I wrote two test applications in the period 2019 – 2021

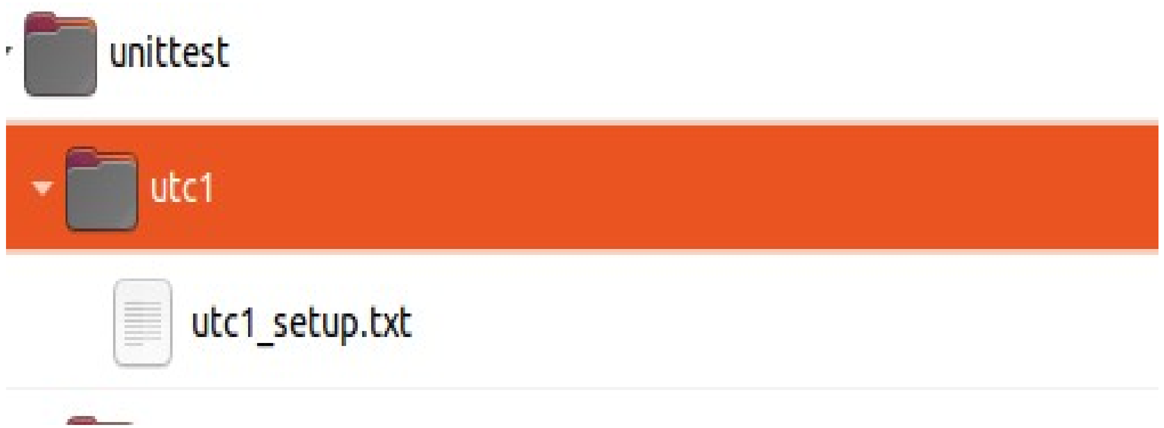
1. A unit test application
2. A data parser

I wrote two versions of the data parser:

1) The basic version parses data and performs consistency checks on the data.
2) The extended version also supports conversions on the data.

In the following example the unit test application is used to test the data parser.

Unit tests are being defined in setup files. They can be found in a unittest folder, which is a required sub directory in the project folder of the application we want to test.
We group unit tests in unit test collections.



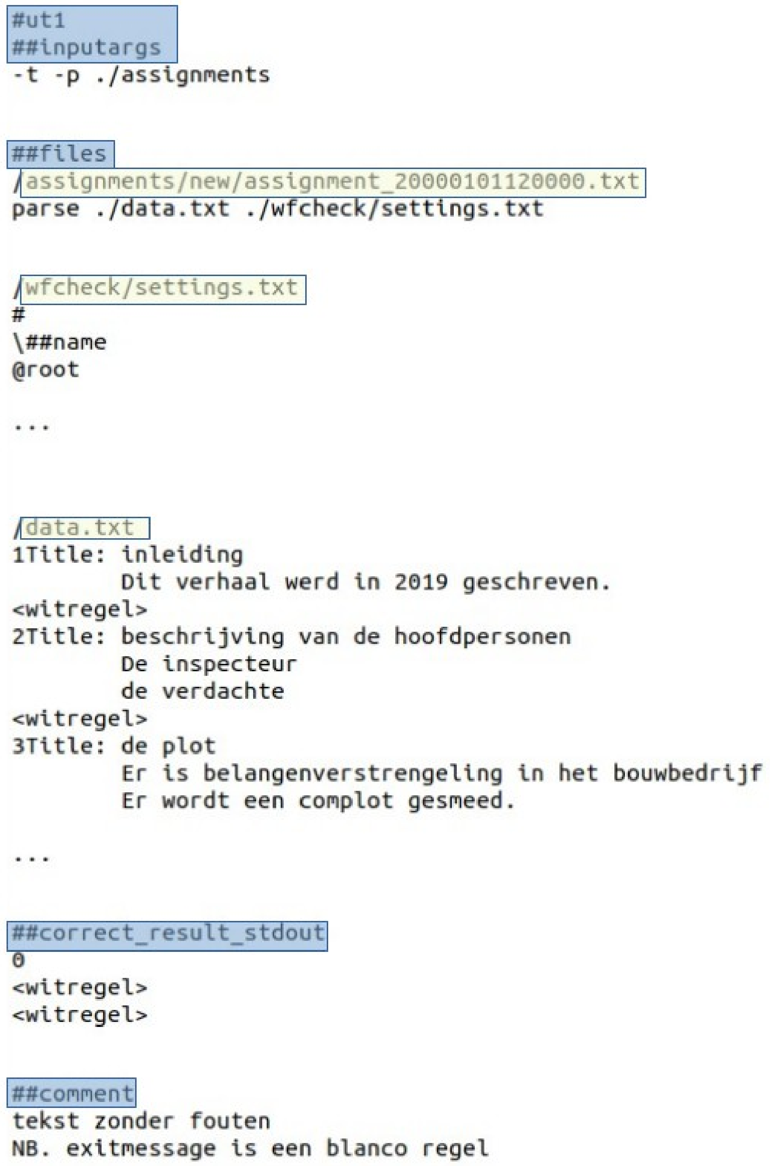
Here we see a setup file for unit test collection 1.

