System Verification and Validation Plan for RwaveDetection

Junwei Lin

May 8, 2025

Revision History

Date	Version	Notes
Feb 21, 2025	1.0	Creation
April 11, 2025	1.1	Complete unit tests

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations, and Acronyms

The definitions of symbols, abbreviations, and acronyms used in this document are consistent with those provided in SRS(Lin, 2025).

This document outlines the Verification and Validation Plan (VnV) for RwaveDetection, structured into four main sections. General Information includes the summary, objectives, challenge level and extras relevant to the VnV process. Plan details the verification strategy for the SRS, Design, VnV, and Implementation phases, along with the use of automated testing and verification tools, as well as software validation plan. System Tests verify both functional and nonfunctional requirements at the system level, ensuring the overall system meets its intended behavior. Unit Test Description covers validating individual modules against both functional and nonfunctional requirements. This plan ensures a comprehensive and systematic approach to verifying and validating the system before deployment.

2 General Information

This section will provide the project summary and objectives for this document.

2.1 Summary

This project focuses on reimplementing the Pan-Tompkins algorithm for R-wave detection in ECG signal processing. This implementation will automatically derive all filter parameters based on requirements without relying on external libraries.

2.2 Objectives

The objectives of this VnV plan include:

- Build confidence in the software correctness.
- Test robustness against noisy ECG signals.

The objectives of this VnV plan do not include:

- User experience testing for a graphical interface or integration into medical workflow systems.
- Clinical trials.

2.3 Challenge Level and Extras

This is a problem with a general-level challenge. Doxygen will be widely used to generate the API documentation.

2.4 Relevant Documentation

In the Verification and Validation Plan, we reference the Software Requirements Specification (SRS) (Lin, 2025) as the primary document. The SRS outlines the system's requirements, forming the foundation for all verification activities. It ensures that the VnV process is aligned with the specified functional and nonfunctional requirements.

3 Plan

This section provides plans to verify SRS, Design, VnV, and Implementation. In addition, some automated tools for testing are listed.

3.1 SRS Verification Plan

The SRS document (Lin, 2025) will be reviewed by a domain expert. The content of the review is based on the following checklist:

- ☐ Completeness: Included GS, A, GD, DD, TM, IM, etc.
- ☐ Formatting: The document has a clear, logical structure, and properly organized.
- ☐ Verifiability: Each requirement is testable with a clear pass/fail standard.
- ☐ Traceability: Each requirement is linked to an IM

3.2 Design Verification Plan

The Design Verification Plan will be reviewed by a domain expert. The content of the review is based on the following checklist:

- ☐ Low module coupling: modules can be reused anywhere and do not affect others.

☐ Good Hierarchy: make sure the hierarchy is clear and easy to maintain.

3.3 Verification and Validation Plan Verification Plan

The Verification and Validation Plan (This document itself) will be reviewed by a domain expert. The content of the review is based on the following checklist:

☐ Completeness : VnV plan includes all necessary sections.

☐ Formatting: The document has a clear, logical structure, and properly organized.

☐ Appropriateness: Confirm that the methods and tools proposed in the VnV plan are appropriate for the project.

☐ Coverage: Make sure the VnV plan covers all the requirements.

3.4 Implementation Verification Plan

☐ Static code check: Tools (cpplint) will be used to automatically detect common issues such as memory leaks, uninitialized variables, and other code quality concerns.

☐ Test coverage: Tools (gcov) will be used to check test coverage and ensure that most of the code is covered by tests, increasing confidence in the correctness of the code.

☐ Unit test: Comprehensive unit tests will be executed in testing framework (gTest) to ensure that every units perform correctly.

☐ Code review: The developer will conduct his own code reviews, and the domain expert will also participate in code reviews if possible.

3.5 Automated Testing and Verification Tools

- Static code checker: cpplint - A tool that enforces Google's C++ style guide by checking for common coding style violations.
- Unit test: gTest - A C++ testing framework that provides a rich set of assertions and test case management.

