System Verification and Validation Plan for Software Engineering

Team #13, ARC
Avanish Ahluwalia
Russell Davidson
Rafey Malik
Abdul Zulfiqar

November 5, 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

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 (?) tables, if appropriate —SS]

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

The purpose of this Verification and Validation (VnV) document is to outline the testing process we'll use to make sure our project meets all its requirements and functions correctly. Verification checks that we're building the product according to our design, while validation ensures the product meets the users' needs and performs as expected. This document will describe both functional and non-functional tests, which help confirm that the project is reliable, safe, and easy to use. By following this VnV plan, we can identify and fix any issues early, ensuring the final product is high-quality and ready for users.

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

The challenge level for the Realm project is **general**. The extras that are going to be tackled in this project are **user documentation** and **usability testing**.

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]

?

[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

This section outlines the structured plan for verifying and validating our project at each stage of development. It first lists the roles of the team members involved in verification, the strategies for ensuring the accuracy of our requirements, design, and implementation, as well as the tools we'll use for automated testing. Each part is designed to ensure our project meets quality standards and fulfills its intended purpose.

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]

- Internal Team Review
- Peer Review
- TA Review
- Requirements Coverage Check

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 walk-throughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walk-through. 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

Refer to section 10 of the development plan for the list of tools. We will use the NUnit-based Unity Testing framework for unit testing and code coverage metrics. Continuous testing will be done by GitHub actions using Unity Builder and Unity Test Runner from the GameCI open source project.

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]

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 Area of Testing1

[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]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs. Output is not how you are going to return the results of the test. The output is the expected result. —SS]

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

   How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   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:

4.1.2 Area of Testing2

...

4.1.3 Profile Testing

The Profile Screen Testing focuses on verifying user interactions related to profile management, password changes, and viewing of profile data. These tests ensure that users can efficiently manage their profile settings and view relevant information, which is critical for maintaining user engagement and security.

1. **Name:** Validate User Authentication
   Test ID: Test-PS1
   Control: Automated
   Initial State: App launched, login screen displayed.
   Input: User enters valid credentials.
   Output: The expected result is that the user successfully logs in and is redirected to their profile page.
   Test Case Derivation: This test is to ensure the system authenticates users with valid credentials.
   How test will be performed: Through an automated script that inputs valid user credentials and checks if redirection to the profile page is successful.

2. **Name:** Password Change Functionality
   Test ID: Test-PS2
   Control: Manual
   Initial State: User logged in, on profile settings page.
   Input: User inputs new password and confirms.
   Output: The expected result is that the system updates the password and provides a confirmation message.
   Test Case Derivation: To verify that the system allows users to change their password securely.
   How test will be performed: Tester manually changes the password and checks for confirmation of the change.

3. **Name:** View Profile Information
   Test ID: Test-PS3
   Control: Automated
   Initial State: User logged in, on profile page.
   Input: None.
   Output: The expected result is that profile information (username,

password, profile picture, status) is displayed correctly.
Test Case Derivation: Confirm that all user profile information is retrievable and displayed correctly.
How test will be performed: Automated test that logs in as a user and verifies that all profile information is displayed as expected.

4. **Name:** Access Help Page
Test ID: Test-PS4
Control: Manual
Initial State: User logged in, on profile page.
Input: User navigates to help page.
Output: The expected result is that a help page with FAQs and additional help information is displayed.
Test Case Derivation: Ensure the help page is accessible and provides useful information.
How test will be performed: Tester navigates to the help page and verifies the presence and accuracy of the information.

4.1.4 Sub-Realm Testing

The Sub-Realms Testing focuses on verifying user interactions related to creating, managing, and interacting within sub-realms. These tests ensure that users can efficiently create sub-realms, manage sub-realm members, and interact with sub-realm-specific content, which is essential for fostering collaboration and community engagement within the app.

