System Verification and Validation Report for Chess Connect

Team #4,
Alexander Van Kralingen
Arshdeep Aujla
Jonathan Cels
Joshua Chapman
Rupinder Nagra

March 8, 2023

1 Revision History

Date	Version	Notes
2023-03-04	Arshdeep Aujla	Added Template for Nonfunctional Requirements
2023-03-05	Arshdeep Aujla	Added Table for functional requirements, traceability matrix
2023-03-07	Jonathan Cels	Added functional requirement test reports
2023-03-08	Joshua Chapman	Added changes due to testing, Trace to Modules
2023-03-07	Alexander Van Kralingen	Added Arduino unit tests example
2023-03-08	Jonathan Cels	Added nonfunctional requirement test reports
2023-03-08	Arshdeep Aujla	Added reflection appendix
2023-03-08	Rupinder Nagra	Added Unit Testing section

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
FSM	Finite State Machine
TBD	To Be Determined
LCD	Liquid Crystal Display
LED	Light Emitting Diode
Engine Move	Good chess move, calculated by a chess engine
Legal Move	Chess move that is allowed according to the rules
Illegal Move	Chess move that is not allowed according to the rules
ADC	Analog to Digital Converter

Refer to SRS Section 1 for an extensive list of used symbols, abbreviations, and acronyms.

Contents

List of Tables

List of Figures

3 Functional Requirements Evaluation

Refer to the VnV Plan for descriptions of the tests derived to evaluate the functional requirements.

3.1 Game Active State

Test	Input	Expected	Actual	Notes	Result
GA-1	Draw/resign button pressed while game active.	System variable 'gameInProgress' set to false.	System variable configured correctly.		Pass
GA-2	Start game button pressed while game active.	System variable 'gameInProgress' remains true.	System variable configured correctly.		Pass
GA-3	User mode button pressed while game active.	System variable 'currMode' changed to represent the selected user mode.	User mode unchanged.	Design changed, user mode not switchable while a game is active.	Rework
GA-4	Start game button pressed while game inactive.	System variable 'gameInProgress' set to true, 'currFEN' variable is set to the starting FEN.	System variables configured correctly.		Pass
GA-5	Move made that results in stalemate or checkmate according to the rules of chess while game inactive.	System variable 'gameInProgress' set to false.	System variable configured correctly.		Pass

Table 1: Active State Functional Requirements Results

3.2 Game Inactive State

Test	Input	Expected	Actual	Notes	Result
GI-1	Start game button pressed while game inactive.	System variable 'gameInProgress' set to true.	System variable configured correctly.		Pass
GI-2	User mode button pressed while game inactive.	User mode unchanged.	System variable configured correctly.	Design changed, user mode is now switchable (only) while a game is inactive.	Rework
GI-3	Draw/resign button pressed while game inactive.	System variable 'gameInProgress' remains false.	System variable configured correctly.		Pass
GI-4	Piece moved while game inactive.	System variable 'currFEN' is unchanged.	System variable configured correctly.		Pass
GI-5	Draw/resign button pressed, or move made that results in stalemate or checkmate according to the rules of chess while game active.	Game termination and winner are displayed on LCD screen.	Display updates correctly.		Pass

Table 2: Inactive State Functional Requirements Results

3.3 Normal Mode

Test	Input	Expected	Actual	Notes	Result
NB-1	Piece moved while in normal mode.	Game state is updated to reflect piece movement.	Game state updated correctly.		Pass
NB-2	Resign button pressed while in normal mode.	System variable 'gameInProgress' set to false.	System variable configured correctly.		Pass
NB-3	Draw button pressed while in normal mode.	System variable 'gameInProgress' set to false.	System variable configured correctly.		Pass
ND-1	Game state updated while in normal mode.	Updated game state is transmitted to the web application via Bluetooth.	Game state transmitted correctly.		Pass
NA-1	Web application receives updated game state while in normal mode.	Update to game state is reflected on web application display.	Display updates correctly.		Pass
NA-2	Game termination occurs while in normal mode.	Game termination and winner are displayed on web application display.	Display updates correctly.		Pass

