System Verification and Validation Plan for Software Engineering

Team #1, Sanskrit Ciphers
Omar El Aref
Dylan Garner
Muhammad Umar Khan
Aswin Kuganesan
Yousef Shahin

October 27, 2025

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 (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

[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

[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

[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

[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]

3.7 Software Validation

[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

4.2 Tests for Nonfunctional Requirements

4.2.1 System Performance Testing

Collection Processing Performance

1. **test-nfr-perf-1**

Type: Dynamic, Automated, Performance

Initial State: System deployed with empty database; sample dataset of 500 representative fragments from the British Library collection prepared for ingestion

Input/Condition: Execute bulk fragment processing pipeline with sample dataset; monitor processing time, resource utilization, and completion rate

Output/Result: Processing completes within 1.15 hours for 500 fragments; all fragments successfully processed with metadata extracted and indexed; system logs show no critical errors; extrapolated processing rate indicates full 21,000-fragment collection would process within $\leq$ 48 hours (per SRS NFR criteria)

How test will be performed: Automated script will initiate the bulk processing workflow on the 500-fragment sample. Performance monitoring tools (e.g., time command, system resource monitors) will track execution time, CPU usage, memory consumption, and I/O operations. Upon completion, validation queries will verify that all fragments are searchable with extracted metadata. Test passes if: (1) processing time $\leq$ 1.15 hours for 500 fragments (extrapolates to $\leq$ 48 hours for full 21,000-fragment collection, meeting SRS requirement), (2) success rate $\geq$ 99%, and (3) no memory leaks or resource exhaustion detected. Processing rate will be calculated (fragments/hour) and documented for scaling projections.

2. **test-nfr-perf-2**

   Type: Dynamic, Automated, Load Testing

   Initial State: System fully operational with sample dataset (500-1000 fragments) loaded; load testing framework configured

   Input/Condition: Simulate 15 concurrent users performing typical research workflows (searches, filtering, canvas operations) over a 20-minute period to verify system can support 15+ concurrent users (per SRS NFR criteria)

   Output/Result: System maintains responsiveness throughout test period with 15 concurrent users; search queries return results within 3 seconds; canvas operations respond within 500ms; 95% of operations

complete successfully without errors; no server crashes or performance degradation

How test will be performed: Load testing tool (e.g., Locust or k6) will simulate 15 concurrent users executing scripted realistic research workflows. Test will ramp up from 1 to 15 users over 2 minutes, then maintain 15 concurrent users for 18 minutes. Test scenarios include: (1) search by fragment ID (10% of operations), (2) metadata filter queries (40%), (3) canvas workspace creation with 5-10 fragments (30%), (4) session save/restore operations (20%). Performance metrics (response times, error rates, throughput, server resource utilization) will be logged continuously. Test passes if: (1) mean search response time $\leq$ 2 seconds under load, (2) 95th percentile search time $\leq$ 3 seconds, (3) canvas operations complete within 500ms, (4) error rate $\leq$ 2%, (5) system successfully handles 15 concurrent users without crashes (meeting SRS requirement), and (6) no unhandled exceptions or out-of-memory errors occur. Results will include response time graphs, throughput metrics, and resource utilization charts.

4.2.2 AI Matching Assistance Quality

Matching Suggestion Utility

1. **test-nfr-ai-1**

   Type: Dynamic, Manual, Expert Evaluation

   Initial State: System operational with AI matching model trained and deployed; test set of 20 fragment queries prepared with known ground truth (10 fragments with confirmed matches, 10 without matches in collection)

   Input/Condition: For each test fragment, request AI matching suggestions; record top 5 suggestions and their confidence scores

   Output/Result: For fragments with known matches: true match appears in top 5 suggestions in $\geq$ 60% of cases; for fragments without matches: system correctly identifies low confidence or provides diverse alternatives; overall suggestion relevance rated as "potentially useful" by domain expert(s) in $\geq$ 50% of cases

   How test will be performed: Project supervisor and/or 1-2 domain experts (e.g., faculty member, graduate student in relevant field) will

evaluate AI matching suggestions for the 30-fragment test set. For each fragment, evaluators will complete a standardized evaluation form.

Evaluation Form (per fragment):

- Fragment ID: [ID]
- Known Ground Truth: [Has match in collection: Yes/No] [If yes, Fragment ID of match: ___]
- Top 5 AI Suggestions: [List 5 suggestion IDs and confidence scores]
- Ground Truth in Top 5?: [Yes/No]
- Visual Feature Relevance: Do suggestions show similar edge patterns, damage, or script characteristics? [Yes/Somewhat/No]
- Overall Utility Rating: Would these suggestions be useful for scholarly investigation? [Useful / Potentially Useful / Not Useful]
- Comments: [Brief explanation of rating]

Results will be documented in a spreadsheet with one row per fragment. Test passes if: (1) known matches appear in top 5 for $\geq 60\%$ of test cases (6 out of 10 fragments with known matches), and (2) overall utility ratings are "useful" or "potentially useful" in $\geq 50\%$ of all 20 cases (10 or more fragments rated positively).

