# CS562: Applied Software Engineering

## *Part 4:Final Report*

Hafed Alghamdi
Email: Alghamha@oregonstate.edu

**Abstract**

The main objective of this project is testing Astropy.table library. The work targeted essential operations of the Table library to ensure their correctness and check if theyapply what have been specified in Astropy documentationabout these operations. Table library is large and pretty much complicated. Therefore, the work did not include the reading and writing to a file operations neither the printing nor some other helper functions. Instead, the main focus was the actual table operations that should guarantee tables' data consistency and reliability such as removing, deleting, updating columns, tuples, etc.

The main tool used for testing is TSTL. TSTL has captured three bugs in the Table library and shows great results. In this report, we will discuss these bugs, TSTL effectiveness, TSTL advantages and disadvantages, and Table Library code coverage.

### Accomplishment

In this project we were able to learn Python, TSTL, code coverage and different software testing techniques. Since, we are using TSTL in this project, the majority of knowledge gain was related to Random Testing technique. Having all that in mind, we were able to capture three bugs related to Astropy.table library using TSTL tool. Moreover, we were able to use python heavily in this project which provided us with a great programming experience. Finally, we got to know what all software testing is about and therefore, not every software can be trusted as in our case large libraries revealed some bugs that will be discussed in the next section.

### Captured Bugs

Testing Astropy.table library using TSTL revealed some unexpected bugs. These bugs varies in criticality from bad API to serious bugs that may damage tables data and make it inconsistence and unreliable. The following is a summary of these captured bugs:

1- ***Updating Columns' Unit Field Bug:****(Figure 1: The code used to capture )*
This bug occursmost of the time when inserting a column more than oncewith new values to update its content. The library is able to update the tuples'values of a column but the unit fieldmost of the time does not get the new updates. Missing the unit update may cause a devastating damage to the table's data when for example a user uses a loop to remove data from the table based on their units' values. User's expectationsis that the update has been implemented, since there was no error messages and further actions can be applied. As a

result, this will eventually results in table's data inconsistency. Someone, could argue that this might have beenpart of the design to easy the development of other staff. However, this kind of argument cannot be taken into account because any library APIs should expect the users' unexpected behaviors. Therefore, a good design could be either updating all the values including the unit field or not updating the values at all **by raising an exception.** Moreover, documenting this kind ofaction is essential to make the user better understand the library APIs and use them correctly.

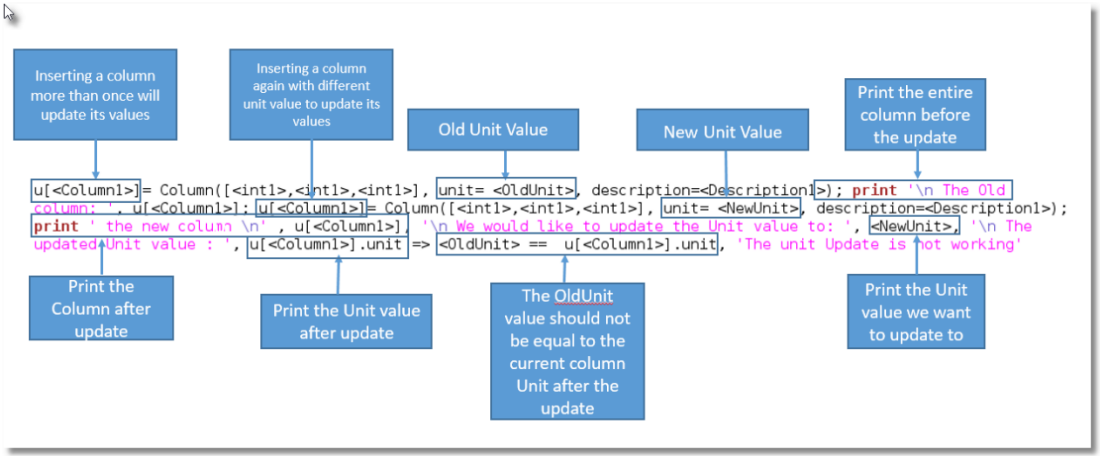


*Figure 1: The code used to capture bug 1*

2- ***Local variable 'new_name' referenced before assignment:****(Figure 2: The code used for to capture bug)*

This uncaught exception occurswhen inserting a column randomly to the table while the rename_duplicate option is enabled and the column is not exist in the table. This bug is impractical as it shows some API weaknesses that may lower user'soverall satisfaction. Usually, user's expectations when adding a column to a table with the option rename-duplicate enabled, the column will be added no matter what, and even if it is not exist in the table it will be at least inserted with the same name. A proper action forsimilarsituation is checking for the column existence first, and based on that apply an action either inserting the column with the same name if it is not exist,otherwiserename_duplicate the column when it is exist.