- Test coverage: gcov - A test coverage analysis tool for C++ that works with GCC.
- Local code quality control: git pre-commit - A framework that runs checks before committing code, helping to enforce coding standards and catch issues early.
- Continuous integration: GitHub Actions - An automation tool that enables CI/CD workflows, allowing code to be built, tested, and deployed automatically.

3.6 Software Validation Plan

The software will be validated using the MIT-BIH dataset, where the system's performance will be assessed by calculating the Root Mean Square Error (RMSE) between the predicted and actual results. This will provide a quantitative measure of the system's accuracy.

4 System Tests

This section will provide the system tests designed to verify requirements of the system, including functional and non-functional requirements.

4.1 Tests for Functional Requirements

4.1.1 TC1: Basic IIR and FIR Filters

1. FIR Filter with Random Coefficients

   Control: Automatic

   Initial State: No prior state is required; the test starts with randomly generated filter coefficients.

   Input:

   - FIR filter coefficients generated using scipy.signal.firwin(10, 0.3).
   - A sinusoidal wave with added Gaussian noise, generated using np.sin(2 * np.pi * 0.05 * np.arange(1000)) + np.random.normal(0, 1, 1000).

Output:

- The filtered signal produced by the implemented FIR filter should closely match the filtered signal from scipy.signal.lfilter.
- The RMSE between the two outputs should be below $1e-5$.

Test Case Derivation: The test ensures that a basic FIR filter implementation behaves consistently with pseudo-oracle scipy.signal.

How test will be performed:

(a) Generate FIR filter coefficients.

(b) Generate an input signal as a sinusoidal wave with Gaussian noise.

(c) Apply the FIR filter implementation.

(d) Apply scipy.signal.lfilter with the same coefficients.

(e) Compute the RMSE between outputs.

(f) Assert that the RMSE is below $1e-5$.

2. IIR Filter with Random Coefficients

Control: Automatic

Initial State: No prior state is required; the test starts with randomly generated filter coefficients.

Input:

- IIR filter coefficients generated using scipy.signal.butter(3, 0.3, btype='low', output='ba').
- A sinusoidal wave with added Gaussian noise, generated using np.sin(2 * np.pi * 0.05 * np.arange(1000)) + np.random.normal(0, 1, 1000).

Output:

- The filtered signal produced by the implemented IIR filter should closely match the filtered signal from scipy.signal.lfilter.
- The RMSE between the two outputs should be below $1e-5$.

The test ensures that a basic IIR filter implementation behaves consistently with scipy.signal.

How test will be performed:

(a) Generate IIR filter coefficients.

(b) Generate an input signal as a sinusoidal wave with Gaussian noise.

(c) Apply the IIR filter implementation.

(d) Apply scipy.signal.lfilter with the same coefficients.

(e) Compute the RMSE between outputs.

(f) Assert that the RMSE is below $1e-5$.

4.1.2 TC2: Butterworth and Chebyshev Filters

1. Butterworth Low-Pass Filter

Control: Automatic

Initial State: No prior state is required; the test starts with selected filter parameters.

Input:

- Cutoff frequency: $f_c = 0.3$
- Filter order $N = 4$
- Filter coefficients generated using scipy.signal.butter(4, 0.3, btype='low', output='ba').

Output:

- The computed filter coefficients should match those generated by scipy.signal.butter.
- The RMSE between the two outputs should be below $1e-5$.

Test Case Derivation: The test ensures that a Butterworth filter implementation behaves consistently with pseudo-oracle scipy.signal.

How test will be performed:

(a) Generate Butterworth filter coefficients by scipy.signal.butter.

(b) Generate Butterworth filter coefficients by implementation.

(c) Compute the RMSE between the implemented and scipy.signal coefficients.

(d) Assert that the RMSE is below 1e-5.

2. Chebyshev Type I Low-Pass Filter

Control: Automatic

Initial State: No prior state is required; the test starts with selected filter parameters.

