# CAS 741: Test Plan

## Aqueous Speciation Diagram Generator

Steven Palmer
palmes4

October 25, 2017

# Revision History

Table 1: Revision History

| **Date** | **Developer(s)** | **Change** |
|---|---|---|
| 10.13.2017 | S. Palmer | First revision of document |

# Symbols, Abbreviations and Acronyms

| symbol | description |
|---|---|
| SpecGen | The Aqueous Speciation Diagram Generator program |
| SRS | Software Requirements Specification |
| T | Test |

# Contents

# 1 General Information

This document provides a detailed description of the testing that will be carried out on the Aqueous Speciation Diagram Generator program (herein referred to as SpecGen).

## 1.1 Purpose

The purpose of this document is to provide a comprehensive plan for testing the SpecGen software against the requirements described in the SpecGen SRS.

## 1.2 Scope

The test plan is narrowed to the following scope:

- The tests outlined in this document are limited to the verification of the requirements given in the SpecGen SRS. The validation of the requirements will be carried out via correspondance with Dr. Scott Smith.
- The tests outlined in this document are limited to dynamic tests only. Due to the small size and low complexity of the SpecGen program, no formal static testing (code walkthroughs, code inspections, etc.) will be carried out.
- The SpecGen software will be written in Python. The testing of implementations in other languages will not be considered in this document.

# 2 Plan

## 2.1 Software Description

Chemical speciation refers to the stable (equilibrium) distribution of chemical species in a given chemical system. Speciation diagrams, which plot species concentrations against an independently varied parameter of the system, are useful tools for displaying speciation data in a concise and easy to use format.

SpecGen will produce a speciation diagram given a set of chemical reactions, equilibrium constants, and element totals that define a chemical system. SpecGen will be specific to speciation of ions in aqueous systems under varying pH, which is of particular importance in the fields of aqueous process engineering and hydrometallurgy. The diagram generated by SpecGen will plot speciation of all aqueous species (excluding $H^+$ and $OH^-$) across the pH range 0 to 14.

## 2.2 Test Team

The test team includes the following members:

- Steven Palmer

## 2.3 Automated Testing Approach

The automated testing for SpecGen will be carried out using a set of unit and integrations tests. A test coverage analysis of these tests will be carried out to ensure that testing is as complete as possible. The target for this analysis is 100% statement coverage.

Regression testing will be used during the implementation stage and for any future changes. Since SpecGen is small in scope and will be implemented by a single developer, other forms of automated testing, such as continuous integration testing, will not be considered.

## 2.4 Verification Tools

The following tools will be used to facilitate testing:

1. **PyUnit** (a unit testing framework for Python) will be used to write and run unit tests
2. **Coverage.py** will be used to assess test coverage
3. **make** will be used to automate the building and execution of the test program

## 2.5 Non-Testing Based Verification

N/A

# 3 System Test Description

[Intial states, inputs, outputs will be formalized as implementation proceeds. —SP]

