Analysis of test data from a WATS database

About me and my area of interest in data

I work as a quality engineer at a company that develops electronic products.
These products are testet and the test results collected in a WATS database.
WATS is a database system tailored at collecting and analysing data from test in productions.

Step 1 - find a dataset

Per produced product, a series of test steps are logged in the dataase together with some other information. These data can be analysed in the WATS system internaly, but it is a limited system. In order to explore the data and analyse deeper data is exported from the database.

These data are exported as zip files containing a tsv with 1 line per test and all information in columns.

The data used have been stripped from sensitive data.

In [1]:

```
from pprint import pprint
from csv import reader as csv_reader
from collections import OrderedDict
from collections import Counter
```

In [2]:

```
# tsv file extracted from zip
filename = 'data\WATSExport_UUT_2019-11-08_15-43-59\Text-export_stripped.txt'
lines_to_load = 50

# 'utf-8' is for reading unicode files.
# utf-8-sig is for reading unicode files with ByteOrderMarker as 2 first bytes in file
with open(filename, 'r', encoding='utf-8-sig') as tsvfile:
    csvreader = csv_reader(tsvfile, delimiter='\t')
    header = next(csvreader)
    # using comprehension for first runs
    # read and dict zip each line to read, to key:value pair
    # using ordered dictionary to keep order of values
    dataset = [
        OrderedDict(zip(header, next(csvreader)))
        for i in range(lines_to_load)
    ]
```

In [3]:

```
# header consist of 2453 columns
header_size = len(header)
header_size
```

Out[3]:

```
2453
```

the first approx 23 columns are basic test header data
the rest are steps in the test.
Each step has : measurement, hi limit, lo limit, unit, comp op, status, report text

In [4]:

```
header[0:32]
```

Out[4]:

```
['Serial number',
 'Part number',
 'Revision',
 'Product name',
 'Test operation',
 'Station name',
 'Location',
 'Purpose',
 'Status',
 'Date/time',
 'UTC date/time',
 'Execution time',
 'Batch serial number',
 'Test socket index',
 'Operator',
 'Fixture ID',
 'Comment',
 'Error code',
 'Error message',
 'WATS Link',
 'Misc Description 1',
 'Misc Data String 1',
 'Misc Data Numeric 1',
 '|<- 00.1 CHECK FIXTURE CODE',
 '00.1 CHECK FIXTURE CODE (High limit)',
 '00.1 CHECK FIXTURE CODE (Low limit)',
 '00.1 CHECK FIXTURE CODE (Unit)',
 '00.1 CHECK FIXTURE CODE (Comp Operator)',
 '00.1 CHECK FIXTURE CODE (Status)',
 '00.1 CHECK FIXTURE CODE (Report text) ->|',
 '|<- 00.2 AIR PRESSURE DETECT',
 '00.2 AIR PRESSURE DETECT (High limit)']
```

Step 2 - Explore

looking at first line

In [5]:

```
list(dataset[0].items())[0:32]
```

Out[5]:

```
[('Serial number', 'S90458ID001W1942'),
 ('Part number', 'Product1'),
 ('Revision', ''),
 ('Product name', ''),
 ('Test operation', 'F1 test sys'),
 ('Station name', 'Tstation6'),
 ('Location', 'SITE1'),
 ('Purpose', 'Production'),
 ('Status', 'Passed'),
 ('Date/time', '2019-11-08T21:50:31'),
 ('UTC date/time', '2019-11-08T14:50:31'),
 ('Execution time', '393'),
 ('Batch serial number', ''),
 ('Test socket index', '-1'),
 ('Operator', 'user1'),
 ('Fixture ID', '24'),
 ('Comment', ''),
 ('Error code', '0'),
 ('Error message', ''),
 ('WATS Link', 'SKYWATSLINK'),
 ('Misc Description 1', 'Product name'),
 ('Misc Data String 1', 'Product1'),
 ('Misc Data Numeric 1', '0'),
 ('|<- 00.1 CHECK FIXTURE CODE', '24'),
 ('00.1 CHECK FIXTURE CODE (High limit)', '24'),
 ('00.1 CHECK FIXTURE CODE (Low limit)', '5'),
 ('00.1 CHECK FIXTURE CODE (Unit)', '-'),
 ('00.1 CHECK FIXTURE CODE (Comp Operator)', 'GELE'),
 ('00.1 CHECK FIXTURE CODE (Status)', 'Passed'),
 ('00.1 CHECK FIXTURE CODE (Report text) ->|', 'No: 1 - 00 FIXTURE'),
 ('|<- 00.2 AIR PRESSURE DETECT', '1'),
 ('00.2 AIR PRESSURE DETECT (High limit)', '1')]
```

Looking at general parts

serial numbers can be devided into id and year-week code

In [6]:

```
dataset[0]['Serial number'],dataset[0]['Serial number'][12:]
```

Out[6]:

```
('S90458ID001W1942', '1942')
```

In [7]:

```
list(dataset[0].items())[0]
```

Out[7]:

```
('Serial number', 'S90458ID001W1942')
```

In [8]:

```
# creating new set with yearweek first in list
dataset = [
    OrderedDict([['YearWeek', d['Serial number'][12:]]] + list(d.items()) )
    for d in dataset]
header.insert(0,'YearWeek')
list(dataset[0].items())[0:32]
```

Out[8]:

```
[('YearWeek', '1942'),
 ('Serial number', 'S90458ID001W1942'),
 ('Part number', 'Product1'),
 ('Revision', ''),
 ('Product name', ''),
 ('Test operation', 'F1 test sys'),
 ('Station name', 'Tstation6'),
 ('Location', 'SITE1'),
 ('Purpose', 'Production'),
 ('Status', 'Passed'),
 ('Date/time', '2019-11-08T21:50:31'),
 ('UTC date/time', '2019-11-08T14:50:31'),
 ('Execution time', '393'),
 ('Batch serial number', ''),
 ('Test socket index', '-1'),
 ('Operator', 'user1'),
 ('Fixture ID', '24'),
 ('Comment', ''),
 ('Error code', '0'),
 ('Error message', ''),
 ('WATS Link', 'SKYWATSLINK'),
 ('Misc Description 1', 'Product name'),
 ('Misc Data String 1', 'Product1'),
 ('Misc Data Numeric 1', '0'),
 ('|<- 00.1 CHECK FIXTURE CODE', '24'),
 ('00.1 CHECK FIXTURE CODE (High limit)', '24'),
 ('00.1 CHECK FIXTURE CODE (Low limit)', '5'),
 ('00.1 CHECK FIXTURE CODE (Unit)', '-'),
 ('00.1 CHECK FIXTURE CODE (Comp Operator)', 'GELE'),
 ('00.1 CHECK FIXTURE CODE (Status)', 'Passed'),
 ('00.1 CHECK FIXTURE CODE (Report text) ->|', 'No: 1 - 00 FIXTURE'),
 ('|<- 00.2 AIR PRESSURE DETECT', '1')]
```

some columns seem to be empty
count different value pr in column
if the first items seen are empty, means that column is probably empty all the way

