Test Plan

Instructions: Pick one testing strategy from each of the following two categories that your team will employ as you implement your software. Note that the suggested methods are in bold and underlined. If your team chooses either Load testing or Integration testing, meet with me to discuss appropriate test plan format in places where it might differ from that below.

1. **Unit testing** OR Code review
2. **Functional testing** OR Load testing OR Integration testing

Document your test plan using the template provided below.

# The Skyentists: GPP and RECO Calibration Software

Mark Matas, Lucas Hamilton, Jake Pennington, Max Thibeau

# Date:

04/17/20

# OVERVIEW

## Purpose of the Test Plan Document

The Test Plan defines the testing strategies and approaches for the completion of testing on the calibration software. This document also includes the results of the testing scenarios. The intended viewers of this test plan are the developers of the calibration software and any future developers and testers, although this document can be shared with any user/client/stakeholder whose input is needed and valued.

## Testing Strategies

The team’s overall approach to testing includes both the unit and functional testing strategies. Unit testing is an approach that includes testing an individual object or function with a single test. Testing in this way allows the tester to set a specific input with an expected value, as unit testing tests a specific behavior, function, or object in the code. This approach includes the testing of all steps within software development that consists of architecture and design, code implementation and debugging, performance, and quality assurance (1). This unit testing blends well with the functional testing strategy since some of these unit tests could be included in functional tests and also can test many of the same parts of the software. Functional testing is an approach that consists of a series of tests that define entry values for an operation and observe if the result is what was expected (2). These tests can both be used to ensure the right error message displays or if the expected result is a value. This type of testing also allows for normality tests, as well as exceptions.

## Test Risks / Issues

Poor time management is the greatest risk the team faces. It would be a waste of time to manually check the data flow is working for such a complex product. We will balance rigorous testing with efficient testing by:

* Looking at data shapes instead of the raw data to verify data transformations work correctly
* Pick several values out of large data to manually verify data transformations work correctly
* Prioritizing high priority test cases over low priority test cases
* Keeping tests within the scope of the product being delivered

# 2. TESTING METHOD 1: UNIT TESTING

## 2.1 Scope

The functions that will be tested are the calculations within the software and the inputs prompted to the user. The items that are not tested are the functions within the graphical user interface (GUI) and the getters/setters within the classes, as these will work properly. These topics are further described in the table below, with a reason why it is tested or not. The tester for the functions is also reported.

|  |  |  |
| --- | --- | --- |
| Unit Test Scope |  |  |
| **In Scope** | **Reason Tested** | **Tester** |
| *ConfigFile class* | *Defines all input datasets* | *Max Thibeau* |
| MeteorlogicalInput class | Key input dataset | Max Thibeau |
| FluxtowerData class | Manages all flux tower data in one class | Max Thibeau |
| SingleFluxTower class | Key input dataset | Max Thibeau |
| BPLUT class | Key input and output dataset | Max Thibeau |
| ReferenceInput class | Key input dataset | Max Thibeau |
| PFTSelector class | Defines what PFT params will be optimized for |  |
| GPP optimization class | Optimizes GPP parameters |  |
| Ramp func class | Guides optimization process |  |
| Gpp funcs | Defines all functions needed for GPP optimization |  |
| SSE funcs | Objective function minimized by optimizers |  |
| RECO class | Optimizes RECO Parameters |  |
| RECO funcs | Defines all functions needed for RECO optimization |  |
| AnalyticalModelSpinup class | Creates initial pools for SOC calculation |  |
| SOC class | Key output dataset |  |
| NumericalModelSpinUp | Key output dataset |  |
| **Out of Scope** | **Reason Not Tested** |  |
| Getters/Setters | These functions are built into python and used in most/all classes in the project. Someone can test only one of these and it would work for all the other, if necessary. |  |
| GUI Functions | These are the GUI functions that make the pages flow and get to the base page. These are the functions that were previously tested in the prototype design phase in the first semester (Fall 2019). These GUI Functions include the “display” functions within the CLI (can test by making sure x and y will be same length) |  |

## 2.2 Methodology

Unit testing tests individual modules of source code for their functionality. In this form of testing, each “unit” are key functions to the calibration software. The goal of this procedure is to isolate each module of the software to prove each module works. This style of testing proves all individual modules of a piece of software work well before stitching them all together.