Table 3: Normal Mode Functional Requirements Results

3.4 Engine Mode

Test	Input	Expected	Actual	Notes	Result
EB-1	Piece moved while in engine mode.	Game state is updated to reflect piece movement.	Game state updated correctly.		Pass
EB-2	Resign button pressed while in engine mode.	System variable 'gameInProgress' set to false.	System variable configured correctly.		Pass
EB-3	Draw button pressed while in engine mode.	System variable 'gameInProgress' set to false.	System variable configured correctly.		Pass
EB-4	Engine moves transmitted from the web application to microcontroller.	Engine moves are displayed on the LCD screen.	Display updated correctly.		Pass
ED-1	Game state updated while in engine mode.	Updated game state is transmitted to the web application via Bluetooth.	Game state transmitted correctly.		Pass
ED-2	Engine moves are calculated by the web application.	Calculated engine moves are transmitted from the web application to the microcontroller via Bluetooth	Moves transmitted correctly.	Only one engine move currently calculated, more planned in future revisions.	Partial Pass

Test	Input	Expected	Actual	Notes	Result
EA-1	Web application receives updated game state while in engine mode.	Update to game state is reflected on web application display.	Display updates correctly.		Pass
EA-2	Engine moves are calculated by the web application.	Calculated engine moves are displayed on web application display.	Engine moves are not displayed.	Not implemented, planned in future revisions.	TBD
EA-3	Game termination occurs while in engine mode.	Game termination and winner are displayed on web application display.	Display updates correctly.		Pass

Table 4: Engine Mode Functional Requirements Results

3.5 Beginner Mode

Test	Input	Expected	Actual	Notes	Result
BB-1	Piece moved while in beginner mode.	Game state is updated to reflect piece movement.	Game state updated correctly.		Pass
BB-2	Piece picked up and held while in beginner mode.	LEDs on board indicate legal moves.	Correct LEDs light up.		Pass
BB-3	Piece moved such that an illegal move is made while in beginner mode.	LEDs on board indicate illegal move.	Correct LEDs light up.	Not implemented, planned in future revisions.	TBD
BB-4	Resign button is pressed while in beginner mode.	System variable 'gameInProgress' set to false.	System variable configured correctly.		Pass
BB-5	Draw button is pressed while in beginner mode.	System variable 'gameInProgress' set to false.	System variable configured correctly.		Pass
BD-1	Game state is updated while in beginner mode.	Updated game state is transmitted to the web application via Bluetooth.	Game state transmitted correctly.		Pass

Test	Input	Expected	Actual	Notes	Result
BA-1	User selects chess instructions in web application.	Web application displays detailed rules for how to play chess.	N/A	Not implemented, planned in future revisions.	TBD
BA-2	Web application receives updated game state while in beginner mode.	Update to game state is reflected on web application display.	Display updates correctly.		Pass

Table 5: Beginner Mode Functional Requirements Results

4 Nonfunctional Requirements Evaluation

Many of the nonfunctional requirements outlined in the VnV plan are left for revision 1, as much of the external testing relies on the product being more completed and accessible to a wider audience. Some external user testing was done, specifically for NFT-4 and NFT-5, and is detailed in the Performance section.

4.1 Look and Feel

Look and feel testing will be performed for revision 1, as though it is functional, this part of the product is not ready for external user testing.

4.2 Usability and Humanity

Usability testing will be performed for revision 1, as though it is functional, this part of the product is not ready for external user testing.

4.3 Performance

Test	Input	Expected	Actual	Notes	Result
NFT-4	Experiment performed as detailed below.	Average LED response time over all trials is less than 0.5 seconds.	Average response time was 0.488 seconds.	This value is close to the 0.5 second limit, and may be subject to human error. Further testing will be done with more trials for revision 1 to ensure this requirement is met.	Pass
NFT-5	Experiment performed as detailed below.	Maximum single response time of any trial is less than 1 second.	Maximum response time was 0.72 seconds.		Pass
NFT-6	N/A	N/A	N/A	Not tested, as the system is currently subject to a 2-second delay before refreshing. Will be tested for revision 1.	TBD
NFT-7	N/A	N/A	N/A	Not tested, as the system is currently subject to a 2-second delay before refreshing. Will be tested for revision 1.	TBD

Table 6: Performance Non-Functional Requirements Results

The following table holds data for NFT-4 and NFT-5. Three external users, all familiar with the rules of chess, were each asked to perform the following experiment in beginner mode 5 times:

Pick up an arbitrary piece and suspend it in the air. Wait until the board visually indicates the possible moves for the suspended piece. The time between the user picking up the piece and the board's LED indicators turning on was measured and recorded.