1. **Name:** Create New Sub-Realm
Test ID: Test-G1
Control: Manual
Initial State: User logged in, on sub-realms page.
Input: User provides a sub-realm name, description, and invites members.
Output: The expected result is that a new sub-realm is created with the specified details, and invited members are notified.
Test Case Derivation: To verify that users can create new sub-realms with relevant details and invite members.
How test will be performed: Tester manually creates a sub-realm and verifies that the sub-realm is created with the correct details and members are notified.

2. **Name:** Add or Remove Sub-Realm Members
   Test ID: Test-G2
   Control: Manual
   Initial State: User logged in, viewing a sub-realm they manage.
   Input: User adds or removes specific members from the sub-realm.
   Output: The expected result is that sub-realm membership updates accordingly, reflecting added or removed members.
   Test Case Derivation: Ensure the sub-realm manager can manage sub-realm membership.
   How test will be performed: Tester adds and removes members from the sub-realm and verifies that the changes are reflected.

3. **Name:** Edit Sub-Realm Settings
   Test ID: Test-G3
   Control: Manual
   Initial State: User logged in, viewing a sub-realm they manage.
   Input: User changes sub-realm name or description.
   Output: The expected result is that sub-realm settings update with the new name or description.
   Test Case Derivation: Verify that sub-realm settings can be modified by the sub-realm manager.
   How test will be performed: Tester edits the sub-realm settings and checks that the changes are applied.

4. **Name:** Delete Sub-Realm
   Test ID: Test-G4
   Control: Manual
   Initial State: User logged in, viewing a sub-realm they manage.
   Input: User selects the option to delete the sub-realm.
   Output: The expected result is that the sub-realm and all associated data are removed from the system.
   Test Case Derivation: Ensure that sub-realm managers can delete sub-realms and remove associated data.
   How test will be performed: Tester deletes a sub-realm and verifies that it no longer exists in the system.

5. **Name:** Interact with Sub-Realm AR Content
   Test ID: Test-G5
   Control: Manual

Initial State: User logged in, viewing a sub-realm.
Input: User interacts with sub-realm-specific AR content.
Output: The expected result is that sub-realm-specific AR content responds to user interactions.
Test Case Derivation: Verify that users can interact with shared AR content within the sub-realm.
How test will be performed: Tester interacts with sub-realm-specific AR content and observes responses.

4.1.5 Friends Testing

The Friends Screen Testing focuses on verifying user interactions related to managing friends, including sending requests, accepting or rejecting requests, and viewing or removing friends. These tests ensure that users can effectively manage their friend connections within the app, which is important for social interaction and network building.

1. **Name:** Send Friend Request
   Test ID: Test-FS1
   Control: Manual
   Initial State: User logged in, viewing friends screen.
   Input: User sends a friend request to another user.
   Output: The expected result is that the friend request is sent, and the recipient receives a notification.
   Test Case Derivation: To ensure users can initiate friend requests.
   How test will be performed: Tester sends a friend request and verifies that it reaches the recipient.

2. **Name:** Accept or Reject Friend Request
   Test ID: Test-FS2
   Control: Manual
   Initial State: User logged in, viewing pending friend requests.
   Input: User accepts or rejects a friend request.
   Output: The expected result is that the friend list updates with the accepted friend, or the request is removed if rejected.
   Test Case Derivation: Confirm that users can manage incoming friend requests.
   How test will be performed: Tester accepts and rejects friend requests, verifying updates to the friend list.

3. **Name:** View Friend List
   Test ID: Test-FS3
   Control: Manual
   Initial State: User logged in, viewing friends screen.
   Input: None
   Output: The expected result is that a list of current friends is displayed.
   Test Case Derivation: Ensure users can view their list of friends.
   How test will be performed: Tester views the friends screen to check for an accurate friend list.

4. **Name:** Remove Friend
   Test ID: Test-FS4
   Control: Manual
   Initial State: User logged in, viewing friends screen.
   Input: User selects a friend to remove.
   Output: The expected result is that the friend is removed from the user's friend list.
   Test Case Derivation: To verify users can remove friends from their friend list.
   How test will be performed: Tester removes a friend and verifies the friend list updates.

