System Verification and Validation Plan for Software Engineering

Team #11, OKKM Insights
Mathew Petronilho
Oleg Glotov
Kyle McMaster
Kartik Chaudhari

November 2, 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

Please refer to Section 4 of Software Requirements Specification found here.

This document is intended to provide a description of the specific validation and verification activites that will be completed throughout the development of the GeoWeb System. The purpose of these activities is to ensure the system requirements agree with stakeholder needs, and to certify the implementation of the system satisifes the stakeholder requirements. This document will start with a review of the system, a description of the testing plan, and a list of tests to be completed.

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 state-

ment. 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]

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

...

4.2 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 Operational and Environmental

Release Tests

1. Road Map Consistency: T-OE0

   Related Reqs: NFR-OE7

   Type: Manual, Static

   Initial State: Application has a release road map that is publicly accessible.

Input/Condition: Team member conducts a review.

Output/Result: At least MIN_ON_TIME_MILESTONE% of the listed milestones have been met on time.

How test will be performed: The team member looks over the road map and cross references the completion date of milestones to the dates listed in the road map.

2. Beta Testing: T-OE1

   Related Reqs: NFR-OE8

   Type: Dynamic, Exploratory

   Initial State: Beta version of application is deployed and accessible

   Input/Condition: At least BETA_TESTERS beta testers are provided access to use the application.

   Output/Result: Feedback on any bugs, navigation issues, or aesthetic problems is provided. Less than MAX_BUGS_FOUND bugs are found.

   How test will be performed: Testers will be recruited and identified. They will be from fields of interest that include scientists, labelers, and domain experts. Then, the beta testing environment will be set up and the url will be distributed to the testers along with any other set up resources. Specific tasks are provided for testers to complete that focus on the annotation tools, sign up process and project creation. Feedback will be collected through direct comments from the tester.

3. Regression Testing: T-OE2

   Related Reqs: NFR-OE9

   Type: Dynamic, Automated

   Initial State: Application is deployed.

   Input/Condition: Run regression test suite, consisting of unit tests.

   Output/Result: All regression tests are passed.

   How test will be performed: An automated script with regression tests will run when updates are made to the production build.

4.2.2 Maintainability and Support

Maintenance Tests

1. Ease of Change: T-MS0

   Related Reqs: NFR-MR0

   Type: Manual, Static

   Initial State: Application's source repository contains complete documentation.

   Input/Condition: Competent software developer who has not previously worked on the app reviews documentation and attempts to perform tasks.

   Output/Result: The developer can easily make a minor update to a specified part of the application.

   How test will be performed: Give the developer time to read through the documentation. Give them a maintenance task, such as updating the size of the title font to 20px. Observe them and document how long it takes them and if they encountered any troubles.

4.2.3 Security

Access Tests

1. Logged Out Permissions: T-SE0

   Related Reqs: NFR-SE0

   Type: Manual, Dynamic, White-box

   Initial State: Application is deployed.

   Input/Condition: Tester who is not signed in tries to access application paths for project creation and image labeling (Ex. /projects or /label).

   Output/Result: The tester is denied access to these paths and is told to sign in.

   How test will be performed: On the deployed application, the tester will visit all possible paths as a logged out user.

2. Labeler Permissions: T-SE1
   Related Reqs: NFR-SE1

   Type: Manual, Dynamic, White-box

   Initial State: Application is deployed.

   Input/Condition: Tester who is signed in as a labeler tries to access application paths for project creation.

   Output/Result: The tester is denied access to these paths. However, the tester has access to paths related to image labeling.

   How test will be performed: On the deployed application, the tester will visit all possible paths as a labeler.

3. Invalid Email Format: T-SE2
   Related Reqs: NFR-SE2, NFR-SE5

   Type: Automatic, Dynamic

   Initial State: Front-end registration page is created and integrated with the database.

   Input/Condition: Email with invalid format, such as an empty string or a string missing '@', is entered.

   Output/Result: Application rejects email and tells the user that the email format is wrong.