Input:

- Cutoff frequency: $f_c = 0.3$
- Filter order $N = 4$
- Passband ripple: $0.5dB$
- Filter coefficients generated using scipy.signal.cheby1(4, 0.5, 0.3, btype='low', output='ba').

Output:

- The filtered signal produced by the implemented Chebyshev filter should closely match the filtered signal from scipy.signal.lfilter.
- The RMSE between the two outputs should be below $1e-5$.

The test ensures that a Chebyshev implementation behaves consistently with scipy.signal.

How test will be performed:

(a) Generate Chebyshev filter coefficients by scipy.signal.cheby1.

(b) Generate Chebyshev filter coefficients by implementation.

(c) Compute the RMSE between the implemented and scipy.signal coefficients.

(d) Assert that the RMSE is below 1e-5.

4.1.3 TC3: Squaring function

1. Sequence input test

   Control: Automatic

   Initial State: No prior state is required.

   Input:

   - $[1.2, -2.1, -1.0, 2.0, 3.0]$

   Output:

   - $[1.44, 4.41, 1.0, 4.0, 9.0]$

   Test Case Derivation: The purpose of this test case is to verify that the squaring function correctly squares each value in the input list, regardless of whether the numbers are positive or negative. The function should return the squared value of each element from the input list.

   The input given here is just an example. In practice, the input will be a large amount of data randomly generated by the program for testing.

   How test will be performed:

   (a) The input list will be provided to the squaring function.

   (b) The function will process each element in the list and square it.

   (c) The output will be compared against the expected result.

   (d) There is a global floating point comparison threshold ϵ to determine whether floating point numbers are equal.

4.1.4 TC4: Thresholding function

1. Sequence input test

   Control: Automatic

   Initial State: No prior state is required.

   Input:

- input sequence: $[1.44, 4.2, 1.0, 4.0, 9.0]$
- dynamic threshold: $[3.7, 3.8, 3.9, 3.6, 3.2]$

Output:

- $[0, 4.2, 0, 4.0, 9.0]$

Test Case Derivation: This test case aims to verify that the thresholding function correctly applies dynamic thresholds to each element in the input sequence. For each element in the sequence, if the value is greater than the corresponding threshold, it should remain unchanged; otherwise, it should be replaced with 0.

The input given here is just an example. In practice, the input will be a large amount of data randomly generated by the program for testing.

How test will be performed:

(a) The input sequence and dynamic thresholds will be provided to the thresholding function.

(b) The function will process each element in the list.

(c) The output will be compared against the expected result.

(d) There is a global floating point comparison threshold ϵ to determine whether floating point numbers are equal.

4.1.5 TC5: RMSE calculating function

1. Sequence input test

Control: Automatic

Initial State: No prior state is required.

Input:

- input sequence 1: $[100, 200, 300, 400, 500]$
- input sequence 2: $[99, 203, 300, 398, 501]$

Output:

- $RMSE = 1.732$

Test Case Derivation: This test case verifies that the RMSE function correctly calculates the root mean squared error between two input sequences. The RMSE is calculated as the square root of the mean of the squared differences between corresponding elements of the two sequences.

The input given here is just an example. In practice, the input will be a large amount of data randomly generated by the program for testing.

How test will be performed:

(a) Two input sequences will be provided to the thresholding function.

(b) The function will process two input sequences and return the RMSE.

(c) The output will be compared against the expected result.

(d) There is a global floating point comparison threshold ϵ to determine whether floating point numbers are equal.

4.1.6 TC6: R-wave position calculating

1. MIT-BIH Database test

Control: Automatic

Initial State: No prior state is required.

Input:

- sampling frequency $f_s = 360$ Hz
- MIT-BIH data from https://www.physionet.org/content/mitdb/1.0.0/, as shown in figure 1. This is a curve consisting of discrete sample points and contains the annotated points from cardiologists. The blue vertical lines in the figure 1 are annotated points.

Output:

- The index of annotated points from cardiologists as shown in figure 1.

