System Verification and Validation Plan for Stoichiometry Mass-Mass Program

Deemah Alomair

December 23, 2019

1 Revision History

Date	Version	Notes
28/10/2019	1.0	First version of document
23/12/2019	1.0	Second version of document

Contents

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
SRS	Software Requirements Specification
MIS	Module Interface Specification
MG	Module Guide
SMMP	Stoichiometry Mass-Mass Program
R	Functional requirement
NF	Non-functional requirement

3 General Information

This document is to build verification and validation plan for Stoichiometry Mass-Mass Program and provide description about the testing that will be carried out on the system. The main goal of this document is to check whether SMMP meets the SRS and fulfill its intended purpose. This document will be used as the reference and guidance for testing SMMP.

3.1 Summary

Stoichiometry Mass-Mass Program is a software that convert unbalanced chemical reaction to balanced one. Then do some necessary calculations to get the mass of one of the reactants involved in that chemical reaction.

3.2 Objectives

The objectives of the verification and validation plan is to ensure that SMMP is Reliable. Building the confidence of the correctness of the software. In addition, enhance the maintainability for the traceability of the potential changes. Moreover, ensure usability of the SMMP to novel users.

3.3 Relevant Documentation

Relevant Documents that need to be visited while reading this document include the following:
SRS, which can be found SRS [1]
System Design that includes both MIS and MG, which can be found Design [2]
Unit Verification and Validation Plan, which can be found UnitVnVPlan [3]
System and Unit Verification and Validation Report , which can be found VnVReport [4]

4 Plan

This section details the plan to be followed when testing the software, including those involved in the testing, the testing approach, and the verification tools which will be used.

4.1 Verification and Validation Team

The test team includes the following members:

- Main contributor: Deemah Alomair.
- Primary reviewers: Dr. Spencer Smith, Sharon Wu.
- Secondary reviewers: Bo Cao, Ao Dong, Peter Michalski.

4.2 SRS Verification Plan

- Feedback: Classmates, including all primary and secondary reviewers listed above, will provide feedback on GitHub by submitting an issue in the issue tracker. They will read the document and provide a guidelines on how to enhance the document.
- Initial Review: The document will be manually reviewed by Dr. Spencer Smith using the SRS checklist upon its initial creation, as found in the CAS741 GitLab repository

4.3 Design Verification Plan

- Feedback: Classmates, including all primary and secondary reviewers listed above, will provide feedback on GitHub by submitting an issue in the issue tracker. They will read the document and provide a guidelines on how to enhance the document.
- Initial Review: The documents will be manually reviewed by Dr. Spencer Smith using the MG/MIS checklists upon its initial creation, as found in the CAS741 GitLab repository

4.4 Implementation Verification Plan

Implementation Verification Plan will include the followings:

- Manual testing.
- Automatic testing that includes unit testing for each individual function and coverage testing using "unittest package" and "coverage package" that work with PyCharm environment.

- Pylint Linter will be considered throughout the implementation of SMMP.
- Parallel testing.

4.5 Software Validation Plan

The main goal of this project is to build the product right and get a correct calculated output. As the developer and the user is the same person software validation plan is not applicable.

5 System Test Description

This section lists the system tests to be performed to verify whether or not the program fulfills the functional requirements and to test how well it meets the non-functional requirements. Note that some of the tests for the functional requirements are unit tests.

5.1 Tests for Functional Requirements

5.1.1 Input Tests

Input conversion tests

1. MassInputTest-id1

   Control: Functional, dynamic, automatic.

   Initial State: Not Applicable.

   Input:

input	response
Mass $\notin$ number	error massage
Mass ≤ 0	error massage
Mass > 0	accepted

Table 1: Input possible value's level

Output: error massage indicating that mass value is not a number or mass value ≤ 0.

Test Case Derivation: the expected value is a positive integer number.

How test will be performed:

- the system will get the value.
- process the value to ensure it's a number. if not error massage will be shown.
- if its a number it will test if its greater than 0 or not. if not error massage will be shown.