	User 1	User 2	User 3
Trial 1	0.49s	0.48s	0.36s
Trial 2	0.65s	0.69s	0.27s
Trial 3	0.53s	0.33s	0.52s
Trial 4	0.72s	0.34s	0.49s
Trial 5	0.42s	0.51s	0.52s
Total Average	**0.488s**		

Table 7: Experimental Results of Performance Testing

The following scatter plot shows the experimental results visually. The average, shown in green, falls just below the required 0.5 second threshold, and all points are well below the maximum acceptable response time of 1 second.

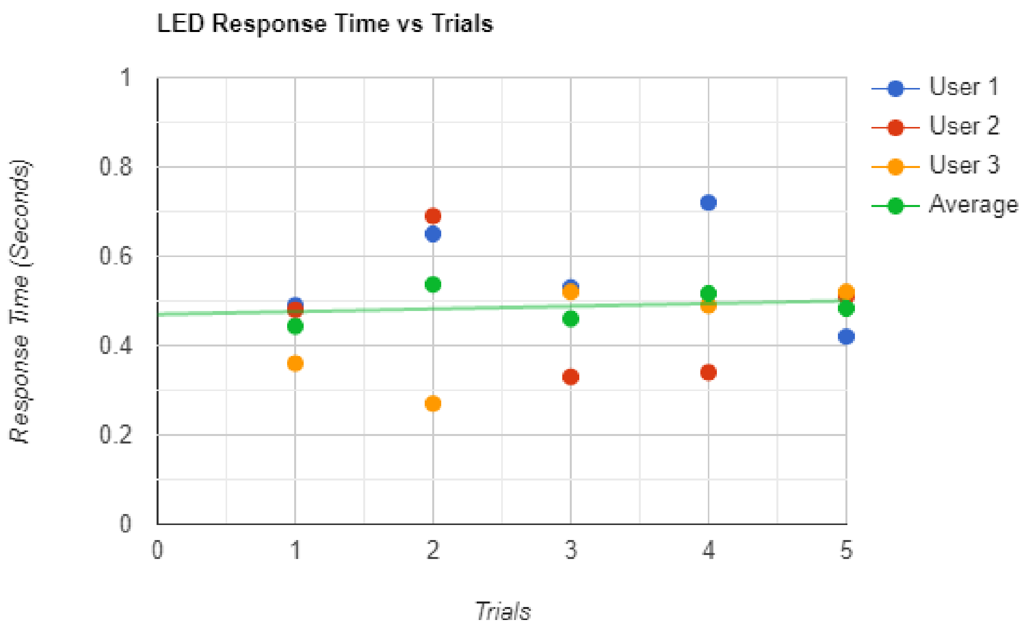

Figure 1: Experimental Results of Performance Testing

4.4 Health and Safety

Test	Input	Expected	Actual	Notes	Result
NFT-8	10 sample wires were chosen while the product was running. Their current and voltage were measured, and power was calculated using those measurements.	All power measurements below safe limits.	All power measurements below safe limits.	Measurements far below any safety thresholds.	Pass

Table 8: Health and Safety Non-Functional Requirements Results

The following table holds data for NFT-8.

Wire #	Wire Description	Gauge	Current (Amps)	Voltage (Volts)	Calculated Power (Watts)	Maximum power (Watts)
1	Hall sensor 1 with black piece	20	0.04	1.6	0.064	7.5
2	Hall sensor 32 with no piece	20	0.03	1	0.03	7.5
3	Hall sensor 64 with white piece	20	0.02	0.6	0.012	7.5
4	Arduino power line	10	0.5	5	2.5	75
5	ADC 1	20	0.01	5	0.05	7.5
6	ADC 4	20	0.02	5	0.1	7.5
7	ADC clock	20	0.05	5	0.25	7.5
8	Wall power supply (L)	8	0.75	110	47.631	120
9	Wall power supply (G)	8	0.01	0	0	120
10	Bluetooth RX	20	0.02	5	0.1	7.5

Table 9: Experimental Results of Health and Safety Testing

4.5 Precision and Accuracy

Precision and accuracy testing will be performed for revision 1, as though it is functional, this part of the product is not ready for external user testing.