## 3.1 Tests for Functional Requirements

### 3.1.1 Input Testing

**T1: Equation Parser, Integration**

**Type:** Functional, Automatic, Integration

**Initial State:** ?

**Input:** Input file with general chemical reaction

**Output:** Data structure of reaction equation components

**How test will be performed:** Automated integration test

**T2: Input Data Structure, Integration**

**Type:** Functional, Automatic, Integration

**Initial State:** ?

**Input:** Input file

**Output:** Data structure works – probably several tests

**How test will be performed:** Automated integration test

### 3.1.2 Derived Input Testing

**Derived Species Tests**

**T3: Species List from Equation**

**Type:** Functional, Automatic, Unit

**Initial State:** ?

**Input:** String representation of chemical reaction

**Output:** List of species contained in reaction

**How test will be performed:** Automated unit test

**T4: Species List from Equation, Integration**

**Type:** Functional, Automatic, Integration

**Initial State:** ?

**Input:** Input file

**Output:** List of species contained in reaction

**How test will be performed:** Automated integration test

### 3.1.3 Input Constraint Testing

**Element Concentration Tests**

**T5: Negative Element Total Concentration**

**Type:** Functional, Automatic, Unit

**Initial State:** ?

**Input:** Negative concentration

**Output:** Error

**How test will be performed:** Automated unit test

**T6: Zero Element Total Concentration**

**Type:** Functional, Automatic, Unit

**Initial State:** ?

**Input:** Zero concentration

**Output:** Error

**How test will be performed:** Automated unit test

**T7: Positive Element Total Concentration**

**Type:** Functional, Automatic, Unit

**Initial State:** ?

**Input:** Positive concentration

**Output:** Positive concentration

**How test will be performed:** Automated unit test

**T8: Element Total Concentration, Integration**

**Type:** Functional, Automatic, Integration

**Initial State:** ?

**Input:** Input file

**Output:** Concentration

**How test will be performed:** Automated integration test

### 3.1.4 Input Conversion Testing

[SpecGen will convert inputs into a set of non-linear equations to be solved by a non-linear solver. This will involve some sort of module/code, but none of this corresponds to an actual requirement. Should I have explicit tests for this even though it doesn't map to a requirement? Even without tests here, this will still end up being testing in the integration tests for calculations and output. —SP]

### 3.1.5 Calculation Testing

**Non-Linear Equation Solver Tests**

#### T9: Non-Linear Equation System Solution

**Type:** Functional, Automatic, Unit

**Initial State:** ?

**Input:** System of non-linear equations

**Output:** Solution that satisfies original system of equations

**How test will be performed:** Automated unit test

#### T10: Non-Linear Equation System Solution, Integration

**Type:** Functional, Automatic, Integration

**Initial State:** ?

**Input:** Input file

**Output:** Solution that satisfies original system of equations

**How test will be performed:** Automated integration test

### 3.1.6 Output Testing

**Speciation Diagram Generation Tests**

#### T11: Speciation Diagram

**Type:** Functional, Manual, Unit

**Initial State:** ?

**Input:** Output specifications

**Output:** Speciation diagram

**How test will be performed:** The generated speciation diagram will be manually reviewed to ensure that it matches the specified output.

**T12: Speciation Diagram, Integration**

**Type:** Functional, Manual, Integration

**Initial State:** ?

**Input:** Input file

**Output:** Speciation diagram

**How test will be performed:** The generated speciation diagram will be manually reviewed to ensure that it matches the diagram expected by the input specifications.

## 3.2 Tests for Nonfunctional Requirements

### 3.2.1 Output Readability Tests

**T13: Readability of generated speciation diagram**

**Type:** Nonfunctional, Manual

**Initial State:** ?

**Input:** Input file

**Output:** Speciation diagram

**How test will be performed:** This is a qualitative test to ensure that the diagrams generated by SpecGen are readable (axis labels visible, curves distinguishable from eachother, legend/labelling of curves, etc.).

# 4 Traceability Between Test Cases and Requirements

A trace between tests and requirements is provided in Table 2.

Table 2: Requirements Traceability

| Requirement | Test(s) |
|---|---|
| R1 | T14, T15, T16, T1, T2 |
| R2 | T3, T4 |
| R3 | T5, T6, T7, T8 |
| R4 | T9, T10 |
| R5 | T11, T12 |

# 5 Unit Testing Plan

## 5.1 Input Testing

### T14: Equation Parser, General

**Type:** Functional, Automatic, Unit

**Initial State:** ?

**Input:** String representation of chemical reaction

**Output:** Data structure of reaction equation components

**How test will be performed:** Automated unit test

### T15: Equation Parser, Edge Case 1

**Type:** Functional, Automatic, Unit

**Initial State:** ?

**Input:** String representation of chemical equation that hits edges cases (want full coverage)

**Output:** Data structure of reaction equation components

**How test will be performed:** Automated unit test

[Edge cases will be added during implementation (after data structures and exact format of input is decided). For simplicity going forward, edge cases are ignored (since at this point I don't know the number/nature of the edge cases). They will be added as the implementation is developed. —SP]

**T16: Input Data Structure**

**Type:** Functional, Automatic, Unit

**Initial State:** ?

**Input:** Input data (?)

**Output:** Data structure works – probably several tests

**How test will be performed:** Automated unit test