For each test to be completed, all of the following steps must be completed:

1. All requirements are finalized and analyzed
2. The goal of the test is defined
3. The test is created, reviewed, and approved
4. The test is executed
5. The actual data of the test is compared with the expected data
6. Any differences between actual and expected results are debugged and resolved
7. The actual data matches the expected data
8. Final test case is created
9. Test is formally signed off

## Test Readiness

As soon as the ConfigFile class is created, unit testing can commence. As more parts of the project are completed, more unit tests will be ready. The project is being completed in a requirements oriented manner, as each requirements is completed it can be immediately tested afterward. As soon as the whole process is completed from beginning to end, several functional test requirements can be defined and tested.

## Control Procedures

### 2.4.1 Problem Reporting

When an incident is encountered during the testing process the individual testing the function will adhere to the Testing Incident Report included in the appendix of this document. When an incident is encountered, the individual finding the incident is expected to fill out the form. The individual initially assigned to the broken function will be responsible for fixing the function.

### 2.4.2 Change Requests

When one of the members makes a change request, they will do so as a pull request on the GitHub repository. The team members will discuss the pull request during a team meeting. Team members will discuss the feasibility, impact, and effort of the request. If the change request is approved, the individual responsible for the change request will be in charge of implementing the change request. If the request is not approved, it will be documented as a denied pull request on the GitHub repository.