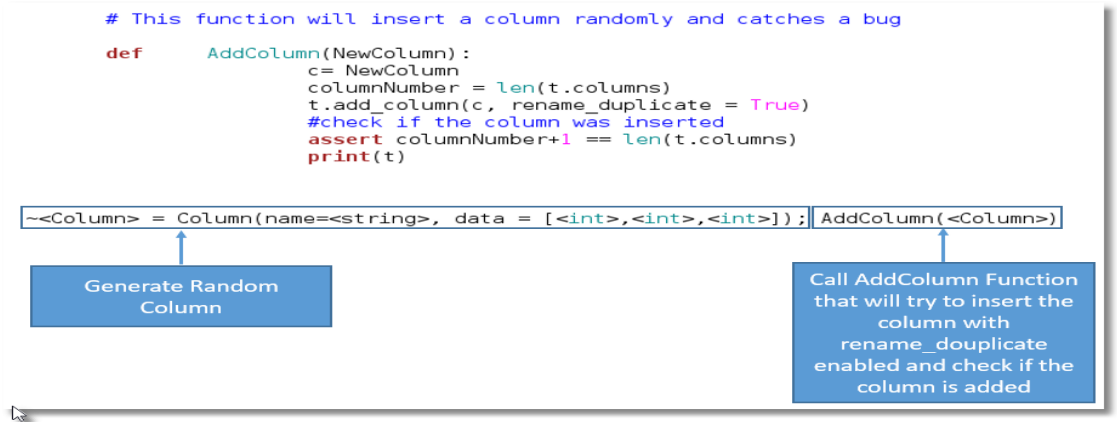


*Figure 2: The code used for to capture bug 2*

3- ***Updating Tuples (Indices) of Type (String) Bug:*** *(Figure 3: The code used for to capture bug 3)*

This bug occurs when updating a tuple value of type (string). In Astropy.table library, there are different types of columns such as Integers, float, string, object, and more that can be used for each column. String type wasinteresting to test because it uses the length of the first insertion of columns' tuples and hard coded the longest string length as a type. For example, if you have 'Hafed' as the longest string tuple, then the corresponding type of a column would be S5. This was an interesting technique to test and a bug was detected when updating tuples. The detected bug appears when updating a tuple with longer string after normal column insertion where the updated tuple will only have a portion of the new string. Apparently, it is only updating the field based on the originalcolumn type when it was first inserted, ignoring that it can be updated with a longer string which in turn results in table data inconsistency and unreliability. This is a major bug because the user is expecting the field to be updated with the new string value but in reality it is not, because of the hardcoded length of the original stringcolumn type. We are not sure what the reason behind this approach is, but a good way to fix this bug can be possibly achieved by first checking the old string length type and the new inserted string length, then adjust the string type of the column accordingly.

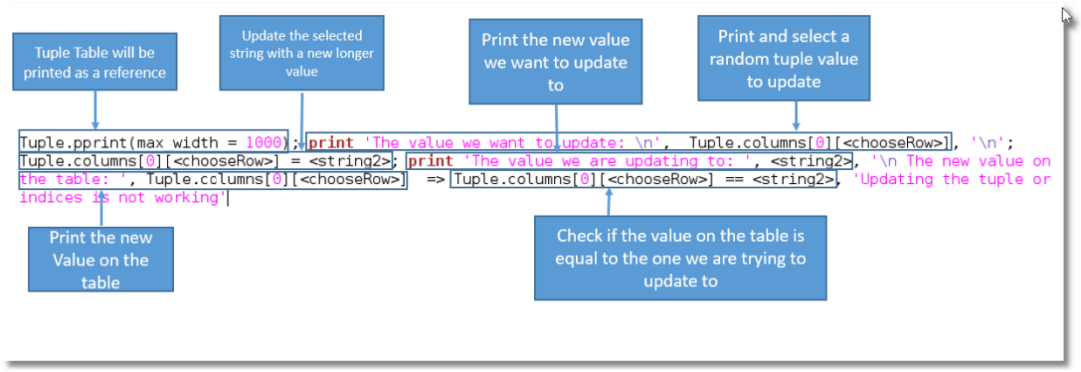


*Figure 3: The code used for to capture bug 3*

## How well the tester work

Since TSTL is using random testing technique, we were able to capture some bugs. We have noticed that TSTL guides towards the errors because of the effective random tester technique it is using. Moreover, it is providingcode coverage statistics based on python coverage.py that can be used as a useful reference when for instance someone would like to know how much have been covered from the code and what's been left. Of course, there were some usage issues related to the TSTL syntaxand lack of documentation that will be discussed in later sections, however, as long as it captures bugs in a very large and popular libraries we think that is great.