In [9]:

```
# collumn_to_test = [d['YearWeek'] for d in dataset]
collumn_to_test = [d['Revision'] for d in dataset]
itemcount = Counter()
itemcount.update(collumn_to_test)
itemcount
```

Out[9]:

```
Counter({'': 50})
```

```
# this is now set to RAW cell - can be swich back to CODE cell to run again

# counting all columns at once
itemcount = OrderedDict((h,Counter()) for h in header)
[itemcount[h].update(d[h] for d in dataset) for h in header];

# encounters error
# KeyError: ''01.01.14 Measure Voltage on CLM Input (High limit)''
# switch to loop method for finding and elimination
```

```
# this is now set to RAW cell - can be swich back to CODE cell to run again

# counting all columns at once
# take II
itemcount = OrderedDict((h,Counter()) for h in header)

# doing single line count to find error...
previous_field = ''
for hi,h in enumerate(header):
    columndata = []
    for di,d in enumerate(dataset):
        try:
            columndata.append(d[h])
        except Exception:
            raise Exception(f'Err in line no {di}, field no {hi}, {h}, prev
{previous_field}')
    previous_field = h
    itemcount[h].update(columndata)
# Error message
# Exception: Err in line no 48, field no 116, 01.01.14 Measure Voltage on CLM Input (High
limit)
# adding previous field to find the spot
# Err in line no 48, field no 116, 01.01.14 Measure Voltage on CLM Input (High limit),
#   prev |<- 01.01.14 Measure Voltage on CLM Input
```

```
# this is now set to RAW cell - can be swich back to CODE cell to run again

# looking at failing line
dataset[48]
```

```
# this is now set to RAW cell - can be swich back to CODE cell to run again

hmmm
When test fails, the rest of the header is not added - it is not there.
('Serial number', 'S89487ID001W1942'),
...
('Status', 'Failed'),
...
('|<- 01.01.12 Measure Voltage on XI7=L', '5.052'),
('01.01.12 Measure Voltage on XI7=L (High limit)', '5.05'),
('01.01.12 Measure Voltage on XI7=L (Low limit)', '4.95'),
('01.01.12 Measure Voltage on XI7=L (Unit)', 'V'),
('01.01.12 Measure Voltage on XI7=L (Comp Operator)', 'GELE'),
('01.01.12 Measure Voltage on XI7=L (Status)', 'Failed'),
('01.01.12 Measure Voltage on XI7=L (Report text) ->|',
'No: 13 - 01 PS TEST'),
('|<- 01.01.14 Measure Voltage on CLM Input', '')])
```

In [10]:

```
# counting all columns at once
# take III
# attempt at finding all dataset with missing - aka those with failures
itemcount = OrderedDict((h,Counter()) for h in header)

# doing single line count to find error...

for di,d in enumerate(dataset):
    for hi,h in enumerate(header):
        try:
            test = d[h]
        except Exception:
            print(f'ERROR in dataset {di}({d["Serial number"]}={d["Status"]}), field no {hi
            break
```

```
ERROR in dataset 28(S90188ID001W1942=Failed), field no 704, 04.02.04 Test AI
_1 (High limit)
ERROR in dataset 45(S90125ID001W1942=Failed), field no 2083, 08.13.05 Measur
e voltage TA1-TA2 (High limit)
ERROR in dataset 48(S89487ID001W1942=Failed), field no 116, 01.01.14 Measure
Voltage on CLM Input (High limit)
```

In [11]:

```
# counting all columns at once
# take IV
# if the previous test step has failed this step will not have data
# and an error is encountered.
# thsese data are the ignored
itemcount = OrderedDict((h,Counter()) for h in header)
for hi,h in enumerate(header):
    columndata = []
    for di,d in enumerate(dataset):
        try:
            columndata.append(d[h])
        except Exception:
            pass
    itemcount[h].update(columndata)
```

In [12]:

```
itemcount['Status']
```

Out[12]:

```
Counter({'Passed': 47, 'Failed': 3})
```

In [13]:

```
# first findings said Counter({'24': 181, '18': 146, '5': 172, '': 1})
# '':1 needs to be cleaned out or investigated?
itemcount['Fixture ID']
```

Out[13]:

```
Counter({'24': 17, '18': 16, '5': 17})
```

In [14]:

```
[d for d in dataset if d['Fixture ID'] == '']
# so valid after all - just failed in first step as it was not possible to get fixture ID
```

Out[14]:

```
[]
```

In [15]:

```
# print columns which has no data
# amount of empty value are listed
for h in header:
    if itemcount[h][''] > 0:
        print('"',h,'"',':',itemcount[h][''],':')
# 200 means no data at all
# 1 means that 1 line in that column has empty value
```

```
" Revision " : 50 :
" Product name " : 50 :
" Batch serial number " : 50 :
" Comment " : 50 :
" Error message " : 50 :
" |<- 01.01.14 Measure Voltage on CLM Input " : 1 :
" |<- 04.02.04 Test AI_1 " : 1 :
" |<- 08.13.05 Measure voltage TA1-TA2 " : 1 :
"  " : 47 :
```

In [16]:

```
# Fields with only 1 value
singleitemlist = [(h,len(itemcount[h])) for h in header if len(itemcount[h]) == 1]
f'{len(singleitemlist)} of {header_size} has only 1 value'
```

Out[16]:

```
'2239 of 2453 has only 1 value'
```

In [17]:

```
# Fields with multiple values
multiitemlist = [(h,len(itemcount[h])) for h in header if len(itemcount[h]) > 1]
print(f'{len(multiitemlist)} of {header_size} has multiple values')
```

```
215 of 2453 has multiple values
```

In [18]:

```
# sort on number of occurance of item
# multiitemlist.sort(key=lambda item: item[1], reverse=True)
multiitemlist[0:20]
```