   How test will be performed: A unit test will be performed where the input is entered into the email section of the registration form.

4. Duplicate Email: T-SE3
   Related Reqs: NFR-SE2, NFR-SE5

   Type: Automatic, Dynamic

   Initial State: Front-end registration page is created and integrated with the database.

   Input/Condition: Email that is already in database is entered.

   Output/Result: Application rejects email and tells the user that the email is in use.

   How test will be performed: A unit test will be performed where the input is entered into the email section of the registration form.

5. Invalid Password Format: T-SE4
   Related Reqs: NFR-SE3, NFR-SE5

   Type: Automatic, Dynamic

   Initial State: Front-end registration page is created and integrated with the database.

   Input/Condition: Password with invalid format, such as an empty string or a string with no numbers, is entered.

   Output/Result: Application rejects password and tells the user what requirements they have not met.

   How test will be performed: A unit test will be performed where the input is entered into the password section of the registration form.

6. System Error: T-SE5
   Related Reqs: NFR-SE4, NFR-SE6

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Purposely invoke a system failure, and attempt to perform an action such as a label submission.

   Output/Result: Application provides an error message on the user interface. The database has not changed in anyway.

   How test will be performed: Go on to the application, start a labeling task, purposely disconnect from the internet, and try to submit a labeled image.

Integrity Tests

1. Duplicate Entries: T-SE6

   Related Reqs: NFR-SE7

   Type: Manual, Dynamic

   Initial State: Database is deployed.

   Input/Condition: Duplicate database entry is inserted into the database.

   Output/Result: Database has only one of the inputted entry and the duplicate has been removed.

How test will be performed: Attempt to insert the same entry twice into the database through the database UI.

Privacy Tests

1. Encrypted User Data: T-SE7

   Related Reqs: NFR-SE8

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester registers an account.

   Output/Result: All sensitive user data that is stored in the database is encrypted.

   How test will be performed: Tester will create a new account, then check the corresponding user entry in the database and see if the sensitive information is encrypted.

2. Encrypted Payments: T-SE8

   Related Reqs: NFR-SE9

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester enters sample payment details to pay for a labeling project that has been created.

   Output/Result: These details are encrypted and can not be read through packet analyzers. The amount in the request can not be modified by an adversary.

   How test will be performed: Tester will enter sample payment details, and submit their payment. Using a packet analyzer (such as Wireshark), packets from this request will be looked at to ensure all information is encrypted.

Immunity Tests

1. SQL Injection: T-SE9

   Related Reqs: NFR-SE10

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: A malicious SQL statement is entered into a text field.

   Output/Result: The system raises an error telling the user that it is invalid.

   How test will be performed: Tester will enter a SQL statement such as "{valid email}'–" into an input such as the email input. This example has the potential to bypass a password check by commenting out the rest of the SQL query. The tester will check that when this statement is entered, the system gives feedback that it is invalid.

4.2.4 Cultural

Language Tests

1. Support of Different Languages: T-CU0

   Related Reqs: NFR-CU0

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester selects a language from a list of available languages.

   Output/Result: All text on the website is translated and displayed in the selected language.

   How test will be performed: Tester will check that the language selection list is accessible, and that the most popular languages are included. When a language is selected, the tester will check that the translation has been applied and there is no untranslated or gibberish text. This can be checked for each language.

4.2.5 Compliance

Financial Tests

1. Compliant Payment Process: T-CO0

   Related Reqs: NFR-SE9

   Type: Manual, Static

   Initial State: Application is deployed.

   Input/Condition: Qualified Security Assessor (QSA) assesses the application.

   Output/Result: They determine that it meets the PCI-DSS standard.

   How test will be performed: A QSA will be found and contacted to perform an assessment. The QSA will be shown all parts of the application that deal will financial transactions and will be able to make a determination on if it meets the standard.

Legal Tests

1. System Availability: T-CO1

   Related Reqs: NFR-CO0

   Type: Manual, Dynamic

   Initial State: Application is deployed.

   Input/Condition: Tester changes country they are accessing the application from using a tool such as a VPN.