## Test Cases

Table 1: Unit Test Cases

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Unit Test Cases | | | |  | |  | |  | |  | |
| **ID** | **Test Case** | **Input** | **Expected Output** | | **Actual Output** | | **Tester Name & Date** | | **Pass/Fail** | | **Resolution**   |  | | --- | | *Testing incident report #1.0 created (see Appendix A). Retested as ID 9.1* | |
| 1.0 | ConfigFile Function: test\_read() | “config\_test.cfg” | <input\_files.ConfigFile.ConfigFile object at 0x...> | | <input\_files.ConfigFile.ConfigFile object at 0x7f9cecaf4f10> | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 1.1 | ConfigFile Function:  get\_soc() | None | an array with an SOC val for each flux tower site (356) | | get\_soc.shape() is 356 | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 2.0 | Meteorological Input Function:  \_subset\_data(data\_list) | Data\_list: [‘MET’,’vpd’] | 2D array that’s n\_days by n\_tower\_sites (6575, 356) | | result.shape = (6575, 356) | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 2.1 | Meteorological Input Function:  \_test\_\_find\_pft\_to\_claimed\_sites() | None | Returns true when every site is claimed by at least 1 pft | | True | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 2.2 | Meteorological Input Function:  sites\_claimed\_by\_pft(pft) | pft = 1 | a list of flux tower sites claimed by pft 1 | | [92, 93, 94, 138, 178, 179, 187, 190, 212, 230, 234, 294, 295, 339] | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 2.3 | Meteorological Input Function:  test\_climatological\_year(start\_date,end\_date,meteor\_variables) | None | Returns true when every climatological var has 365 days worth of data | | *True* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 2.4 | Meteorological Input Function:  lat\_long() | None | Result = returns valid array of [long,lats] | | *[[ 31.5167 121.961 ]*  *[ 31.5847 121.903 ]*  *[ 31.5169 121.972 ]*  *[ 39.346 -0.31881]*  *[ 41.7043 12.3573 ]*  *[ 41.7052 12.3761 ]*  *[ 40.606 8.151 ]*  *[ 42.5839 10.0784 ]*  *[ 53.3989 6.356 ]*  *[ 71.5943 128.8878 ]*  *[ 68.36239 18.79475]*  *[ 28.4583 -80.6709 ]*  *[ 28.6086 -80.6715 ]*  *[ 41.4646 -82.9962 ]]* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 3.0 | FluxTowerData Function:  set\_coords(coord\_array) | coord\_array: valid array of [long,lats] | Result = calls \_set\_weights() | | *Set weights is called* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 3.1 | FluxTowerData Function:  \_set\_weights() | Empty weights array | Result = array of weights (nonzero) | | *Every tower site has a weight* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 3.2 | FluxTowerData Function: smooth\_outliers(met) | met: “gust” | Result = user inputs a window size succesfully (3.3 and 3.4) | | *User inputs size of 20 sucessfully* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 3.3 | FluxTowerData Function: smooth\_gpp\_outliers(met,window) | met: “gust”, window: 10,  Valid towers array | Result =  Calls SingleFluxTower gpp smooth | | *Gpp.smooth() is called for every flux tower* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 3.4 | FluxTowerData Function:  smooth\_reco\_outliers() | met: “gust”, window: 10,  Valid towers array | Result =  Calls SingleFluxTower reco smooth | | *RECO smooth() is called for every flux tower* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 4.0 | SingleFluxTower Function:  gpp() | None | Result = prints single tower’s data gpp | | *Got 6575 values for GPP* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 4.1 | SingleFluxTower Function:  smooth\_outliers\_single\_var(var,met,window) | var: valid RECOs,  met: “gust”, window: 10 | Result = outliers from var are filtered | | *scipy’s filfilt() has been succesfully applied to the tower’s reco data* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 5.0 | NewBPLUT Function:  print() | Relative csv path for BPLUT | Result = prints BPLUT values | | *An 8x44 array of values(pftt by