Out[18]:

```
[('Serial number', 47),
 ('Station name', 3),
 ('Status', 2),
 ('Date/time', 50),
 ('UTC date/time', 50),
 ('Execution time', 10),
 ('Fixture ID', 3),
 ('|<- 00.1 CHECK FIXTURE CODE', 3),
 ('|<- 00.2 AIR PRESSURE DETECT', 2),
 ('|<- 01.01.03 Measure Voltage on CN17.1', 46),
 ('|<- 01.01.04 Measure Voltage on TP30', 47),
 ('|<- 01.01.05 Measure Voltage on CN17.17', 29),
 ('|<- 01.01.06 Measure Voltage on TP26', 41),
 ('|<- 01.01.11 Measure Voltage on CN17.3 - PC16 = H', 4),
 ('|<- 01.01.12 Measure Voltage on XI7=L', 34),
 ('01.01.12 Measure Voltage on XI7=L (Status)', 2),
 ('|<- 01.01.14 Measure Voltage on CLM Input', 46),
 ('|<- 01.01.17 Measure Voltage on CN17.1', 26),
 ('|<- 01.01.19 Measure Voltage on CN17.1 - PA8 = L', 38),
 ('|<- 01.01.22 Measure battery charge PA19 = L', 2)]
```

Looking at test step part

Each step has : header, hi limit, lo limit, unit, comp op, status, report text.
steps start at field 24 and has each 7 fields

In [19]:

```
# test listning of data
#from first teststep(24) and 2 steps (+7+7)
list(dataset[0].items())[24:24+7+7]
```

Out[19]:

```
[('|<- 00.1 CHECK FIXTURE CODE', '24'),
 ('00.1 CHECK FIXTURE CODE (High limit)', '24'),
 ('00.1 CHECK FIXTURE CODE (Low limit)', '5'),
 ('00.1 CHECK FIXTURE CODE (Unit)', '-'),
 ('00.1 CHECK FIXTURE CODE (Comp Operator)', 'GELE'),
 ('00.1 CHECK FIXTURE CODE (Status)', 'Passed'),
 ('00.1 CHECK FIXTURE CODE (Report text) ->|', 'No: 1 - 00 FIXTURE'),
 ('|<- 00.2 AIR PRESSURE DETECT', '1'),
 ('00.2 AIR PRESSURE DETECT (High limit)', '1'),
 ('00.2 AIR PRESSURE DETECT (Low limit)', '0'),
 ('00.2 AIR PRESSURE DETECT (Unit)', '-'),
 ('00.2 AIR PRESSURE DETECT (Comp Operator)', 'GELE'),
 ('00.2 AIR PRESSURE DETECT (Status)', 'Passed'),
 ('00.2 AIR PRESSURE DETECT (Report text) ->|', 'No: 2 - 00 FIXTURE')]
```

In [20]:

```
# setup range in dataset
teststep = 2
teststepstart = 24 + teststep * 7
teststepend = teststepstart + 7
```

In [21]:

```
# look at headers for range
testheaders = header[teststepstart:teststepend]
testheaders
```

Out[21]:

```
['|<- 00.3 FIXTURE CLOSED OR CANCELED',
 '00.3 FIXTURE CLOSED OR CANCELED (High limit)',
 '00.3 FIXTURE CLOSED OR CANCELED (Low limit)',
 '00.3 FIXTURE CLOSED OR CANCELED (Unit)',
 '00.3 FIXTURE CLOSED OR CANCELED (Comp Operator)',
 '00.3 FIXTURE CLOSED OR CANCELED (Status)',
 '00.3 FIXTURE CLOSED OR CANCELED (Report text) ->|']
```

In [22]:

```
# better single test headers
#split text to list by comma
testheaders = 'Measured,High limit,Low limit,Unit,CompOperator,Status,ReportText'.split(','
testheaders
```

Out[22]:

```
['Measured',
 'High limit',
 'Low limit',
 'Unit',
 'CompOperator',
 'Status',
 'ReportText']
```

In [23]:

```
# single line of data
dataset_test = dataset[0]
# single teststep lines
[di for di in list(dataset_test.items())[teststepstart:teststepend]]
```

Out[23]:

```
[('|<- 00.3 FIXTURE CLOSED OR CANCELED', '1'),
 ('00.3 FIXTURE CLOSED OR CANCELED (High limit)', '1'),
 ('00.3 FIXTURE CLOSED OR CANCELED (Low limit)', '1'),
 ('00.3 FIXTURE CLOSED OR CANCELED (Unit)', '-'),
 ('00.3 FIXTURE CLOSED OR CANCELED (Comp Operator)', 'GELE'),
 ('00.3 FIXTURE CLOSED OR CANCELED (Status)', 'Passed'),
 ('00.3 FIXTURE CLOSED OR CANCELED (Report text) ->|', 'No: 3 - 00 FIXTUR
E')]
```

In [24]:

```
# data only
[di[1] for di in list(dataset_test.items())[teststepstart:teststepend]]
```

Out[24]:

```
['1', '1', '1', '-', 'GELE', 'Passed', 'No: 3 - 00 FIXTURE']
```

In [25]:

```
# combine to dict
teststeps = dict(zip(testheaders, [di[1] for di in list(dataset_test.items())[teststepstart
teststeps
```

Out[25]:

```
{'Measured': '1',
 'High limit': '1',
 'Low limit': '1',
 'Unit': '-',
 'CompOperator': 'GELE',
 'Status': 'Passed',
 'ReportText': 'No: 3 - 00 FIXTURE'}
```

In [26]:

```
#putting it together
teststeps_start = 24
teststeps_offset = 7
teststeps_count = 10

teststeps_end = len(header)

dataset_teststeps = []
for data in dataset:
    teststeps = OrderedDict()
    for index in range(teststeps_start, teststeps_end, teststeps_offset):
        stepname = header[index]
        # print(stepname)
        full_data_list = list(data.items())
        measdata = [d[1] for d in full_data_list[index:index+teststeps_offset]]
        stepdata = OrderedDict(zip(testheaders, measdata))
        # pprint(stepdata)
        teststeps[stepname] = stepdata
    dataset_teststeps.append(teststeps)
```

In [27]:

```
# listing test step names
pprint(list(dataset_teststeps[5].keys())[0:4])
```

```
['|<- 00.1 CHECK FIXTURE CODE',
 '|<- 00.2 AIR PRESSURE DETECT',
 '|<- 00.3 FIXTURE CLOSED OR CANCELED',
 '|<- 01.01.03 Measure Voltage on CN17.1']
```

In [28]:

```
#Test dataset of teststeps
# find all measurements and of air pressure detected
air_pres_meas = [d['|<- 00.2 AIR PRESSURE DETECT']['Measured'] for d in dataset_teststeps[0
air_pres_status = [d['|<- 00.2 AIR PRESSURE DETECT']['Status'] for d in dataset_teststeps[0
```

In [29]:

```
count_meas = Counter()
count_status = Counter()
count_meas.update(air_pres_meas)
count_status.update(air_pres_status)
count_meas, count_status
```

Out[29]:

```
(Counter({'1': 34, '0': 16}), Counter({'Passed': 50}))
```

```
air pressure is detected in only 101 but all 150 are pass???
(Counter({'1': 101, '0': 49}), Counter({'Passed': 150}))
Maybe investigate...
```

Step 3 - Areas of interest

Before diving into the analysis, the data is reloaded and cleanup and transformations are done at loading. And a larger portion of data is loaded.

