitor system performance metrics, including stability and response time.

3. Confirm system maintains stable performance.

- PR-ST-CR2

Type: Static

Initial State: System is initialized with required storage capacity.

Input/Condition: Data stored in the database approaches 10TB annually.

Output/Results: The expected result is that the system accommodates 10TB of data without loss of performance.

How the test will be performed:

1. Load 10TB data into database.
2. Monitor database performance metrics, such as access time and error rate.
3. Confirm system's ability to manage 10TB without performance impact.

The test cases below confirm that the system can scale effectively to accommodate an increasing user base, data volume, and computational needs over time.

- PR-ST-SE1

Type: Static

Initial State: System operational with current allocated user amount.

Input/Condition: Increase user base by 10% annually.

Output/Results: The expected result is that the system maintains performance while handling user growth.

How the test will be performed:

1. Simulate a 10% increase in the user base by increase user base parameters by 10%.
2. Monitor system metrics such as response time and error rate.

3. Confirm that the system operates within acceptable performance metrics post-increase.

The test cases below verify the system's reliability across updates and compatibility with major operating systems.

- PR-ST-LR1

Type: Static

Initial State: System release build in use, with ongoing development updates.

Input/Condition: System stability monitored over successive updates.

Output/Results: The expected result is that the system maintains a failure rate below 1% in release builds.

How the test will be performed:

1. Conduct routine tests on the release build during ongoing development updates.
2. Monitor system logs for errors or malfunctions.
3. Verify that the failure rate remains below 1%.

- PR-ST-LR2

Type: Static

Initial State: System configured for compatibility testing across platforms.

Input/Condition: System loaded on Windows, macOS, Linux, Android, and iOS.

Output/Results: The expected result is that the system functions correctly across all platforms.

How the test will be performed:

1. Run the system on each specified operating system.
2. Perform standard operations and monitor user experience on

each platform.
3. Confirm that the system operates without issues on all platforms.

4.2.2 Area of Testing2

...

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Author Author. System requirements specification. https://github.com/..., 2019.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

[This section is not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?
2. What pain points did you experience during this deliverable, and how did you resolve them?
3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.
4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?