4.2.3 Usability Testing

Research Workflow Task Completion

1. **test-nfr-usability-1**

   Type: Dynamic, Manual, User Testing

   Initial State: System fully operational with sample dataset loaded; test participants (3-4 users: project supervisor and/or 2-3 classmates) recruited; standardized task scenarios prepared

   Input/Condition: Each participant performs 5-6 representative research tasks: (1) search for specific fragment by ID, (2) apply metadata filters, (3) create canvas workspace, (4) arrange 3-5 fragments on canvas, (5) save session, (6) restore previous session

   Output/Result: Task completion rate $\geq 80\%$ across all participants and tasks (per SRS NFR criteria); average task completion time within

specified targets; no more than 1 critical usability issue preventing task completion; participants rate overall ease of use as "easy" or "very easy" for $\geq 70\%$ of tasks

How test will be performed: Participants will complete tasks in a moderated usability testing session (30-45 minutes per participant). For each task, observers will record: completion status (success/fail/partial), time-on-task, number of errors, assistance required, and participant comments. Task-level success criteria: Task 1-2 (search/filter): ≤ 2 minutes each; Task 3-4 (canvas): ≤ 3 minutes each; Task 5-6 (save/restore): ≤ 1 minute each.

Post-test questionnaire (5-point Likert scale for each task):

- Q1-Q6: "How easy was it to complete Task [1-6]?" (1=Very Difficult, 2=Difficult, 3=Neutral, 4=Easy, 5=Very Easy)
- Q7: "What was the most confusing aspect of the system?" (open-ended)
- Q8: "What feature did you find most helpful?" (open-ended)
- Q9: "Would you recommend this system to colleagues for manuscript research?" (Yes/No/Maybe)

Test passes if: (1) overall task completion rate $\geq 80\%$ (aligns with SRS NFR requirement), (2) $\geq 70\%$ of individual task ratings (Q1-Q6) are 4 or 5 ("easy" or "very easy"), and (3) no more than 1 critical usability issue identified that completely blocks task completion.

2. **test-nfr-usability-2**

   Type: Static, Manual, Interface Evaluation

   Initial State: System interface fully implemented and accessible for evaluation; usability evaluation criteria checklist prepared

   Input/Condition: 2 team members (who did not design the specific interface components being evaluated) independently evaluate the interface against core usability principles

   Output/Result: Consolidated report identifying usability issues categorized by principle and severity (critical, major, minor); no more than 3 critical violations identified; prioritized list of recommended improvements documented

How test will be performed: Each team member evaluator will spend 1-2 hours systematically exploring the interface and documenting usability issues. Evaluators will use a standardized checklist covering core usability principles:

Evaluation Checklist:

(a) **Clarity**: For each major interface element (search bar, filter panel, canvas controls), is it clear what actions are available and what they do? Rate: Pass/Fail with examples.

(b) **Feedback**: When performing actions (search, drag fragment, save session), does the system provide clear visual or textual feedback? Rate: Pass/Fail with examples.

(c) **Consistency**: Are similar operations (e.g., all "save" actions, all "delete" actions) performed in similar ways? Rate: Pass/Fail with examples.

(d) **Error Prevention & Recovery**: Can users prevent errors (confirmation dialogs) and recover from mistakes (undo, clear error messages)? Rate: Pass/Fail with examples.

(e) **Efficiency**: Can users accomplish core tasks (search $\rightarrow$ filter $\rightarrow$ canvas $\rightarrow$ save) with minimal unnecessary steps? Count steps for each workflow. Rate: Pass/Fail.

(f) **Learnability**: Can a new user understand how to perform basic search and canvas operations without documentation? Rate: Pass/Fail with observations.

For each issue identified, document: (1) specific location/feature, (2) principle violated, (3) severity rating (critical/major/minor), (4) screenshot or description, (5) suggested fix. Severity definitions: critical (prevents task completion), major (causes significant frustration or confusion), minor (cosmetic or minor inconvenience). Team will meet to consolidate findings and create prioritized issue list. Test passes if: (1) no more than 3 critical violations identified, (2) evaluation report completed documenting all findings with specific examples, and (3) critical issues have documented remediation plans before final release.