Figure 1: input wave

- The RMSE between calculated and annotated R-wave index.

Test Case Derivation: The test will evaluate the accuracy of the R-wave detection function by comparing the detected R-wave index to the annotated index, with the RMSE between the two sets of index expected to be below 3.0.

How test will be performed:

(a) The input f_s and wave data will be provided to the calculating function.

(b) The annotated data will be provided to the calculating function.

(c) The function will process the original wave data and calculate the R-wave index.

(d) The function will return the RMSE.

(e) The correctness of this function is evaluated by the output RMSE, assert that the RMSE is below 3.0.

4.2 Tests for Nonfunctional Requirements

4.2.1 TC7: Usability

1. Evaluate software usability through user interaction and a survey

Type: Non-functional, Dynamic, Manual

Initial State: The software is fully functional and ready for user testing.

Input/Condition:

(a) Users interact with the software under typical usage conditions.

(b) Conduct a user survey (section 6) to collect feedback on usability.

Output/Result: Survey results and feedback from users.

How test will be performed:

(a) Distribute the program to target users for use.

(b) Conduct a user survey to gather structured feedback.

(c) Analyze survey responses and summarize findings.

2. Verify Doxygen generates a structured and readable API documentation

Type: Non-Functional, Dynamic, Manual

Initial State: The software documentation is written in a format supported by Doxygen.

Input/Condition: Run Doxygen on the project's documentation source files.

Output/Result: A structured and readable API documentation is generated in the expected format (e.g., HTML, PDF).

How test will be performed:

(a) Execute Doxygen with the configured settings.

(b) Inspect the generated documentation for completeness and readability.

(c) Verify that all sections (e.g., class descriptions, function documentation) are properly formatted.

(d) Conduct a peer review to assess usability.

4.2.2 TC8: Maintainability

1. Ensure high test coverage

   Type: Non-Functional, Dynamic, Automatic

   Initial State: The software project contains multiple modules.

   Input/Condition: Run a test coverage analysis tool (gcov).

   Output/Result: A report showing the percentage of code covered by tests.

   How test will be performed:

   (a) Execute the test suite with coverage tracking enabled.
   (b) Generate a test coverage report.
   (c) Verify that all modules have test coverage above a predefined threshold.
   (d) If any module lacks coverage, update tests accordingly.

4.2.3 TC9: Portability

1. Confirm the software builds and runs correctly on multi-platform

   Type: Non-Functional, Dynamic, Manual

   Initial State: The source code is available and ready for compilation.

   Input/Condition: Attempt to build and run the software on both Linux and Windows environments.

   Output/Result: The software compiles successfully and functions correctly on both platforms.

   How test will be performed:

   (a) Set up test environments for Linux and Windows.
   (b) Compile the software on both platforms.
   (c) Run key functionalities and verify expected behavior.
   (d) Identify and document any platform-specific issues.

4.2.4 TC10: Reusability

1. Check if the code is modular and reusable in different contexts

   Type: Non-Functional, Static, Manual

   Initial State: The software project contains multiple modules.

   Input/Condition: Review code structure and check module usage.

   Output/Result: A checklist ensures that modules are used in different contexts.

   How test will be performed:

   (a) Inspect code structure for clear separation of concerns.

   (b) Identify modules that are used in different parts of the software.

   (c) Complete a module reuse checklist, and suggest improvements for better modularity.

4.3 Traceability Between Test Cases and Requirements

	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	NFR1	NFR2	NFR3	NFR4
TC1	X											
TC2		X	X									
TC3				X								
TC4					X							
TC5						X						
TC6							X	X				
TC7									X			
TC8										X		
TC9											X	
TC10												X

Table 1: Traceability Matrix Showing the Connections Between Requirements and System Test Cases

5 Unit Test Description

This section outlines the unit testing strategy and plans for verifying the functional components of the R-peak detection system. The tests are designed to cover both normal and edge-case behavior for each module.