Here we see (a shortened version of) the definition of unit test 1 in the setup file.

We make clear with which arguments the parser should be started.

As the parser also needs a settings file and a data file, we also define which files need to be created.

Under correct_result_stdout we write the expected outcome in the terminal, in this case an exit code 0.



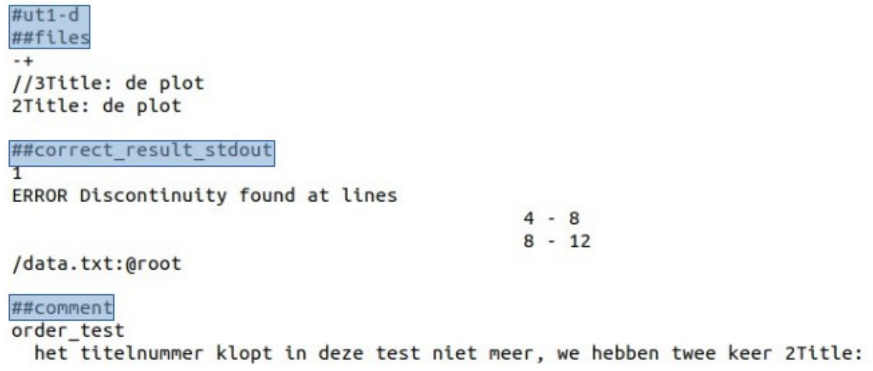
```
#ut1
##inputargs
-t -p ./assignments


##files
/assignments/new/assignment_20000101120000.txt
parse ./data.txt ./wfcheck/settings.txt


/wfcheck/settings.txt
#
\##name
@root

...



/data.txt
1Title: inleiding
        Dit verhaal werd in 2019 geschreven.
<witregel>
2Title: beschrijving van de hoofdpersonen
        De inspecteur
        de verdachte
<witregel>
3Title: de plot
        Er is belangenverstrengeling in het bouwbedrijf
        Er wordt een complot gesmeed.

...


##correct_result_stdout
0
<witregel>
<witregel>


##comment
tekst zonder fouten
NB. exitmessage is een blanco regel
```

Here we define a variant on unit test 1. This can be done very easily as the unit test application supports inheritance: a unique feature of this application!

We see that a specific (title)line in the previously defined file data.txt is being replaced with another (title)line.

We expect an error in the terminal this time and therefor set a new value for correct_result_stdout.

As the value for inputargs remains unchanged, we do not have to be concerned about it, as it is being inherited from unit test 1.