4.2.4 System Availability and Reliability

Uptime and Availability Testing

1. **test-nfr-reliability-1**

   Type: Dynamic, Automated, Reliability Testing

   Initial State: System deployed to test/staging environment; GitHub Actions workflow configured for automated monitoring

   Input/Condition: Continuous operation over 7-day period with automated health checks every 30 minutes via GitHub Actions scheduled workflow; simulated user activity via automated test script

   Output/Result: System availability $\geq 95\%$ during the test period (per SRS NFR criteria); no more than 2 unplanned outages lasting $>$ 1 hour; system recovers automatically or can be restarted successfully after any failures

   How test will be performed: GitHub Actions scheduled workflow (using cron syntax) will run every 30 minutes to perform health checks: (1) HTTP request to home page (expect 200 status), (2) execute test search query via API (verify JSON response), (3) verify database connectivity through health endpoint. Each workflow run logs results to a file committed to repository with timestamp and status. Additionally, a separate workflow runs hourly to simulate realistic user activity (perform searches, create canvas session, save/restore operations) and verify end-to-end functionality. All workflow runs are recorded in GitHub Actions history. For any failed health check: workflow creates GitHub issue automatically with failure details and timestamp. Test passes if: (1) uptime $\geq 95\%$ over 7 days (calculated as successful health checks / total health checks; allowing maximum 8.4 hours downtime), (2) no more than 2 unplanned outages lasting > 1 hour, and (3) system recovers within 1 hour of failure detection (verified through subsequent health check passes). Results documented in availability report generated from workflow logs showing: total checks, successful checks, failed checks, outage periods, and uptime percentage.

2. **test-nfr-reliability-2**

   Type: Dynamic, Manual, Disaster Recovery

Initial State: System operational with sample dataset loaded; backup procedures documented and tested; test environment available for recovery simulation

Input/Condition: Simulate critical failure scenario (e.g., accidental database deletion, container/server crash); initiate recovery procedures using documented backup strategy

Output/Result: System restored to functional state within 2 hours; sample dataset and user data successfully recovered with no loss; all core functionality (search, canvas, authentication) verified operational post-recovery

How test will be performed: Recovery test will be conducted in isolated test environment to avoid impacting production. Test scenario: (1) create backup of current system state (database, uploaded images, configuration), (2) simulate failure by stopping services and removing/corrupting critical data, (3) execute documented recovery procedures from backup, (4) measure time from "failure" to full restoration. Post-recovery verification checklist: database accessible and contains expected records, search returns correct results for sample queries, canvas workspace can be created and manipulated, user authentication works, uploaded images accessible. Test passes if: (1) recovery completed within 2 hours, (2) all verification checks pass, (3) no data loss detected (verified by record count and sample data spot-check), and (4) recovery procedure documentation is sufficient for team member unfamiliar with process to execute successfully.

4.2.5 Data Integrity and Security

Data Preservation and Attribution

1. **test-nfr-data-1**

   Type: Functional, Automated, Data Validation

   Initial State: British Library collection metadata and user-submitted images loaded into system

   Input/Condition: Execute comprehensive data validation checks across entire collection

Output/Result: 100% of fragments retain original British Library catalog information (shelfmark, collection, provenance); all user-submitted images maintain uploaded attribution; no corruption or loss of metadata fields

How test will be performed: Automated validation script will query all fragments in the database and verify metadata integrity.

Required Metadata Fields:

- **Fragment ID**: Unique identifier (non-null, unique constraint)
- **Collection Source**: "British Library" or user-uploaded indicator (non-null)
- **Shelfmark**: Original catalog reference (required for British Library fragments)
- **Provenance**: Geographic/historical origin information (if available in source)
- **Image URL/Path**: Location of fragment image (non-null, valid path)
- **Upload Attribution**: Username or source for user-submitted fragments (non-null for user uploads)
- **Timestamp**: Date added to system (non-null)

Validation checks: (1) all required fields present and non-null, (2) British Library fragments cross-referenced against source catalog CSV (shelfmark, collection matches), (3) user-uploaded fragments have valid attribution field, (4) referential integrity verified (foreign keys valid, no orphaned records). Script generates exception report listing any fragments with missing/inconsistent data. Manual spot-check of 100 randomly selected fragments verifies field accuracy. Test passes with 100% of fragments having all required fields populated and $\geq$ 99% matching source data exactly.

2. **test-nfr-data-2**

   Type: Static, Manual, Code Review

   Initial State: Data handling, authentication, and API code modules identified for review; security review checklist prepared based on OWASP guidelines

Input/Condition: Security-focused code review conducted by team members not primarily responsible for implementing the reviewed modules, supplemented by automated security scanning tools

Output/Result: Review checklist completed with findings documented; no critical security vulnerabilities identified; common security best practices verified (input validation, SQL injection prevention, authentication, authorization, API security)

How test will be performed: Two team members (preferably those who did not write the security-critical code) will conduct peer code review focusing on security aspects. Review checklist will cover: (1) input validation and sanitization for user-provided data, (2) SQL injection prevention (parameterized queries/ORM usage), (3) authentication mechanisms (password hashing, session management), (4) authorization controls (proper access checks before sensitive operations), (5) secure handling of API keys and secrets (not committed to repository), (6) protection against common web vulnerabilities (XSS, CSRF). Additionally, run automated security scanning tool appropriate for tech stack (e.g., npm audit for Node.js, Bandit for Python, pip-audit, or built-in IDE security warnings). All findings documented with severity ratings (critical/high/medium/low). Test passes if: (1) no critical or high-severity vulnerabilities identified in manual review, (2) checklist completed for all security-critical modules (authentication, data access, API endpoints), (3) automated scanning shows no critical issues, and (4) any medium/low findings have documented remediation plans or accepted risk justifications.