2. ReactionInputTest-id2

Control: Functional, dynamic, automatic.

Initial State: Not Applicable

Input: 1. reaction element = selection from drop down menu.
2. atom value:

atom value	response
atom value $\notin$ number	error massage
atom value ≤ 0	error massage
atom value > 0	accepted

Table 2: Input possible value's level

Output: error massage indicating that atom value is not a number or atom value ≤ 0.

Test Case Derivation: the expected value is chemical element selected from a menu and atom value belongs to positive integer number.

How test will be performed:

- the system will get the values.
- the system will ensure user has already selected an element.
- process the atom value to ensure it's a number. if not error massage will be shown.

- if its a number it will test if its greater than 0 or not. if not error massage will be shown.

5.1.2 Output Tests

Output constrains tests throughout the rest of this document we will use one chemical reaction as an example. in real testing we can use different chemical reactions as long as it consists of two reactants, each consists of two element at most. reaction used : "Fe_2O_3 + C → Fe + CO_2".

1. ReactionOutputTest-id3

   Control: Functional, dynamic, automatic.

   Initial State: N.A

   Input: chemical reaction : "Fe_2O_3 + C → Fe + CO_2" two reactants and two products each with maximum two elements.

   Output: balance reaction : 2Fe_2O_3 + 3C → 4Fe + 3CO_2

   Test Case Derivation: the expected value is balance chemical reaction by adding the appropriate coefficients in front of each reactant and product.

   How test will be performed:

   - the system will get chemical reaction elements and atoms value.
   - if the entered reaction is not balance then the system will output the balanced reaction instead.
   - the balance result will be compared to [5] as parallel testing.

2. MassOutputTest-id4

   Control: Functional, dynamic, automatic.

   Initial State: Not Applicable

   Input:
   chemical reaction : "Fe_2O_3 + C → Fe + CO_2"
   mass of known reactant : 2
   name of known reactant: "Fe_2O_3"

Output: C mass = 0.1 g.

Test Case Derivation: the expected value is a positive integer number.

How test will be performed: the system will calculate mass value using the calculation method and ensure it's greater than 0.

5.1.3 Calculation Method Tests

1. CheckBalanceTest-id5

   Control: Functional, dynamic, automatic.

   Initial State: N.A

   Input:
   chemical reaction : "Fe_2O_3 + C $\rightarrow$ Fe + CO_2" two reactants and two products each with maximum two elements.

   Output: "not balance !"

   Test Case Derivation: the expected value is balance when chemical reaction is balance and not balance otherwise.

   How test will be performed:

   - the system will get chemical reaction elements and atoms value.
   - the system will calculate the total atom value for each element in each side of the reaction.
   - the system will compare the total of each element in both side and repeat this process for each element involved in the reaction.
   - if total atoms value in reactant side = total atoms value in product side then the chemical reaction considered balance .
   - "balance" will be displayed to the user using GUI widget.
   - if not balance then balance form will be calculated and "not balance !" will be displayed to the user using GUI widget with the new form of the reaction.

   more tests will be performed on balancing chemical reaction as unit testing in unit testing plan. [3]

2. MolecularWeightCalculationTest-id6

   Control: Functional, dynamic, automatic.

   Initial State: Not Applicable

   Input: name of known reactant: "Fe_2O_3"

   Output: Molecular weight = 55.845*2 + 15.9994*3 = 159.6882 g/mol.

   Test Case Derivation: the expected value is a molecular weight which is equal to the sum of atomic mass multiplied by the total atom value for each element building up the reactant.

   How test will be performed:

   - the system will get the atomic mass for each element of the reactant.
   - multiply each atomic mass by the total atoms of the element
   - add the result of each element together.
   - save the final result to be used in other function.

3. Mole1CalculationTest-id7

   Control: Functional, dynamic, automatic.

   Initial State: Not Applicable

   Input: Mass = 2 g , molecular weight = 159.6882 g/mol

   Output: Mole = 2 / 159.6882 = 0.0125 mol.

   Test Case Derivation: the expected value is mole for reactant with known mass which is equal to given mass value divided by calculated molecular weight.