num\_bplut\_vars)* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 5.1 | NewBPLUT Function:  gpp\_params(pft) | pft: 1 | Result = gets valid BPLUT GPP params for pft 0 | | *[1.38, 230.0, 303.0, 15.0, 7000.0, 18.0, 26.0, 0.36, 1.0]* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 5.2 | NewBPLUT Function:  reco\_params(pft) | pft: 1 | Result = gets valid BPLUT RECO params for pft 0 | | *[-46.0, 59.0, 619.5600000000001, 0.43]* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 5.3 | NewBPLUT Function:  kmult\_params(pft) | pft: 1 | Result = gets valid TSOIL, SMtop min and max for pft 0 | | *[619.5600000000001, -46.0, 59.0]* | | *Max Thibeau, 04-15-2020* | | *Pass* | | *Complete* |
| 5.4 | NewBPLUT Function:  after\_optimization(gpp\_or\_reco, pft, vars\_optimized) | gpp\_or\_reco: “GPP”, pft: 0, vars\_optimized: all 8 parameters (new vals) | Result = updated BPLUT shows optimized values for pft 0 | |  | |  | | *Pass* | | *Complete* |
| 5.5 | NewBPLUT Function:  test\_write() | None | Result = updated bplut outputted in csv format | |  | |  | | *Pass* | | *Complete* |
| 6.0 | ReferenceInput Function:  subset\_by\_pft(pft,tower\_sites\_claimed\_by\_pft) | pft: 0, tower\_sites\_claimed\_by\_pft: valid array of tower sites | Result = subsets gpp and reco for these valid tower sites | |  | |  | | *Pass* | | *Complete* |
| 6.1 | ReferenceInput Function:  gpp() | None | Result = valid array of observed gpp form towers | |  | |  | | *Pass* | | *Complete* |
| 6.2 | ReferenceInput Function:  reco() | None | Result = valid array of observed reco from towers | |  | |  | | *Pass* | | *Complete* |
| 6.3 | ReferenceInput Function:  \_subset\_data(data\_list) | data\_list: [‘GPP’, ’gpp\_pft1\_mean’] | Result = valid arrays for observed gpp | |  | |  | | *Pass* | | *Complete* |
| 7.0 | PFTSelector Function:  select\_pft(meteor\_input,prev\_pfts) | meteor\_input: from previous test, prev\_pfts: [1,2] | Result =  Displays valid pfts 1-8 and displays previous pfts next to this | |  | |  | | *Pass* | | *Complete* |
| 8.0 | GPP Function:  clean\_nans(array) | array: [1.0,8.7,nan,9.8,5.76,nan,nan,19.8] | Result = array with no nans | |  | |  | | *Pass* | | *Complete* |
| 8.1 | GPP Function:  set\_apar\_bounds() | FPAR: array of floats,  PAR: array of floats | Result = ability to set bounds (int) | |  | |  | | *Pass* | | *Complete* |
| 8.2 | GPP Function:  func\_to\_optimize(gpp\_param) | gpp\_param:  bplut.gpp\_params(pft) where pft:0 | Result = returns valid SSE (11.0) | |  | |  | | *Pass* | | *Complete* |
| 8.3 | GPP Function:  gpp\_v\_emult(pft,bplut,gpp\_param) | pft: 0, bplut: valid BPLUT, gpp\_param:  bplut.gpp\_params(pft) | Result = displays prompt if wanting to view optional graph | |  | |  | | *Pass* | | *Complete* |
| 9.0 | Ramp\_Func Function:  downward\_ramp\_func(x,x\_min\_max) | x: 10.0, x\_min\_max:  (5.0,20.0) | Result = 0.5 | |  | |  | | *Pass* | | *Complete* |
| 9.1 | Ramp\_Func Function:  upward\_ramp\_func(x,x\_min\_max) | x: 10.0, x\_min\_max: (5.0,30.0) | Result = 0.25 | |  | |  | | *Pass* | | *Complete* |
| 9.2 | Ramp\_Func Function:  kmult\_arrhenius\_curve(x,bt\_soil) | x: 10.0,  bt\_soil: 5.00 | Result = 1.1038 | |  | |  | | *Pass* | | *Complete* |
| 9.3 | Ramp\_Func Function:  arrhenius\_curve(x,mult,a,b) | x: 10.0,  mult: 5.00 ,  a: 25.0,  b: 100.0 | Result = 1.2911 | |  | |  | | *Pass* | | *Complete* |
| 10.0 | GPP\_Func Function:  gpp\_apar(apar, vpd, tmin, smrz, tsurf, lue, vpd\_min, vpd\_max, tmin\_min, tmin\_max, smrz\_min, smrz\_max, ft\_mult\_frozen, ft\_mult\_thawed) | Apar: 10.0, vpd: 12.5, tmin: 15.0, smrz: 17.5, tsurf: 280.0, lue: 1.5, vpd\_min: 10.0, vpd\_max: 20.0, tmin\_min: 12.5, tmin\_max: 22.5, smrz\_min: 15.0, smrz\_max: 25.0, ft\_mult\_frozen: 0.5, ft\_mult\_thawed: 1 | Result = 1.07811 | |  | |  | | *Pass* | | *Complete* |