4.6 Capacity

Test	Input	Expected	Actual	Notes	Result
NFT-10	A relatively complex chess FEN position was transmitted via Bluetooth to the web application while in engine mode.	The measured memory usage of the web application is less than 1GB.	The maximum measured memory value was 1187.4 MB, as measured by Windows task manager.	Memory usage will increase in future revisions, as more engine moves will be calculated at a higher depth.	Pass

Table 10: Capacity Non-Functional Requirements Results

4.7 Security

Test	Input	Expected	Actual	Notes	Result
NFT-11	Bluetooth connection to web application severed.	Web application alert that the connection has been lost.	N/A	Not implemented, planned in future revisions.	TBD
NFT-12	Power connection to system is severed, and then restored after a short time frame.	The game state data is stored in local memory and is unchanged after power is restored.	Game state data is properly stored.		Pass

Table 11: Security Non-Functional Requirements Results

5 Unit Testing

Unit testing is a crucial aspect of software development that involves testing individual units or components of a software application in isolation from the rest of the system. It provides us with a way to ensure that each unit of code is functioning as intended and that it integrates seamlessly with other parts of the software application. By identifying defects and bugs early in the development cycle, it helped reduce the overall time spent on software development while also improving the quality and reliability of our final product. Additionally, our testing served as a form of documentation, helping us understand how different components of the system interact and ensuring that future modifications do not break existing functionality. We implemented unit testing to ensure our project was robust, maintainable, and of high quality. As mentioned in the VnV Plan, the React Testing Library was used for the Javascript unit tests. Additionally, with the inclusion of many hardware related components in our project, most of our testing is done manually, leaving very few tests that require being automated.

Creating unit tests for the Embedded software required several of Arduino's built in functionality to be simulated. This included serial communication functions, pin setup (input or output), reading from and writing to pins, and time delays. Additionally, binary values neded to be setup to simulate a sequence of events such as values recorded from a hall sensor, or LEDs turning on or off. All of this is handled in the MockArduinoController.cpp file, which holds the SerialStream and PinSimulation classes, as well as several functions for interacting with the hardware.

Rather than unit testing every function in normal operation, individual functions were tested to ensure correct outputs from simulated inputs. Integration with the system was completed physically with the Arduino exectuing the program. An example of the test for hardware reading is given below.

```
void testReadPiece()
{
   setupBoard();

   // Simulate picking a piece
   organizedHallValues[0][1] = randHall(NO_COLOUR);
   mapHallValuesToSensors();
   PinSim.reWritePin(hallRx[0]);
   writeAdcRow(hallRx[0], rawHallValues[0]);

   // NO_PIECE, NO_COLOUR
   Square expectedSquare = Square(0,1);

   // Checkpick() function inside Arduino's loop should catch this,
   // updating the pieces on the board
   loopArduino();

   // Make sure the state changes to PIECE_LIFTED ('l')
   check(assert_equal('l', gameState), __FUNCTION__, __LINE__);

   // Make sure the square in the board array is updated successfully
   check(assert_equal(expectedSquare, currentBoard[0][1]), __FUNCTION__, __LINE__)
;
}
```

The rest of the tests follow a similar format of setting up the initial state, simulating an input and comparing the expected output. Several similar tests were performed with different piece types, positions and piece colours to cover edge cases. Only one test of each type in a group testing the same function with different inputs and outputs has been included in the table below.

Test	Input	Expected	Actual	Result
testReadPiece	random Hall sensor value between 140 and 310 (no piece reading) corresponding to square B1	Square B1 resets piece value to NO_PIECE	currentBoard[0][1] holds NO_PIECE, NO_COLOUR	pass
testHighlightPawnValidMoves	Pawn lifted from starting position (A2)	A3, A4 light up	A3, A4 pins read output HIGH	pass
testHighlightKnightValidMoves	Knight lifted from starting position (B1)	A3, C3 light up	pins read output HIGH	pass
testHighlightBishopValidMoves	Bishop lifted from C1 after D2 Pawn to D3, and G7 Pawn to G5	C2, E3, F4, G5 light up	C2, E3, F4, G5 pins read output HIGH	pass
testHighlightRookValidMoves	Rook lifted from	light up	pins read output HIGH	pass
testHighlightQueenValidMoves	Queen lifted from	light up	pins read output HIGH	pass
testHighlightKingValidMoves	King lifted from	light up	pins read output HIGH	pass
testPieceToChar	Piece(QUEEN, BLACK)	'q'	'q'	pass
testValidateFENString	King lifted from	rnbqkbnr/pppp1ppp/ 4p3/8/4P3/5N2/ PPPP1PPP/RNBQKB1R w KQkq - 1 4	rnbqkbnr/pppp1ppp/ 4p3/8/4P3/5N2/ PPPP1PPP/RNBQKB1R w KQkq - 1 4	pass
testGameStartValid	black piece on row 1 while all other pieces are white	false	false	pass
inStalemate	rnbqkbnr/pppppppp/ 8/8/8/8/PPPPPPPP/ RNBQKBNR w KQkq - 0 1	false	false	pass
inCheckmate	rnb1kbnr/pppp1ppp/ 8/4p3/5PPq/8/ PPPPP2P/ RNBQKBNR w KQkq - 1 3	true	true	pass

Table 12: Unit Test functions

6 Changes Due to Testing

The results detailed in the above tests prompted a number of design changes in the hardware, embedded and web application systems. Outlined below are the changes made as result.