In [30]:

```
from pprint import pprint
from csv import reader as csv_reader
from collections import OrderedDict
from collections import Counter
```

In [31]:

```
# start fresh with load of data after all cleanup and transformation has been implemented
filename = 'data\WATSExport_UUT_2019-11-08_15-43-59\Text-export_stripped.txt'
lines_to_load = 1000
# Generel fields end and teststep starts at column 23
teststeps_start = 23
teststeps_offset = 7

with open(filename, 'r', encoding='utf-8-sig') as tsvfile:
    csvreader = csv_reader(tsvfile, delimiter='\t')
    header = next(csvreader)
    # split header in generel and teststeps

    # generel header
    header_generel = header[0:teststeps_start]
    # adding yearweek after serialnr 'S90458ID468W1942' -> '1942'
    header_generel.insert(1,'YearWeek')

    header_generel_size = len(header_generel)

    # teststep hader
    header_steps = header[teststeps_start:]
    header_steps_size = len(header_steps)
    testheaders = 'Measured,high limit,Low limit,Unit,CompOperator,Status,ReportText'.split


    dataset = []
    for i in range(lines_to_load):
        data = next(csvreader)
        # split in test headers and test steps

        # Generel data
        data_generel = data[0:teststeps_start]
        # adding yearweek after serialnr 'S90458ID468W1942' -> '1942'
        data_generel.insert(1,data[0][12:])


        # test step data
        data_steps = data[teststeps_start:]
        teststeps = OrderedDict()
        for index in range(0, header_steps_size, teststeps_offset):
            stepname = header_steps[index]
            # print(stepname)
            measdata = [d for d in data_steps[index:index+teststeps_offset]]
            stepdata = OrderedDict(zip(testheaders, measdata))
            # pprint(stepdata)
            teststeps[stepname] = stepdata

        # adding teststeps to generel data
        data2 = data_generel + [teststeps]
        header2 = header_generel + ['Test Steps']

        # creating main dataset
        d = OrderedDict(zip(header2, data2))
        dataset.append(d)
```

```
dataset[0]
```

In [32]:

```
# counting all columns at once
itemcount = OrderedDict((h,Counter()) for h in header_generel)

for hi,h in enumerate(header_generel):
    columndata = []
    for di,d in enumerate(dataset):
        try:
            columndata.append(d[h])
        except Exception:
            pass
    itemcount[h].update(columndata)
```

In [33]:

```
# Fields with multiple values
multiitemlist = [(h,len(itemcount[h])) for h in header_generel if len(itemcount[h]) > 1]
print(f'{len(multiitemlist)} of {header_generel_size} has multiple values')
pprint(multiitemlist)
```

```
9 of 24 has multiple values
[('Serial number', 763),
 ('YearWeek', 6),
 ('Station name', 4),
 ('Status', 2),
 ('Date/time', 995),
 ('UTC date/time', 995),
 ('Execution time', 48),
 ('Operator', 8),
 ('Fixture ID', 4)]
```

The list of fields with multiple items, with more than 1 occurance are interesting for analysis.

The field from 'Serial Number' to 'Fixtue ID' are generel on the test, whereas the rest is specific to teststeps.

Both sets of data are interesting and will be analyzed.

Analysis of generel fields

These are the general parts that might be interesting.

- ('YearWeek', 4),
- ('Station name', 3),
- ('Status', 2),
- ('Execution time', 28),
- ('Operator', 6),
- ('Fixture ID', 4),
- ('WATS Link', 500),

In [34]:

```
# the actual serial number column is not so interesting, so we just look at
itemcount['YearWeek']
# why are there older items?
```

Out[34]:

```
Counter({'1942': 980, '1508': 3, '1934': 5, '1925': 2, '1621': 1, '1939':
9})
```

In [35]:

```
#date span of tests in list
dataset[0]['Date/time'], dataset[-1]['Date/time']
# 1925+1934 are probably retests due to repair
# but 1508?
```

Out[35]:

```
('2019-11-08T21:50:31', '2019-10-25T09:43:07')
```

In [36]:

```
# 07-11-2019 is week 45, so 1942 is newest and the others(1925+1934) probably retests due t
# but 1508 is a very old one - why is it tested?
# maybe this was a returned defect, to be tested ?
investigate = [d for d in dataset if d['YearWeek']== '1508']
[(d['Serial number'],d['Status']) for d in investigate]
# not investigated further here.
```

Out[36]:

```
[('S13978ID001W1508', 'Failed'),
 ('S13978ID001W1508', 'Passed'),
 ('S13978ID001W1508', 'Failed')]
```

In [37]:

```
# the tests are spread on 4 stations stations
itemcount['Station name']
```

Out[37]:

```
Counter({'Tstation6': 253,
         'Tstation8': 363,
         'Tstation5': 236,
         'Tstation7': 148})
```

In [38]:

```
# crossing with status to see if any station is more prone to error than others
# count failed/pass pr station
status_count = Counter()
status_count.update((d['Status'],d['Station name']) for d in dataset)

for station in itemcount['Station name'].keys():
    failpct = status_count[('Failed', station)]/status_count[('Passed', station)]
    print(f'{station} has {failpct:0.3f}% fail')

# Station 6 has a little more fails than others...
```

```
Tstation6 has 0.140% fail
Tstation8 has 0.058% fail
Tstation5 has 0.073% fail
Tstation7 has 0.057% fail
```

In [39]:

```
itemcount['Operator']
```

Out[39]:

```
Counter({'user1': 78,
         'user2': 267,
         'user3': 73,
         'user4': 15,
         'user5': 42,
         'user6': 395,
         'user7': 74,
         'user8': 56})
```

In [40]:

```
# let do same test for fails pr operator
# crossing with status to see if any station is more prone to error than others
# count failed/pass pr station
status_count = Counter()
status_count.update((d['Status'],d['Operator']) for d in dataset)

for op in itemcount['Operator'].keys():
    failpct = status_count[('Failed', op)]/status_count[('Passed', op)]
    print(f'{op} has {failpct:0.3f}% fail')
# 2 user has a little more failure than the others
# It might be the users who handles the repairs
```

```
user1 has 0.083% fail
user2 has 0.077% fail
user3 has 0.043% fail
user4 has 0.000% fail
user5 has 0.135% fail
user6 has 0.088% fail
user7 has 0.057% fail
user8 has 0.120% fail
```

In [41]:

```
itemcount['Fixture ID']
# it was seen in Explore section that '' was due to failure in reading fixture ID.
# this product is probably being tested again.
```