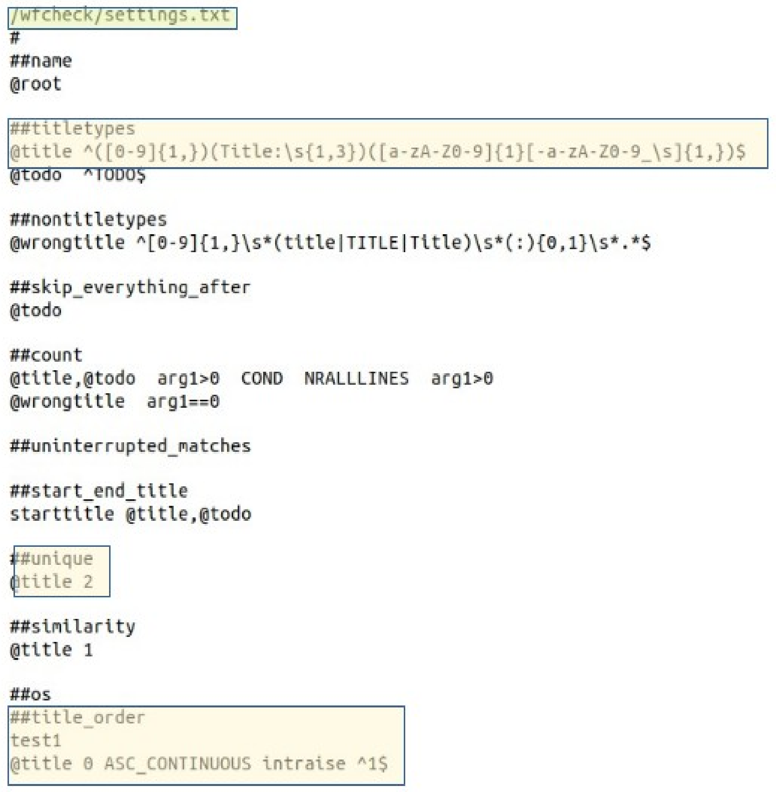
```
#ut1-d
##files
-+
//3Title: de plot
2Title: de plot

##correct_result_stdout
1
ERROR Discontinuity found at lines
                                             4 - 8
                                             8 - 12

/data.txt:@root

##comment
order_test
  het titelnummer klopt in deze test niet meer, we hebben twee keer 2Title:
```

The parser identifies the titlelines in data.txt with an in the settingsfile defined regular expression.

This allows the parser to group data. Quite complex groupings are possible, but is not demonstrated here.

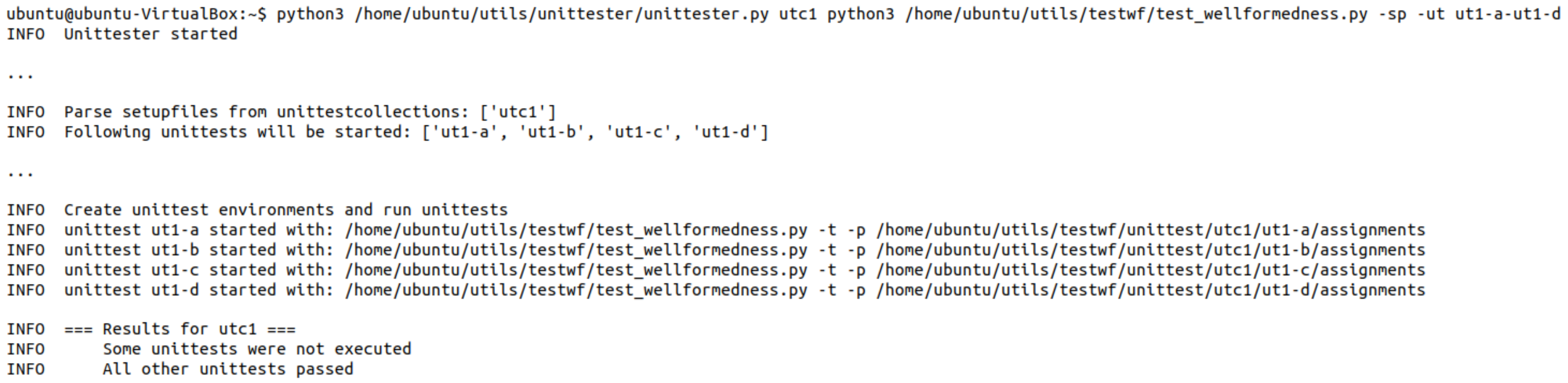
The settingsfile tells the parser also how to test the data.
The unique test for example (see highlight), tests if every titleline in the collection of titlelines is unique.