5.1 Unit Testing Scope

Modules in the scope:

- General Digital Filter Module
- Specified IIR Filter Module
- Pan Tompkins Algorithm Module
- Annotated Data RMSE Module
- Rwave Detect Module
- Math Module

- Input Output Processing Module
- Error Handler Module
- Logger Module

Modules outside the scope:

- Third-party libraries used for filtering or plotting (e.g., SciPy, Matplotlib).

5.2 Tests for Functional Requirements

5.2.1 General Digital Filter Module (GDFTest)

The General Digital Filter Module handles filter coefficient regularization and signal filtering. The tests will consist of these two parts.

1. UT-1: RegularizeTest

   Type: Functional, Automatic

   Initial State: Filter uninitialized

   Input: A set of filter parameters without regularization

   Output: The regularized b and a coefficient sequences, where $a[0]$ must be 1.

   Test Case Derivation: Each element of the b and a sequences is divided by $a[0]$.

   How test will be performed: Using the GoogleTest framework.

2. UT-2: LfilterTestSuite

   Type: Functional, Automatic

   Initial State: Filter initialized

   Input: The filter parameters derived from SciPy, a signal contaminated with Gaussian noise, and the corresponding filtered signal obtained using the SciPy-derived parameters.

Output: The filtered signal computed by the General Digital Filter Module should closely match the pseudo oracle output produced by SciPy.

Test Case Derivation: The test cases are substituted into the filter difference equation for computation.

How test will be performed: Using the GoogleTest framework.

5.2.2 Specified IIR Filter Module (SIIRTest)

The Specified IIR Filter Module includes automatic derivation of parameters for various filters, so tests have been designed to verify the parameter derivation.

1. UT-3: ButterTest

   Type: Functional, Automatic

   Initial State: Filter uninitialized

   Input: Cutoff frequency and filter order

   Output: The calculated b and a coefficients for the Butterworth filter

   Test Case Derivation: The correct derivation of the b and a coefficients based on the specified cutoff frequency and filter order.

   How test will be performed: Using the GoogleTest framework.

5.2.3 Pan Tompkins Algorithm Module (AlgTest)

The Pan Tompkins Algorithm Module is responsible for detecting R-waves from preprocessed ECG signals. Tests have been designed to verify the accuracy of R-wave detection under various conditions.

1. UT-4: RWaveDetectionTest

   Type: Functional, Automatic

   Initial State: ECG signal preprocessed

   Input: Preprocessed ECG signal

   Output: The detected R-wave locations

Test Case Derivation: The test cases will verify the correctness of R-wave detection by comparing the output (detected R-waves) to known ground truth locations in the input signals.

How test will be performed: Using the GoogleTest framework.

5.2.4 Annotated Data RMSE Module (RMSETest)

The Annotated Data RMSE Module calculates the Root Mean Square Error (RMSE) between program-generated index sequences and expert-annotated sequences. Tests have been designed to verify the correctness of the RMSE calculation.

1. UT-5: calRMSESuite

   Type: Functional, Automatic

   Initial State: Input sequences (program-generated and expert-annotated) available

   Input: Program-generated index sequence and expert-annotated index sequence

   Output: Calculated RMSE value

   Test Case Derivation: The RMSE will be calculated by comparing the program-generated index sequence with the expert-annotated sequence, using the standard RMSE formula.

   How test will be performed: Using the GoogleTest framework.

5.2.5 Rwave Detect Module (RwaveDetectTest)

The Rwave Detect Module is responsible for detecting R-wave locations from raw, unfiltered ECG signals. The test ensures that the module correctly processes the signal through the full filtering chain and accurately detects the R-wave indices.

1. UT-6: RWaveDetectionFilterTest

   Type: Functional, Automatic

   Initial State: Raw, unfiltered ECG signal

Input: Unfiltered ECG signal

Output: Detected R-wave indices after complete filtering process

Test Case Derivation: The test will check whether the R-wave detection process accurately identifies the R-wave locations after the entire filtering chain is applied, comparing the detected indices to annotated data.