4.1.6 Settings Testing

The Settings Testing focuses on verifying user interactions related to modifying various settings, including accessibility, display, privacy, profile, and sub-realm settings. These tests ensure that users can customize their app experience according to their preferences and privacy requirements, enhancing usability and personalization.

1. **Name:** Modify Accessibility Settings
   Test ID: Test-S1
   Control: Manual
   Initial State: User logged in, viewing settings page.
   Input: User adjusts text size, enables/disables viewing of object names, or changes language.
   Output: The expected result is that accessibility settings apply as configured by the user.

Test Case Derivation: Confirm that accessibility settings are configurable.
How test will be performed: Tester changes each accessibility setting and verifies the changes apply.

2. **Name:** Adjust Display Settings
Test ID: Test-S2
Control: Manual
Initial State: User logged in, viewing settings page.
Input: User changes display settings such as light/dark mode or AR object visibility.
Output: The expected result is that display settings reflect user preferences.
Test Case Derivation: To ensure display settings are customizable by the user.
How test will be performed: Tester modifies display settings and observes changes.

3. **Name:** Manage Privacy Settings
Test ID: Test-S3
Control: Manual
Initial State: User logged in, viewing settings page.
Input: User modifies privacy settings to control visibility of profile, friends list, and AR interactions.
Output: The expected result is that privacy settings update based on user preferences.
Test Case Derivation: Verify users can control privacy settings for their profiles and interactions.
How test will be performed: Tester changes privacy settings and checks for corresponding updates.

4. **Name:** Update Profile Settings
Test ID: Test-S4
Control: Manual
Initial State: User logged in, viewing settings page.
Input: User changes username, password, profile picture, or status.
Output: The expected result is that profile settings are updated and saved.

Test Case Derivation: Ensure users can update personal profile information.
How test will be performed: Tester updates profile settings and verifies changes.

5. **Name:** Access Sub-Realm Settings
   Test ID: Test-S5
   Control: Manual
   Initial State: User logged in, viewing settings page.
   Input: User navigates to sub-realm settings to modify sub-realm options.
   Output: The expected result is that sub-realm settings are accessible and configurable.
   Test Case Derivation: To confirm users can manage settings for sub-realms they belong to.
   How test will be performed: Tester accesses and modifies sub-realm settings, verifying updates.

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 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   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:

4.2.2 Area of Testing2

...

4.2.3 Portability Testing

1. **Name:** Validate Cross-Platform Compatibility
   Test ID: Test-PT1
   Type: Non-Functional, Manual
   Initial State: Application is built for both iOS and Android platforms.
   Input/Condition: Run the app on iOS and Android devices.
   Output/Result: The app is functional and displays correctly on both platforms.
   How test will be performed: Tester will install the app on both an iOS and an Android device, verifying consistent functionality and UI.

2. **Name:** Common Codebase Validation
   Test ID: Test-PT2
   Type: Non-Functional, Code Review
   Initial State: The app's codebase is ready for review.
   Input/Condition: Inspect the codebase to ensure shared files are correctly configured with minimal platform-specific files.
   Output/Result: Codebase only differs in configuration files for platform-specific settings.
   How test will be performed: Developer will conduct a code walkthrough, focusing on configuration files to confirm minimal platform-specific variations.

3. **Name:** Build Verification on iOS and Android
   Test ID: Test-PT3
   Type: Non-Functional, Automated
   Initial State: The cross-platform codebase is ready for automated build testing.
   Input/Condition: Initiate automated builds for both iOS and Android.
   Output/Result: Both builds succeed without errors.
   How test will be performed: An automated CI/CD pipeline will attempt to build the app for both platforms, confirming compatibility.