Out[41]:

```
Counter({'24': 253, '18': 362, '5': 384, '': 1})
```

In [42]:

```
# about retesting...
# lets look at how many serial numbers have been retested?
trial_count = 0
total_count = 0
passed_n = []
for passnum in range(1,6):
    passcount =  len([d for d in itemcount['Serial number'].items() if d[1] == passnum])
    passed_n.append(passcount)
    total_count += passcount
    trial_count += passcount * passnum

    print(f'Passed after {passnum} try : {passcount}')
print(f'A total of {total_count} products have been testet {trial_count} times')
```

```
Passed after 1 try : 563
Passed after 2 try : 166
Passed after 3 try : 31
Passed after 4 try : 3
Passed after 5 try : 0
A total of 763 products have been testet 1000 times
```

In [43]:

```
itemcount['Execution time']
```

Out[43]:

```
Counter({'393': 220,
         '392': 83,
         '394': 253,
         '395': 173,
         '396': 58,
         '397': 50,
         '391': 17,
         '211': 4,
         '212': 2,
         '199': 5,
         '213': 5,
         '398': 41,
         '209': 4,
         '214': 2,
         '200': 1,
         '197': 3,
         '198': 2,
         '208': 6,
         '123': 1,
         '122': 1,
         '11': 3,
         '43': 1,
         '390': 1,
         '46': 1,
         '45': 1,
         '454': 1,
         '411': 1,
         '195': 1,
         '194': 1,
         '206': 1,
         '400': 4,
         '-85980': 3,
         '402': 1,
         '192': 12,
         '205': 1,
         '193': 2,
         '191': 1,
         '10': 1,
         '399': 20,
         '462': 1,
         '222': 2,
         '221': 1,
         '220': 1,
         '218': 3,
         '219': 1,
         '160': 1,
         '12': 1,
         '210': 1})
```

Analysis of parts of test steps

The test steps are mainly numerical measured values

In [44]:

```
# list all steps in a test
teststeps = dataset[0]['Test Steps']
testnames = [name for name,tests in teststeps.items()]
print(f'There are {len(teststeps)} steps in test')
pprint(testnames[0:5])
```

```
There are 348 steps in test
['|<- 00.1 CHECK FIXTURE CODE',
 '|<- 00.2 AIR PRESSURE DETECT',
 '|<- 00.3 FIXTURE CLOSED OR CANCELED',
 '|<- 01.01.03 Measure Voltage on CN17.1',
 '|<- 01.01.04 Measure Voltage on TP30']
```

In [45]:

```
# lets find the step with most failure
data_with_fails = [d for d in dataset if d['Status'] == 'Failed']
print(f'{len(data_with_fails)} fails of {len(dataset)} total')
```

```
75 fails of 1000 total
```

In [46]:

```
# probing data to see format
data_with_fails[0]['Test Steps']['|<- 02.01.17 NPCS, OPT1 H']['Status']
```

Out[46]:

```
'Passed'
```

In [47]:

```
# Counter only counts. I need the serials to go with for identification
# so yousing defaultdict to find all and the manualy count.
from collections import defaultdict
steps_that_fails = defaultdict(list)
steps_no_status = []

for d in data_with_fails:
    # print(d['Serial number'])
    for stepname in testnames:
        try:
            status = d['Test Steps'][stepname]['Status']
        except:
            status = ''
            #print('No status' , stepname)
        #print(status)
        if status == 'Failed':
            steps_that_fails[stepname].append(d['Serial number'])
            break
        if not status:
            steps_no_status.append(stepname)
            break
            #print(f'Failed in {stepname}')
print(len(steps_that_fails))
print(len(steps_no_status))
count_fails = sorted([(name,len(sers)) for name,sers in steps_that_fails.items()],key=lambd
pprint(count_fails[0:5])
```

```
22
1
[('|<- 01.01.07.01 Chip Erase', 18),
 ('|<- 03.03.02.02 Test comm.RS232 DEBUG UART1 - READ', 14),
 ('|<- 08.13.04 Read E', 6),
 ('|<- 12.02 Programming Xmodem 1', 5),
 ('|<- 01.01.06 Measure Voltage on TP26', 4)]
```

In [48]:

```
# first with this fail
steps_that_fails['|<- 01.01.07.01 Chip Erase'][0]
```

Out[48]:

```
'S89755ID001W1942'
```

In [49]:

```
test = [d for d in data_with_fails if d['Serial number'] == 'S89755ID001W1942'][0]
```

In [50]:

```
test['Test Steps']['|<- 01.01.07.01 Chip Erase']
# this is some digital test that can only be 0 o1 1
```

Out[50]:

```
OrderedDict([('Measured', '0'),
             ('high limit', '1'),
             ('Low limit', '1'),
             ('Unit', '-'),
             ('CompOperator', 'GELE'),
             ('Status', 'Failed'),
             ('ReportText', 'No: 8 - 01 PS TEST')])
```

Step 4 - visualisations

In [51]:

```
import matplotlib.pyplot as plt
from matplotlib import colors
from matplotlib.dates import DateFormatter
import numpy as np
```

In [52]:

```
# it is interesting to see this
itemcount['Execution time']
```

Out[52]:

```
Counter({'393': 220,
         '392': 83,
         '394': 253,
         '395': 173,
         '396': 58,
         '397': 50,
         '391': 17,
         '211': 4,
         '212': 2,
         '199': 5,
         '213': 5,
         '398': 41,
         '209': 4,
         '214': 2,
         '200': 1,
         '197': 3,
         '198': 2,
         '208': 6,
         '123': 1,
         '122': 1,
         '11': 3,
         '43': 1,
         '390': 1,
         '46': 1,
         '45': 1,
         '454': 1,
         '411': 1,
         '195': 1,
         '194': 1,
         '206': 1,
         '400': 4,
         '-85980': 3,
         '402': 1,
         '192': 12,
         '205': 1,
         '193': 2,
         '191': 1,
         '10': 1,
         '399': 20,
         '462': 1,
         '222': 2,
         '221': 1,
         '220': 1,
         '218': 3,
         '219': 1,
         '160': 1,
         '12': 1,
         '210': 1})
```

In [53]:

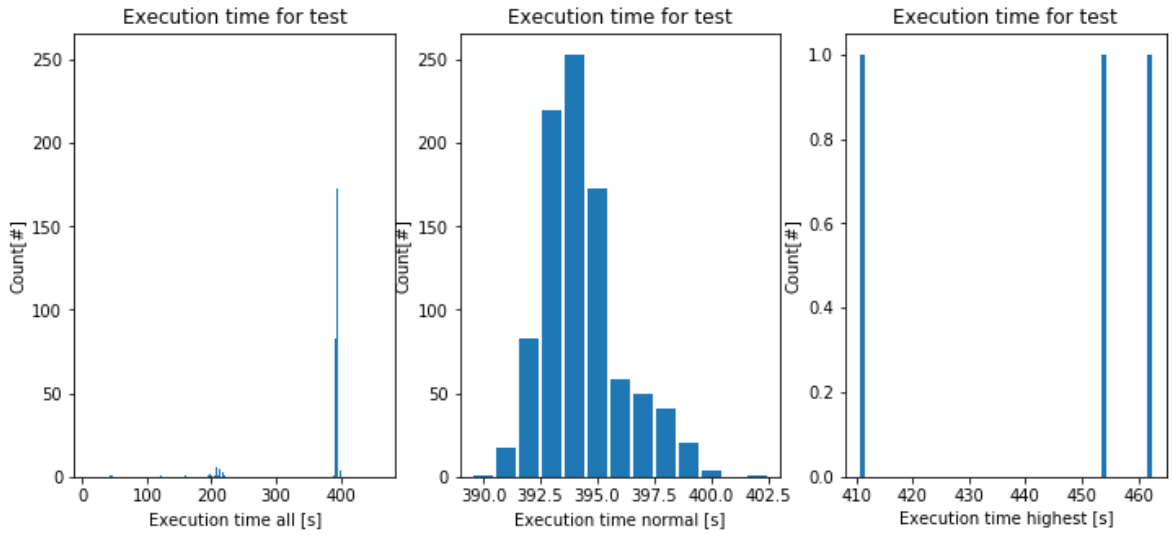
```
# execution times
# above zero to remove "-85980"
# this will be total time record
x1,y1 = zip(*[(x,y) for x,y in itemcount['Execution time'].items() if float(x)>0 ])

# above 250 to only see passed
# below 405 to disregard higher outliers
# These will be the passed
x2,y2 = zip(*[(x,y) for x,y in itemcount['Execution time'].items() if float(x)>250 and floa

x3,y3 = zip(*[(x,y) for x,y in itemcount['Execution time'].items() if float(x)>405 ])
```

In [54]:

```
x1 = np.array(x1,float)
y1 = np.array(y1,float)
x2 = np.array(x2,float)
y2 = np.array(y2,float)
x3 = np.array(x3,float)
y3 = np.array(y3,float)
```

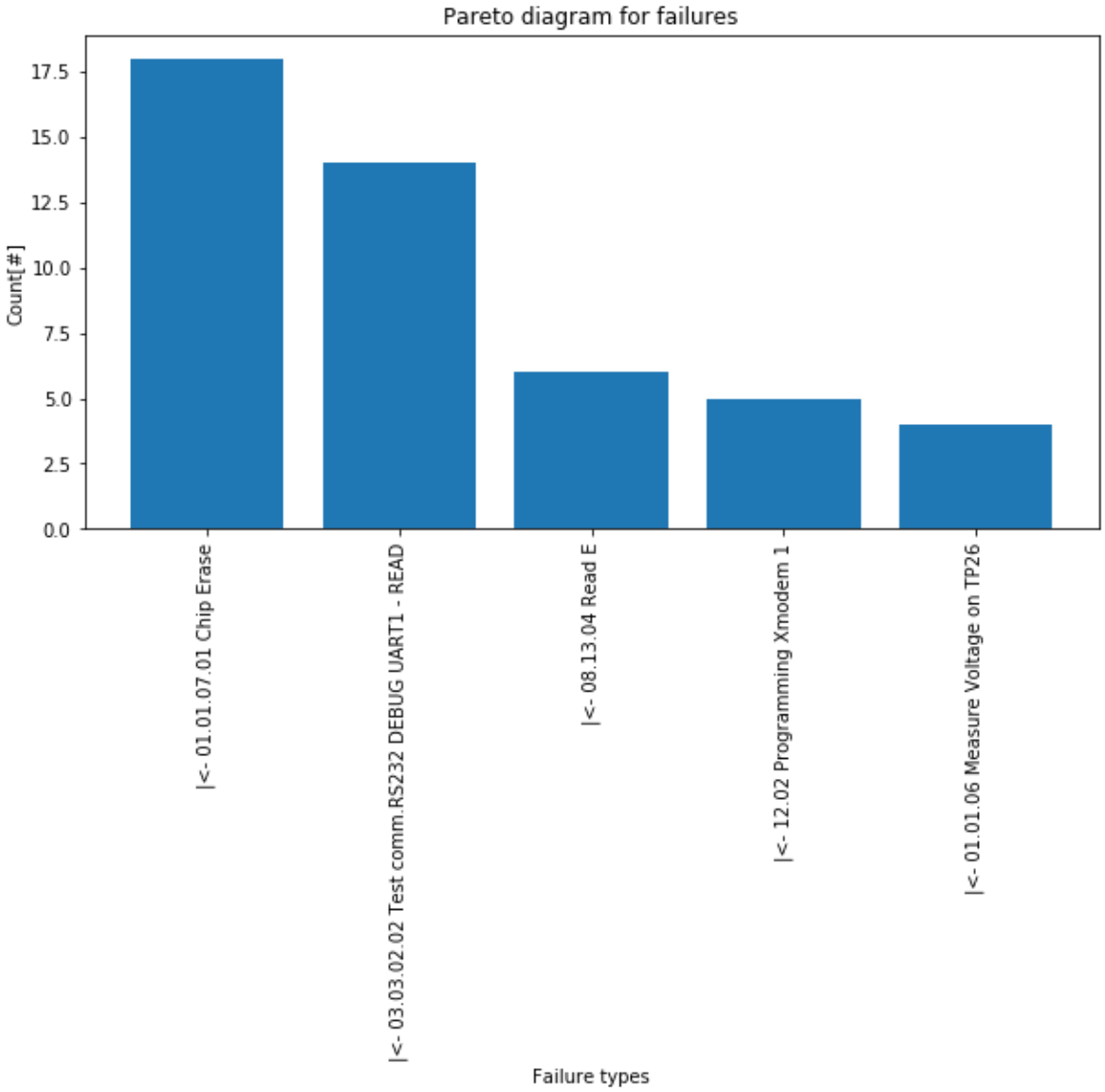
In [55]:

```
plt.figure(figsize=(12,5));
plt.subplot(1,3,1);
plt.bar(x1,y1);
plt.gca().set(
    xlabel='Execution time all [s]',
    ylabel='Count[#]',
    title='Execution time for test',
);
plt.subplot(1,3,2);
plt.bar(x2,y2);
plt.gca().set(
    xlabel='Execution time normal [s]',
    ylabel='Count[#]',
    title='Execution time for test',
);
plt.subplot(1,3,3);
plt.bar(x3,y3);
plt.gca().set(
    xlabel='Execution time highest [s]',
    ylabel='Count[#]',
    title='Execution time for test',
);

# The majority lies around 395, with some outliers
# the ones below are failed
# The higher oens are probably repairs where test is haltet for longer periods of time.
```