| 10.1 | GPP\_Func Function:  gpp(fpar, par, vpd, tmin, smrz, tsurf, lue, vpd\_min, vpd\_max, tmin\_min, tmin\_max, smrz\_min, smrz\_max, ft\_mult\_frozen, ft\_mult\_thawed) | fpar: 5.0,  par: 38.0, vpd: 12.5, tmin: 15.0, smrz: 17.5, tsurf: 230.0, lue: 1.5, vpd\_min: 10.0, vpd\_max: 20.0, tmin\_min: 12.5, tmin\_max: 22.5, smrz\_min: 15.0, smrz\_max: 25.0, ft\_mult\_frozen: 0.5, ft\_mult\_thawed: 1 | Result = 10.242045 | |  | |  | | *Pass* | | *Complete* |
| 10.2 | GPP\_Func Function:  emult(vpd, tmin, smrz, tsurf, lue, vpd\_min, vpd\_max, tmin\_min, tmin\_max, smrz\_min, smrz\_max, ft\_mult\_frozen, ft\_mult\_thawed) | vpd: 12.5, tmin: 15.0, smrz: 17.5, tsurf: 230.0, lue: 1.5, vpd\_min: 10.0, vpd\_max: 20.0, tmin\_min: 12.5, tmin\_max: 22.5, smrz\_min: 15.0, smrz\_max: 25.0, ft\_mult\_frozen: 0.5, ft\_mult\_thawed: 1 | Result = 0.035937 | |  | |  | | *Pass* | | *Complete* |
| 11.0 | SSE Function:  sse (all\_obs, all\_pred, non\_missing\_obs, weights) | all\_obs: flux\_tower\_data.gpp(), all\_pred: simulated\_gpp(), non\_missing\_obs: flux\_tower\_data.non\_missing\_observations(), weights: flux\_tower\_data.weights() | Result = valid list of gpp values | |  | |  | | *Pass* | | *Complete* |
| 11.1 | SSE Function:  rmse(n\_s, obs, pred) | n\_s: 2,  obs: [5.25,6.32], pred: [5.34,6.54] | Result = 4.20703 | |  | |  | | *Pass* | | *Complete* |
| 12.0 | RECO Function:  set\_prh\_and\_pk() | Specify prh as .75 and pk as .5 | Result = no error message to display for re-prompting | |  | |  | | *Pass* | | *Complete* |
| 12.1 | RECO Function:  \_simulate\_reco(reco\_param) | reco\_param: bplut.reco\_params(pft) for pft 0 | Result = valid array of simulated RECO values | |  | |  | | *Pass* | | *Complete* |
| 12.2 | RECO Function:  rhc\_v\_kmult() | None, input y | Result = displays optional graph | |  | |  | | *Pass* | | *Complete* |
| 12.3 | RECO Function:  func\_to\_optimize(reco\_param) | reco\_param:  bplut.reco\_params(pft) where pft:0 | Result =  returns valid SSE (11.0) | |  | |  | | *Pass* | | *Complete* |
| 13.0 | RECO\_Func Function:  kmult(t\_soil, smsf, bt\_soil, smsf\_min, smsf\_max) | t\_soil: 10.0, smsf: 15.0, bt\_soil: 5.0, smsf\_min: 12.5, smsf\_max: 22.5 | Result = 0.36793 | |  | |  | | *Pass* | | *Complete* |
| 13.1 | RECO\_Func Function:  reco(gpp, t\_soil, smsf, c\_bar, f\_aut, bt\_soil, a, b, smsf\_min, smsf\_max) | Gpp: 25.0, t\_soil: 10.0 , smsf: 15.0 , c\_bar: 12.5, f\_aut: 0.5 , bt\_soil: 5.0 , a: 25.0, b: 100.0, smsf\_min: 12.5, smsf\_max: 22.5 | Result = 16.535 | |  | |  | | *Pass* | | *Complete* |
| 13.2 | RECO\_Func Function:  reco(gpp, kmult, c\_bar, f\_aut) | Gpp: 6.08, kmult: 1.35,  c\_bar: 0.57,  f\_aut: 0.78 | Result = 6.40224 | |  | |  | | *Pass* | | *Complete* |
| 14.0 | AnalyticalModelSpinUp Function:  calc\_npp(towers\_gpp,fraut) | towers\_gpp: [[5.5,3.45],[2.34,7.68]], fraut: .78 | Result = [ [1.21,0.759],[0.5148,1.6896] ] | |  | |  | | *Pass* | | *Complete* |
| 14.1 | AnalyticalModelSpinUp Function:  summed\_kmults() | None | Result = valid array values | |  | |  | | *Pass* | | *Complete* |
| 14.2 | AnalyticalModelSpinUp Function:  get\_npps() | None | Result = valid array of array of npp values | |  | |  | | *Pass* | | *Complete* |
| 15.0 | SOC Function:  calc\_sigmas() | Num\_towers: 2, kmult: [1.5,3.0], npp: [2.5,4.0] | Result = [0.72,0.72] | |  | |  | | *Pass* | | *Complete* |
| 15.1 | SOC Function:  calc\_beta\_soc(fmet,fstr,kstr,krec,ropt) | Fmet: 0.456, fstr: 0.67, kstr: 0.34, krec: 0.5, ropt: 0.98 | Result = 0.002842 | |  | |  | | *Pass* | | *Complete* |
| 15.2 | SOC Function:  calc\_estimate() | sigmas = [0.72,0.54]  beta\_soc = .002842 | Result = [.00204624, .00153468] | |  | |  | | *Pass* | | *Complete* |
| 16.0 | NumericalModelSpinUp Function:  set\_iterations() | None, input an integer of 10 | Result = calling of \_forward\_run() 10 times | |  | |  | | *Pass* | | *Complete* |
| 16.1 | NumericalModelSpinUp Function:  \_forward\_run() | None, set in NumericalModelSpinUp creation | Result = calculation of pools | |  | |  | | *Pass* | | *Complete* |
|  | *…Add all rows necessary for a complete plan of test cases.* |  |  | |  | |  | |  | |  |