   How test will be performed:

   - the system will get mass and molecular weight for the reactant.
   - divide mass by molecular weight
   - save the final result to be used in other function.

4. MoleRatioCalculationTest-id8

   Control: Functional, dynamic, automatic.

   Initial State: Not Applicable

   Input: coefficient1 = 2 , coefficient2 = 3

   Output: coefficient2/coefficient1 = 3/2 = 1.5

   Test Case Derivation: the expected value is Mole Ratio which is equal to derived coefficient value of reactant with unknown mass divided by coefficient value of reactant with known mass.

   How test will be performed:

   - the system will get coefficient2 and coefficient1 derived from balancing the reaction..
   - divide coefficient2 by coefficient1
   - save the final result to be used in other function.

5. Mole2CalculationTest-id9

   Control: Functional, dynamic, automatic.

   Initial State: Not Applicable

   Input: Mole1 = 0.0125 , MoleRatio = 1.5

   Output: Mole1/MoleRatio = 0.0125/1.5 = 0.008

   Test Case Derivation: the expected value is Mole2 (mole for reactant with unknown mass) which is equal to Mole1 value divided by MoleRatio.

   How test will be performed:

   - the system will get Mole1 and MoleRatio derived from other functions
   - divide Mole1 by MoleRatio
   - save the final result to be used in other function.

6. MassCalculationTest-id10

   Control: Functional, dynamic, automatic.

   Initial State: Not Applicable

   Input: Mole2 = 0.0062 , Molecular weight = 12.0107

   Output: Mole2 * Molecular weight = 0.008 * 12.0107 = 0.1 g.

   Test Case Derivation: the expected value is final Mass which is equal to Mole2 value multiplied by reactant Molecular weight.

   How test will be performed:

   - the system will get Mole2 and Molecular weight derived from other functions
   - multiply Mole2 by Molecular weight
   - send the final result to GUI to be displayed.

5.2 Tests for Nonfunctional Requirements

1. UsabilityTesting-id11

   Type: Nonfunctional,Dynamic, Manual.

   Initial State: Not Applicable.

   Input/Condition: chemical reaction , mass of one reactant

   Output/Result: Balance reaction , Second reactant mass.

   How test will be performed: user need to complete small survey. see appendix A

2. Reliability-id12

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State: Not Applicable.

   Input: testing 25 unbalanced chemical reaction with given mass for one reactant.

Output: the percentage of correct answer. (aim for 100 % correct answer)

How test will be performed: the developer manually will test the overall system using 25 different examples of unbalanced chemical reaction consists of two reactant each consist of at most two elements with given mass of one of reactant, and get the percentage of correct answer. correct answer will include: (right balancing + correct mass value)

5.3 Traceability Between Test Cases and Requirements

A trace between system tests and requirements is provided in Table 3.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
R1	X	X										
R2			X	X								
R3					X	X	X	X	X	X		
R4	X	X	X	X	X	X	X	X	X	X		
NF1											X	
NF2	X	X	X	X	X	X	X	X	X	X	X	X
NF3												X

Table 3: Traceability Matrix Showing the Connections Between Requirements and system tests

References

[1] https://github.com/deemaalomair1/CAS741project/tree/master/docs/SRS

[2] https://github.com/deemaalomair1/CAS741project/tree/master/docs/Design

[3] https://github.com/deemaalomair1/CAS741project/blob/master/docs/VnVPlan/UnitVnVPlan/UnitVnVPlan.pdf

[4] https://github.com/deemaalomair1/CAS741project/tree/master/docs/VnVReport

[5] http://www.endmemo.com/chem/balancer.php

6 Appendix

Survey to measure the Satisfaction of user on SMMP system:

	1	2	3	4	5
The ability to understand the propose of the system					
The ability to fill the input easily					
The system is easy to use					
The output is clear					
The GUI is visible and consistent					
Flexibility and efficiency of use					
I would like to use the system again					

- 1 is the lowest Satisfaction level and 5 is the highest.

Figure 1: user satisfaction survey.