The title order test (see highlight) tests the ordering of titlelines. It demands that the chapter numbers of the titlelines is an uninterrupted sequence.
This ordering for example, is incorrect:
1, 2, 2, 4, 5



```
/wfcheck/settings.txt
#
##name
@root

##titletypes
@title ^([0-9]{1,})(Title:\s{1,3})([a-zA-Z0-9]{1}[-a-zA-Z0-9_\s]{1,})$
@todo  ^TODO$

##nontitletypes
@wrongtitle ^[0-9]{1,}\s*(title|TITLE|Title)\s*(:){0,1}\s*.*$

##skip_everything_after
@todo

##count
@title,@todo  arg1>0  COND  NRALLLINES  arg1>0
@wrongtitle  arg1==0

##uninterrupted_matches

##start_end_title
starttitle @title,@todo

##unique
@title 2

##similarity
@title 1

##os
##title_order
test1
@title 0 ASC_CONTINUOUS intraise ^1$
```

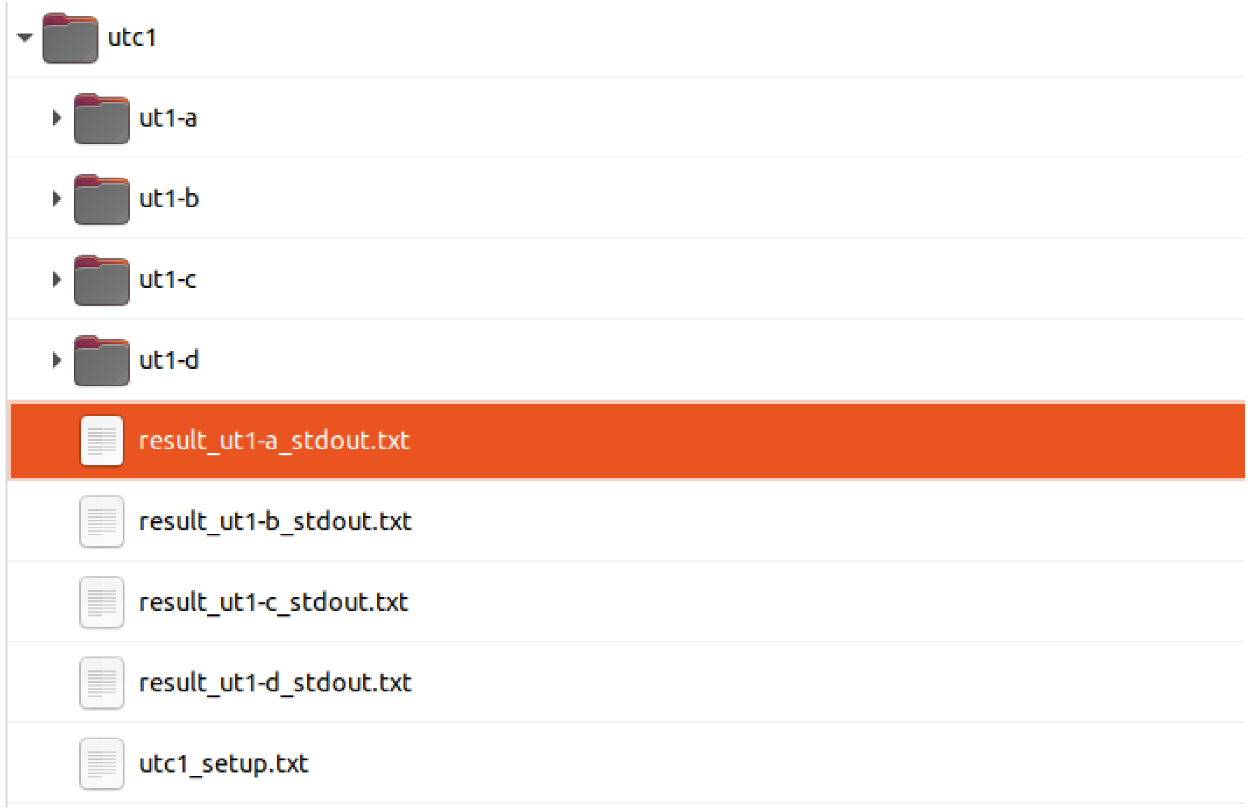
We now start the unit test application to test the parser:



```
ubuntu@ubuntu-VirtualBox:~$ python3 /home/ubuntu/utils/unittester/unittester.py utc1 python3 /home/ubuntu/utils/testwf/test_wellformedness.py -sp -ut ut1-a-ut1-d
INFO  Unittester started

...

INFO  Parse setupfiles from unittestcollections: ['utc1']
INFO  Following unittests will be started: ['ut1-a', 'ut1-b', 'ut1-c', 'ut1-d']

...

INFO  Create unittest environments and run unittests
INFO  unittest ut1-a started with: /home/ubuntu/utils/testwf/test_wellformedness.py -t -p /home/ubuntu/utils/testwf/unittest/utc1/ut1-a/assignments
INFO  unittest ut1-b started with: /home/ubuntu/utils/testwf/test_wellformedness.py -t -p /home/ubuntu/utils/testwf/unittest/utc1/ut1-b/assignments
INFO  unittest ut1-c started with: /home/ubuntu/utils/testwf/test_wellformedness.py -t -p /home/ubuntu/utils/testwf/unittest/utc1/ut1-c/assignments
INFO  unittest ut1-d started with: /home/ubuntu/utils/testwf/test_wellformedness.py -t -p /home/ubuntu/utils/testwf/unittest/utc1/ut1-d/assignments

INFO  === Results for utc1 ===
INFO      Some unittests were not executed
INFO      All other unittests passed
```