In [56]:

```
# Creating a pareto of which faisl are seen at most
xlabels,failcount = zip(*count_fails[0:5])
plt.figure(figsize=(10,5));
plt.bar(xlabels,failcount);
plt.gca().set(
    title='Pareto diagram for failures',
    xlabel='Failure types',
    ylabel='Count[#]',
);
plt.xticks(rotation='vertical');
```



In [57]:

```
# there are a couple of failures on
# |<- 01.01.06 Measure Voltage on TP26
# Lets see the measurements
stepname = '|<- 01.01.06 Measure Voltage on TP26'
d = dataset[0]
ts = d['Test Steps']
ts_010106 = ts[stepname]
ts_010106_meas = ts_010106['Measured']
ts_010106_unit = ts_010106['Unit']
ts_010106_meas, ts_010106_unit
```

Out[57]:

```
('5.193', 'V')
```

In [58]:

```
dataset[0]['Date/time']
```

Out[58]:

```
'2019-11-08T21:50:31'
```

In [59]:

```
# create a list of date, measurement, high and low limit
ts_010106_meas_list = [
    (x,y,h,l) for x,y,h,l in [
        (
            d['Date/time'],
            d['Test Steps'][stepname]['Measured'],
            d['Test Steps'][stepname]['high limit'],
            d['Test Steps'][stepname]['Low limit'],
        )
        if len(d['Test Steps'][stepname]) > 2
        # and d['Status'] == 'Passed'
        else (d['Date/time'],None,None,None)
        for d in dataset
    ]
    if y is not None
]
```

In [60]:

```
# probe the list - looks fine
ts_010106_meas_list[0:5]
```

Out[60]:

```
[('2019-11-08T21:50:31', '5.193', '5.5', '4.95'),
 ('2019-11-08T21:49:08', '5.201', '5.5', '4.95'),
 ('2019-11-08T21:48:11', '5.188', '5.5', '4.95'),
 ('2019-11-08T21:43:44', '5.204', '5.5', '4.95'),
 ('2019-11-08T21:42:19', '5.071', '5.5', '4.95')]
```

In [61]:

```
# test date conversion in Numpy - works
dt = np.dtype('M')
x = np.array([dataset[0]['Date/time'],], dtype=dt)
x
```

Out[61]:

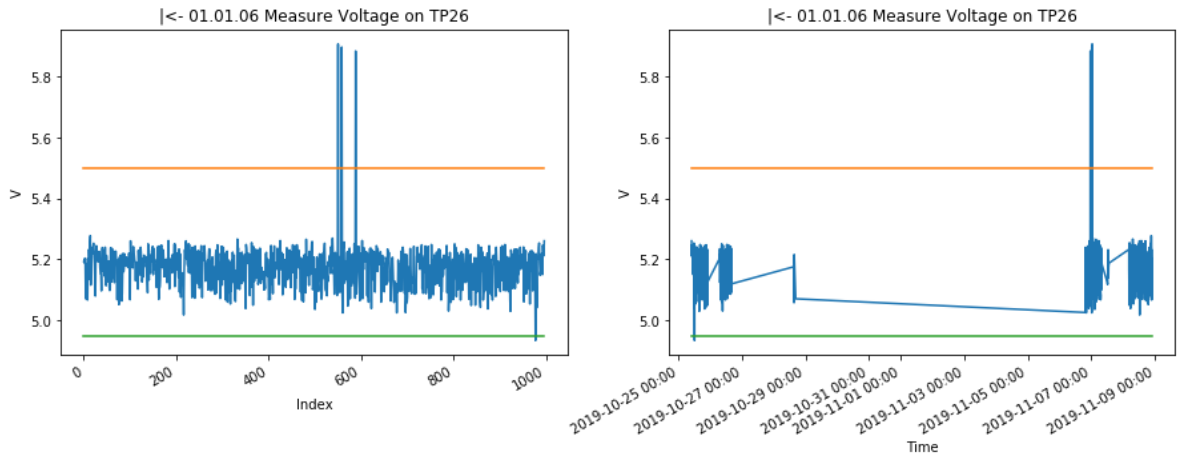
```
array(['2019-11-08T21:50:31'], dtype='datetime64[s]')
```

In [62]:

```
# create simple indexlist for plotting
idx = range(len(ts_010106_meas_list))
# create numpy arrays for plotting
x = np.array([xy[0] for xy in ts_010106_meas_list], np.datetime64)
y = np.array([xy[1] for xy in ts_010106_meas_list], np.float)
h = np.array([xy[2] for xy in ts_010106_meas_list], np.float)
l = np.array([xy[3] for xy in ts_010106_meas_list], np.float)
```

In [63]:

```
# just to see how the measurements are in generel
# only plotting meausured value on index
myFmt = DateFormatter("%Y-%m-%d %H:%M")
plt.figure(figsize=(15,5));
#plt.plot([idx,idx,idx],[y,h,l]);
plt.subplot(1,2,1);
plt.plot(y);
plt.gca().set(
    title=stepname,
    xlabel='Index',
    ylabel=ts_010106_unit,
);
plt.plot(h);
plt.plot(l);

plt.subplot(1,2,2);
plt.plot(x,y);
plt.gca().xaxis.set_major_formatter(myFmt)
plt.gca().set(
    title=stepname,
    xlabel='Time',
    ylabel=ts_010106_unit,
);
plt.plot(x, h);
plt.plot(x, l);

## Rotate date labels automatically
plt.gcf().autofmt_xdate()
# Except for fails outside limits, it looks ok from index plot
# from date plot we see that tests happens in blocks.
```



In [64]:

```
# lets group in date-blocks
np_data = list(zip(x,y))

segment1 = [
    (xx,yy) for xx,yy in np_data
    if xx < np.datetime64('2019-10-26T03:00:00')
]
segment2 = [
    (xx,yy) for xx,yy in np_data
    if xx > np.datetime64('2019-10-26T03:00:00')
    and xx < np.datetime64('2019-10-27T06:00:00')
]
segment3 = [
    (xx,yy) for xx,yy in np_data
    if xx > np.datetime64('2019-11-05T00:00:00')
    and xx < np.datetime64('2019-11-07T10:00:00')
]
segment4 = [
    (xx,yy) for xx,yy in np_data
    if xx > np.datetime64('2019-11-08T10:00:00')
    and xx < np.datetime64('2019-11-11T00:00:00')
]
```