## What is wrong with TSTL?

TSTL has a special syntax that sometimes confuses the user and makes him/shetakes a very long time to figure out something that could be done in a minute. We believe this kind of issues can be resolved by having some documentations as a reference. Moreover, some enhancements have been added to TSTL during the term such as the exception warning that can be a source of help, although we could not try it because of time limitations, but that will help users a lot when they face exceptions issues that they would like to ignore and just get informed about them. Moreover, some users would like to have the bools get values sometimes from other functions depending on the SUT (For instance, $< columnIndex >:= < [1..foolen(\ \ )] >$). We tried that but it seems that TSTL does accepting it.

Finally, we wish that we haveenable, disable and just let me know options when having assert in the actions (For instance, $a + b => disable(a == b)$). This will make the user able to continue testing without having more helper functions for instance avoid having the assert matches when the disable option is active because some actions may rely on this action even if it breaks.

## What is good about TSTL?

TSTL shows its power in capturing bugs as it is designed for random testing. The syntax as well enforces the users to follow Random Testing technique. However, it is not only Random Testing technique that can be used in TSTL, stress testing, performance testing in addition to some other testing techniques can be used in TSTL as well. However, it will be more difficult to do so in TSTL. Moreover, since it is built based on python, it is very easy to create a helper function in python when stuck and easily use it inside the TSTL code to achieve the required goal. Finally, code coverage in python is really great and give some since of what's been covered and what's been left.

## Code Coverage

Thecode coverage of the Astropy Table Library is 37 percent so far according to python coverage report(Figure 4: Code Testing Coverage). However, allhighly technical used functions that have been mentioned in Part 3 report have been tested thoroughly (Table 1: Code Testing Coverage). We believe the low coverage percent was a result of the untested parts related to printing, importing files and exporting files functions those have many properties, branches and helper functionswhich theywere not part of the scope of this project. Moreover, some functions are meant for python 3 and TSTL work best on version 2.7, so these functions were ignored as well.

| Functions have been tested | Functions have not been tested |
|---|---|
| Adding Columns | More |
| Adding Rows | Next |
| Copy Table | Field |
| Column Index | Filled |
| Remove Columns | Pformat |

| | |
|---|---|
| Group By | pprint |
| Rename Column | Read |
| Reverse Order | Show in browser |
| Sort | Write |
| Remove Rows | convert_bytestring_to_unicode - python3 only |
| As Array returns a copy of the table as np | convert_unicode_to_bytestring - python3 only |
| keep Columns | To_Pandas |

*Table 1: Code Testing Coverage*

**Note:**TSTL coverage worked for this project using the normal import but it was not giving any kind of accuracy. However, when using 'source: <path>where the library code lives under python, because it cannot be copied to the TSTL folder's, it worked and gave better results.

```
STOPPING TEST DUE TO TIMEOUT, TERMINATED AT LENGTH 46
STOPPING TESTING DUE TO TIMEOUT
36.6911764706 PERCENT COVERED
100.169565916 TOTAL RUNTIME
52 EXECUTED
5146 TOTAL TEST OPERATIONS
94.3555512428 TIME SPENT EXECUTING TEST OPERATIONS
0.119736433029 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
5.43757891655 TIME SPENT CHECKING PROPERTIES
99.7931301594 TOTAL TIME SPENT RUNNING SUT
0.0550320148468 TIME SPENT RESTARTING
0.0 TIME SPENT REDUCING TEST CASES
458 BRANCHES COVERED
330 STATEMENTS COVERED
```

*Figure 5: Coverage report*

```
1  Name                                                  Stmts   Miss Branch BrPart  Cover   Missing
2  ------------------------------------------------------------------------------------------------------
3  /usr/lib64/python2.7/site-packages/astropy/table/table.py     876    562    484     92    37%   2-46, 57, 61-78, 82-88, 91, 105-
4
```

*Figure 6: Coverage.out file*

## References:


1- http://www.astropy.org/
2- Main source for All Astropy.table functions: http://docs.astropy.org/en/stable/api/astropy.table.Table.html
3- http://docs.astropy.org/en/stable/install.html
4- http://docs.astropy.org/en/stable/index.html
5- http://docs.astropy.org/en/stable/table/index.html#module-astropy.table
6- http://docs.astropy.org/en/stable/units/index.html
7- https://github.com/astropy/astropy
8- https://github.com/agroce/tstl
9- http://docs.astropy.org/en/stable/table/modify_table.html
10- https://docs.python.org/2/tutorial/index.html