Table 2: Requirements Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Requirements Test Cases | | | | | |
| **ID\*** | **Test Case Summary** | **Prerequisites** | **Test Procedure** | **Input Data** | **Expected Result** |
| 1.0 | Calculate the dominant PFT for each tower site | Meteor\_input class and fluxtowerdata class works | Every tower is claimed by > 1 pft | Dict of pft to tower sites | True |
| 1.1  (MP01) | Calculate percentage of area that is occupied for each 9km tower site | meteor\_input class works | Each tower site has a %claimed for every PFT | meteor\_input data | True |
| 2.0 | Read in the input datasets from the configuration file for calibration | config\_file class is defined | A ConfigFile class is created | “config\_test.cfg” | A ConfigFile class is successfully created |
| 2.1 (MP03) | Subset time series variables to configurable period |  |  |  |  |
| 3.0 | Gather L4C Meteorological inputs and ground-truth/observed data from the flux towers | Fluxtowerdata class works | SingleFluxTower class is created | A single flux tower fname | An array for gpp, reco, and nee |
| 4.0 | User chooses one of the PFTs | SelectPFT class | SelectPFT class is called | 5 | 5 |
| 4.1  (MP04) | Subset tower sites to the chosen PFT | Requirement 1.0 | Print tower sites | 5 | 32 tower sites |
| 4.2  (MP06) | Compile important data for optimization of GPP and RECO (ex: observed GPP, VPD, TMIN, etc) | meteor\_input class works | meteor\_vars attribute is fully stocked | meteor\_input fname | 6 vars, all [365xnum\_sites] |
| 4.3  (MP05) | Compile ancillary info on each site for selected PFT (ex: SOC stock size for the site) |  |  |  |  |
| 5.0 | Guide user through removing outliers in average annual GPP and RECO calculations | Gpp and reco optimizer classes work | Ramps are displayed, params chosen, ramps redisplayed | None | A list of new optimized params and ramp displays |
| 5.1  (LP01) | Remove negative values in annual GPP and RECO for each flux tower site | Fluxtowerdata class works as well as singlefluxtower | All GPP >= 0, all RECO >= 0 | Single flux tower | True |
| 5.1  (MP07) | Average data for PFT for each day of the year | meteor\_input and flux\_tower\_data work | [365xn\_sites] array for each var | FluxTowerData, StartDate, EndDate | [365xn\_sites] |
| 5.2  (MP19) | Calculate GPP\*, Kmult\*, NPP\* based off of optimized calculations | Gpp and reco optimizers work,  spinup classes work | Gpp, kmult, and npp are succesfully summed along the time axis | [365xn sites] arrays for gpp, kmult, and npp | n\_sites long arrays for gpp, kmult, and npp |
| 6.0 | Allow user to optimize GPP and RECO parameters (8 GPP params, 4 RECO params) | Gpp and reco optimizer classes work | Scipy.minimize func succesfully called | Two choice vectors, 1 8 long and 1 4 long | 8 floats returned for gpp, 4 floats returned for reco |
| 6.1  (MP11) | Allow user to specify Pk and Prh for RECO | Reco optimizer works | Let the user go through the pk, prh procedure | ,5. 9 | .5, .9 |
| 7.0 | Calculate linear ramp functions given current BPLUT | Ramp func classes work, meteor\_input class works | Scatter plot with ramp function superimposed | VPD, | Scatter plot with ramp function superimposed |
| 7.1  (MP08,  MP12) | Calculate linear ramp functions using the updated values after optimization (and be able to save) | gpp\_optimizer class works | Difference between old and new plot | gpp\_optimized params , meteor\_input vars | Difference between old and new plot |
| 7.2  (LP02) | Calculate lower and upper limits for APAR (user can set bounds) | Gpp optimizer class works | all\_apar > low\_bound and all\_apar < high\_bound | Apar, (apar\_bound\_tuple) | True |
| 7.3  (MP09,  LP04) | Allow user to plot GPP against Emult and Rh/Cbar against Kmult as an optional graph | GPP and RECO optimizer classes work | Prompt user if they want to plot | Yes | Scatter plot |
| 7.4  (MP10) | Report differences between original and updated/optimized values | GPP and RECO optimizer classes work | Param diffrerence is reported | Optimizer classes | Old param: x. new param: x |
| 7.5  (MP13) | Calculate Cbar for each tower site after optimization |  |  |  |  |
| 7.6  (LP03) | Allow the user to repeat optimizations after the initial optimization |  |  |  |  |
| 8.0 | Allow the user to specify the number of Numerical Spin-Up iterations | NumericalModelSpinup class works | NumericalModelSpinup class is called | 10 | 10 spinups are performed |
| 8.1  (MP15) | Run Analytical and Numerical Model Spin-Ups | NumericalModelSpinup and AnalyticalSpinUp class works | NumericalModelSpinup class is called | 10 | 10 spinups are performed |
| 8.2  (MP16) | Run preliminary spin up arbitrary period over full operational record 2000-2019 | PreliminarySpinup class works | PreliminarySpinUp class is called | None | Kmult, NPP, daily litterfall |
| 9.0 | Compute comprehensive validation (SOC) and fit statistics for each PFT | Soc and fit\_statistics classes work | RMSE stats are reported for every SOC | Estimated, ground truth SOC | Pearson coefficient |
| 9.1  (MP14) | Display Estimated SOC vs Ground-Truth SOC | SOC class works | Scatter plot is called | Two n\_sites long arrays | Scatter plot of estimated vs ground-truth SOC is displayed |
| 9.2  (MP20) | Calculate RMSE for each subset of towers for GPP and RECO |  |  |  |  |
| 10.0 | Output updated BPLUT, 4 SOC pool calculations, and report SMRZ min/max for years 2000-2018 (total forward model run) | Whole calibration procedure works | write\_output() function is called | Updated BPLUT table, 4 SOC pools, and SMRZ min/max | 3 text files |
| 10.1  (MP21) | Calculate NEE (NEE= RECO – GPP) based on optimized parameters then report |  |  |  |  |
| 11.0 | Calculate flux tower weights as some tower sites are located within the same 9km area | flux\_tower\_data nd meteor\_input classes work | set\_weights() method is called | meteor\_input class and fluxtowerdata class | Every flux tower is assigned a weight |
| ***Test Case IDs link back to SRS*** | High Priority Requirements are the #.0, not labeled with HP  Dependencies on the HP Requirements:  MP = Medium Priority Requirements  LP = Low Priority Requirements |  |  |  |  |