Only a selection of the unit tests defined in unit test collection 1 is executed, and we see that all unit tests pass.

What happened?
For each unit test a working environment was made in the unit test folder.



```
utc1
    ut1-a
    ut1-b
    ut1-c
    ut1-d
    result_ut1-a_stdout.txt
    result_ut1-b_stdout.txt
    result_ut1-c_stdout.txt
    result_ut1-d_stdout.txt
    utc1_setup.txt
```

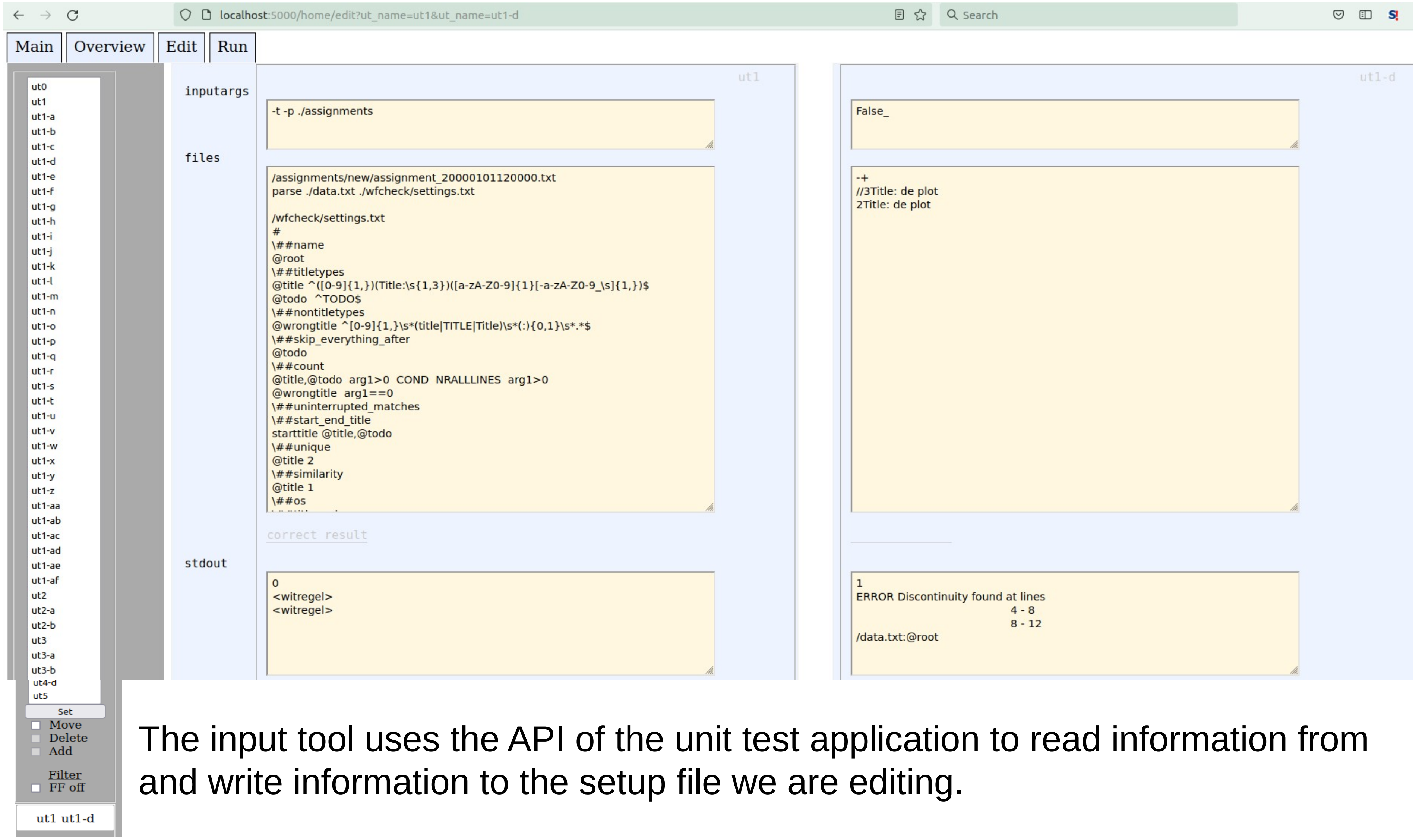
In each environment the parser is being started with specific arguments that were defined in the corresponding unit test.

The output of the parser to the terminal (stdout) is captured, and compared with the expected result.

With this unit test application we can write a lot of unit tests in quite a short time!

In particular, if we use the Flask input tool, a very handy tool to edit setup files.

In this example, in the left part of the screen ut1 is shown, and in the right part unit test variant ut1-d



ut1 — inputargs:

```
-t -p ./assignments
```

ut1 — files:

```
/assignments/new/assignment_20000101120000.txt
parse ./data.txt ./wfcheck/settings.txt

/wfcheck/settings.txt
#
\##name
@root
\##titletypes
@title ^([0-9]{1,})(Title:\s{1,3})([a-zA-Z0-9]{1}[-a-zA-Z0-9_\s]{1,})$
@todo  ^TODO$
\##nontitletypes
@wrongtitle ^[0-9]{1,}\s*(title|TITLE|Title)\s*(:){0,1}\s*.*$
\##skip_everything_after
@todo
\##count
@title,@todo  arg1>0  COND  NRALLLINES  arg1>0
@wrongtitle  arg1==0
\##uninterrupted_matches
\##start_end_title
starttitle @title,@todo
\##unique
@title 2
\##similarity
@title 1
\##os
```

ut1 — stdout:

```
0
<witregel>
<witregel>
```

ut1-d — inputargs:

```
False_
```

ut1-d — files:

```
-+
//3Title: de plot
2Title: de plot
```

ut1-d — stdout:

```
1
ERROR Discontinuity found at lines
                               4 - 8
                               8 - 12
/data.txt:@root
```

The input tool uses the API of the unit test application to read information from and write information to the setup file we are editing.