4.2.6 Scalability Testing

Fragment Collection Scalability

1. **test-nfr-scale-1**

   Type: Dynamic, Manual, Scalability Analysis

   Initial State: System operational with sample dataset (500 fragments); ability to add additional test data

   Input/Condition: Test system performance with incrementally larger datasets: 500 fragments (baseline), 2,000 fragments, 5,000 fragments

Output/Result: Performance metrics documented for each dataset size showing how system scales; search response times remain acceptable (≤ 5 seconds) at 5,000 fragments; documented analysis of expected performance at full 21,000-fragment scale based on observed trends

How test will be performed: Using existing sample data (500 fragments), create duplicate test data to reach 2,000 and 5,000 fragments (duplicates acceptable for testing purposes). At each dataset size, execute standardized queries: (1) fragment ID search (10 queries, measure average time), (2) single metadata filter (10 queries, measure average time), (3) multi-criteria filter with 2-3 filters (5 queries, measure average time). Record response times and database query execution times. Document results in simple table showing dataset size vs. average response time for each query type. Analyze trend (linear, logarithmic, exponential) and extrapolate to 21,000 fragments. Test passes if response times remain usable (≤ 5 seconds) at 5,000 fragments and trend analysis suggests acceptable performance at full scale.

4.2.7 Cross-Browser and Device Compatibility

Interface Compatibility Testing

1. **test-nfr-compat-1**

   Type: Manual, Dynamic, Compatibility Testing

   Initial State: System deployed and accessible via URL

   Input/Condition: Access system and perform core workflows on multiple browser and device combinations: Chrome, Firefox, Safari (desktop); Chrome, Safari (mobile); tablets

   Output/Result: All core functionality (search, filter, canvas) operates correctly across tested platforms; visual layout renders appropriately; no critical browser-specific bugs identified

   How test will be performed: Test team will execute standardized test suite on each browser/device combination.

   Test Suite (perform on each platform):

   (a) **Search Test**: Enter fragment ID "BL-12345" in search bar, verify results display correctly

(b) **Filter Test**: Apply metadata filter (e.g., "Script: Brahmi"), verify filtered results

(c) **Canvas Creation**: Click "New Canvas", verify canvas workspace opens

(d) **Drag-and-Drop**: Drag 2 fragment images onto canvas, verify they appear and can be repositioned

(e) **Image Rendering**: Verify fragment images load at full resolution, zoom controls work

(f) **Session Save**: Save canvas session, verify success message

(g) **Session Restore**: Reload page, restore saved session, verify fragments reappear in correct positions

(h) **Responsive Layout**: Resize browser window to mobile width (375px), verify layout adapts without breaking

Target Platforms: Chrome (latest, primary development browser), Firefox (latest, secondary browser), Safari (latest macOS if team has access).

For each test-platform combination, record: Pass/Fail, any errors/warnings, visual issues, performance notes. Issues documented with severity (critical/major/minor) and browser-specific details. Test passes if all critical functionality works on Chrome and ≥ 1 additional browser. Critical issues on Chrome must be resolved; issues on secondary browsers documented for future work. Mobile testing is optional/out of scope for this project phase.

4.2.8 Performance Benchmarking

Response Time Measurement

1. **test-nfr-bench-1**

   Type: Dynamic, Manual, Benchmark Testing

   Initial State: System operational with sample dataset (500-1000 fragments)

   Input/Condition: Execute standardized set of operations representing typical usage patterns: (1) fragment ID search, (2) single metadata

filter, (3) multi-criteria filter (2-3 criteria), (4) canvas operations with 5-10 fragments

Output/Result: Performance report documenting average response times for each operation type; simple table showing mean response time for each operation; baseline metrics documented for future regression testing

How test will be performed: For each operation type, execute 10 test runs and record response times using browser developer tools or simple timing script. Calculate average (mean) response time for each operation type. Document results in simple table format: Operation Type — Average Response Time — Notes. Example acceptable targets: ID search $\leq$ 1 second, single filter $\leq$ 2 seconds, multi-filter $\leq$ 3 seconds, canvas operations $\leq$ 2 seconds. This benchmark establishes baseline performance metrics for future comparison rather than strict pass/fail criteria. Results provide data for identifying performance regressions in future releases.

...

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?