   Output/Result: Application is blocked in countries facing economic sanctions by the Government of Canada.

   How test will be performed: A list of the countries facing economic sanctions by Canada will be compiled. Then, the tester will simulate that they are accessing the application from these countries, and ensure it is unreachable.

2. Taxes: T-CO2

Related Reqs: NFR-CO1

Type: Manual, Dynamic

Initial State: Application is deployed.

Input/Condition: Tester redeems a cash balance.

Output/Result: If the cash balance exceeds a threshold, a tax form will be issued.

How test will be performed: Tester creates a test account with an account balance over the threshold. When they withdraw, they check that a tax form has been emailed to the email associated with the account. The tax form should reflect the withdrawal balance.

3. Project Availability: T-CO3

Related Reqs: NFR-CO2

Type: Manual, Dynamic

Initial State: Application is deployed.

Input/Condition: Tester changes country they are accessing the application from using a tool such as a VPN.

Output/Result: Specific project is not shown.

How test will be performed: A project will be specified to only be distributed in a specific country. Then, the tester will simulate that they are accessing the application from other countries, and ensure the project does not show up.

4.2.6 User Documentation and Training

1. Helpfulness of User Aids: T-UDT0

Related Reqs: NFR-UD0, NFR-UD1, NFR-UD2, NFR-TR0

Type: Manual, Dynamic

Initial State: Application is deployed with all help features. Tutorials and user documentation have been created.

Input/Condition: Users attempt to complete a basic labeling task using only the platform's built-in help resources (help system, quick start guide, tutorials and contextual tooltips).

Output/Result: At least MIN_USER_HELP_SATISFACTION% of users who used a help feature found that feature helpful. With the assistance of the help tools, the user was able to perform the task within MAX_TASK_TIME minutes.

How test will be performed: The purpose of the platform will be explained to the users and the built-in help features will be shown. Then, the labeling task will be given to them. Time to complete task is observed and the help tools they use are recorded. Participants will then fill out a usability survey, which can be viewed in the appendix.

2. Usefulness of Sandbox: T-UDT1

   Related Reqs: NFR-TR1

   Type: Automatic, Dynamic

   Initial State: Application is deployed with all help features. Tutorials and user documentation has been created.

   Input/Condition: A new user has accessed the platform.

   Output/Result: At least MIN_PRACTICE_USAGE% of new users utilize the practice environment, with self-assessment scores indicating an average improvement of IMPROVE_IN_ACC% in labeling accuracy over their first three attempts.

   How test will be performed: Practice environment utilization will be tracked by the application. Improvement in accuracy will also be tracked. If the metrics meet or succeed our thresholds, then we can conclude the sandbox is useful.

...

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.

Parameter	Value	Unit	Description
MIN_ON_TIME_MILESTONE	80	%	Minimum percent of milestones that have been met on time
BETA_TESTERS	50	People	Number of beta testers
MAX_BUGS_FOUND	10	Bugs	Number of software bugs found
MIN_USER_HELP_SATISFACTION	80	%	Minimum percent of users satisfied with help feature
MAX_TASK_TIME	15	Minutes	Maximum time it takes a user to complete a task
MIN_PRACTICE_USAGE	80	%	Minimum percent of new users who have used the practice sandbox
IMPROVE_IN_ACC	20	%	Improvement in accuracy of a user after practicing

6.2 Usability Survey Questions

1. Did you use the help system to aid in completing your task? Yes/No
2. If you answered yes, please rate how useful it was in helping you accomplish your task: 1 (Not Useful) - 5 (Very Useful)

3. Did you use the quick start guide to aid in completing your task? Yes/No

4. If you answered yes, please rate the clarity and helpfulness of it: 1 (Not Helpful) - 5 (Very Helpful)

5. Did you notice the tool-tips or pop-ups providing contextual help as you worked? Yes/No

6. If you answered yes, please rate the clarity and usefulness of the in-app help indicators: 1 (Not Helpful) - 5 (Very Helpful)

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?