Project Title: System Verification and Validation Plan

Robert White

October 16, 2018

1 Revision History

Date	Version	Notes
2018-10-13	1.0	Creation of first draft for VnV plan presentation.

2 Symbols, Abbreviations and Acronyms

symbol	**description**
T	Test

[symbols, abbreviations or acronyms – you can simply reference the SRS tables, if appropriate —SS]

Contents

List of Tables

List of Figures

This document discusses the verification and validation requirements for SpecSearch. The Project Management Body of Knowledge (PMBOK) guide provides unambigious definitions for verification and validation. It defines verification as the evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. It defines validation as the assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. (PMBOK)

3 General Information

3.1 Summary

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS]
The software being tested is called SpecSearch. SpeacSearch will search for the spectrum of a particular lax equation from a lax pair that is compatbilible with solutions to the Non-Linear Schrodinger (NLS) Equation. It will also use the spectral information to determine the stability of the solutions.

3.2 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]

- Build confidence in software correctness.
- Ensure maintainability and manageability (can easily have more features/numerical algorithms incorporated).
- Satisfy the requirements of my thesis supervisor and those outlined in the SRS.
- Verify effectiveness and ease of use (usability).

3.3 References

[Reference relevant documentation. This will definitely include your SRS —SS]

- Dr. Smith's VnV Template
- PMBOK
- https://www.wqusability.com/articles/more-than-ease-of-use.html
- http://people.ucalgary.ca/ design/engg251/First
- Bernard Deconinck and Benjamin L.Segal.

4 Plan

4.1 Verification and Validation Team

The verification and validation team consists of my theis supervisor, Dr. Dmitry Pelinovsky, and I.

4.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may just be ad hoc feedback from reviewers, like your classmates, or you may have something more rigorous/systematic in mind.. —SS]

- Feedback from thesis supervisor about mathematical theory, assumptions, constraints and goals.
- Feedback from classmates and Dr. Smith about document outline, readability, clarity and requirements.

4.3 Design Verification Plan

[Plans for design verification —SS]

- Inspection of the software by my thesis supervisor.

4.4 Implementation Verification Plan

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

- A software verification checklist: Does the software allow the user to fufill their responsibilities? Does the software fufill its intended responsibilities? Can the intended user understand the software? Are the system constraints violated? Does the software achieve its intended goals? Does the solution have the required properties outlined in the SRS?
- Did the functional and nonfunctional requirement tests pass (see below)?

4.5 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]

- The stability spectrum for elliptic solutions to the focusing NLS equation. (paper)

5 System Test Description

5.1 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]

1. test-Rin1

   Control: Enough versus Not Enough Information

   Initial State: Software with no inputed data.

   Input: Varying initial data configurations (ie missing certain inputs and having all inputs)

Output: Error or pass message.

How test will be performed: All possible combinations of ommited variables (and writing variables non-numerically, such as accidently characters or symbols) in the input will be considered. A successful test in this case will be an error message upon running the software with these inputs. On the other hand we will test cases with each variable in the input having a numerical value. These numerical values will vary between negative,positive and zero values. A successful test in this case will be a pass message.

2. test-Rfind1

   Control: Full versus non-full eigenvalue array.

   Initial State: Software with prescribed input data.

   Input: The array from SpecSearch that stores the eigenvalues.

   Output: Binary output (Full or non-full array)

   How the test will be performed: A program will go through the eigenvalue array to ensure that each entry is filled with a complex number.

3. test-Rcon1

   Initial State: The software system with prescribed input.

   Input: The approximated continuous spectrum with a tag on explicitly calculated values.

   Output: Binary Variable (Connected or disconnected)

   How test will be performed: This test will check to see if there is a sufficient amount of points between the tagged portions of the spectrum. This will require a criteria for 'computer continuous'.

4. test-Rplt

   Initial State: The software system with prescribed input.

   Input: A plot of the spectrum.

   Output: A plot with regions of calculated eigenvalues emphasized/circled.

   How test will be performed: I will write a program that takes in the spectrum as input and puts explicity labels on explicitly caculated eigen values.

5. test-Rstl

   Initial State: A completed run of SpecSearch with prescribed input.

   Input: The stability results from SpecSearch.

   Output: Binary Variable (Verification of stability)

   How test will be performed: The stability results will be compared with the stability analysis in (ref1).

5.2 Tests for Nonfunctional Requirements

1. test-NFR1

   Type: Static

   Initial State: -

   Input/Condition: SpecSearch MATLAB code

   Output/Result: Pass or Fail

   How test will be performed: The software will be manually read by the developer to see if there is a more effective code structure to allow implementation of new numerical algorithms.

2. test-NFR2a

   Type: Manual

   Initial State: Software system with prescribed input.

   Input: Matrix generated from an instance of SpecSearch.

   Output: Pass or fail.

   How test will be performed: The matrix created with the prescribed inputs in SpecSearch will be extracted and its eigenvalues will be found using another software package. The eigenvalues calculated from SpecSearch and the other package will be compared. The test will be successful if SpecSearch deviates within a certain percentage of the other software's eigenvalues.

3. test-NFR2b

   Type: Manual

   Initial State: Software system with prescribed input.

   Input: Matrix derived from the time indepent lax equation.

   Output: Pass or fail.

   How test will be performed: The matrix output ought to have a specific form for each instance of SpecSearch. This test will loop through each element of the matrix to ensure it has the correct form (ie two of the quadrants are diagonal matrices and the other have a certain 'diagonal pattern')

4. test-NFR2c

   Type: Manual

   Initial State: Software system with prescribed input.

   Input: Matrix derived from the time indepent lax equation.

   Output: Pass or fail.

   How test will be performed: The output will be tested against the boundary value eigen-values derived analytically in Deconinck and Segal. The standard of necessary accuracy will be determined by my supervisor.

5.3 Traceability Between Test Cases and Requirements

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

6 Static Verification Techniques

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

- Code inspection : I will go through the code to see if each step is correct with respect to the mathematical theory. In particular I will ensure that:
  - variables are being used in the right context.
  - any discretization of functions is performed accurately about the origin. For example, equal step sizes in either directions.
  - functions from other packages are being used in the right context. For example, some packages have different standards for constants. Theoretical convention may square a constant while the code may take the squared value directly as the constant.
  - the dimensions of the vectors and matrices are appropriate. For example, multiplication of row with column versus column with row. Or that rows in matrices are seperated with semi-colons.
  - ensure a varible is not accidently overwritten or cleared.
- Code walkthrough: My supervisor and I will go through the code together to ensure that:
  - I correctly implemented the mathematical theory and numerical algorithms.
  - I made the code manageable and maintainable for future use.

References

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

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

7.2 Usability Survey Questions?

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

- How long did it take to learn how to run the software?
- Was it easy to interpret the output?
- Was this program useful for your research?
- What aspects of this software do you feel need improvement?
- Was the output recieved and processed in an adequate amount of time?
- How does this program compare with other software that finds this particular spectrum?