## Test Results (this section to be completed by Week 9)

The results of unit testing are added to the Unit Test Cases table in Section 2.5 (above), designated with the columns Actual Output, Tester Name & Date, Pass/Fail, and Resolution.

The results of functional testing are based off of the Requirements Test Cases in Section 2.5 (above), but having only the Summary, Input Data, and Expected Result. This table gives the results of these functional requirement test cases.

Table 3: Functional Testing Results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Functional Test Results | | | | | | | |
| **ID\*** | **Test Case Summary** | **Input Data** | **Expected Result** | **Actual Result** | **Tester name & date** | **Pass/Fail** | **Resolution** |
| 1.0 | *Login with a user incorrect account.* | *Username: abc*  *Password: abc* | *An error will be displayed for the wrong credentials.* |  | *Jane Doe, 2-15-2019* |  |  |
| 2.2 | *Add Challenge on Admin page with overlapping dates* | *Title: Title*  *Description: Description*  *Start Date: 03/01/19*  *End Date:03/01/18* | *Error Message displayed that start date must be before end date* |  |  |  |  |
| 3.5 | *Create new user* | *first name: abc*  *last name: abc*  *username: abc*  *email: abc@gmail.com password: abc*  *password again: abc*  *rent or own: rent*  *home size: 1000*  *household size: 2*  *other energy: no* | *User is successfully logged in, added to database, has all functionality available* |  | *Jane Doe, 2-15-2019* |  |  |
|  | *…Add all rows necessary for a complete plan of test cases.* |  |  |  |  |  |  |

***\*Test Case IDs link back to SRS***

# Acceptance and Approval

The team members hereby indicate by their signatures below that they have read and agree with the specifications of this document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Mark Matas – 4/19 |  | Max Thibeau - 4/17 |  | [Insert name & date here] |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| [Insert name & date here] |  | [Insert name & date here] |  |  |

# References

1. Hamill, Paul. “Unit Test Frameworks.” *O'Reilly Online Learning*, O'Reilly Media, Inc. [www.oreilly.com/library/view/unit-test-frameworks/0596006896/ch01.html](http://www.oreilly.com/library/view/unit-test-frameworks/0596006896/ch01.html)
2. “Functional Testing.” *Functional Testing - an Overview*, ScienceDirect. [www.sciencedirect.com/topics/computer-science/functional-testing](http://www.sciencedirect.com/topics/computer-science/functional-testing)

# Appendices

*[Include any extra documentation in the appendices that supplements the main document text.]*