6.1 Embedded Testing

Testing the finite state machine located in the electrical system resulted in incorrect functionality according to the design. A finite state machine was designed to control the functionality of the electrical system dictated by the user. It allows users to change modes and have control over the performance of the board. The initial implementation of the FSM utilized a slow clock speed and certain inputs were missed as a result. The new solution is to increase the clock speed of the FSM and include buffers within the states to increase the robustness of the system.

Easy mode provides the functionality of possible moves for pieces on the board. When a piece is lifted, the available squares are highlighted. During testing of the function, there were a number of edge cases that were not covered such as castling, en passant and check scenarios. The proposed solution was to increase the robustness of the algorithm and maintain logs of the game to account for these edge cases.

6.2 Hardware Testing

Testing the hall sensor piece detection circuits at scale uncovered issues with sensitivity of the sensors. At scales of four or more sensors in series, the sensitivity of the reading was too large. Initial solutions included improvement of the hardware. This included improvement of the power supply, grounding and sensors themselves. These solutions did not solve the issue. As a result, the requirements of the sensors have been changed to sense north and south poles exclusively. The functionality of individual piece detection was not feasible.

Assembly of the revision one chess board displayed issues with robustness of the connections between mechanical and electrical assemblies. Temporary connection was established with electrical and duct tape. The electrical system suspended below the mechanical system. After testing of the board, connection points began to fail and separation occured. The solution for revision two is assembly of the mechanical and electrical systems simultaneously. The two systems are assembled together and the robustness of the system is increased.

6.3 Web Application Testing

Integration of the web application with the embedded system revealed an error with the implementation of polling in the web application. Polling required handling of constant data transmission between systems. Issues were clear with required data and processing dedicated to this communicatin. One of the proposed solutions is the implementation of a refresh to erase the stored data. This eliminates the data size issue. A second solution is the introduction of sockets which eliminates polling entirely and solves the data size and processing issues.

- Initially, the game was only allowed to start while white and black were on specific sides. Code changed to support either side as starting position.

7 Automated Testing

7.1 C++

Since Arduino files are .ino and cannot be build using a g++ compiler, a build script has been created to convert the file to a .cpp filetype, compile it with g++, execute it and watch the result. The error code is the number of errors which used by GitHub Actions to determine the success of all of the unit tests. Errors are written to a log file and recorded for future reference. Unit tests are run in this method to verify changes to the code. Tests must all pass before merging into main. Please refer to Unit Testing for details on the tests ran in build script.

7.2 React Testing Library

The React Testing Library was used for testing our React components. Its function was to assist writing tests that simulate user interactions with their application, allowing us to ensure that the components are working as intended. The library provides a set of utilities that make it easy to test React components by interacting with them as a user would, rather than relying on implementation details. This means that tests written with this library are more resilient to changes in the underlying codebase, and provide better coverage of the user experience. Overall, the React Testing Library was an important tool to ensure the quality and reliability of our front-end application.

8 Trace to Requirements & Modules

Test	Requirement	Module
GA-1	GA1	M2
GA-2	GA2	M2
GA-3	GA3	M2
GA-4	GA6	M2
GA-5	GA7	M2, M5, M11
GI-1	GI1	M2
GI-2	GI2	M2, M11
GI-3	GI3	M2
GI-4	GI4	M2, M4
GI-5	GI5, GI6	M2, M5, M11
NB-1	NB1	M2, M11
NB-2	NB2	M2, M11
NB-3	NB3	M2, M11
ND-1	ND1	M2, M11
NA-1	NA1, NA2	M2, M6, M8, M11
NA-2	NA3	M2, M5, M11
EB-1	EB1	M2, M5, M11
EB-2	EB2	M2, M11
EB-3	EB3	M2, M11
EB-4	EB4	M10, M14
ED-1	ED1	M5, M6, M8, M12
ED-2	ED2	M14
EA-1	EA1, EA2	M5, M6, M8, M11
EA-2	EA3, EA4, EA5	M14
EA-3	EA6	M5, M6, M8, M9
BB-1	BB1	M4, M5, M11
BB-2	BB2	M4, M5, M11