How test will be performed: Using the GoogleTest framework.

5.2.6 Math Module (MathTest)

The Math Module focuses on performing mathematical operations, specifically testing the squaring function. The test will ensure that the module correctly computes the square of each element in a given input sequence.

1. UT-7: calSquareSuite

   Type: Functional, Automatic

   Initial State: Sequence of floating numbers available for squaring

   Input: A sequence of floating numbers

   Output: A sequence where each element is the square of the corresponding input element

   Test Case Derivation: The test cases will verify that each element in the input sequence is squared correctly, ensuring the expected output for each input value.

   How test will be performed: Using the GoogleTest framework.

5.2.7 Input Output Processing Module (IOTest)

The Input Output Processing Module is responsible for reading input data from files and processing it into internal data structures. The test will ensure that the module correctly handles file input and converts the data into the appropriate structure.

1. UT-8: read

Type: Functional, Automatic

Initial State: Input file containing floating-point data ready for processing

Input: A file containing a sequence of floating-point numbers

Output: The corresponding data structure representing the sequence of floating-point numbers

Test Case Derivation: The test cases will verify that the module correctly reads the data from the input file and converts it into the expected internal data structure without errors.

How test will be performed: Using the GoogleTest framework.

2. UT-9: write

Type: Functional, Automatic

Initial State: Internal data structure containing a sequence of floating-point numbers

Input: The internal data structure with a sequence of floating-point numbers

Output: A file containing the corresponding sequence of floating-point numbers

Test Case Derivation: The test cases will verify that the module correctly writes the data from the internal structure into a file, ensuring the output file contains the expected sequence.

How test will be performed: Using the GoogleTest framework.

5.2.8 Error Handler Module (ErrHandlerTest)

The Error Handler Module is responsible for catching and logging exceptions. This test will verify that the module correctly handles exceptions and generates appropriate log messages.

1. UT-10: ErrTest

Type: Functional, Automatic

Initial State: Exception mechanism set up in the system

Input: An exception (e.g., a divide-by-zero error or a file-not-found error)

Output: A log entry containing the exception details

Test Case Derivation: The test cases will simulate exceptions being thrown, and the test will verify that the module correctly logs the exception message with the appropriate details.

How test will be performed: Using the GoogleTest framework.

5.2.9 Logger Module (LoggerTest)

The Error Handler Module is responsible for catching and logging exceptions. This test will verify that the module correctly handles exceptions and generates appropriate log messages.

1. UT-11: outputTest

   Type: Functional, Automatic

   Initial State: Logger initialized with a specified log file

   Input: Log message and severity level (e.g., "HIGH", "LOW")

   Output: The log message written to a log file with the correct severity level

   Test Case Derivation: The test will verify that the module correctly logs the message with the specified severity level to the output file. It will check that the log file contains the expected message in the correct format.

   How test will be performed: Using the GoogleTest framework.

References

Junwei Lin. System requirements specification. https://github.com/Lychee-acaca/CAS741/blob/main/docs/SRS/SRS.pdf, 2025.

6 Usability Survey Questions

1. On a scale of 1-5, how easy is it to use the software?

   ☐ 1 - Very difficult

   ☐ 2 - Somewhat difficult

   ☐ 3 - Neutral

   ☐ 4 - Somewhat easy

   ☐ 5 - Very easy

2. Were you able to complete your tasks efficiently?

   ☐ Yes

   ☐ No

   ☐ Somewhat

3. Which areas of the software need improvement? (Check all that apply)

   ☐ User interface design

   ☐ Navigation and workflow

   ☐ Error handling and feedback messages

   ☐ Other (please specify): _____

4. How intuitive are the software controls and menus?

   ☐ Very intuitive

   ☐ Somewhat intuitive

   ☐ Neutral

   ☐ Not very intuitive

   ☐ Not intuitive at all

5. Did you encounter any usability issues? If yes, please describe them.

   _____

6. Any additional comments or suggestions?

   _____