4.2.4 Safety Testing

1. **Name:** Distraction to Surroundings Assessment
   Test ID: Test-SA1
   Type: Non-Functional, Survey-Based
   Initial State: App is functional and ready for user testing.
   Input/Condition: Conduct a user survey after users engage with the app in a controlled environment.
   Output/Result: Survey results show that users do not find the app dangerously distracting them from their surroundings while using it.
   How test will be performed: A group of users will be observed using the app, followed by a survey asking them to rate their distraction levels from surrounding objects. Results will be analyzed to confirm minimal distraction.

2. **Name:** No Bright Flashes or Loud Noises

Test ID: Test-SA2
Type: Non-Functional, Manual Inspection
Initial State: The app is fully developed with all interfaces available for review.
Input/Condition: Navigate through all screens and interactions within the app.
Output/Result: No bright flashes or loud noises are present in any of the app interfaces.
How test will be performed: Tester will manually explore the app, paying special attention to visual and audio elements, ensuring that no features could trigger discomfort or seizures in sensitive users.

4.2.5 Installation Testing

1. **Name:** Verify App Store Availability
   Test ID: Test-I1
   Type: Non-Functional, Manual
   Initial State: App has been submitted and approved on both iOS and Android app stores.
   Input/Condition: Search for the app on the Apple App Store and Google Play Store.
   Output/Result: The app is available for download on both app stores.
   How test will be performed: Tester will verify the presence of the app by searching for it on the respective app stores and confirming it is listed and downloadable.

2. **Name:** Simple Installation Process
   Test ID: Test-I2
   Type: Non-Functional, Manual
   Initial State: App is available on both app stores.
   Input/Condition: Attempt to install the app on a device from both the Apple App Store and Google Play Store.
   Output/Result: The app installs directly without any additional steps or configurations.
   How test will be performed: Tester will initiate the installation from each app store, ensuring the app installs seamlessly without requiring extra configurations or settings adjustments.

4.3 Traceability Between Test Cases and Requirements

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

Test-ID	**Test Name**	**Requirements**
Test-PS1	Validate User Authentication	PS-FR1
Test-PS2	Password Change Functionality	PS-FR3
Test-PS3	View Profile Information	PS-FR4
Test-PS4	Access Help Page	PS-FR6
Test-G1	Create New Group	G-FR1
Test-G2	Add or Remove Group Members	G-FR2
Test-G3	Edit Group Settings	G-FR3
Test-G4	Delete Group	G-FR4
Test-G5	Interact with Group AR Content	G-FR5
Test-FS1	Send Friend Request	FS-FR1
Test-FS2	Accept or Reject Friend Request	FS-FR2
Test-FS3	View Friend List	FS-FR3
Test-FS4	Remove Friend	FS-FR4
Test-S1	Modify Accessibility Settings	S-FR1
Test-S2	Adjust Display Settings	S-FR2
Test-S3	Manage Privacy Settings	S-FR3
Test-S4	Update Profile Settings	S-FR4
Test-S5	Access Group Settings	S-FR5
Test-PT1	Validate Cross-Platform Compatibility	DI-P1
Test-PT2	Common Codebase Validation	DI-P2
Test-PT3	Build Verification on iOS and Android	DI-P1
Test-SA1	Distraction Level Assessment	QS-SA1
Test-SA2	No Bright Flashes or Loud Noises	QS-SA2
Test-I1	Verify App Store Availability	DI-I1
Test-I2	Simple Installation Process	DI-I2

Table 1: Mapping of Tests to Requirements

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

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?

On a scale of 1 - 10, how distracted were you from your surroundings when using the app?

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?

   Making the tests were very easy after we had the initial template down and distributed the work effectively amongst ourselves based on the functional and non-functional requirements.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   One pain point we experienced was that we had a lot to do for this deliverable that we couldn't work on the next one, which is POC. Ultimately, what we decided to do was to focus mainly on this, because the deadline is earlier, and focus mainly on getting our project set up for POC. That way, we at least have started on the next deliverable.

   Another thing was deciding which tests to automate and which to make manual. We resolved this by seeing which test would be easiest or quickest to implement. If a manual test was easiest, we would choose that, but if a manual test would take long, we'd make it automated.

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?