BB-3	BB3	M4, M5, M11
BB-4	BB4	M2, M5, M11
BB-5	BB5	M2, M5, M11
BD-1	BD1	M2, M5, M11
BA-1	BA1	M10, M11
BA-2	BA2	M2, M8, M9
NFT1	LF3	
NFT2	UH5	
NFT3	UH6	
NFT4	PR1	
NFT5	PR2	
NFT6	PR3	
NFT7	PR4	
NFT8	PR6	
NFT9	PR7	
NFT10	PR10	
NFT11	SR4	
NFT12	SR3	

Table 13: Requirements Traceability Matrix

9 Trace to Modules

10 Code Coverage Metrics

A Reflection Appendix

The majority of the tests planned out in the VnV Plan were carried out as expected and reported in the VnV Report. Some differences in the tests were due to a rework of the requirements, and modifying what is in scope for the first iteration of the project.

Two tests were not carried out as planned because the requirements changed since the creation of the VnV Plan. These tests were GA-3 and GI-2. These tests were both related to the changing the game state. Initially, functional requirements outlined that the user should be able to change the games mode regardless if they are currently in the active state or inactive state. On a second consideration, it was decided that the mode should only be able to be changed if the user is in the inactive mode. Therefore the tests GA-3 and GI-1 were no long applicable and not reported in this report.

A few tests for functional and non-functional requirements were not completed due to the scope of the project changing. The requirements changed because they were decided that they weren't mandatory for a working product. It was also not feasible for these requirements to be met during the timeline of the course. These tests that were descoped were ED-2, EA-2, BB-3, BA-1, NFT-6, NFT-7, and NFT-11.

In a future VnV planning, emphasis will need to be made to ensure that the requirements are solidified before the tests are created, and that all of the requirements need to be feasible to be met during the project's timeline.

References

Author Author. System requirements specification. https://github.com/..., 2019.

CSA. *Canadian Electrical Code*. CSA Group, 2021.

FEN. Fen (forsyth-edwards notation) - chess terms. https://www.chess.com/terms/fen-chess. Accessed: 2023-01-18.

Interal Chess Federation FIDE. Fide laws of chess. https://www.fide.com/FIDE/handbook/LawsOfChess.pdf, 2018.

Canadian Centre for Occupational Health and Safety. Hazard and risk: Osh answers. https://www.ccohs.ca/oshanswers/hsprograms/hazard_risk.html, 2022. Accessed: 2022-10-05.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

List of Chess Variants. List of chess variants, Sep 2022. URL https://en.wikipedia.org/wiki/List_of_chess_variants.

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

David L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, 12(2):251–257, February 1986.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.

James Robertson and Suzanne Robertson. *Volere Requirements Specification Template*. Atlantic Systems Guild Limited, 16 edition, 2012.

W. Spencer Smith. Systematic development of requirements documentation for general purpose scientific computing software. In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006*, pages 209–218, Minneapolis / St. Paul, Minnesota, 2006. URL http://www.ifi.unizh.ch/req/events/RE06/.

W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ȧgerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, 13(1): 83–107, February 2007.

W. Spencer Smith, John McCutchan, and Jacques Carette. Commonality analysis of families of physical models for use in scientific computing. In *Proceedings of the First International Workshop on Software Engineering for Computational Science* *and Engineering (SECSE 2008)*, Leipzig, Germany, May 2008. In conjunction with the 30th International Conference on Software Engineering (ICSE). URL http://www.cse.msstate.edu/~SECSE08/schedule.htm. 8 pp.

W. Spencer Smith, John McCutchan, and Jacques Carette. Commonality analysis for a family of material models. Technical Report CAS-17-01-SS, McMaster University, Department of Computing and Software, 2017.

Bill Wall. History of chess sets and symbols. https://www.chesscentral.com/pages/chess-sets-pieces-boards/a-history-of-chess-pieces-and-chess-sets.html, 2003. Accessed: 2022-10-04.

WCAG. Web content accessibility guidelines 2 overview. https://www.w3.org/WAI/standards-guidelines/wcag/#intro, 2018. Accessed: 2022-10-05.