In [65]:

```
segment1[0][0] < np.datetime64('2019-11-09T21:50:31')
```

Out[65]:

```
True
```

In [66]:

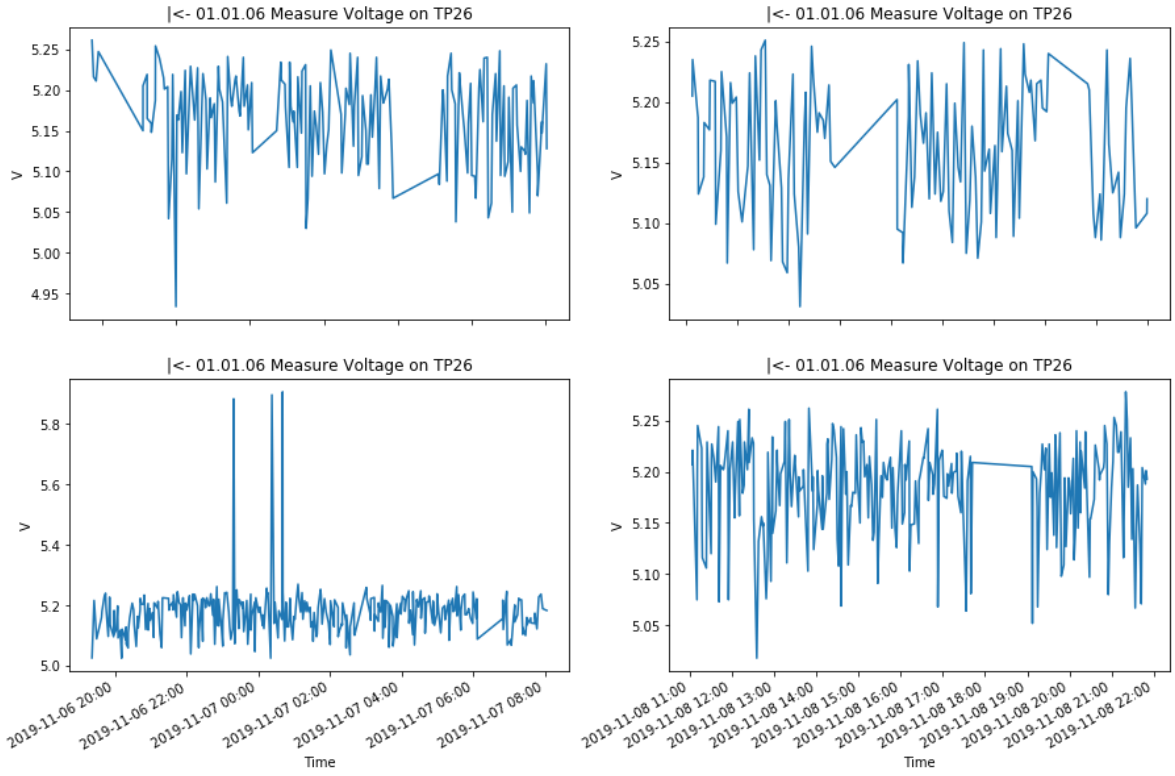
```
#list(zip(*segment2))[1][0:5]
segment3[0]
```

Out[66]:

```
(numpy.datetime64('2019-11-07T08:02:55'), 5.183)
```

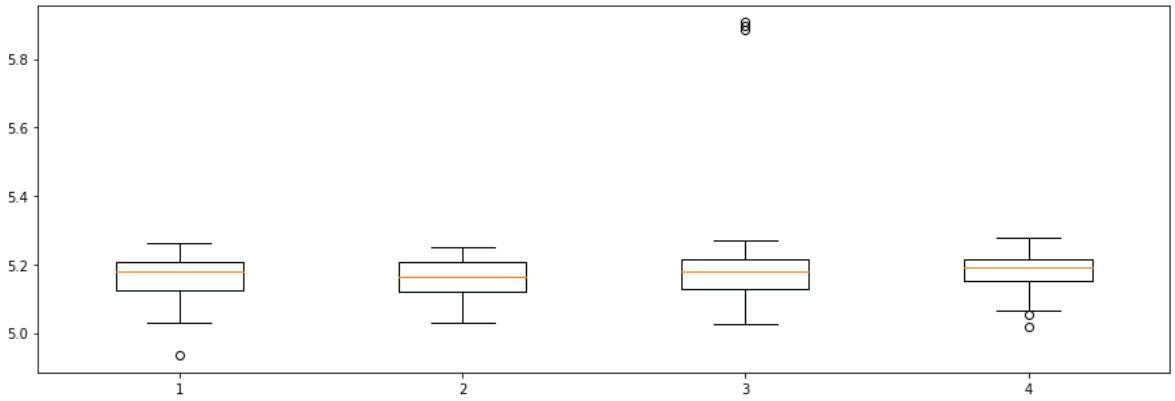
In [67]:

```
# just to see how the measurements are in generel
# only plotting meausured value on index
x1,y1 = list(zip(*segment1))
x2,y2 = list(zip(*segment2))
x3,y3 = list(zip(*segment3))
x4,y4 = list(zip(*segment4))

myFmt = DateFormatter("%Y-%m-%d %H:%M")
plt.figure(figsize=(15,10));

plt.subplot(2,2,1);
plt.plot(x1,y1);
plt.gca().set(
    title=stepname,
    xlabel='Time',
    ylabel=ts_010106_unit,
);
plt.gca().xaxis.set_major_formatter(myFmt)
plt.gcf().autofmt_xdate()
plt.subplot(2,2,2);
plt.plot(x2,y2);
plt.gca().set(
    title=stepname,
    xlabel='Time',
    ylabel=ts_010106_unit,
);
plt.gca().xaxis.set_major_formatter(myFmt)
plt.gcf().autofmt_xdate()
plt.subplot(2,2,3);
plt.plot(x3,y3);
plt.gca().set(
    title=stepname,
    xlabel='Time',
    ylabel=ts_010106_unit,
);
plt.gca().xaxis.set_major_formatter(myFmt)
plt.gcf().autofmt_xdate()
plt.subplot(2,2,4);
plt.plot(x4,y4);
plt.gca().set(
    title=stepname,
    xlabel='Time',
    ylabel=ts_010106_unit,
);
plt.gca().xaxis.set_major_formatter(myFmt)

## Rotate date labels automatically
plt.gcf().autofmt_xdate()
```



In [68]:

```
# creating boxplot of measuredata of the 4 date ranges
plt.figure(figsize=(15,5))
plt.boxplot([y1,y2,y3,y4]);
# except for the failed (outliers) thay all look nicely equal
#this means that they are produced and tested in same way.
```



In [ ]:

```

```